Model likelihoods via measurements

A tour of Rivet and Contur

Andy Buckley
University of Glasgow

Jonathan Butterworth
University College London

YETI Precision ⊗ Collider
IPPP Durham, 13 July 2026

Outline plan

- **This hands-on tutorial follows and solidifies Chris' lecture**

- **After this, you should be able to:**
  - **Select 4-lepton events with Rivet**
  - **Histogram event observables with Rivet & YODA**
  - (Generate hard-process events with MadGraph, if you want)
  - (Including some loop-induced processes, if you want)
  - **Shower the MG events with Pythia**
  - **Run your Rivet analysis routine and visualise the output**
  - **Run data-comparison Rivet routines**
  - **See the effect of Rivet data-MC comparisons on a BSM model with Contur**

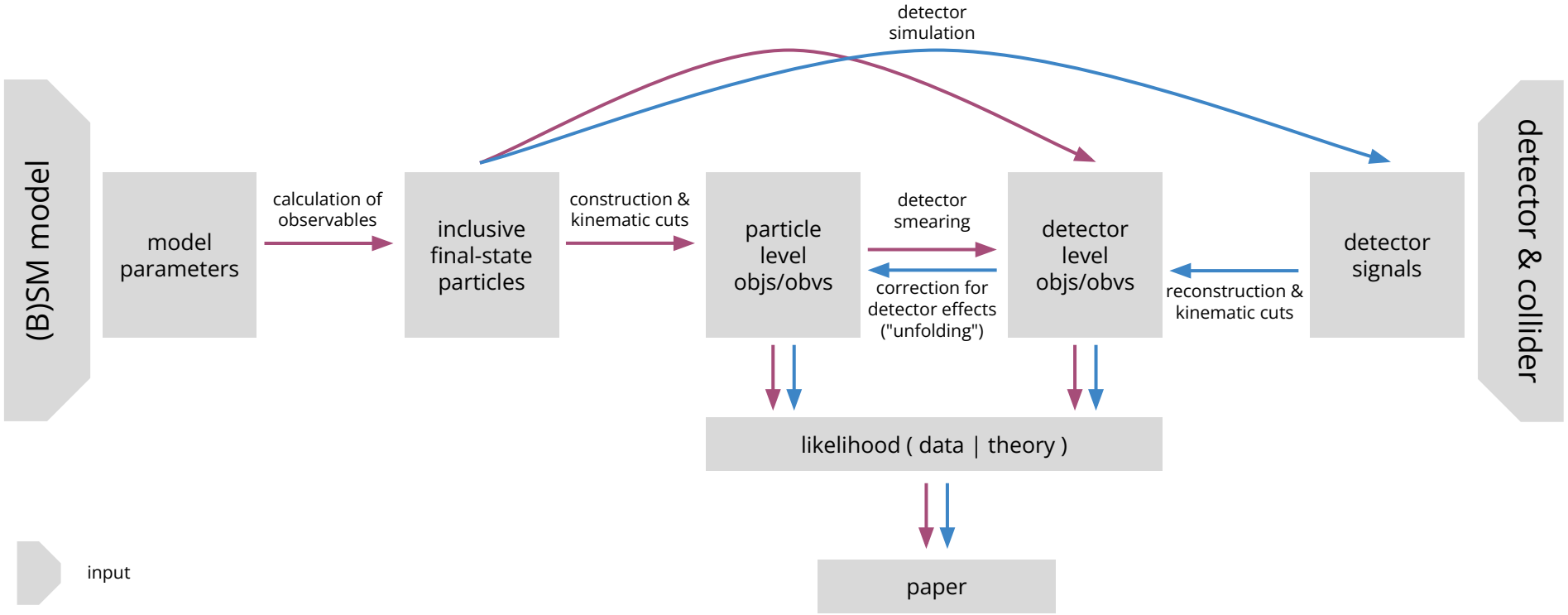
Setup

- **We will be running event generators via Docker/Singularity**
  - Like a Linux virtual machine that you run inside your own PC
  - VM *image* files are O(GBs): download *now*!

- **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… Run with --rm to make it auto-delete, or use 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+MG5_aMC@NLO image**
  - $ docker pull hepstore/contur-tutorial
  - $ docker run -it --rm -v $PWD:/host hepstore/contur-tutorial
  - # cd /work #< for the non-Contur part
  - # rivet -h
    # pythia8-main144 -h
    # contur -h

MC→inference toolchain



Getting started with Rivet

```
rivet-mkanalysis MY_ROUTINE
```

This will create

- A **`MY_ROUTINE.cc`** file for the analysis logic ("routine")
- A **`MY_ROUTINE.plot`** file for plot formatting
- A **`MY_ROUTINE.info`** file for storing metadata about the analysis

The main body of the Rivet routine is within this **`MY_ROUTINE.cc`** file, containing

1. An **`init()`** method that runs once at the start to initialise the routine
2. An **`analyze()`** method that is executed for every single event
3. A **`finalize()`** method that is called once at the end to do post-analysis operations such as scaling the histograms

Build the analysis .cc source file (or files) into a compiled “plugin library” like this:

```
rivet-build MY_ROUTINE.cc
```

Rivet analysis-code example

Rivet's analysis code is intentionally simple and physics-focused:


```
#include "Rivet/Analysis.hh"
#include "Rivet/Projections/FastJets.hh"

namespace Rivet {
  class MySimpleAnalysis : public Analysis {
  public:
    RIVET_DEFAULT_ANALYSIS_CTOR(MySimpleAnalysis);

    void init() {
      FastJets fj(FinalState(), JetAlg::ANTIKT, 0.4);
      declare(fj, "Jets");
      book(_h_jetPt, "Jet_Pt", 50, 0, 500);
    }

    void analyze(const Event& event) {
      const Jets& jets = apply<FastJets>(event, "Jets").jetsByPt();
      for (const Jet& jet : jets) _h_jetPt->fill(jet.pT()/GeV);
    }

    void finalize() {
      normalize(_h_jetPt);
    }

  private:
    Histo1DPtr _h_jetPt;
  };

  RIVET_DECLARE_PLUGIN(MySimpleAnalysis);
}
```

Projections

Projections reduce the full Event to something simpler, e.g. sets of final-state particles or jets.

They are constructed and *declared* in `init()` and *applied* in `analyze()`:

```
#include "Rivet/Projections/FinalState.hh" //Make sure the header file is included
…
…
void init() { // Declare the projections here
     …
     FinalState fs(Cuts::pT > 0.5*GeV);
     declare(fs, "my_first_projection");
}

void analyze(const Event& event) {
     …
     const FinalState& fs = apply<FinalState>(event, "my_first_projection");

     const Particles& fsParticles = fs.particlesByPt();
     …
}
```

Declare `FinalState` object and give it a name that can be used in `analyze()`.

Apply the `FinalState` projection with the name defined in `init()` to the event and store this as `fs`.

Use `fs` to retrieve all final-state particles ordered by transverse momentum (hard to soft).

Example projections

```
#include "Rivet/Projections/FinalState.hh" //Make sure the header file is included
#include "Rivet/Projections/PromptFinalState.hh"
…
…
void init() { // Declare the projections here
      …
      // Get all charged particles
      FinalState chargedTracks(Cuts::charge != 0);

      // Get all charged particles + additional pseudorapidity cut
      FinalState IDtrack(Cuts::charge != 0 && Cuts::abseta < 2.5);

      // Get all muons with transverse momentum >20GeV
      FinalState allMuons(Cuts::abspid == PID::MUON && Cuts::pT > 20*GeV);

      // Get a subset of prompt muons from the previous muon projection
      PromptFinalState promptMuons(allMuons);

      declare…
}
```

Lepton finding

The **`LeptonFinder`** projection allows you to dress the bare-lepton four-momentum with all photon four-momenta within a cone (or by clustering.)

These are referred to as **dressed leptons.**

```
#include "Rivet/Projections/LeptonFinder.hh"
…
void init(){ // Declare the projections here
      …
      // Define (Prompt)FinalState photons and electrons
      FinalState photons(Cuts::abspid == PID::PHOTON);
      PromptFinalState electrons(Cuts::abspid == PID::ELECTRON);

      // Define some cuts on electrons
      Cut electronCut = (Cuts::pT > 7*GeV && Cuts::abseta < 2.47);

      // Define dressed electrons
      LeptonFinder dressedElectrons(electrons, photons, 0.1, electronCut);
      declare…
}
```

cone size

Jet finding

```
#include "Rivet/Projections/FastJets.hh" // Make sure relevant header file is included
…
void init() { // Declare the projections here
      …
      // Define the FinalState and apply jet algorithms
      FinalState fs;
      FastJets jets(fs, JetAlg::ANTIKT, 0.4, JetMuons::DECAY, JetInvisibles::NONE);

      declare(jets, "FJets");
      …
}

void analyze(const Event& event) {
      …
      // Apply jetsByPt method in analyze section
      const Jets& allJets = apply<FastJets>(event, "FJets").jetsByPt();
      …
}
```

jet algorithm

radius parameter

configure how muons (and invisible particles) are dealt with in the clustering:

- `NONE`: allow no muons in jets
- `DECAY`: allow only non-prompt/ muons from decays in jets
- `ALL`: allow any kind of muons in jets

Histograms

Histograms in Rivet use the YODA library. Typical “live” histogram behaviour.
Event weights and systematic variations handled automatically.

```
Histo1DPtr _histA;
map<string, Profile1DPtr> _profs;

void init(){ // Book the histograms here
     …
     book(_histA, “hist_name_a”, 20, 0., 10.);
     book(_profs[“foo”], “ymean_vs_x”, logspace(10, 1e-2, 20));
}

void analyze(){ // Fill the histograms here
     _histA->fill(x);
     _profs[“foo”]->fill(x, y);
}

void finalize(){ // Process the histograms here
     scale(_histA, crossSection()/femtobarn/sumW());
}
```

Histos: the init() method

```
/// Book histograms and initialise projections before the run
void init() {
  // Book histograms
  vector<double> mll_bins = { 66., 74., 78., 82., 84., 86., 88., 89., 90., 91.,
                              92., 93., 94., 96., 98., 100., 104., 108., 116. };

  book(_h["mll"], "mass_ll", mll_bins);
}
```

...

Specify bin edges for the histogram (there are also functions like linspace, logspace, bwspace to help)

Book histograms as
`book(<histogram object>, <histogram name>, <bin edges>)`

```
/// @name Histograms
/// @{
map<string, Histo1DPtr> _h;
/// @}
```

Histos: the analyze() and finalize() methods

```
/// Perform the per-event analysis
void analyze(const Event& event) {
  /// Todo: Reconstruct the dilepton invariant mass to fill the histogram
  //
  _h["mll"]->fill(1.0);
}
```

Fill the histogram with "1" for every event.
Well, that isn't too useful...

```
/// Normalise histograms etc., after the run
void finalize() {
  const double scaleFactor = crossSection() / sumOfWeights();
  scale(_h, scaleFactor);
}
```

Scale all histograms by the cross section.

Exercise time!

Your turn!

- Use `rivet-mkanalysis` to create an analysis-routine skeleton
- Edit the `.cc` file and use `LeptonFinder` to find direct electrons and muons
- Require the leptons to be within |eta| < 2.5 and with p_T > 10 GeV
- Book and fill a pair of histograms for electron and muon p_T
- Build the routine
- Style the plots using the .plot file if feeling virtuous
- List its info with `rivet --pwd --show-analysis YOUR_ANALYSIS_NAME`

Making ME events (optional)

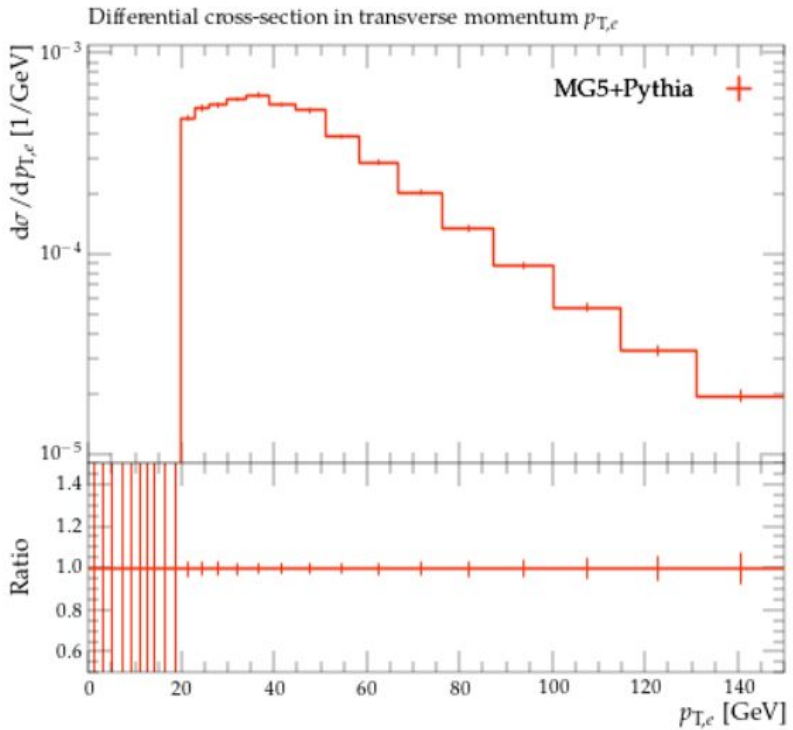
- **(Download zipped LHE files from the Indico if you prefer)**
- **Run MG5:** `MG5aMC/bin/mg5_aMC`
- **Specify the LO pp → 4-lepton process**
  - We'll avoid 4e/4μ pairing ambiguity, and just use a 2e2μ final state
  - `generate p p > e+ e- mu+ mu-`
- **This only includes tree-level. You might be interested to add loop-induced gg→ZZ and gg→H→ZZ channels via EFT models:**
  - *Before* generate: `import model heft` or `import model loop_sm`
- **Set output, launch, tweak run card**
  - `output PROC_4L_TREE` (rename as you wish for SM-loop and HEFT models)
  - `launch`

Showering and Rivet-analysis

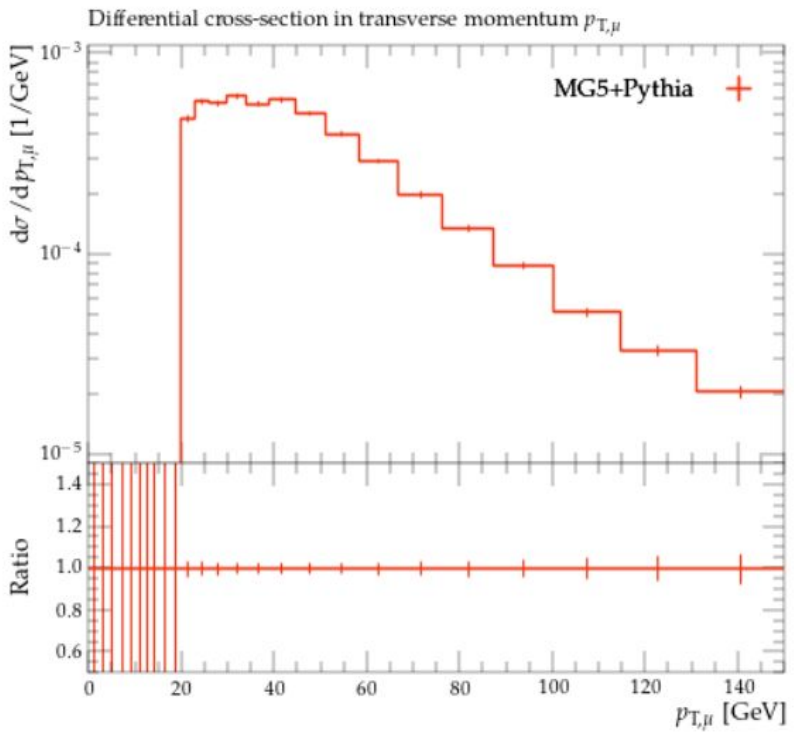
- **Run Pythia+Rivet on the generated MEs**
  - Use either your MG5 events, or pre-generated events from Indico
    - The MG5 events are located in `PROC_4L_*/Events/run_01/unweighted_events.lhe.gz`
  - Take a look in `main144lhe.cmnd`
  - Set your analysis name in the list of Rivet analyses to be run
    - `Rivet:analyses = {MY_ANALYSIS_NAME}`
  - Shower and analyse:
    - `export RIVET_ANALYSIS_PATH=$PWD` #< make plugin discoverable
    - `pythia8-main144 -c main144lhe.cmnd`
- **Plot the observables**
  - `rivet-mkhtml --pwd MG5+Pythia.yoda -o /host`
  - Open in a web browser in the host, e.g. `firefox rivet-plots/index.html`

Example output: 1ℓ

⚓ ⌘ 📊 1e_pt:



⚓ ⌘ 📊 1m_pt:



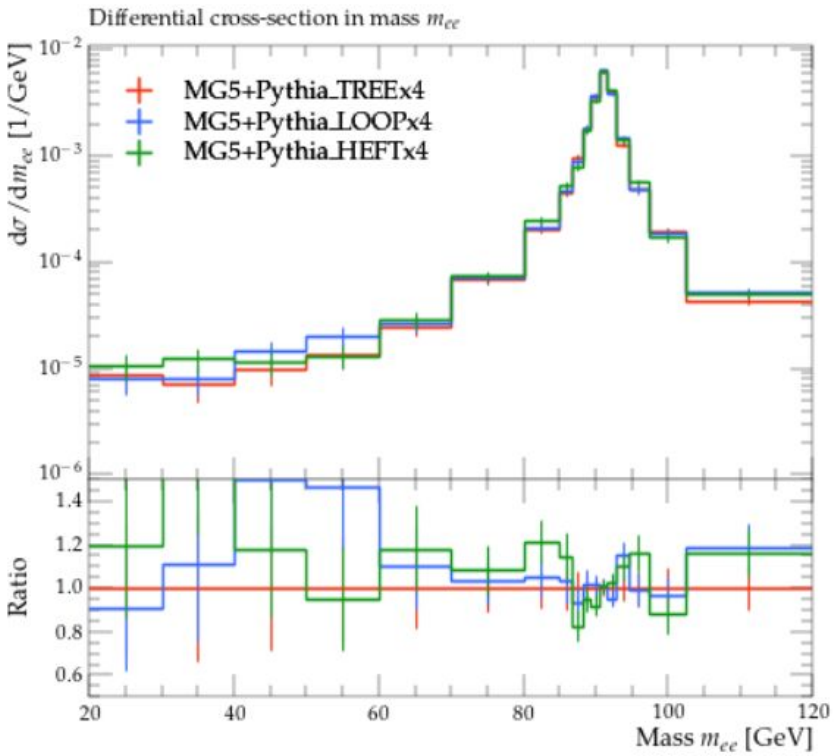
More coding exercise!

Your turn!

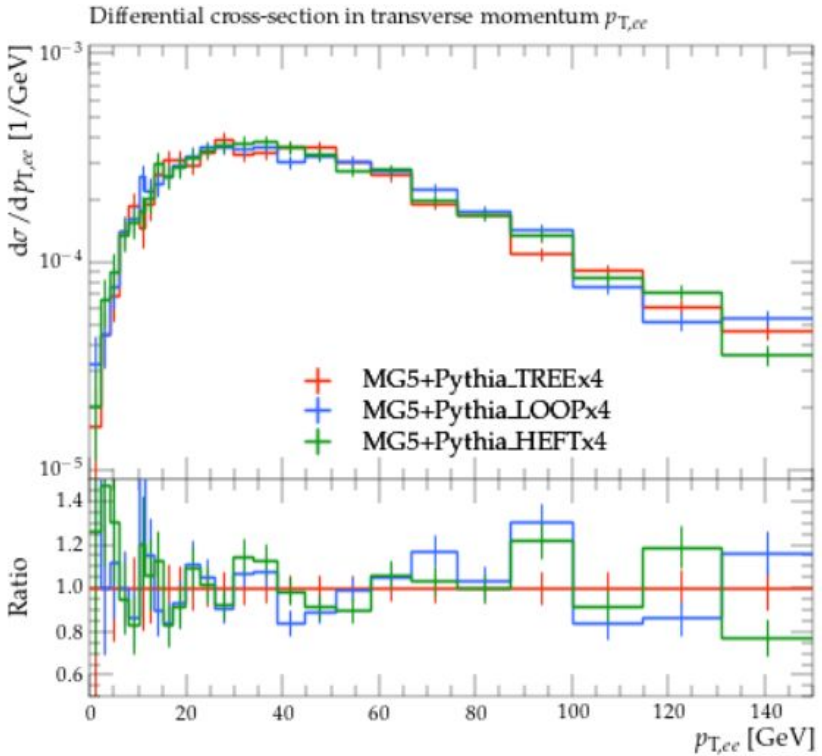
- Edit the `.cc` file again and require exactly two electrons and exactly two muons
- Additionally require that each pair is opposite-signed
- For each pair, get the particles 4-momenta with `.mom()` and sum them
- Add and fill histograms for the di-electron and di-muon p_T's and masses
- Add the dilepton pair momenta together, too, and plot the 4ℓ p_T and mass
- Build the routine and run+inspect again

Example output: 2ℓ

⚓ ⌘ 📊 2e_mass:

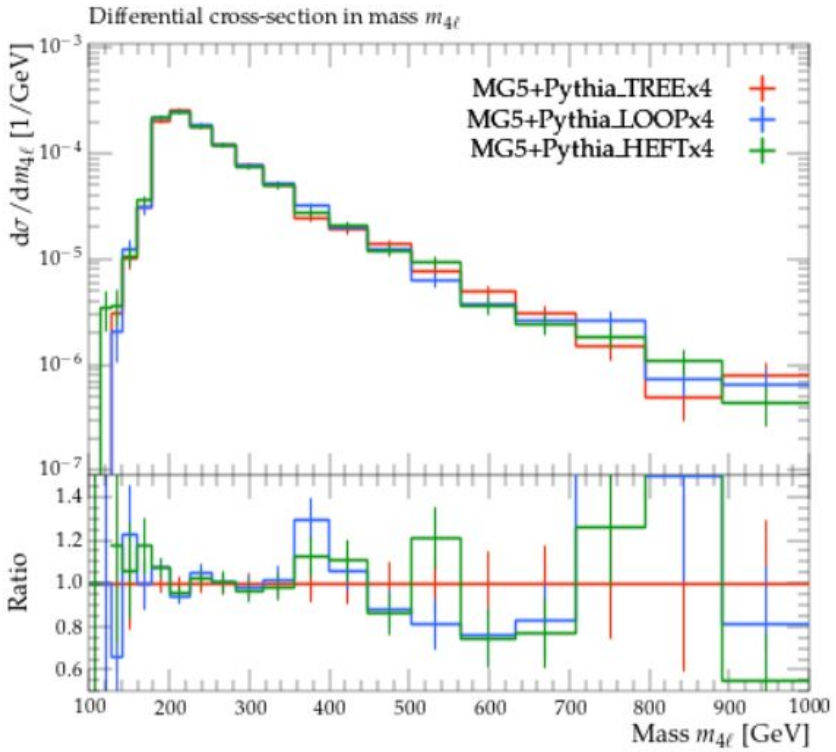


⚓ ⌘ 📊 2e_pt:

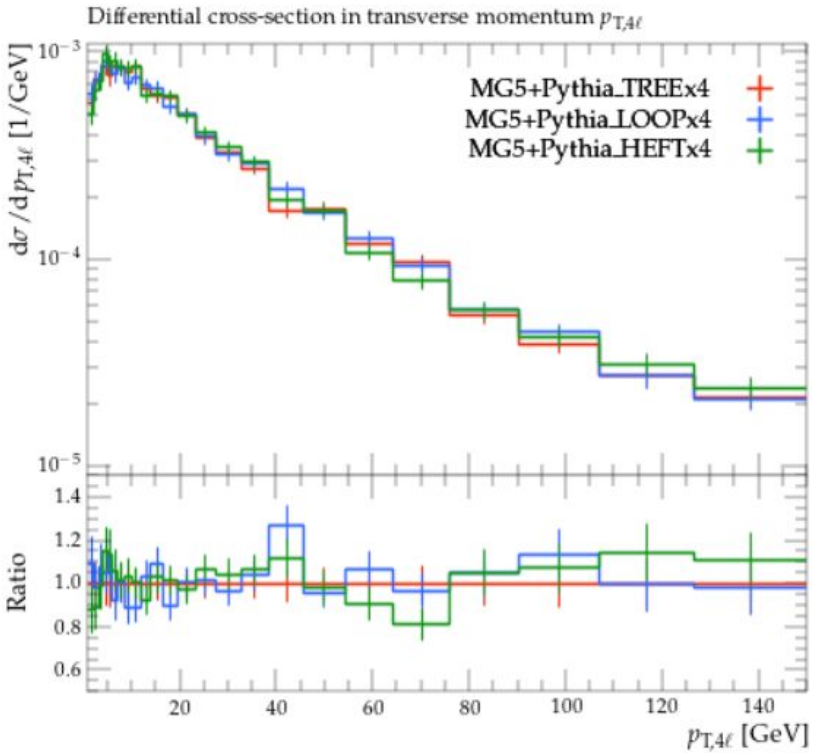


Example output: 4ℓ

4l_mass:



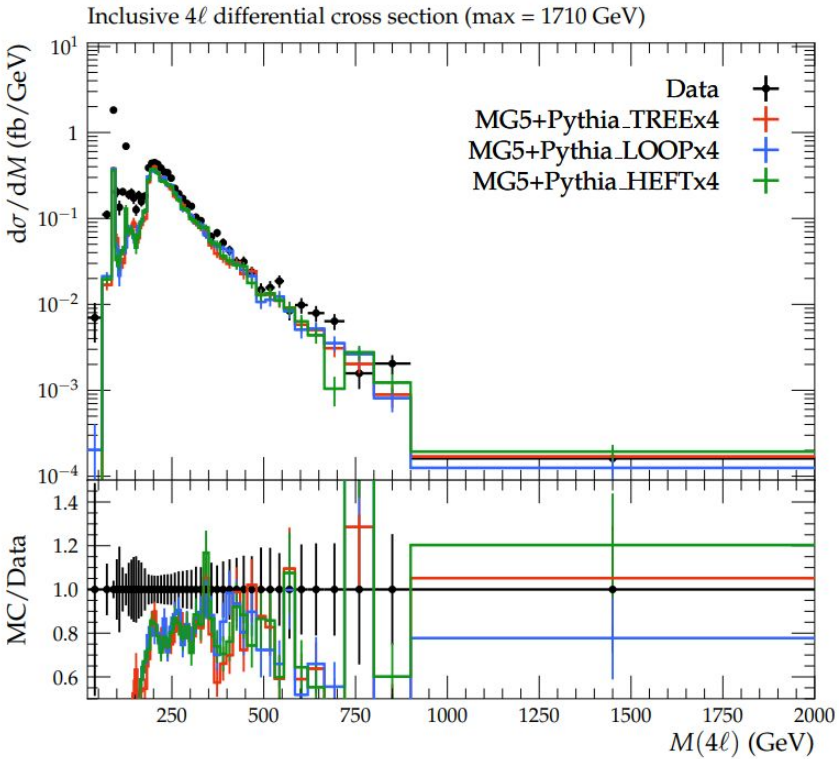
4l_pt:

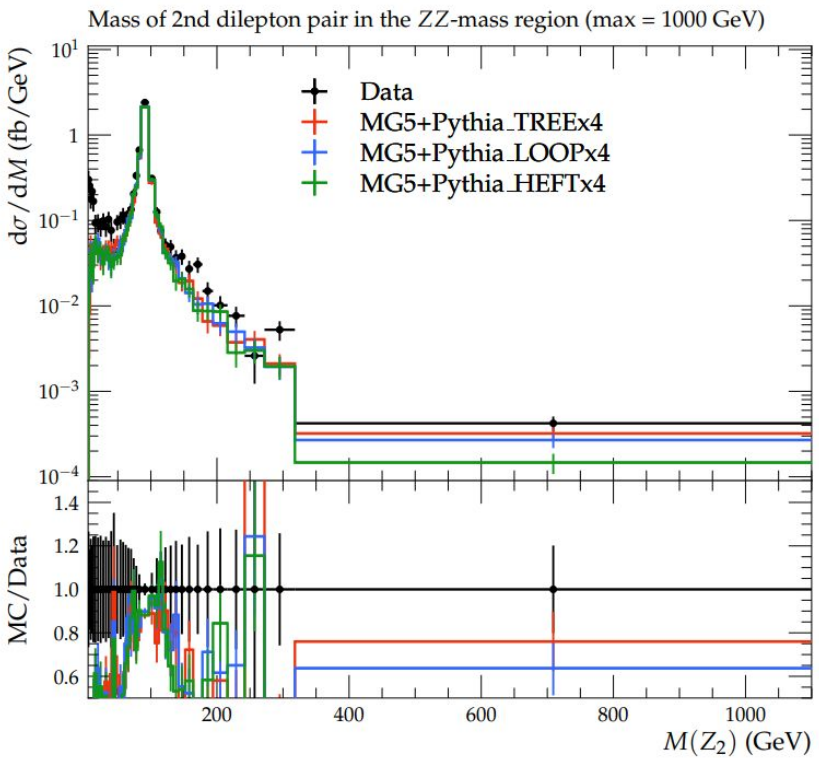


Running data analyses

- **Look at all the analyses**
  - `rivet --list-analyses`
- **Let's run one**
  - Add `ATLAS_2021_I1849535` to the list of analyses
  - Run and view again: look in the new analysis' page
  - What can you see?
    - The Z pole?
    - The Higgs pole?
    - The on-shell ZZ region?
- **We're missing a cross-section factor due to 2e2μ: rescale**
  - `rivet-merge MG5+Pythia.yoda:x2.0 -o MG5+Pythiax4.yoda`
- **Now we're ready for Contur…**

Example output: ATLAS 2021 4ℓ

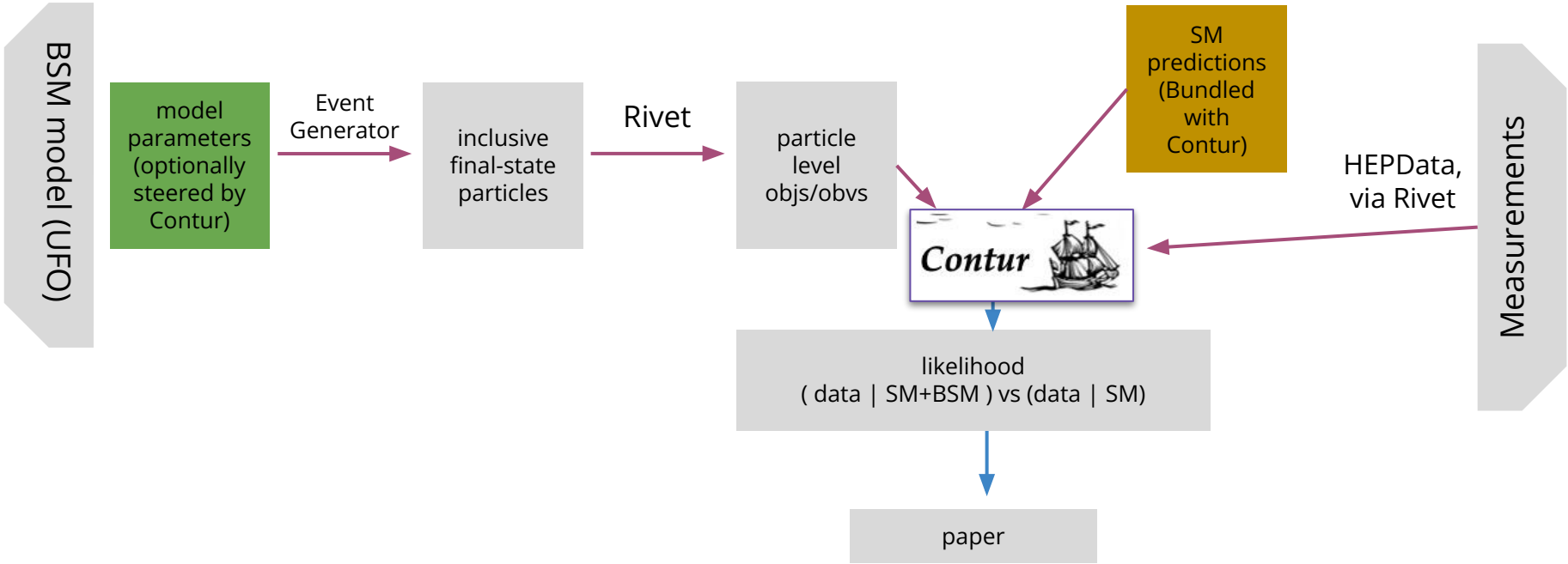




Contur

- **Constraints On New Theories Using Rivet**
  - Signal injection from a BSM model into a huge range of measured fiducial phase spaces
  - Focus on particle level (or smeared) observables
  - Ever-growing library of LHC measurements, new measurements can be integrated quickly once in Rivet
  - Speed and coverage means can test specific models with complex phenomenology, move away from simplified benchmarks

Contur workflow



The ΣSM

Take an example model

Jon Butterworth, Hridoy Debnath, Pavel Fileviez Perez, and Francis Mitchell. Custodial symmetry breaking and Higgs boson signatures at the LHC. *Phys. Rev. D*, 109(9):095014, 2024. arXiv:2309.10027, doi:10.1103/PhysRevD.109.095014.

Extended Higgs sector (real scalar triplet) motivated by the slight freedom in the ϱ parameter (custodial symmetry) suggested by scatter in W mass measurements.

$$M_h^2 = \lambda_0 v_0^2 \left(1 + \frac{1}{\cos 2\theta_0}\right) + \left(\frac{a_1 v_0^2}{8x_0} + b_4 x_0^2\right)\left(1 - \frac{1}{\cos 2\theta_0}\right),$$

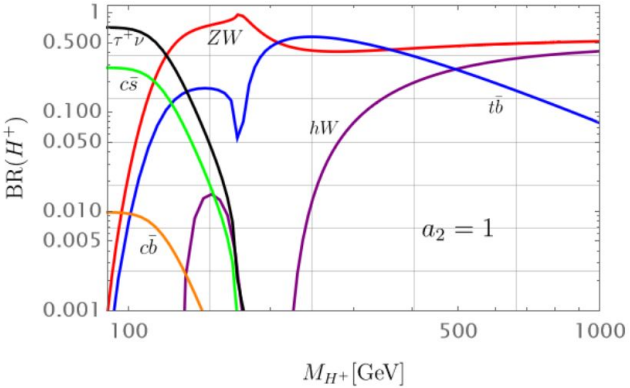
$$M_H^2 = \lambda_0 v_0^2 \left(1 - \frac{1}{\cos 2\theta_0}\right) + \left(\frac{a_1 v_0^2}{8x_0} + b_4 x_0^2\right)\left(1 + \frac{1}{\cos 2\theta_0}\right),$$

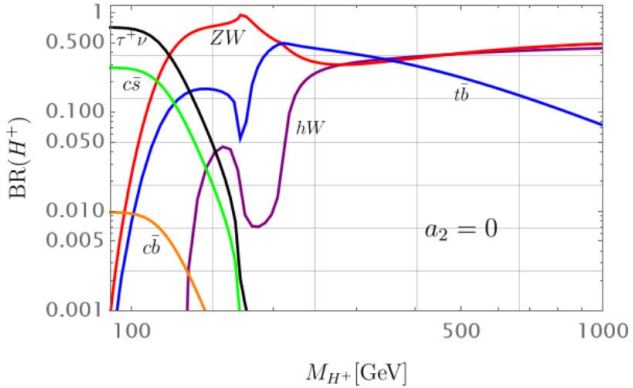
$$M_{H^\pm}^2 = a_1 x_0 \left(1 + \frac{v_0^2}{4x_0^2}\right),$$

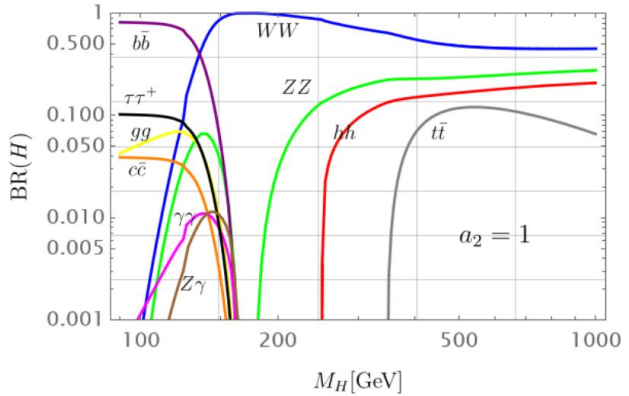
The ΣSM

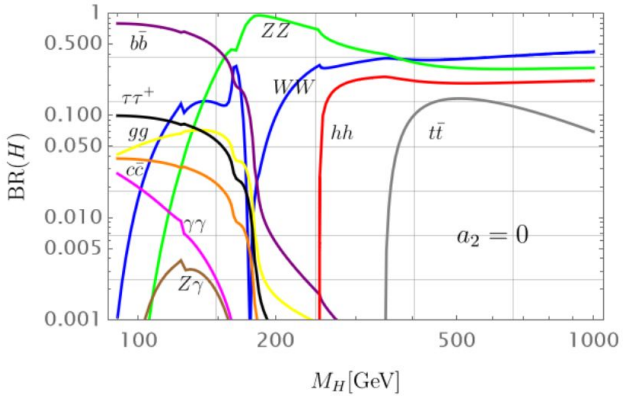
Complex phenomenology, with many SM like signatures

Including multilepton production, for some parameter choices









Contur and the ΣSM

Running on some pre-generated yoda files:

- Outside of docker, make yourself a clean working directory (e.g. mycontur) and cd into it.

  ```
  $ mkdir mycontur; cd mycontur
  ```

- Download contur-tutorial-files.tgz from indico into this directory and uncompress it.

  ```
  $ tar xvfz contur-tutorial-files.tgz; cd YETI
  $ docker run -it --rm -v $PWD:/host hepstore/contur-tutorial
  # cd /host
  ```

Contur and the ΣSM: single parameter point

In this directory you'll see a grid of pre-prepared yoda containing a scan over m_0 (the Higgs mass) and the triplet scalar vev, x_0. The details can be seen in param_file.dat. Go down into this file tree.

```
# cd scan_a2_0/13TeV/; ls
```

You'll see 100 numbered directories. Each contains the result of running rivet on 30k events generated using the SigmaSM, at a particular set of parameter values. Go down into a subdirectory and run contur on the yoda file you'll see there.

```
# cd 0021
# contur runpoint_0021.yoda.gz
```

Contur and the ΣSM: single parameter point

Ignore the linalg warning messages; these come from one analysis with a difficult covariance matrix. You should see: →

This means the exclusion for this point was 98.9%, and was expected to be 99.8%. An directory called ANALYSIS has been created with detailed results. We can use this to look at the histograms.

```
# cd ANALYSIS
# contur-rivetplots -r contur_run.db
```

```
      - SMBG=0.9893830585995079
      - EXP=0.9979795287021263
      - HLEXP=1.0

INFO -
        Model point sampled at these parameter values:
      - a2 : 0.0
      - b4 : 0.0001
      - m0 : 172.22222222222223
      - x0 : 1.2000000000000002
```

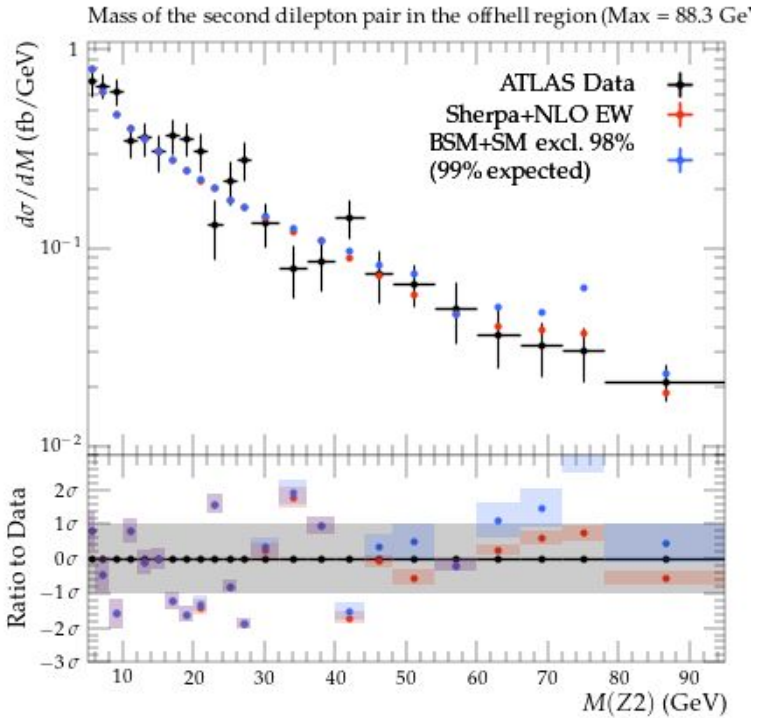
Contur and the ΣSM: single parameter point

You should now have a “plot” subdirectory

Exit docker and go back down:

```
$ cd scan_a2_0/13TeV/0021/ANALYSIS
```

You should now be able to open the index.html file in the plot directory with your web browser, and browse the histograms which give the overall exclusion.



Contur and the ΣSM: parameter point scan

Go back into docker and run on the grid

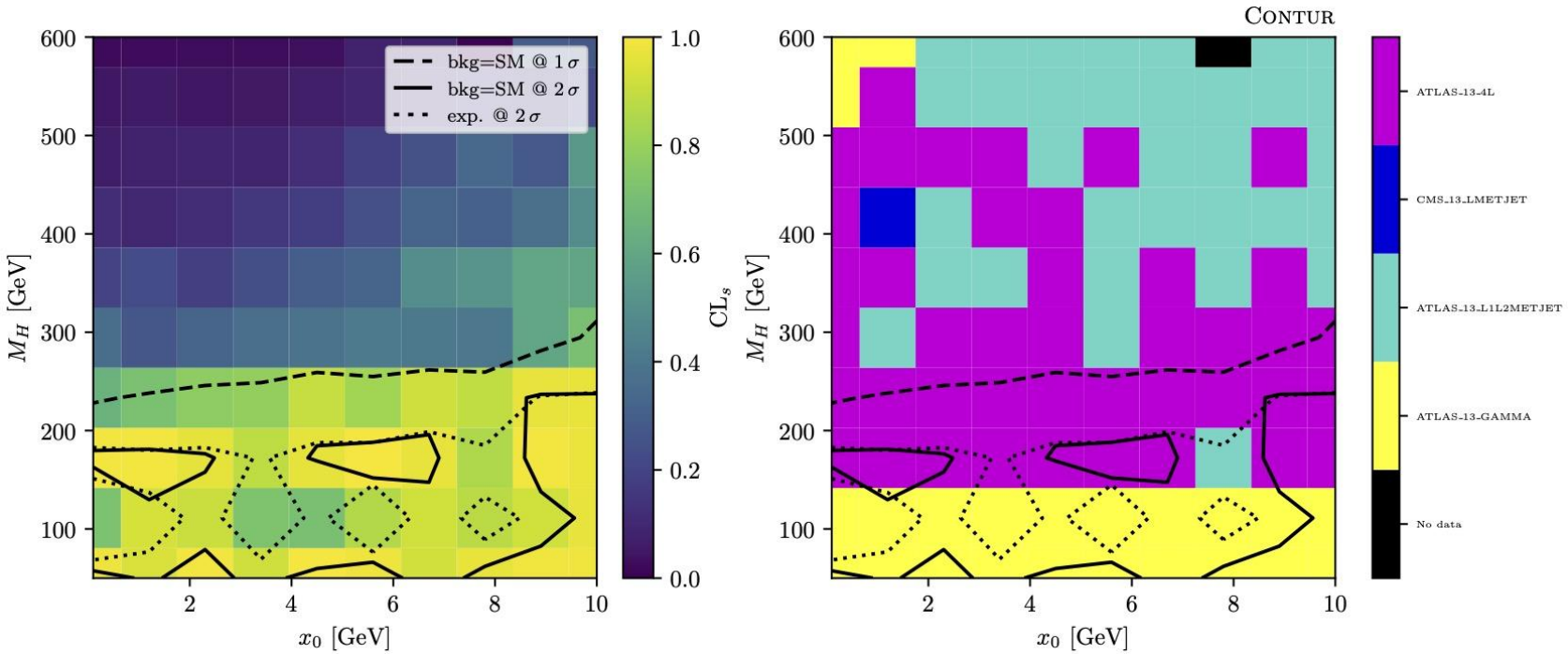
```
$ docker run -it --rm -v $PWD:/host hepstore/contur-tutorial
# cd /host
# contur -g scan_a2_0 -o ANALYSIS_GRID
```

This will run on all 100 result files in the scan_a2_0 directory - will take several minutes - producing a results file as before, but now in a directory called ANALYSIS_GRID. You can make a suit of pre-defined heatmap plots with this:

```
# cd ANALYSIS_GRID
# contur-plot contur_run.db x0 m0
```

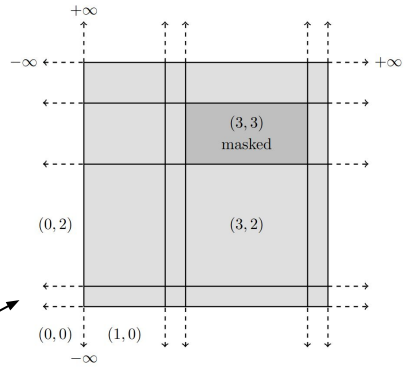
This will create a directory called conturPlots with some image files in that we can discuss.

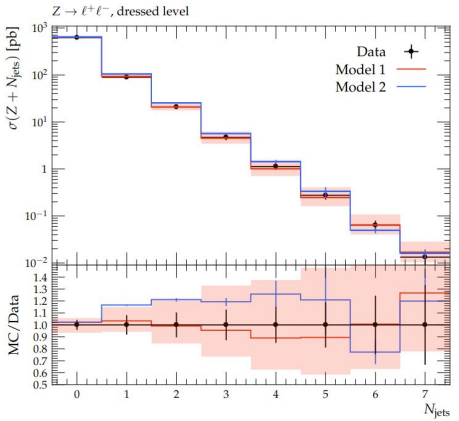
Contur and the ΣSM: parameter point scan

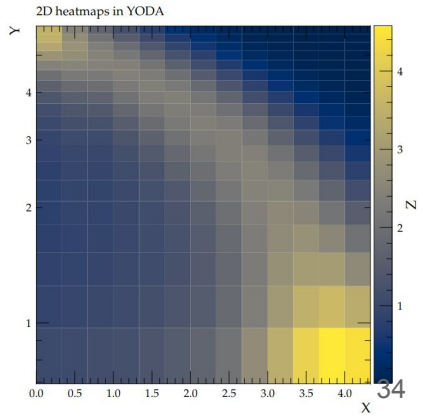


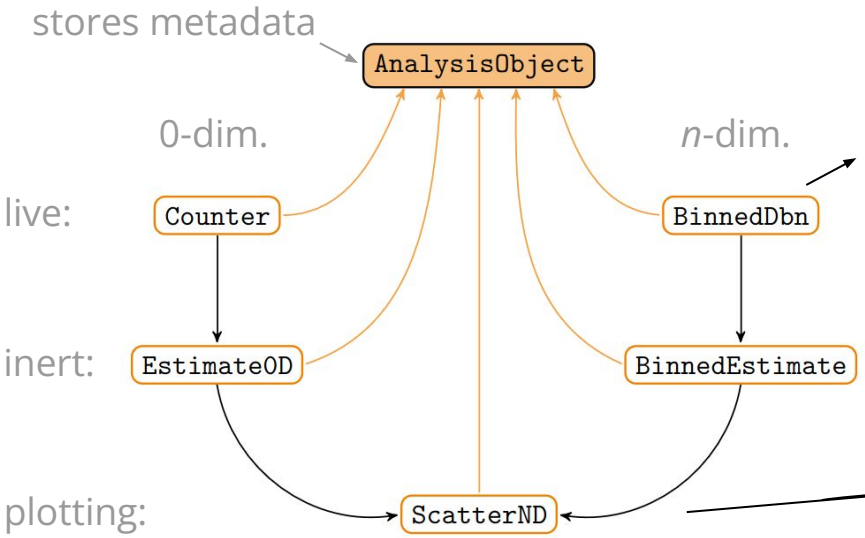
Backup / extras

YODA design

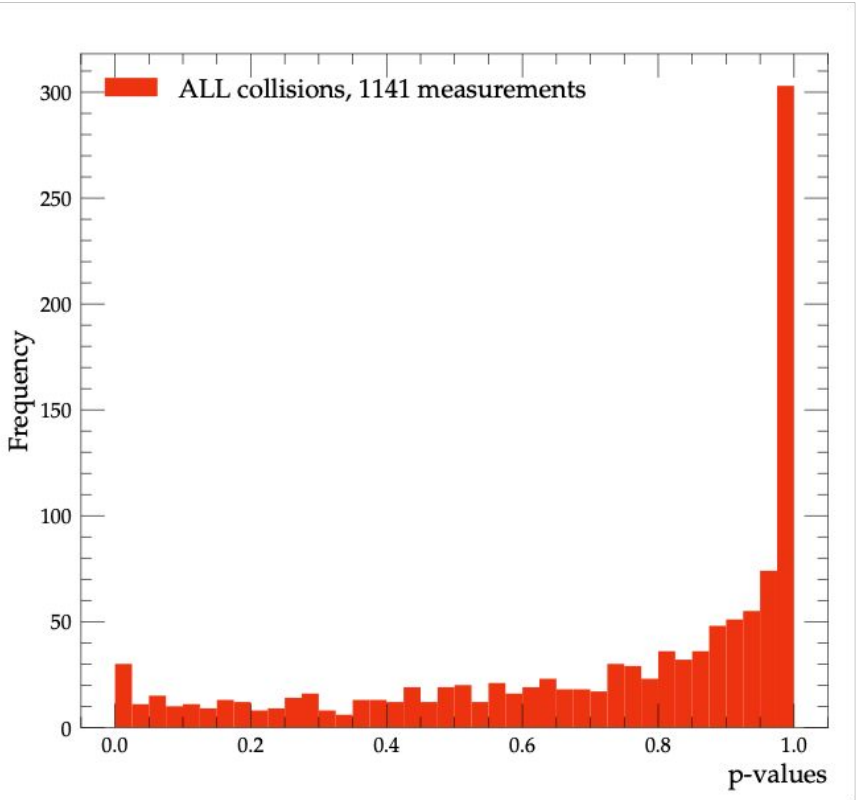

+∞
−∞
+∞
(3, 3)
masked
(0, 2)
(3, 2)
(0, 0)
(1, 0)
−∞


Z → ℓ⁺ℓ⁻, dressed level
Data
Model 1
Model 2
MC/Data
N_jets


2D heatmaps in YODA
Y
X
Z


stores metadata
AnalysisObject
0-dim.
n-dim.
live:
Counter
BinnedDbn
inert:
Estimate0D
BinnedEstimate
plotting:
ScatterND

Contur Extra



Contur and the ΣSM: Generating BSM events

Let's generate some events

- $ docker run -it --rm -v $PWD:/host hepstore/contur-herwig
- # cd /host/RunInfo/
- # ufo2herwig Sigma_SM_UFO
- # cd ..
- # contur-batch –single

Would create a single run directory with a runpoint_0000.sh script which could be executed to generate some events

However, Herwig doesn't work with python 3.14