

MC event generation tutorial

Andy Buckley
University of Glasgow

CHACAL 2024

Johannesburg, 19 Jan 2024 (updated 10 Feb 2024)



University
of Glasgow

Setup: Docker images and containers

❖ We will be running event generators via Docker

- Like a virtual Linux machine that you run inside your own PC
- VM *image* files are O(1 GB): download these in advance!



❖ Containers and volume binding

- You can **run** an image multiple times: each copy is a *container*
- By default the data from each container stays on your machine... this eats a *lot* of disk space! use **--rm** to make it auto-delete, or periodically **docker system prune**
- To make it easy to get data in and out of your container, make a “portal” directory: **-v /some/host/dir:/some/container/dir**

❖ Rivet+Pythia image

- `docker pull hepstore/rivet-pythia:chacal24`
- `docker run -it --rm hepstore/rivet-pythia:chacal24`
- Test: `# rivet -h` `# pythia8-main93 -h`

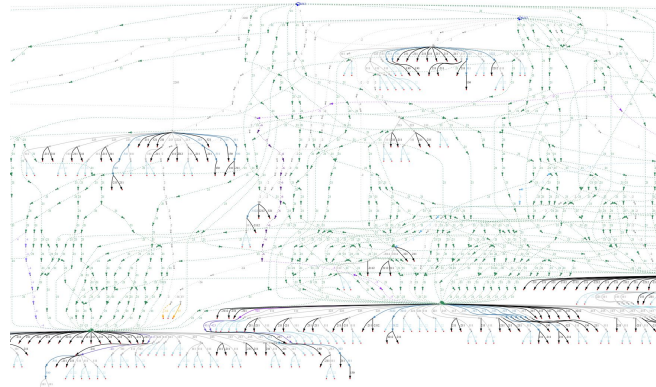
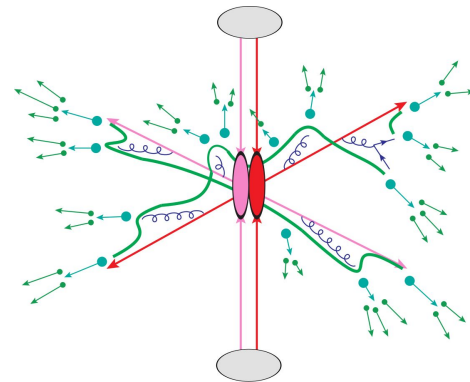


❖ Rivet+MG5_aMC@NLO image

- `docker pull hepstore/rivet-mg5amcnlo:chacal24` + `docker run ...`
- Test: `# rivet -h` `# MG5_aMC_v3_5_3/bin/mg5_aMC -h`

MC generation

- ❖ **MC generation: where theory meets experiment**
 - The fundamental pp collision, without a surrounding detector
- ❖ **Components of a fully exclusive SHG chain**
 - QFT matrix element sampling at fixed order in QCD etc.
 - *Dressed* with approximate collinear splitting functions, iterated in factorised Markov-chain “parton showers”
 - FS parton evolution terminated at $Q \sim 1$ GeV: phenomenological hadronisation modelling. Mixed with MPI modelling.
 - Finally particle decays, and other niceties
- ❖ **Today**
 - hands-on tutorial with Pythia8 and MadGraph5
 - for background principles see the lecture slides
 - introduction to running generators and studying their output
 - generation biasing for efficient phase-space population
 - ME/PS merged generation with extra ME jets
 - ~~BSM model configuration and generation~~



Generator basics

❖ First, get your Pythia Docker container started

- `$ docker pull hepstore/riwet-pythia:chacal24`
- `$ docker run -it --rm -v $PWD:/host hepstore/riwet-pythia:chacal24`



purple = command shell

❖ Pythia8: shower-hadronisation generator (SHG) with many LO processes built-in

- Pythia 8.3 docs: <https://pythia.org/latest-manual/Welcome.html>
- We'll use the "main93" example interface. Open a blank command file: `# nano py8-top.cmnd`
- Add the lines:
 - Beams:eCM = 13000
 - Top:all = on
 - Main:writeHepMC = on
- And run: `# pythia8-main93 -c py8-top.cmnd -o TOP -n 100`



blue = generator configs

❖ Examine the output

- `less TOP.hepmc`
- Run a basic physics analysis on it: `# rivet -a MC_FSPARTICLES TOP.hepmc -H TOP.yoda`
- View the histogram data: `$ less TOP.yoda; # yodals -v TOP.yoda`
- `# rivet-mkhtml TOP.yoda -o /host/riwet-plots-top`
- And point your Web browser at it, e.g. `$ firefox rivet-plots-top/index.html`

More statistics = no more event files

❖ The HepMC ASCII files are very large!

- They waste space, and CPU due to the writing/re-reading time
- Useful for debugging, though

❖ Better that we pass the events to Rivet in memory instead

- `# nano py8-top.cmnd`
- And change to:
 - `Beams:eCM = 13000`
 - `Top:all = on`
 - `Main:runRivet = on`
 - `Main:analyses = MC_TTBAR,MC_JETS,MC_FSPARTICLES,MC_ELECTRONS,MC_MUONS`
- `# pythia8-main93 -c py8-top.cmnd -o TOP -n 5000`
- `# rivet-mkhtml TOP.yoda -o /host/rivet-plots-top`

❖ Inspect the output

- Do the lepton distributions make sense?
- The jets?
- What happens to the statistics at high p_T ?

Jet-event generation

❖ Let's make some inclusive-jet events

- In Pythia, this just means a $pp \rightarrow jj$ ME. Everything else comes from the PS, especially ISR
- It does remarkably well for that (thanks to a few tricks)
- But mostly we use higher-order generators for the ME nowadays. Py8 is quick, though!

❖ We start with the obvious configuration

- ```
nano py8-jets.cmnd
 Beams:eCM = 13000
 HardQCD:all = on
 PhaseSpace:pThatMin = 10
 Main:runRivet = on
 Main:analyses = MC_JETS
```
- ```
# pythia8-main93 -c py8-jets.cmnd -o JETS -n 6000
```

 (there's a reason for this number of events!)

❖ View the output

- ```
rivet-mkhtml JETS.yoda -o /host/rivet-plots-jets
```
- And view: what's happened to the  $p_T$  tails and 3rd, 4th jet distributions?
- We can improve this with ME phase-space slicing and/or enhancement

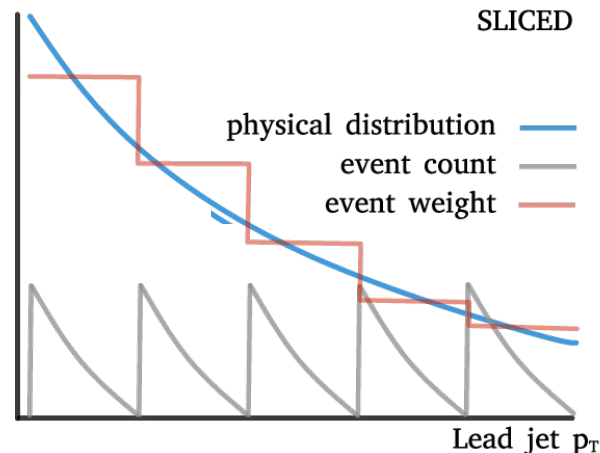
# Jet-event slicing

## ❖ The statistics died off at high $p_T$

- The unweighted events are asymptotically distributed like the physical  $d\sigma/dp_T$
- $\Rightarrow$  far too many low- $p_T$  events for our needs! Rapidly drop below systematics threshold
- Simple solution: stick together several runs in orthogonal *slices* of ME phase-space

## ❖ Three slices, the top-one open-ended

- Add a max  $p_T^{\text{hat}}$  to py8-jets.cmnd:  
PhaseSpace:pThatMin = 10  
PhaseSpace:pThatMax = 50  
# pythia8-main93 -c py8-jets.cmnd -o JETS0 -n 2000
- Then a min/max pair above that:  
PhaseSpace:pThatMin = 50  
PhaseSpace:pThatMax = 100  
# pythia8-main93 -c py8-jets.cmnd -o JETS1 -n 2000
- And a final min-only:  
PhaseSpace:pThatMin = 100  
# pythia8-main93 -c py8-jets.cmnd -o JETS2 -n 2000
- Plot and study: # rivet-merge JETS?.yoda -o JETS\_SLICE.yoda  
# rivet-mkhtml JETS{0,1,2}.yoda:LineStyle=dotted JETS\_SLICE.yoda:Sliced -o /host/rivet-plots-jets



# Jet-event enhancement

## ❖ The statistics work better now, and the correctly xs-normalised sum is smooth

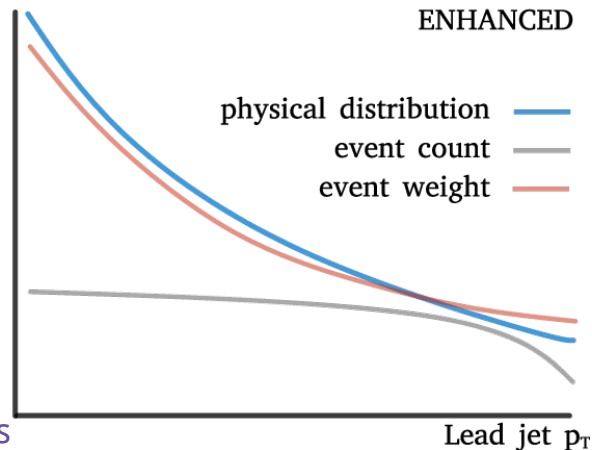
- We still have falling stats in each slice, though: “sawtooth” statistical error
- Can we “continuously slice”? Yes! Sample from  $p_T^{\text{hat},n} d\sigma/dp_T^{\text{hat}}$ , with weights  $1/p_T^{\text{hat},n}$
- Since LO  $2 \rightarrow 2$  process,  $p_T^{\text{hat}}$  is unambiguous

## ❖ Enhanced dijet generation

- Enable biasing in py8-jets.cmd:  

```
PhaseSpace:pThatMin = 10
PhaseSpace:bias2Selection = on
pythia8-main93 -c py8-jets.cmd -o JETS_ENH -n 3000
```
- Pretty-printing of all methods:  

```
rivet-mkhtml JETS.yoda:Raw:LineColor=red \
JETS{0,1,2}.yoda:LineColor=purple:LineStyle=dotted \
JETS_SLICE.yoda:Slice:LineColor=green \
JETS_ENH.yoda:Enh:LineColor=orange -o /host/rivet-plots-jets
```
- Study the output. Which is better at phase-space coverage?  
Compare the numbers of events generated





# V+jets production

## ❖ W/Z+jets are the biggest and most CPU-consuming MC samples at the LHC

- Followed by ttbar, single-top, diboson, ...
- The “classic” development lab for beyond-LO methods, because
  - Born process at 2→1 tree level; jets (and hence all Z  $p_T$ ) is beyond LO
  - colour-singlet boson is unproblematic for QCD
  - vector boson: symmetry protection ⇒ small NLO corrections wrt Higgs
  - massive boson = naturally “anchored” scale choices: more stable than massless jets or photons

## ❖ First, let’s make a Pythia8 version, then go to MG5

- # nano py8-zmm.cmnd  
Beams:eCM = 13000  
WeakBosonAndParton:qqbar2gmZg = on  
WeakBosonAndParton:qg2gmZq = on  
PhaseSpace:pThatMin = 20  
23:onMode = off  
23:onIfAny = 13  
Main:runRivet = on  
Main:analyses = MC\_JETS
- # pythia8-main93 -c py8-zmm.cmnd -o ZMM -n 5000
- # mv ZMM.yoda /host/Py-ZJ.yoda

# V+jets production: MG5

## ❖ Get the MG5 image and open it in a separate terminal

- `$ docker pull hepstore/rivet-mg5amcnlo:chacal24`
- `$ docker run -it --rm -v $PWD:/host hepstore/rivet-mg5amcnlo:chacal24`  
`# cd MG5_aMC_v3_5_3/`  
`# bin/mg5_aMC`
- MG5 is a fixed-order ME generator that interfaces with Pythia's showers, decays, etc.

## ❖ Generate the lowest-order jet-multiplicity sample

- `> generate p p > mu+ mu- j`  
`> output PROC-ZJ`  
`> launch`  
`> ... (enable Pythia, run)`  
`> quit`
- `# cp -r PROC-ZJ /host/`  
⇒ look at diagrams in the host file browser, xsec in web browser
- `# cd PROC-ZJ/Events/run_01/`  
⇒ look at the LHE (and HepMC) event files:  
`# zless unweighted_events.lhe.gz`

JPG Feyn diagrams will be generated automatically in the SubProcesses (sub)folders. You can also use the `> display diagrams` command... but not very effectively in Docker since there's no graphics

# V+jets production: MG5 jet-merging

## ❖ We can also make higher-order MEs (here just tree-level)

```
➤ # ...
bin/mg5_aMC
> generate p p > mu+ mu- j
> add process p p > mu+ mu- j j
> output PROC-ZJJMERGED
> quit
```

Add a [QCD] suffix to generate a process at QCD NLO. Slow!!

⇒ one-loop matching with MC@NLO; loop and legs merging/matching with FxFx

```
➤ # cp -r PROC-ZJJMERGED PROC-ZJJBORKED
```

⇒ copy setup for broken, overlapping-process hack

```
cd PROC-ZJJBORKED
```

```
nano Cards/proc_card_mg5.dat
```

```
nano Cards/run_card.dat
```

⇒ set ickkw=0 (disables correct merging!)

```
bin/generate_events
```

```
➤ # cd ../PROC-ZJJMERGED
```

```
bin/generate_events
```

## ❖ What's going on???

- The PS makes the different multiplicities overlap in phase-space: have to avoid double-counting
- CKKW(L) and MLM procedures do this by phase-space weights or cuts: we're trying MLM on/off

# V+jets production: analysis and comparison

## ❖ Run Rivet on the (zipped) MG5 HepMC events

- MG5 events have lots of weights, cf. the LHE file. Incorporating scale and PDF variations. But MG5 doesn't specify a default weight, so we need to identify that by hand:

- ```
# rivet -a MC_JETS \  
PROC-ZJ/Events/run_01/tag_1_pythia8_events.hepmc.gz -H MG-ZJ.yoda
```

- ```
rivet -a MC_JETS \
PROC-ZJJBORKED/Events/run_01/tag_1_pythia8_events.hepmc.gz -H MG-ZJJ-sum.yoda
```

- ```
# rivet -a MC_JETS --unmatch-weights='scomp | smin | smax' \  
PROC-ZJMERGED/Events/run_01/tag_1_pythia8_events.hepmc.gz \  
-H MG-ZJJ-merge.yoda
```

 ← Veto MG5's accidental unphysical merging weights: being fixed!

- And plot:

```
# cp /host/Py-Z.yoda .
```

 ← add --no-weights for speed

```
# rivet-mkhtml Py-Z.yoda MG-Z*-filt.yoda -o /host/rivet-plots-z
```

❖ Inspect the output

- See how the samples have different kinematics & N_{jets} ? And the MG5 systematic uncertainty bands?

Writing a custom analysis

❖ Just running pre-made Rivet analyses like MC_JETS would be very limiting

- Now we will very briefly write our own analysis code

❖ Inside your container, create a new C++ source file

- Rather than start from an empty file, we use rivet-mkanalysis to make a template code:

```
rivet-mkanalysis CHACAL  
nano CHACAL.cc
```

- Book a new histogram:

```
book(_h["jjmass"], "jjmass", logspace(20, 1.0, 1000.0));
```

- Require and get the two leading jets, add 4-vectors, histogram the mass:

```
if (jets.size() < 2) vetoEvent;  
FourMomentum pjj = jets[0].mom() + jets[1].mom();  
_h["jjmass"]->fill(pjj.mass()/GeV);
```

- Build, run and plot:

```
# rivet-build CHACAL.cc  
# rivet --pwd -a CHACAL PROC-ZJJMERGED/.../*.hepmc.gz -H chacal.yoda  
# rivet-mkhtml chacal.yoda -o /host/rivet-plots-chacal
```

Documentation on the code & physics objects from here:

<https://rivet.hepforge.org/doc>
<https://gitlab.com/hepcedar/rivet/-/blob/release-3-1-x/README.md#welcome>

← pick up the analysis .so from current working directory \$PWD

That's it!

- ❖ **Thanks for your time!**
- ❖ You now know how to run two of the most popular LHC event generators at Born and merged/matched levels
- ❖ ~~And how to set up and run any UFO new physics model~~
- ❖ This is a superpower — use it wisely!
- ❖ And the devil is in the details: black-box mode will only get you so far
- ❖ Sometimes it goes wrong, sometimes... it's complicated
- ❖ **Good luck!**



BSM physics generation

❖ Pythia8 has several built-in models, e.g. Z', SUSY, XD resonances...

- Many are steered just via Py8 parameters — see the manual
- SUSY in particular requires an SLHA file: use [hepstore/rivet-tutorial](#)
- Set up a command file with

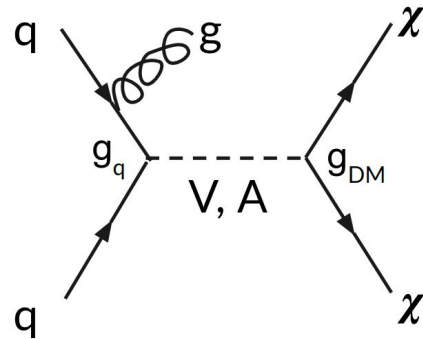
```
SUSY:all = on
SLHA:file = gg_g1500_chi100_g-ttchi.slha
```
- Run and analyse

❖ MG5 is really a generator generator: more flexible

- ⇒ can build new MEs for ~any UFO physics model (as can Sherpa, Herwig)
- E.g. a dark matter model:

```
> import model DMsimp_s_spin1 --modelname
> generate p p > xd xd~ j
```
- etc. DM mass, coupling can be set in the “param card” = SLHA
- Generate and analyse
- More control can be imposed by fixing new-physics couplings at amplitude level e.g. `NP==1` or ME-squared level e.g. `NP^2==1`

hepstore/rivet-tutorial is just the rivet-pythia Docker image with a few extra tutorial files in the work dir



Since the MG5 conversion to use Python3, you may need to run a ‘convert’ command on your UFO, and re-import. The command-line will advise you if this is the case