Analysis preservation with Rivet & model tuning with Professor

Holger Schulz (IPPP Durham)

Fermilab, 5 June 2017

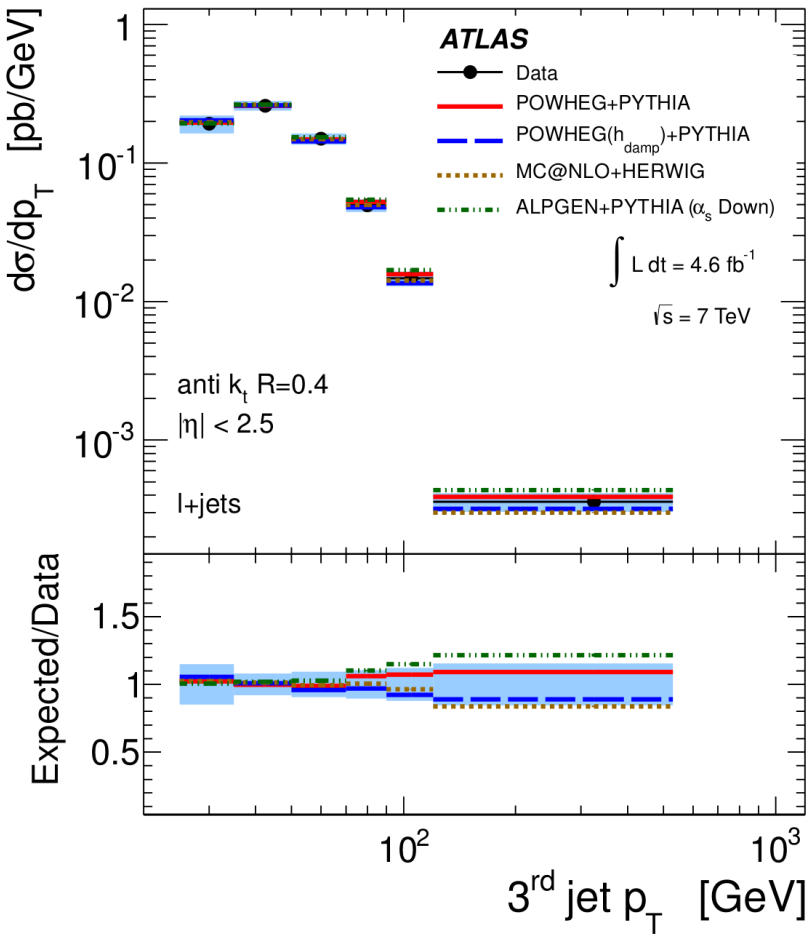
rivet.hepforge.org

professor.hepforge.org

1. Event generators and Rivet
2. Rivet + fast-sim for BSM searches
3. Model tuning with Professor

Event generators and Rivet

Data analyses



- **arXiv:1407.0891[hep-ex]**
- Analysis of ttbar events with ATLAS
- Fully corrected (unfolded) $\rightarrow$ compare with Truth level MC
- Data analysis is expensive in terms of computing and man-power
- How to compare the data with another model prediction?
- How to ensure we can do that also in 10 years?

Event generators for particle collisions

- Monte-Carlo (MC) event generators essential for most experimental work at e.g. LHC
- Event generators based on fundamental QFT — however "only" approximate (can't explicitly calculate all-orders, full multiplicities)
- Last ten years: huge development towards automation and higher accuracy
- Experimental purposes: *exclusive* events
  - Realistic final state (FS) particle multiplicities and composition ("data like")
  - Fortunately *really* possible to simulate e.g. $t\bar{t}$ events with thousands of FS particles
- Important to test MC predictions against data

Anatomy of a hadron collider event

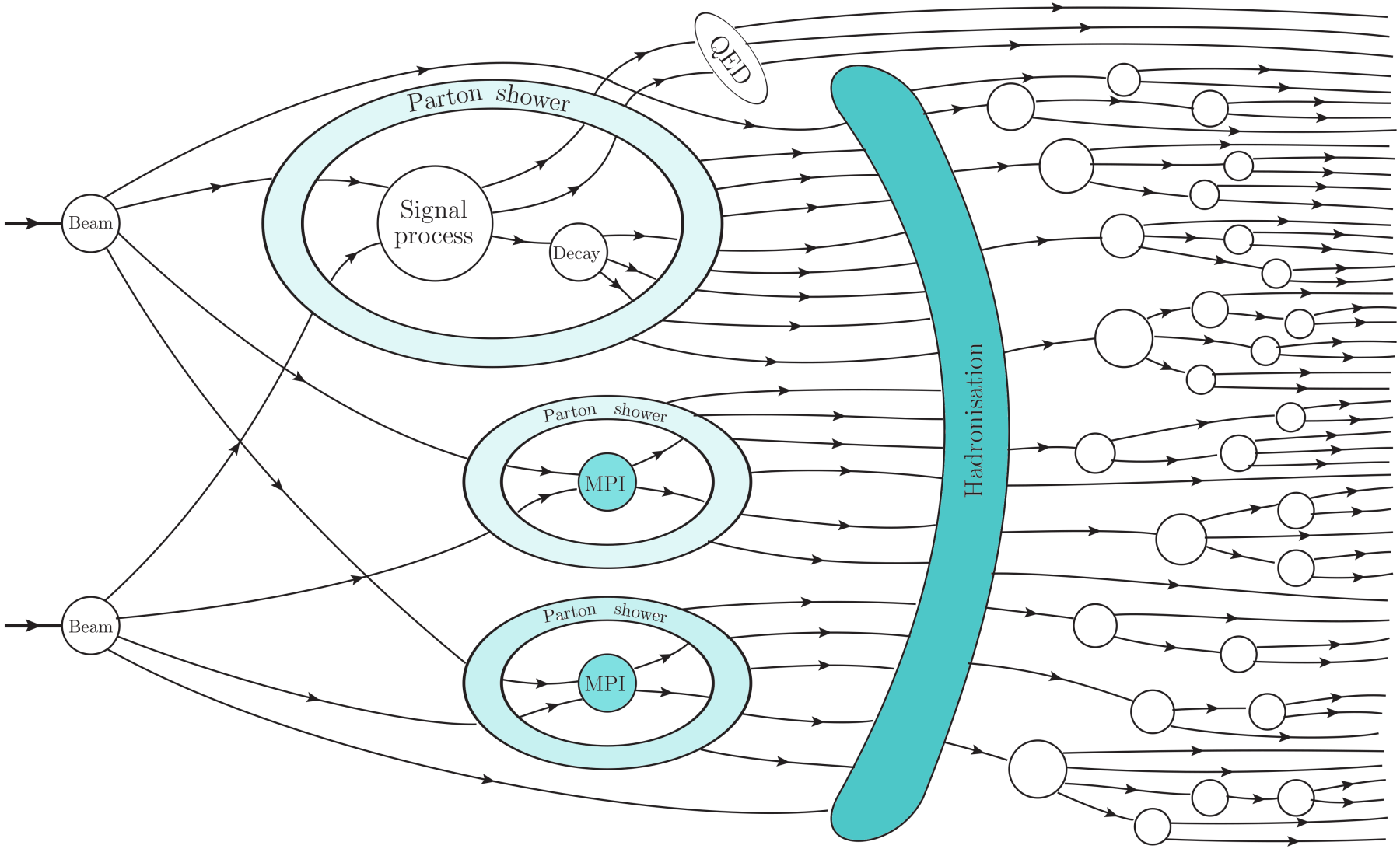


Image: Marek Schönherr

HepMC

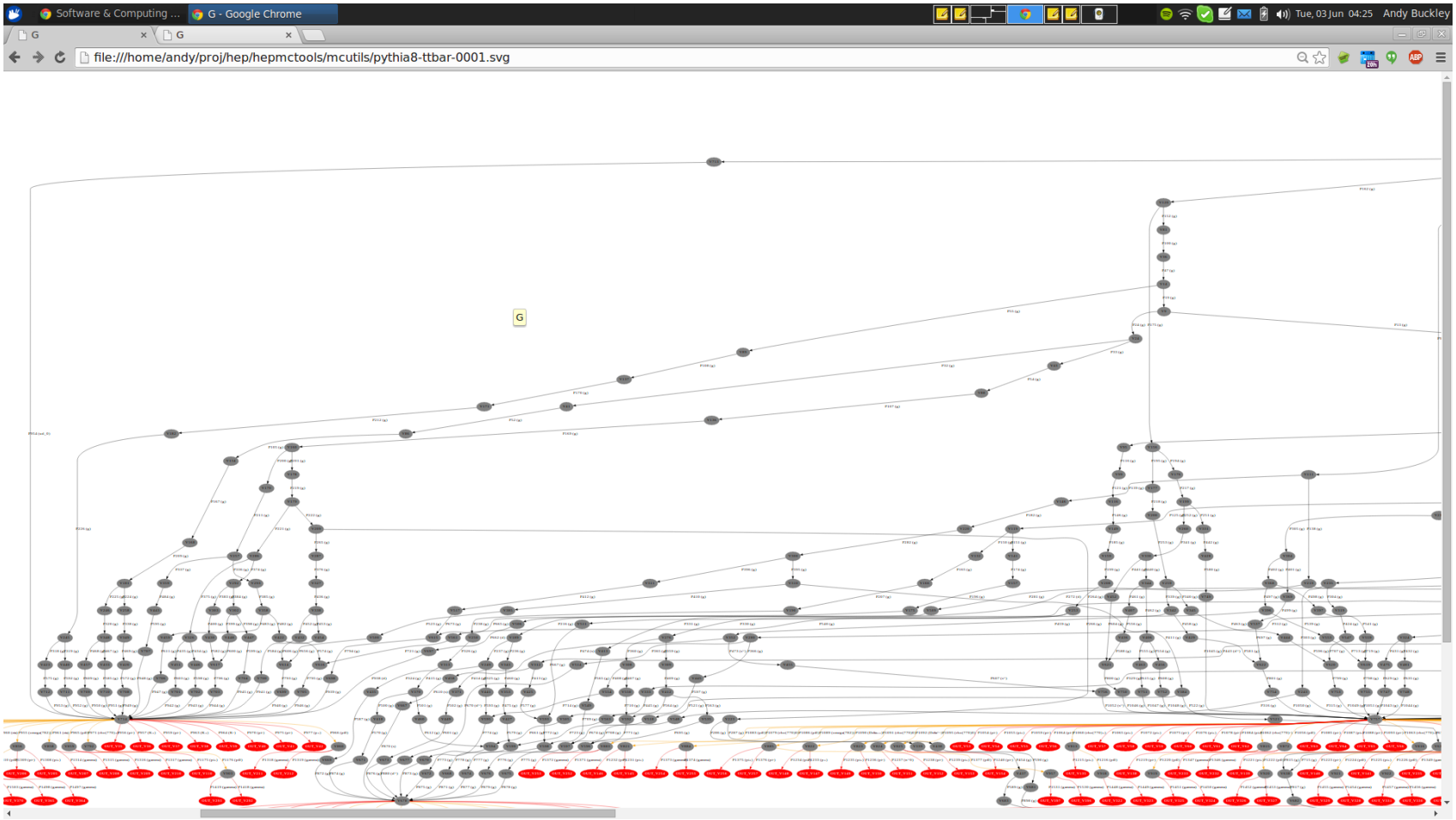
```
HepMC::Version 2.06.09
HepMC::IO_GenEvent-START_EVENT_LISTING
E 0 -1 -1.000000000000e+00 8.871071040596e-02 7.818608287725e-03 0 0 1 10001 10002 0 5 3.416101592648e+03
        7.483422617301e-08 3.416101592648e+03 1.200000000000e+01 0
N 5 "0" "1" "2" "3" "4"
U GEV MM
C 2.846751327207e+02 2.846751327207e+02
F 2 21 4.008893662122e-01 3.729543078303e-02 9.782054163953e+02 1.532449558314e-01 2.594332904812e+00 0 0
V -1 0 0 0 0 0 2 224 0
P 10001 2212 0 0 3.999999889956e+03 4.000000000000e+03 9.382719993929e-01 2 0 0 -1 0
P 10002 2212 0 0 -3.999999889956e+03 4.000000000000e+03 9.382719993929e-01 2 0 0 -1 0
P 10003 52 2.152701458984e+02 -4.008098606740e+01 1.224843865257e+02 2.510978815160e+02 1.000000000000e+01 1 0
P 10004 -52 -1.859496611203e+02 2.683906048726e+02 2.974023320540e+02 4.417679711860e+02 1.000000000000e+01 1 0
P 10005 -211 9.890420965096e-03 -7.232191998081e-02 -4.585734642526e-01 4.848687322907e-01 1.395700000000e
        -01 1
P 10006 211 -4.875521999232e-02 -6.391682129595e-01 -3.244942277751e+00 3.310595603025e+00 1.395700000000e-01 1
P 10007 -211 -5.419747849145e-01 1.109603099151e+00 -5.358114740881e+00 5.500348085938e+00 1.395700000000e-01 1
P 10008 211 -2.371586367548e-01 9.913161177003e-02 -7.510349282885e-01 8.059804860224e-01 1.395700000000e-01 1
P 10009 211 1.081239345290e+00 -1.119198919259e+00 3.588261440166e-01 1.603097230116e+00 1.395700000000e-01 1 0
P 10010 -2212 -6.650888213379e-02 7.369297556192e-01 3.366876238277e+00 3.572631921424e+00 9.382720000000e-01 1
P 10011 -211 8.982907262553e+00 -1.076204480402e+01 5.447163443754e+00 1.504012302540e+01 1.395699999998e-01 1
P 10012 211 1.093319138358e+00 -1.292226110997e+00 6.619801755435e-01 1.822880302695e+00 1.395700000000e-01 1 0
P 10013 211 2.896119078340e-02 -2.408931921491e-01 -1.202631135574e+00 1.234775167287e+00 1.395700000000e-01 1
P 10014 211 -3.568764257915e-01 -2.361806061069e-01 -3.470102086736e+00 3.499175665676e+00 1.395700000000e-01 1
P 10015 2112 -2.003637330934e+01 -2.382464016977e+01 1.759645813122e+02 1.786994303160e+02 9.395659999907e-01 1
P 10016 22 8.357442917026e-01 -2.581824372676e+00 1.269227052127e+01 1.297913774476e+01 -2.384185791016e-07 1 0
...
```

<+title+o

- Particle 4 vectors
- Vertices and genealogy
- Common format for most event generators
- Typically MB per event

Another look at an event graph

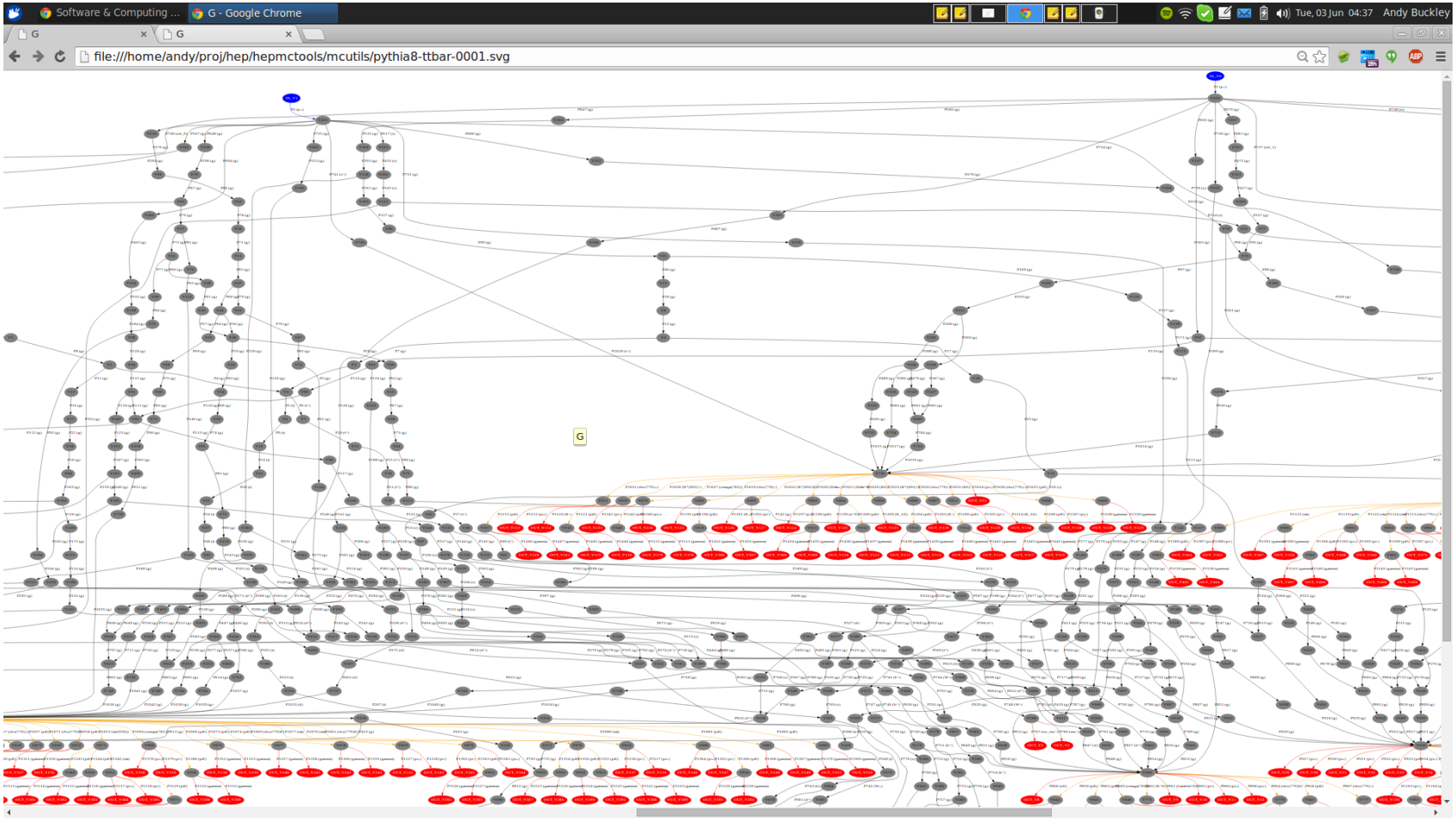
A Pythia8 $t\bar{t}$ event



Most of this is not standardised: Other MCs generating the same physics look *very* different here. But final states and decay chains have to have equivalent meaning.

Another look at an event graph

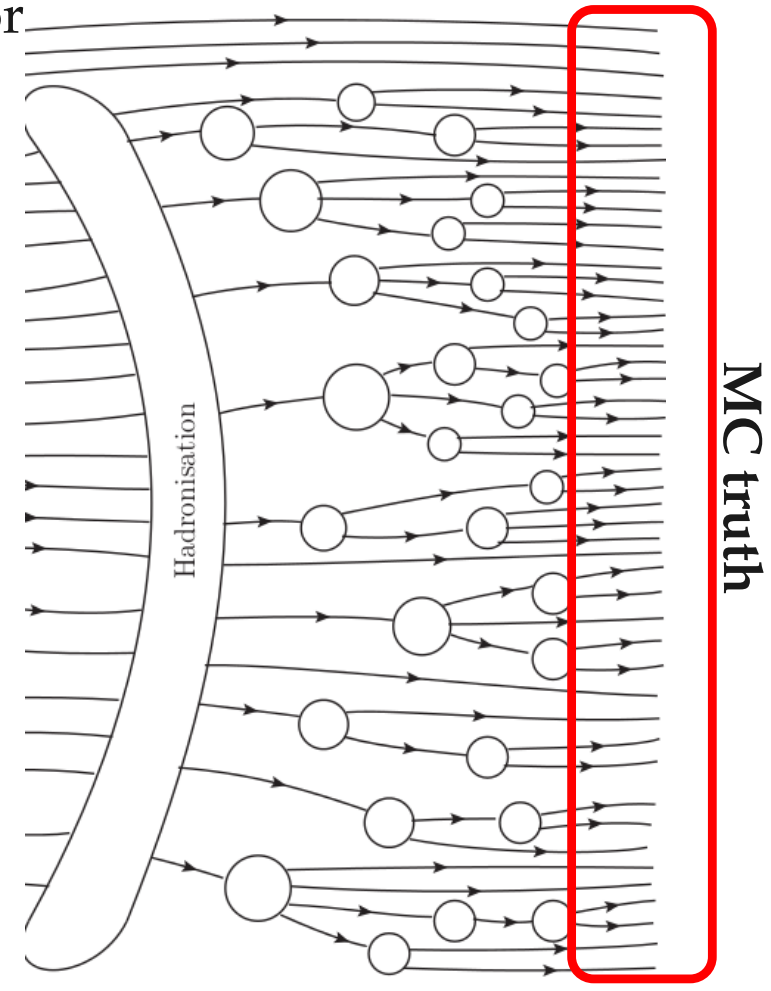
A Pythia8 $t\bar{t}$ event



Most of this is not standardised: Other MCs generating the same physics look *very* different here. But final states and decay chains have to have equivalent meaning.

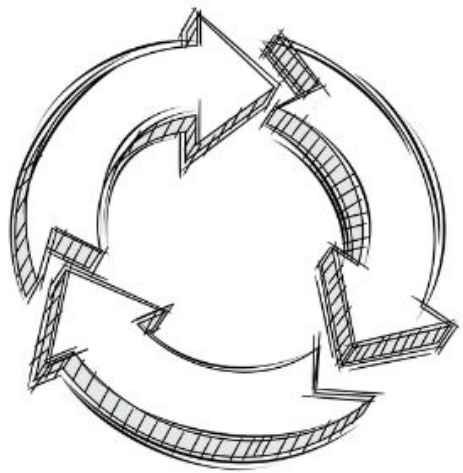
Rivet

- Analysis tool for MC events, generator agnostic via HepMC
- Provides most relevant methods for multi particle final states:
  - Cuts
  - Jets
  - Boson finders
  - Event shapes
  - …
- Writes out histogram (YODA format)
- External infrastructure: HepData, inspire
- Implementation and validation of new (data) analyses now largely provided by experiments
- Easy to write new analyses for signal and background estimates from MC



Rivet

- **Recent big changes in LHC experiment/theory interaction**
  - $\Rightarrow$ more direct collaboration to improve methods and modelling, starting from SM & QCD, now also Top, Higgs, and BSM
- **Rivet** analysis library is part of this: a lightweight way to exchanging analysis details and ideas
- Implementing a Rivet analysis to complement the data analysis is increasingly expected of LHC analyses. **Everyone benefits!**
- One dedicated contact person in CMS and ATLAS



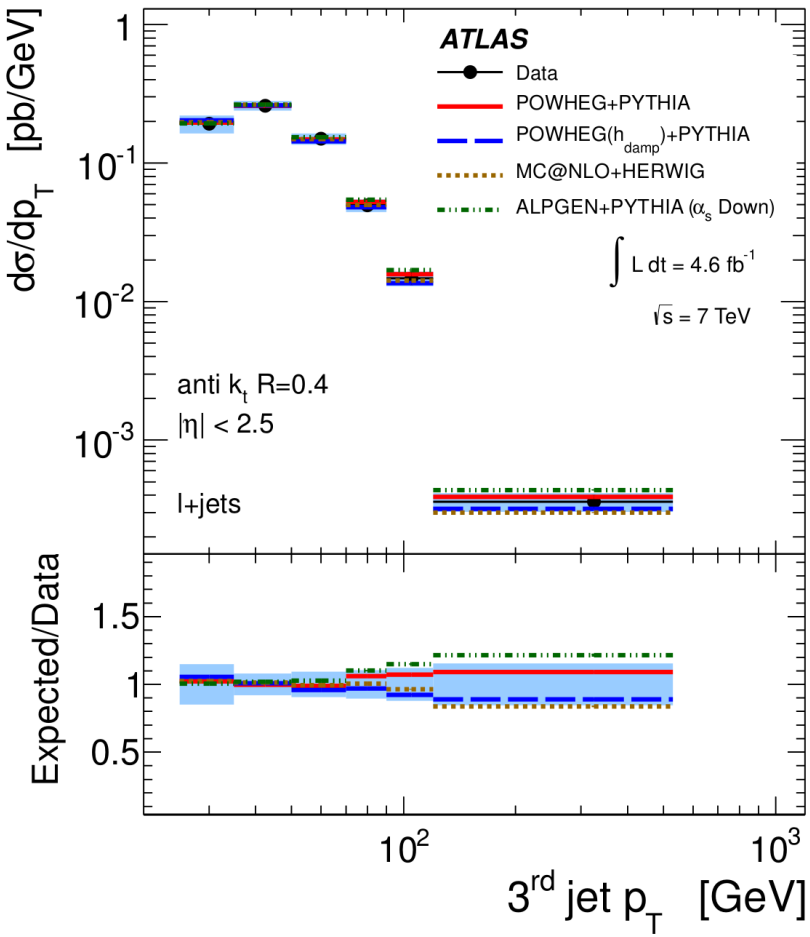
Rivet

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

427 built-in, at today's count! 54 are pure MC, and some double/triple-counting

- *Generator-agnostic* for physics & pragmatics
- A quick, easy and powerful way to get physics plots from lots of MC gens
  - Only requirement: use **HepMC** event record
  - Usually via ASCII, but in-memory exchange is faster
- Rivet has become the LHC standard for archiving LHC data analyses
  - Focus on *unfolded* measurements, esp. QCD and EW+QCD, rather than searches
  - But there are BSM studies using it! **And detector simulation now possible**
  - Key input to MC validation and tuning – increasingly comprehensive coverage
  - Also "recasting" of SM and BSM data results on to new/more general BSM model spaces
  - **Add your analyses, too!**

Analysis naming scheme



- `ATLAS_2014_I1304688`
- `EXPERIMENT_YEAR_INSPIREKEY`
- http://inspire-hep.net/record/1304688
- This particular histogram is `ATLAS_2014_I1304688/d02-x03-y01`
- http://hepdata.cedar.ac.uk/view/ins1304688, table 2, row 3

Design philosophy / pragmatics

Rivet operates on HepMC events, intentionally unaware of who made them. . . so don't "look inside" the event graph.
$\Rightarrow$ reconstruct resonances, dress leptons, avoid partons, etc.

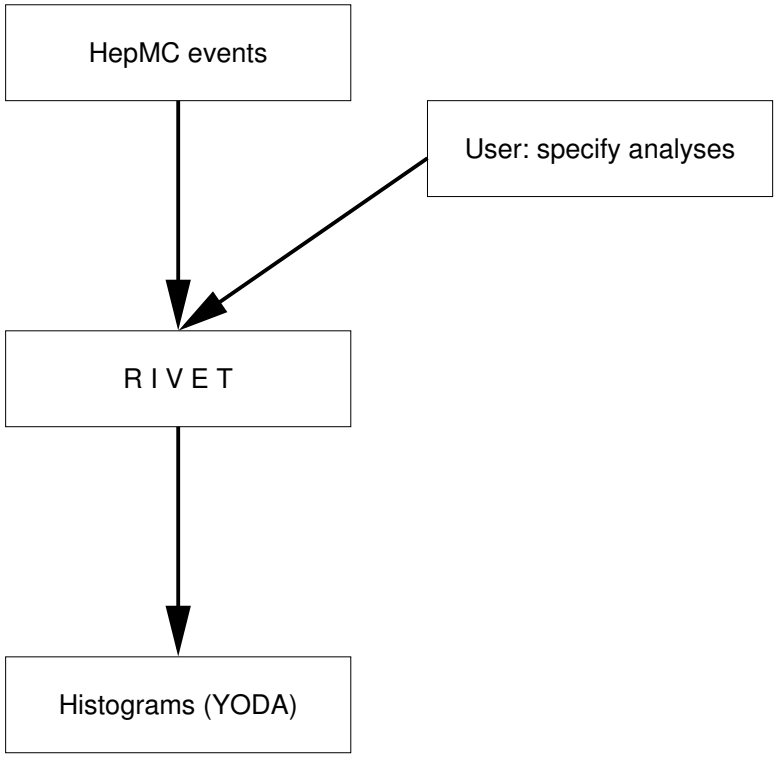
This "hard work" way is actually simpler – fewer gotchas.
Makes you think about physics & helps find analysis bugs/ambiguities

Tech stuff:

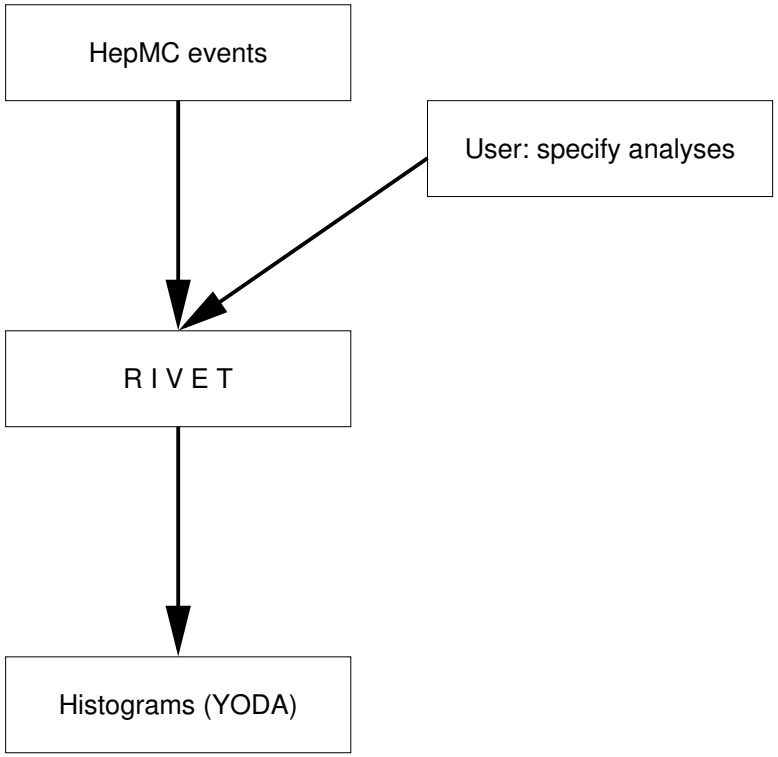
- C++ library with Python interface & scripts
- "Plugins" $\Rightarrow$ write your analyses without needing to rebuild Rivet
  Trivial from user / analysis author point of view
- Tools to make "doing things properly" easy and default
- Computation caching for efficiency
- Histogram syncing: *keep code clean and clear*

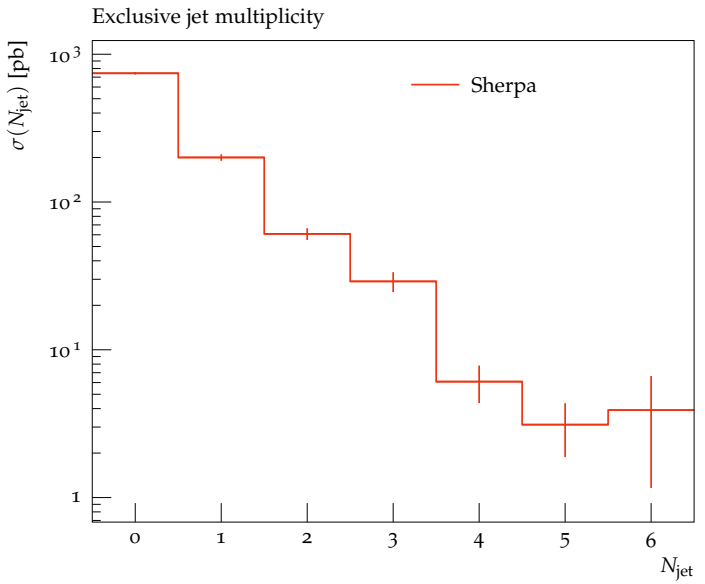
+ helpful developers! New contributors always welcome

Basic principle

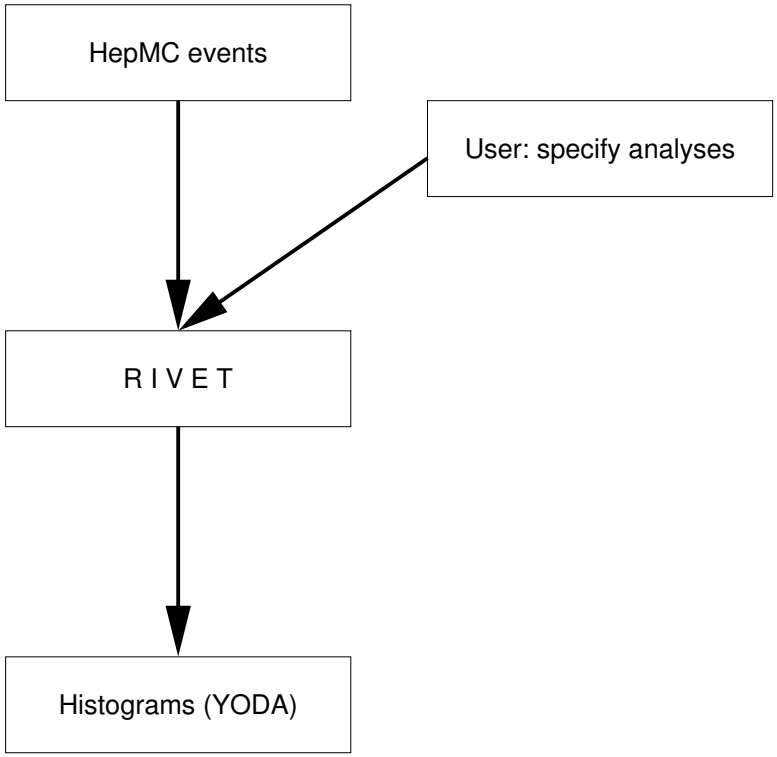


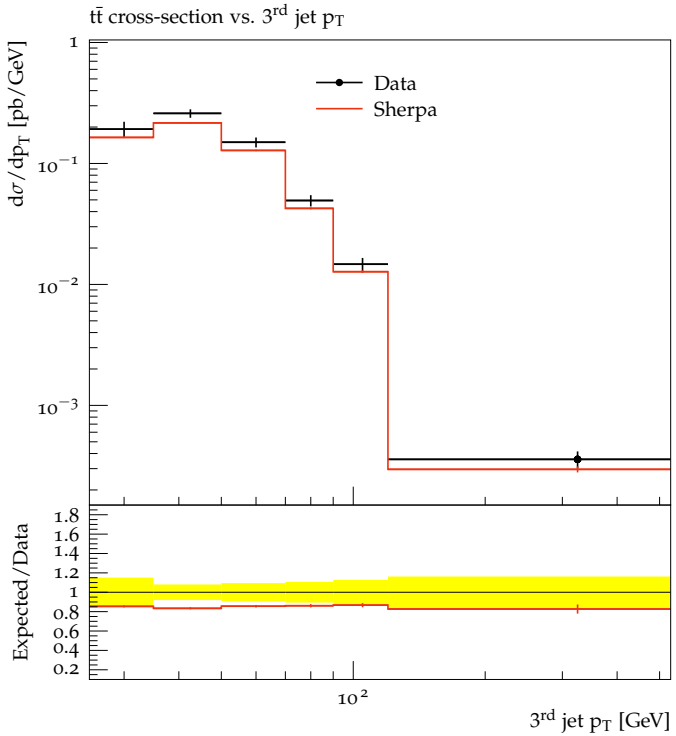
Basic principle





Basic principle





Output example

```
BEGIN YODA_HISTO1D /CMS_2015_I1384119/d01-x01-y01
IsRef=1
Path=/CMS_2015_I1384119/d01-x01-y01
ScaledBy=1.0000000000000005e-04
Title=
Type=Histo1D
XLabel=
YLabel=
# Mean: 2.047907e-03
# Area: 2.128100e+01
# ID  ID  sumw  sumw2  sumwx  sumwx2  numEntries
Total  Total  2.128100e+01 2.128100e-03 4.358152e-02 2.926641e+01 212810
Underflow Underflow 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0
Overflow Overflow 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0
# xlow  xhigh  sumw  sumw2  sumwx  sumwx2  numEntries
-2.000000e+00 -1.800000e+00 1.105200e+00 1.105200e-04 -2.100061e+00 3.994160e+00 11052
-1.800000e+00 -1.600000e+00 1.111800e+00 1.111800e-04 -1.889834e+00 3.216089e+00 11118
-1.600000e+00 -1.400000e+00 1.090400e+00 1.090400e-04 -1.636227e+00 2.458917e+00 10904
-1.400000e+00 -1.200000e+00 1.100500e+00 1.100500e-04 -1.430107e+00 1.862041e+00 11005
-1.200000e+00 -1.000000e+00 1.074200e+00 1.074200e-04 -1.181831e+00 1.303798e+00 10742
-1.000000e+00 -8.000000e-01 1.063700e+00 1.063700e-04 -9.578032e-01 8.659895e-01 10637
-8.000000e-01 -6.000000e-01 1.048100e+00 1.048100e-04 -7.341084e-01 5.176510e-01 10481
-6.000000e-01 -4.000000e-01 1.037100e+00 1.037100e-04 -5.192426e-01 2.634387e-01 10371
-4.000000e-01 -2.000000e-01 1.015100e+00 1.015100e-04 -3.042990e-01 9.460527e-02 10151
-2.000000e-01 0.000000e+00 9.916000e-01 9.916000e-05 -9.947841e-02 1.330623e-02 9916
0.000000e+00 2.000000e-01 1.001600e+00 1.001600e-04 1.008998e-01 1.352624e-02 10016
2.000000e-01 4.000000e-01 9.917000e-01 9.917000e-05 2.981036e-01 9.293782e-02 9917
4.000000e-01 6.000000e-01 1.020100e+00 1.020100e-04 5.094963e-01 2.578570e-01 10201
6.000000e-01 8.000000e-01 1.045300e+00 1.045300e-04 7.323970e-01 5.166559e-01 10453
8.000000e-01 1.000000e+00 1.068300e+00 1.068300e-04 9.613896e-01 8.687205e-01 10683
1.000000e+00 1.200000e+00 1.076400e+00 1.076400e-04 1.184727e+00 1.307530e+00 10764
1.200000e+00 1.400000e+00 1.093400e+00 1.093400e-04 1.422497e+00 1.854267e+00 10934
1.400000e+00 1.600000e+00 1.111300e+00 1.111300e-04 1.665988e+00 2.501220e+00 11113
1.600000e+00 1.800000e+00 1.121100e+00 1.121100e-04 1.904675e+00 3.239614e+00 11211
1.800000e+00 2.000000e+00 1.114100e+00 1.114100e-04 2.116399e+00 4.024084e+00 11141
END YODA_HISTO1D
```

Getting Rivet

Easy to install using our *bootstrap script*:

```
wget http://rivet.hepforge.org/hg/bootstrap/raw-file/2.5.3/rivet-bootstrap
bash rivet-bootstrap
```

Latest version is 2.5.3 Requires C++11

Docker image available:

```
docker pull hepstore/rivet:2.5.3
```

http://rivet.hepforge.org/trac/wiki/Docker

`CVMFS` installations on lxplus.

Getting Rivet

- `rivet` command line tool to query available analyses
- Can be used as a library (e.g. in big experiment software frameworks)
- Can also be used from the command line to read HepMC ASCII files/pipes: very convenient
- Helper scripts like `rivet-mkanalysis`, `rivet-buildplugin`
- Histogram comparisons, plot web albums, etc. very easy

Docs online at **http://rivet.hepforge.org** – PDF manual, HTML list of existing analyses, and Doxygen. Entries in HEPdata point to existing rivet analyses.

Writing an analysis

Writing an analysis is of course more involved. But the C++ interface is pretty friendly: most analyses are short, simple, and readable – details handled in the library + expressive API functions.

A single C++ file is sufficient. Rivet comes with scripts that generate analysis templates and compile the new code into a shared library (plugin).

Mostly "normal":

- Typical init/exec/fin structure
- Histogram titles, labels, etc.: use `.plot` file
- Rivet's own Particle, Jet and FourMomentum classes: some nice things like `abseta()` and `abspid()`, sorting and filtering
- Use of *projections* for computations, with a bit of magic – this is where the caching happens
- Projections are *declared* with a string name, and later are *applied* using the same name
- Final state projections are central: compute from final state or physical decayed particles

Projections

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.

Projections were *declared* with a name in `init()` they are then *applied* to the current event in `analyze()`, by the same name.

E.g.

- Final states (Identified, Charged, Visible, . . .)
- Jets (All native FastJet algorithms)
- Event shapes (Thrust etc.)
- Missing momentum and DIS kinematics

Selection cuts

Combinable `Cut` objects:

- `FinalState(Cuts::pT > 0.5*GeV && Cuts::abseta < 2.5)`
- `fs.particles(Cuts::absrap < 3 || (Cuts::absrap > 3.2 && Cuts::absrap < 5), cmpMomByEta)`

Can also use cuts on PID and charge:

- `fs.particlesByPt(Cuts::abspid == PID::ELECTRON)`, or
- `FinalState(Cuts::charge != 0)`

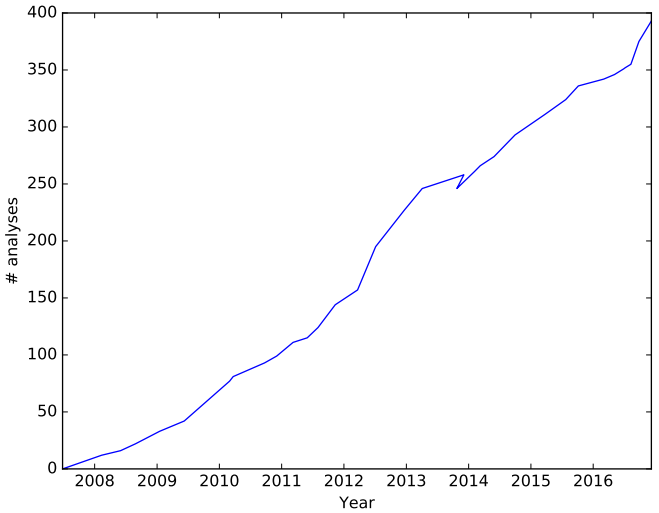
Use of functions/functors for ParticleFinder filtering is coming...

Rivet + fast-sim for BSM searches

BSM analysis coverage

Currently $\sim$ 427 analyses total & $\sim$ 230 LHC alone

- Until recently only 27 dedicated BSM searches – and BSM-sensitive SM measurements
- SM focus on unfolded observables, not sufficient for most BSM studies
- Rivet 2.5.0 introduced detector smearing machinery. *For BSM only!*



NB. glitch is Rivet 1.x $\rightarrow$ 2.x migration.
Note recent acceleration!

- $\Rightarrow$ 9 more BSM routines in last few months:
  - ATLAS: ICHEP 2016 3-lepton & same-sign 2-lepton, 1-lepton + jets, 1-lepton + many jets, jets + MET; 2015 jets + MET and monojet
  - CMS: ICHEP 2016 jets + MET; 8 TeV α_T + b-jets
  - *Partially* validated – not many cutflows available!
  - Also added tools to help with object filtering, cutflows, etc.
  - Important as real-world examples of how to write BSM routines
- **Rivet is in good shape for preserving new physics searches!**

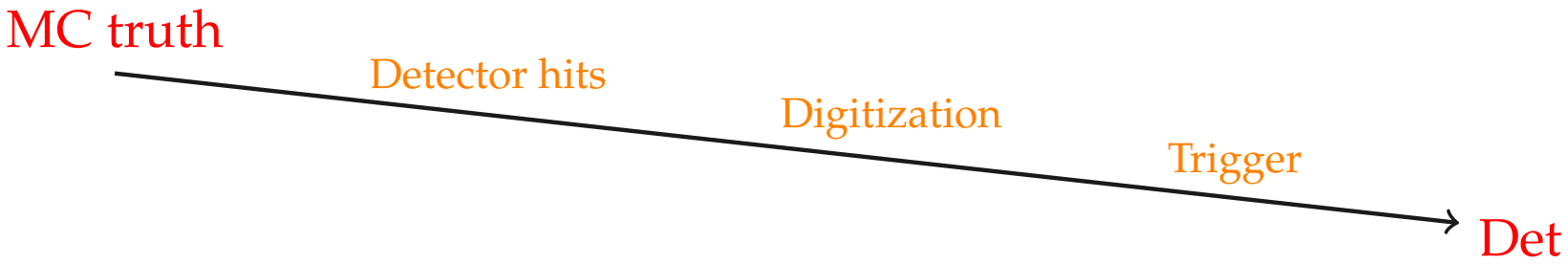
BSM & detector effects

Explicit fast detector simulation vs. smearing/efficiencies

MC truth

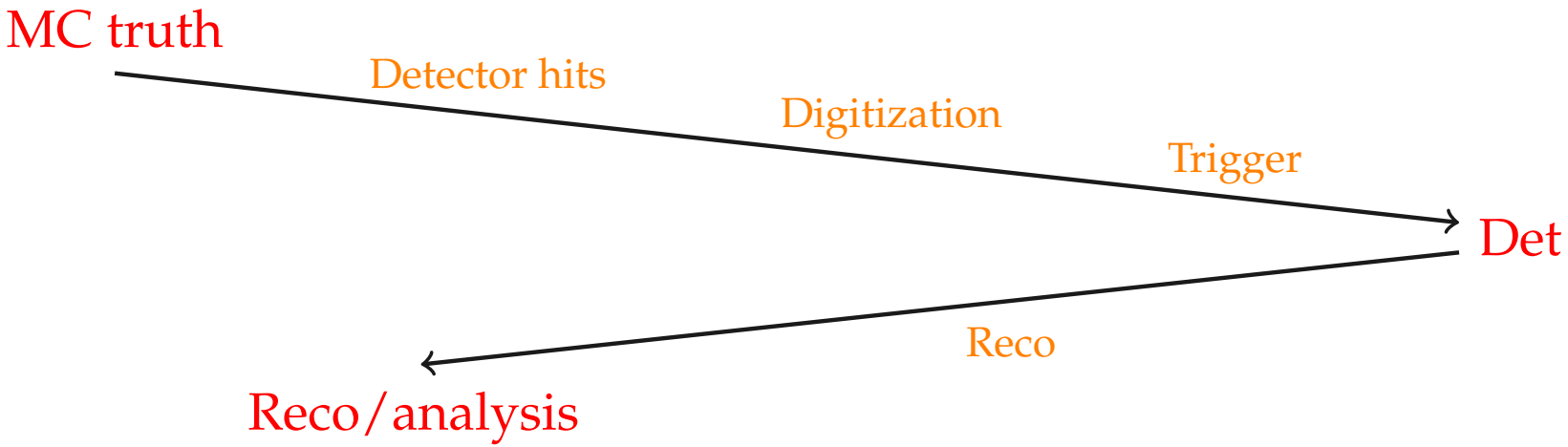
BSM & detector effects

Explicit fast detector simulation vs. smearing/efficiencies



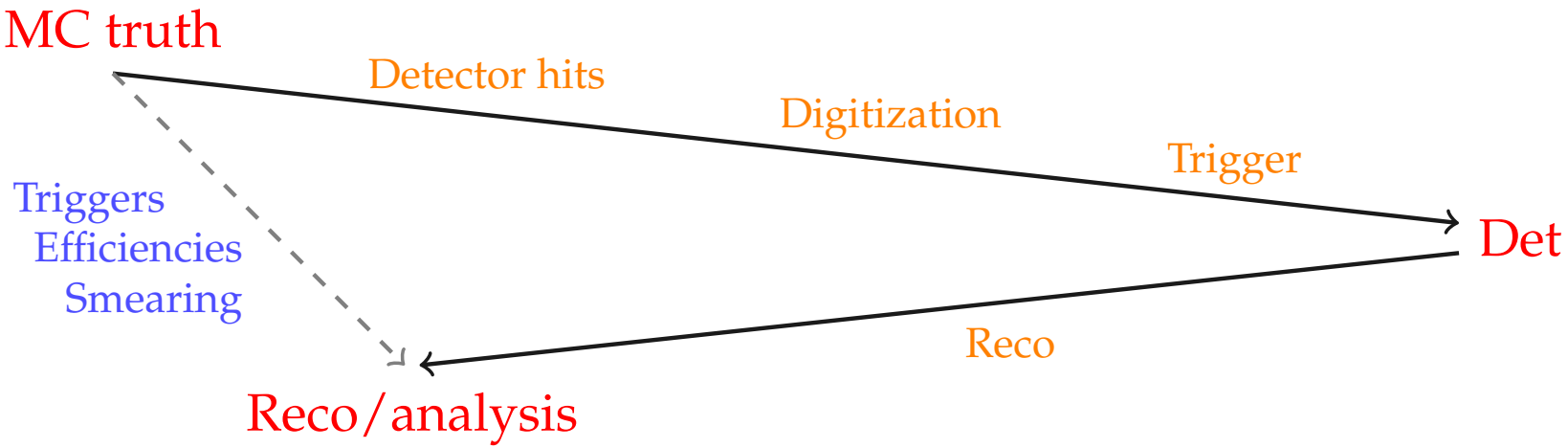
BSM & detector effects

Explicit fast detector simulation vs. smearing/efficiencies



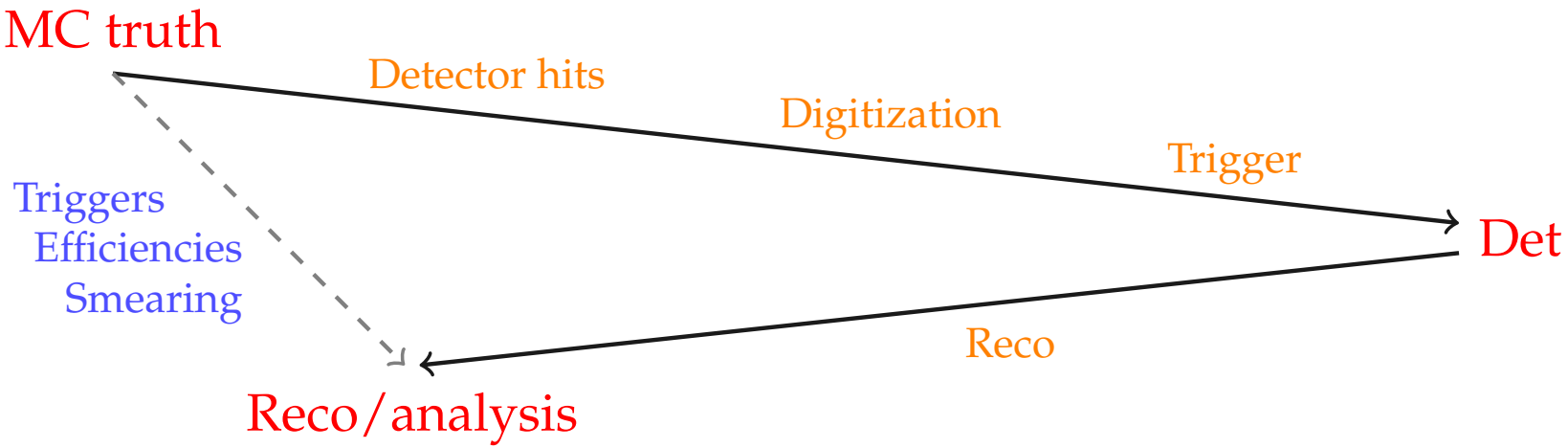
BSM & detector effects

Explicit fast detector simulation vs. smearing/efficiencies



BSM & detector effects

Explicit fast detector simulation vs. smearing/efficiencies

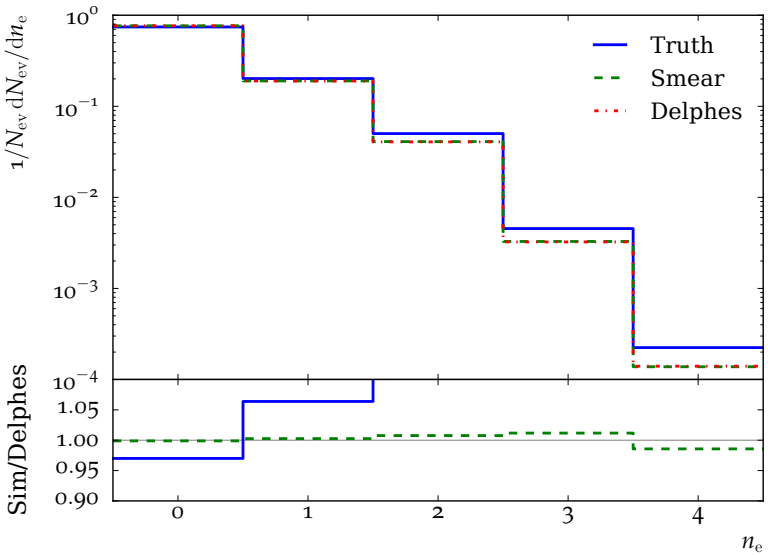


- **Explicit fast-sim takes the "long way round".**
- **Reco already reverses most detector effects!**
- Reco calibration to MC truth: smearing is a few-percent effect
- (Lepton) efficiency & mis-ID functions dominate – and are tabulated in both approaches
- Smearing is more flexible: effs change with phase-space, reco version, run, . . . and need to guarantee *stability* for preservation

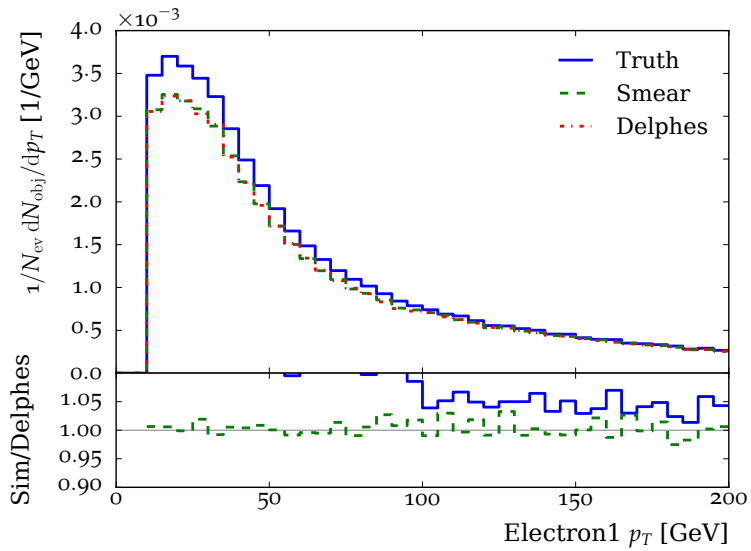
Smearing vs. fast sim vs. MC truth

CMSSM eff/smearing effects from Rivet, in turn using some DELPHES and paper/note calibration functions:

Electron multiplicity



Leading electron p_T

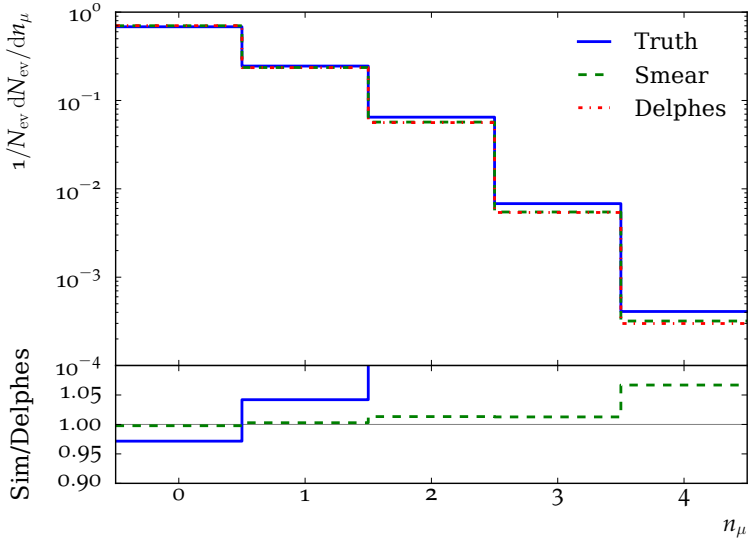


Note major lepton shifts from blue truth to green smeared: difference w.r.t red DELPHES very small

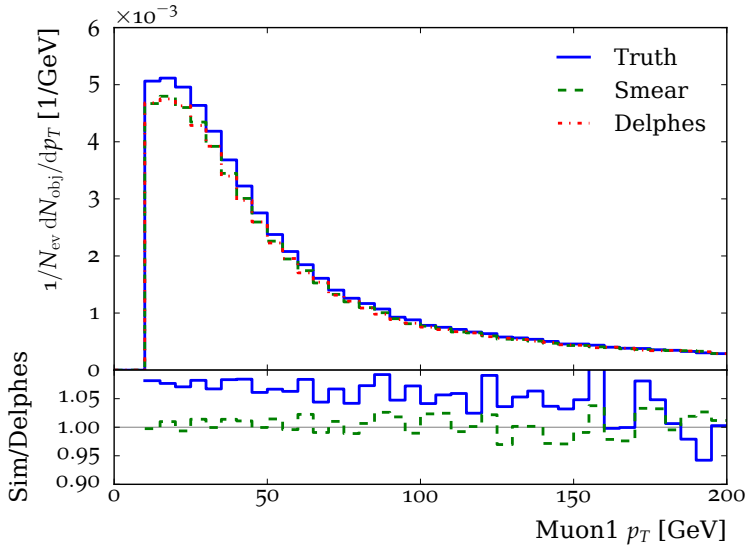
Smearing vs. fast sim vs. MC truth

CMSSM eff/smearing effects from Rivet, in turn using some DELPHES and paper/note calibration functions:

Muon multiplicity



Leading muon p_T



Note major lepton shifts from blue truth to green smeared: difference w.r.t red DELPHES very small

BSM & detector effects (II) ⇒ Rivet 2.5

In addition to last slides, *flexibility* of det-sim is important:

- "Global" fast-sims hence difficult for coverage of multiple experiments, multiple runs, multiple reco calibrations, etc.
- Analysis-specific efficiencies and smearings are more precise and allow use of multiple jet sizes, tagger & ID working points, isolations, . . . ⇒ **many variations in real analyses**

⇒ Rivet det-sim as effs+smearing, localised per-analysis

Rivet internally caches results, so global effect sim still efficient

- Functions for generic ATLAS & CMS performance in Runs 1 & 2
- Inline or analysis-specific functions easy to write & *chain*
- Eff/smearing functions can be used directly, e.g. for object filtering
- Working on embeddability for multithreaded fitters/samplers.

Selection tools for search analyses

Search analyses typically do a lot more "object filtering" than measurements. Rivet 2.5 provides a lot of tools to make this complex logic expressive:

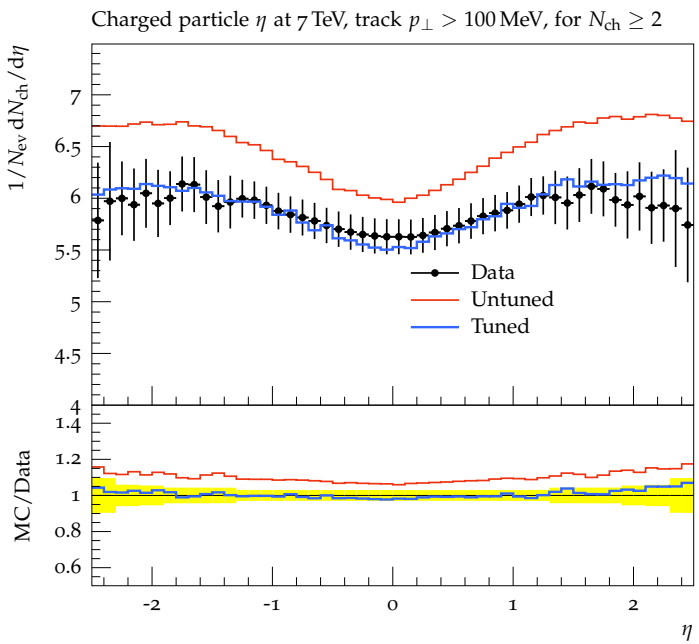
- Filtering functions: `filter_select`, `filter_discard` + `ifilter_*` in-place variants
- Lots of *functors* for common "stateful" filtering criteria:
  `PtGtr(10*GeV)`, `EtaLess(5)`, `AbsEtaGtr(2.5)`, `DeltaRGtr(mom, 0.4)`
- Cut-flow monitor via `#include "Rivet/Tools/Cutflow.hh"`

```
Particles elecs = apply<ParticleFinder>(event, "Electrons").particles(Cuts::pT > 10*GeV);
Jets jets = apply<JetAlg>(event, "Jets").jetsByPt(Cuts::pT > 20*GeV && Cuts::abseta < 2.8);
// Remove electrons within dR = 0.2 of a b-tagged jet
for (const Jet& j : jets)
  if (j.abseta() < 2.5 && j.pT() > 50*GeV && j.bTagged(Cuts::pT > 5*GeV))
    ifilter_discard(elecs, deltaRLess(j, 0.2, RAPIDITY));
// Remove any |eta| < 2.8 jet within dR = 0.2 of a remaining electron
for (const Particle& e : elecs)
  ifilter_discard(jets, deltaRLess(e, 0.2, RAPIDITY));
```

Model tuning with Professor

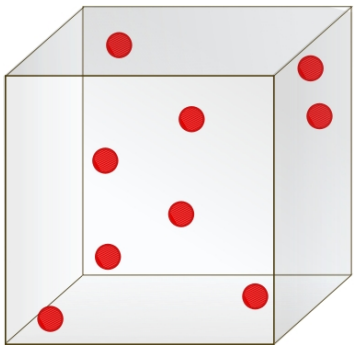
Tuning

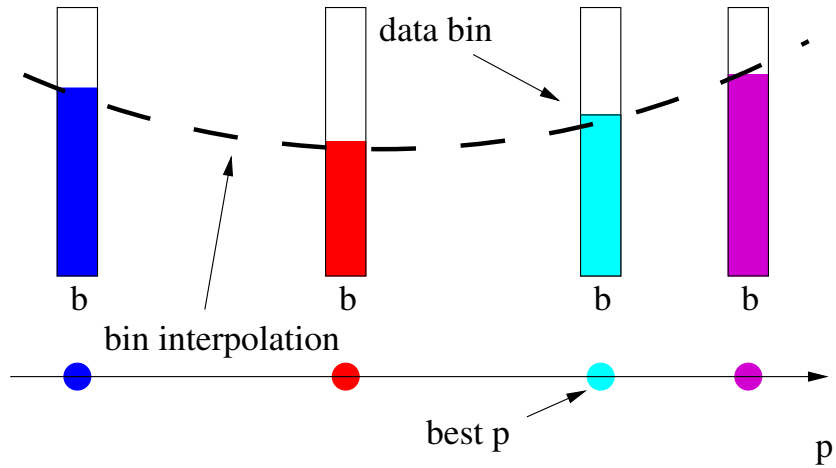
- Realistic events contain physics at low scales where perturbation breaks down
- Rely on model assumptions that introduce many parameters
- Need to find "meaningful" settings
- Can be done manually but hard to on reasonable time-scale



Tuning with Professor in a nutshell

- Random sampling: N parameter points in n-dimensional space
- Run generator and fill histograms (e.g. Rivet) trivial parallel
- For each bin:
  - Don't care about actual dependence on parameters
  - Polynomial approximation *per bin*
- Construct overall (now trivial) $\chi^2(\vec{p}) \approx \sum_{bins} \frac{(D_b - I_b(\vec{p}))^2}{error^2}$
- and numerically *minimise* with **`iminuit`**

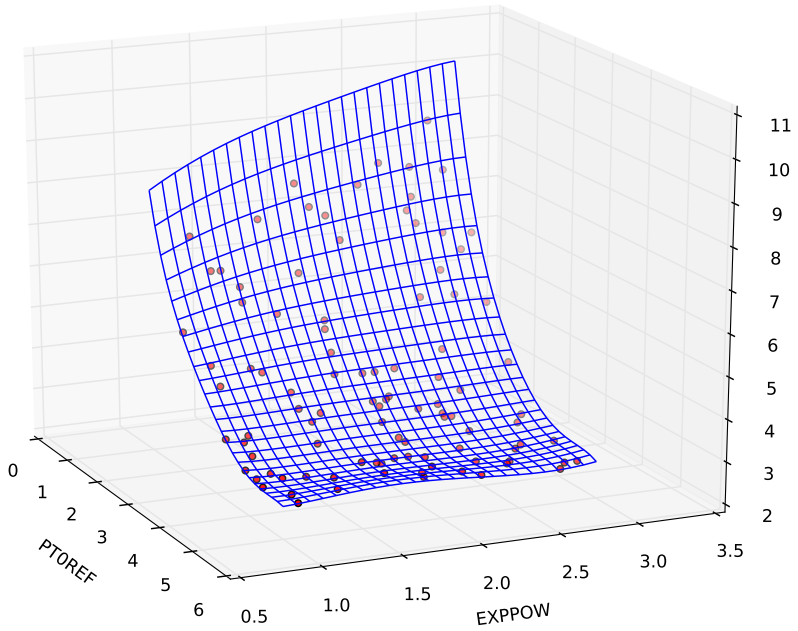




Fitting model

1 bin example, 2 parameters (x,y), 2nd order polynomial

$$\mathrm{MC}_b(\vec{p}) \approx \alpha_0^{(b)} + \sum \beta_i^{(b)}\, p'_i + \sum_{i \le j} \gamma_{ij}^{(b)}\, p'_i\, p'_j$$



$$c^{(\vec{b})} = (\alpha, \beta_x, \beta_y, \gamma_{xx}, \gamma_{xy}, \gamma_{yy})$$

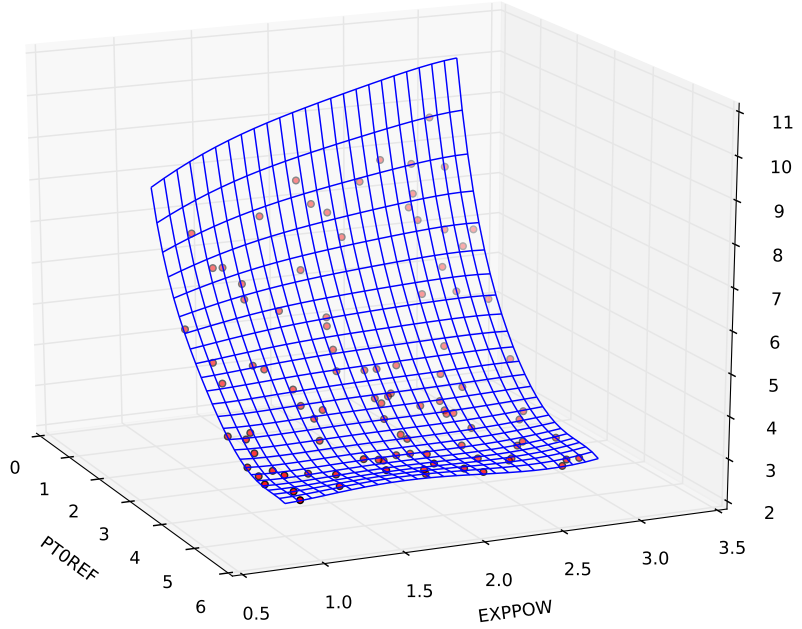
$$\tilde{p}_i = (1, x_i, y_i, x_i^2, x_i y_i, y_i^2)$$

$$\mathrm{MC}_b(\vec{p}) \approx \sum_{i=1}^{N_{\min}(P)} c_i^{(b)}\, \tilde{p}_i$$

Fitting model

1 bin example, 2 parameters (x,y), 2nd order polynomial

$$\mathrm{MC}_b(\vec{p}) \approx \alpha_0^{(b)} + \sum \beta_i^{(b)}\, p'_i + \sum_{i\leq j} \gamma_{ij}^{(b)}\, p'_i\, p'_j$$



$$c^{(\vec{b})} = (\alpha, \beta_x, \beta_y, \gamma_{xx}, \gamma_{xy}, \gamma_{yy})$$

$$\tilde{p}_i = (1, x_i, y_i, x_i^2, x_i y_i, y_i^2)$$

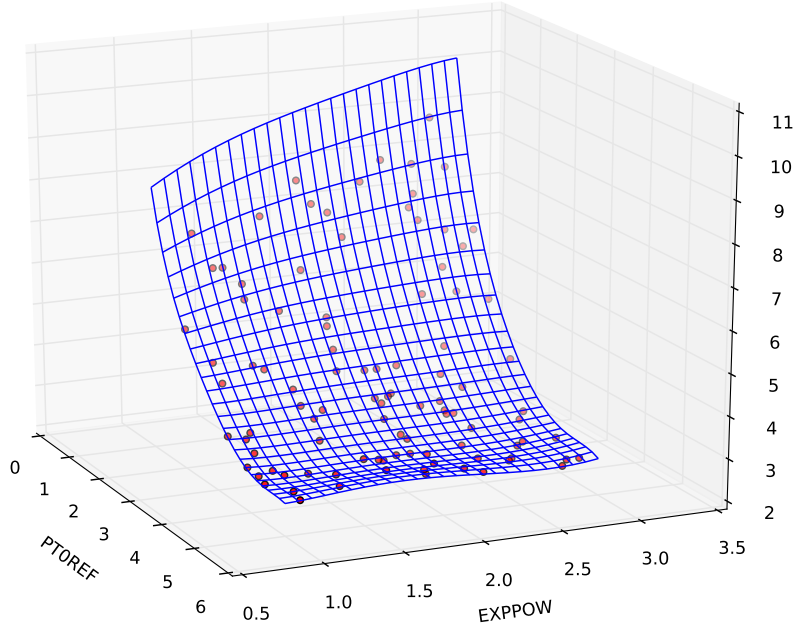
$$\mathrm{MC}_b(\vec{p}) \approx \sum_{i=1}^{N_{\min}(P)} c_i^{(b)}\, \tilde{p}_i$$

$$\underbrace{\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}}_{\vec{\mathrm{MC}}_b} = \underbrace{\begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 \\ & & & \vdots & & \\ 1 & x_N & y_N & x_N^2 & x_N y_N & y_N^2 \end{pmatrix}}_{\tilde{P}=\{\tilde{p}_i\}} \underbrace{\begin{pmatrix} \alpha_0 \\ \beta_x \\ \beta_y \\ \gamma_{xx} \\ \gamma_{xy} \\ \gamma_{yy} \end{pmatrix}}_{\vec{c}^{(b)}}$$

Fitting model

1 bin example, 2 parameters (x,y), 2nd order polynomial

$$\mathrm{MC}_b(\vec{p}) \approx \alpha_0^{(b)} + \sum \beta_i^{(b)}\, p_i' + \sum_{i\le j} \gamma_{ij}^{(b)}\, p_i'\, p_j'$$



$$c^{(\vec{b})} = (\alpha, \beta_x, \beta_y, \gamma_{xx}, \gamma_{xy}, \gamma_{yy})$$

$$\tilde{p}_i = (1, x_i, y_i, x_i^2, x_i y_i, y_i^2)$$

$$\mathrm{MC}_b(\vec{p}) \approx \sum_{i=1}^{N_{\min}(P)} c_i^{(b)}\, \tilde{p}_i$$

$$\underbrace{\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}}_{\vec{\mathrm{MC}}_b} = \underbrace{\begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 \\ & & & \vdots & & \\ 1 & x_N & y_N & x_N^2 & x_N y_N & y_N^2 \end{pmatrix}}_{\tilde{P}=\{\tilde{p}_i\}} \underbrace{\begin{pmatrix} \alpha_0 \\ \beta_x \\ \beta_y \\ \gamma_{xx} \\ \gamma_{xy} \\ \gamma_{yy} \end{pmatrix}}_{\vec{c}^{(b)}}$$

$$c^{(\vec{b})} = \mathcal{I}[\tilde{P}]\vec{\mathrm{MC}}_b.$$

Fitting model

- $\mathcal{I}[\tilde{P}]$ is the pseudo-inverse of $\tilde{P}$
- $\mathcal{I}[\tilde{P}]$ is calculated using singular value decomposition (SVD)
- SVD is least-squares fit
- We need at least as many $\tilde{p}_i$ as there are coefficients

$$\underbrace{\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}}_{\vec{\mathrm{MC}}_b} = \underbrace{\begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 \\ & & & \vdots & & \\ 1 & x_N & y_N & x_N^2 & x_N y_N & y_N^2 \end{pmatrix}}_{\tilde{P}=\{\tilde{p}_i\}} \underbrace{\begin{pmatrix} \alpha_0 \\ \beta_x \\ \beta_y \\ \gamma_{xx} \\ \gamma_{xy} \\ \gamma_{yy} \end{pmatrix}}_{\vec{c}^{(b)}}$$

$$c^{(\vec{b})} = \mathcal{I}[\tilde{P}]\vec{\mathrm{MC}}_b$$

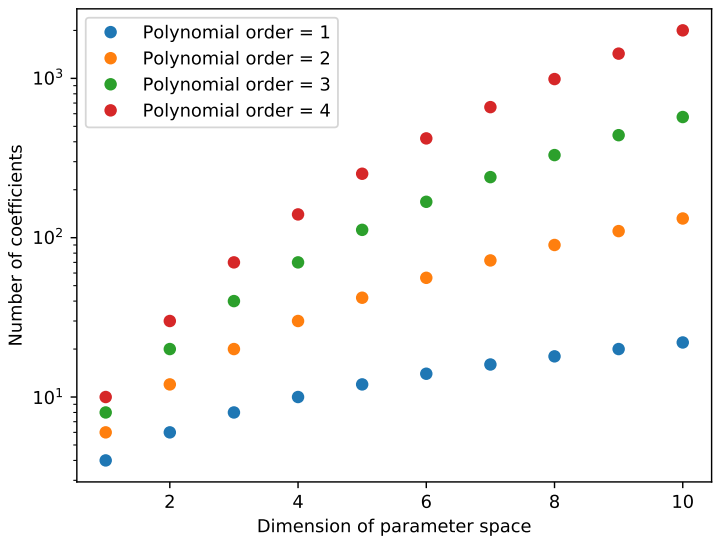
- With $c_i^{(b)}$ calculated $\rightarrow$ prediction

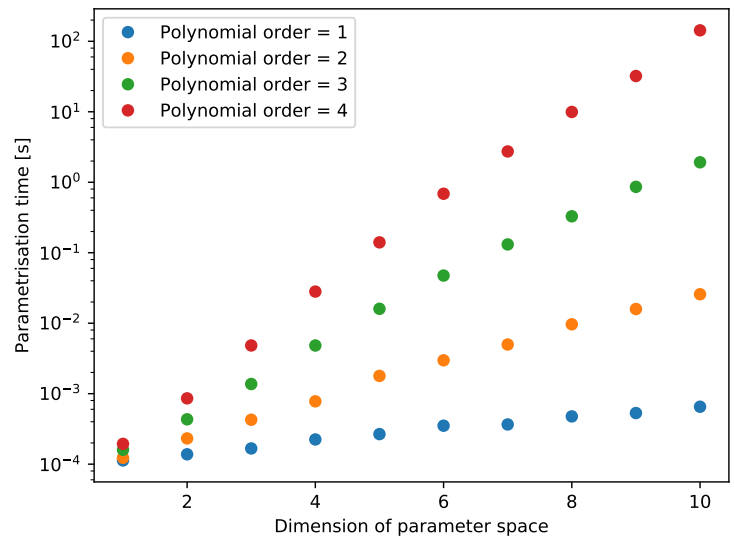
$$\mathrm{MC}_b(\vec{p}) \approx \sum_{i=1}^{N_{\min}(P)} c_i^{(b)}\, \tilde{p}_i$$

  for any $\vec{p}$ in *milliseconds*
- Separate polynomials for central value and uncertainty of a bin

Professor technicalities

- C++ core functionality, python bindings for everything else
- In case of MC, input generation trivial to do in parallel (different points in parameter space)
- Result is fast analytic pseudo-generator
- Storage of coefficients as plain text file $\rightarrow$ can use parameterisation in other C++ codes





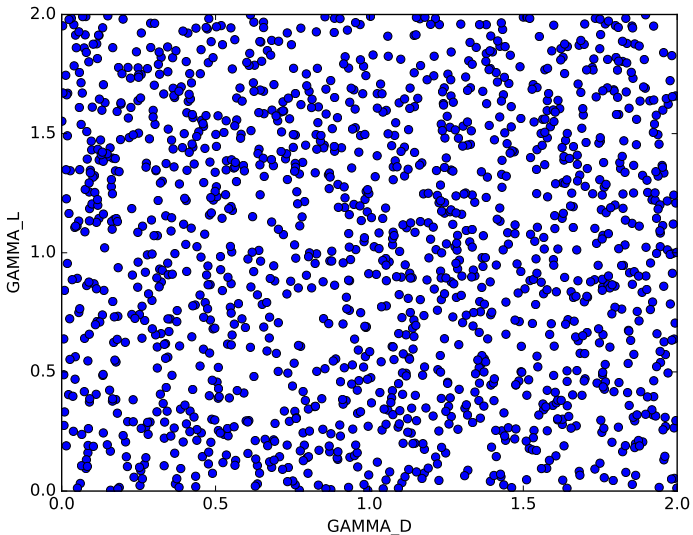
Example output

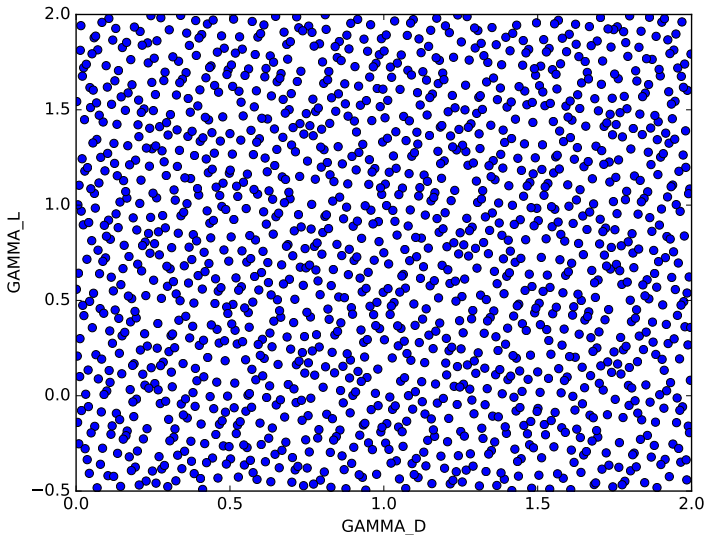
- Plain text file
- Meta information, parameter space, polynomial coefficients

```
DataDir: /home/hschulz/Sherpa/Tuning—NewHad—rel—2—2—4/2017—05—22—bla/scan2
ProfVersion: 2.2.2beta2
Date: 2017—06—01 13:10:01
DataFormat: binned 2
ParamNames: KT_O ALPHA_D ALPHA_L ALPHA_H GAMMA_D GAMMA_L GAMMA_H STRANGE_FRACTION BARYON_FRACTION
Dimension: 9
MinParamVals: 0.000674 —0.497223 —0.499586 0.000757 0.000580 0.002653 0.504251 0.250288 0.250298
MaxParamVals: 2.248679 1.998837 1.997479 3.999912 1.998901 1.998744 7.999943 0.749977 0.749808
DoParamScaling: 1
NumInputs: 1484
/ALEPH_1991_S2435284/d01—x01—y01#0 1.00000e+00 3.00000e+00
  val: 9 2 —0.00360025 —0.00100512 0.000544102 0.000296846 0.000213943 —6.35394e—06 —0.000699236 —0.00021037
          0.000510953 0.000588548 0.00654176 —0.000914022 —0.000716218 0.000180158 —0.00333635 0.00132221 —4.3627e—05
           —0.000422647 —0.000267011 0.000627989 —0.000548664 0.00176204 0.000364814 —0.00053499 0.000451779
          —0.000277334 —0.000844911 —0.00125288 0.0197997 —0.000322684 0.00048657 0.000335561 0.00748714 0.000241382
          —0.000162675 —0.0185692 0.00166507 0.000238913 —0.000310697 —0.000593107 —0.000290523 —0.000245745
          —3.22819e—05 0.000135187 —0.00107326 —0.000518321 0.00102011 0.000345511 —6.48719e—05 —0.00151553
          —7.40607e—05 0.000112481 —0.00291144 0.000185875 2.37507e—05
  err: 9 2 0.000181519 —5.02472e—05 8.08095e—06 0.00013151 3.28828e—05 1.61701e—05 0.000153176 8.04158e—05 1.87033e
          —05 —7.41364e—05 6.47977e—05 0.000105081 —0.00018051 —0.00010014 —7.04912e—05 —5.71483e—05 —2.17563e
          —05 6.00403e—06 —1.02219e—05 —3.50495e—05 —1.21488e—05 3.6774e—05 4.73355e—05 —5.82478e—05 —7.81474
          e—05 —2.7055e—05 2.44168e—05 —0.000104596 —0.000124106 —4.96546e—05 —2.92521e—05 1.82974e—05 9.29349
          e—05 9.0611e—05 7.48066e—05 —2.10497e—05 —4.00846e—05 —3.43163e—05 2.72302e—05 —8.66645e—05 5.17795
          e—05 5.00373e—06 0.000117281 —1.30487e—05 4.24917e—05 0.000133045 —5.13495e—06 —0.000112468 —1.16532e
          —05 3.84505e—05 5.59006e—05 6.78926e—05 9.02897e—05 —1.57199e—05 —0.000171531
```

Sampling

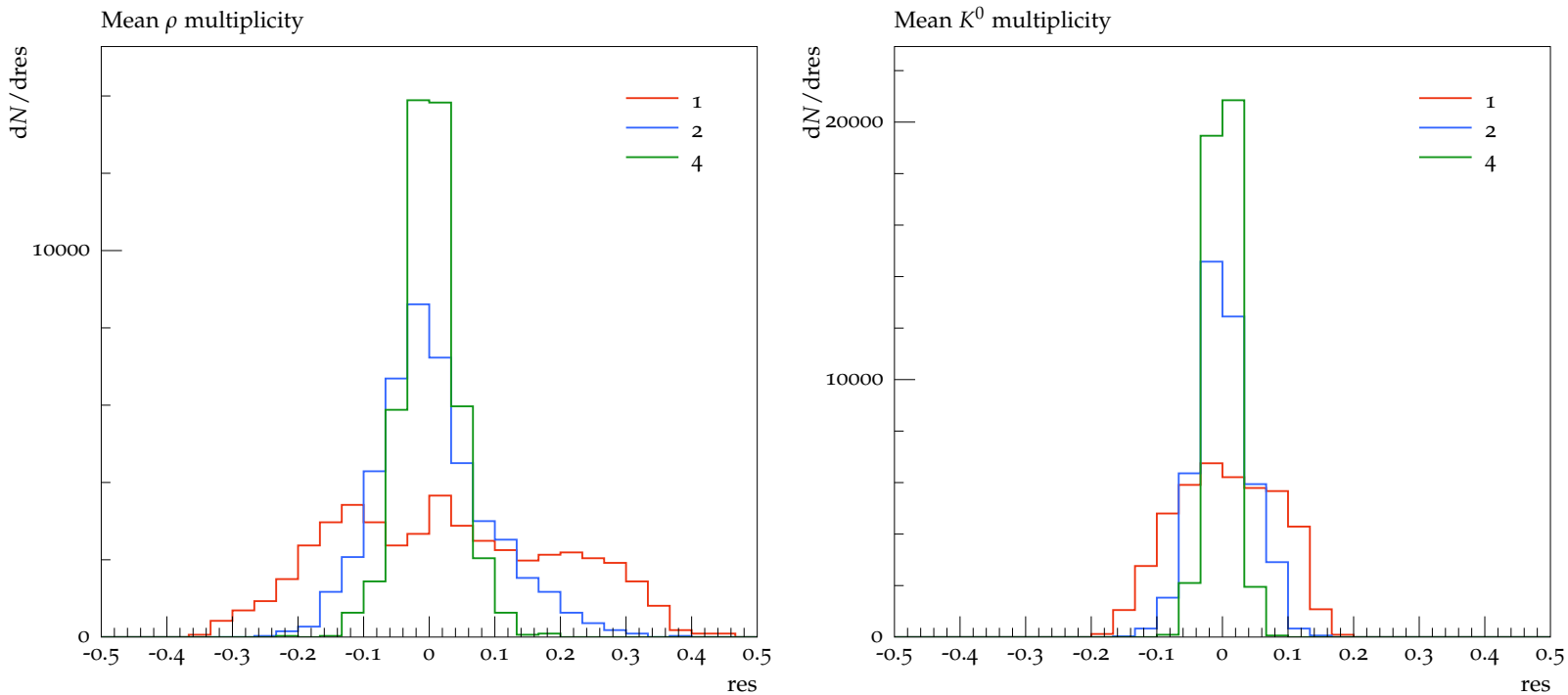
- Define parameter space with input textfile
- Can bias sampling to avoid say unphysical parameter space
- Convenient template instantiation for e.g. generator steering cards
- Random uniform or Sobol sampling
- So far no clear preference





Residuals

- SVD is a least squares fit through points $\vec{p}$
- For each of those we know the exact value $\mathrm{MC}(\vec{p})$
- Define residual as distance between the two:
  $\mathrm{res} = \left[I(\vec{p}) - \mathrm{MC}(\vec{p})\right] / I(\vec{p})$
- Put all res into histogram, expect something symmetrical around 0

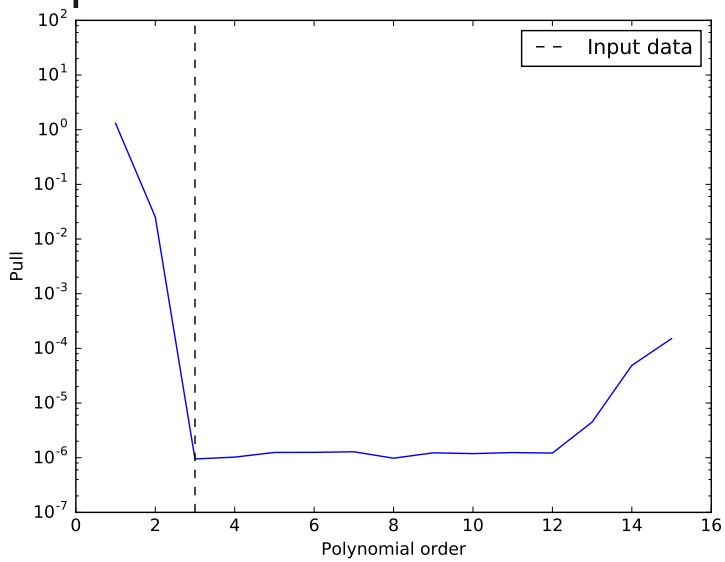


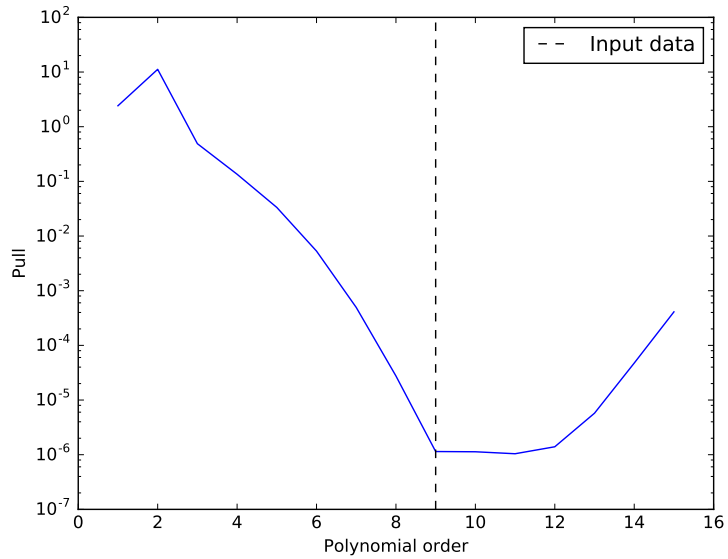
Jackknifing

- Separate input data into training and test sample
- Run parameterisation for *training* sample
- Calculate pull

$$\langle \text{pull} \rangle = \frac{1}{N_{\text{testpoints}}} \sum_{\vec{p} \in \text{ testsample}} \frac{I(\vec{p}) - \text{MC}(\vec{p})}{I(\vec{p})}$$

- Shows which polynomial order is best suited
- Also shows limitation of polynomials when exceeding machine precision





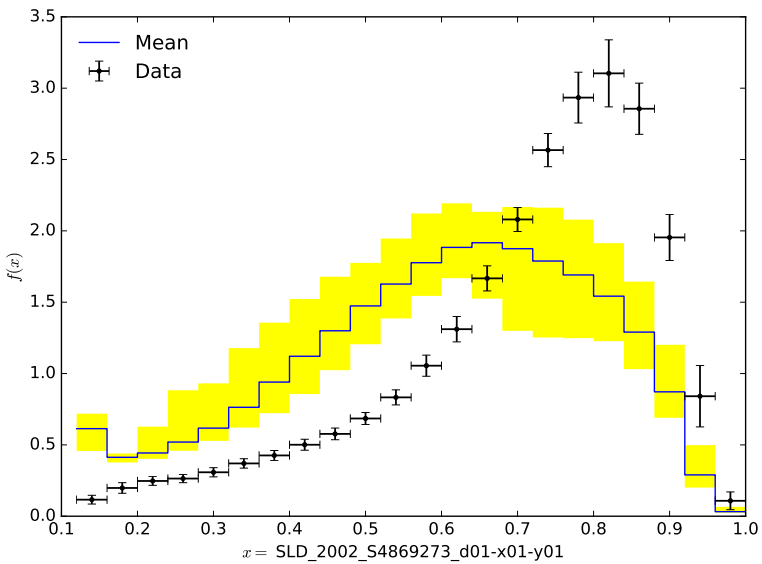
Left: Input data generated from 3$^{\text{rd}}$ order polynomial
Right: Input data generated from 9$^{\text{th}}$ order polynomial

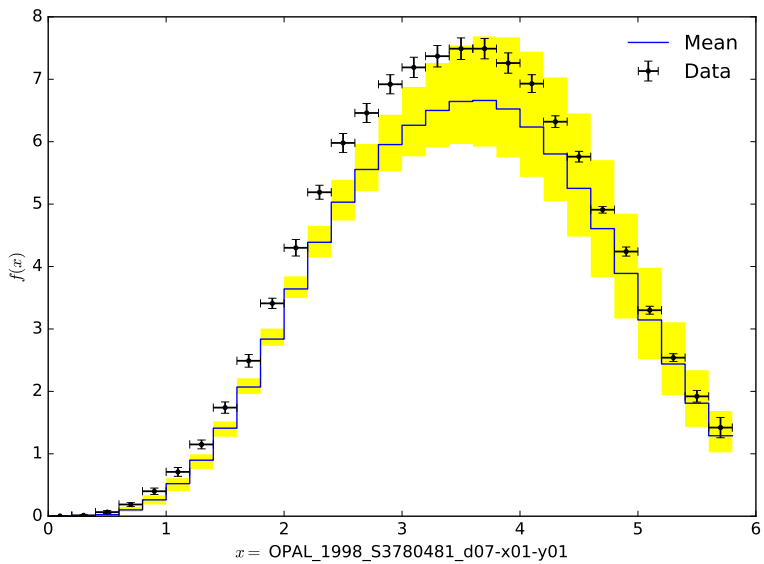
Weights and choice of observables

- Tuning: reasonable measure between model prediction and data
- By default, ad-hoc "chi square" inspired goodness-of-fit

$$\chi^2(\vec{p}) = \sum_{\mathcal{O}} \sum_{b \in \mathcal{O}} w_b \cdot \frac{(f^{(b)}(\vec{p}) - \mathcal{R}_b)^2}{\Delta_b^2(\vec{p})}$$

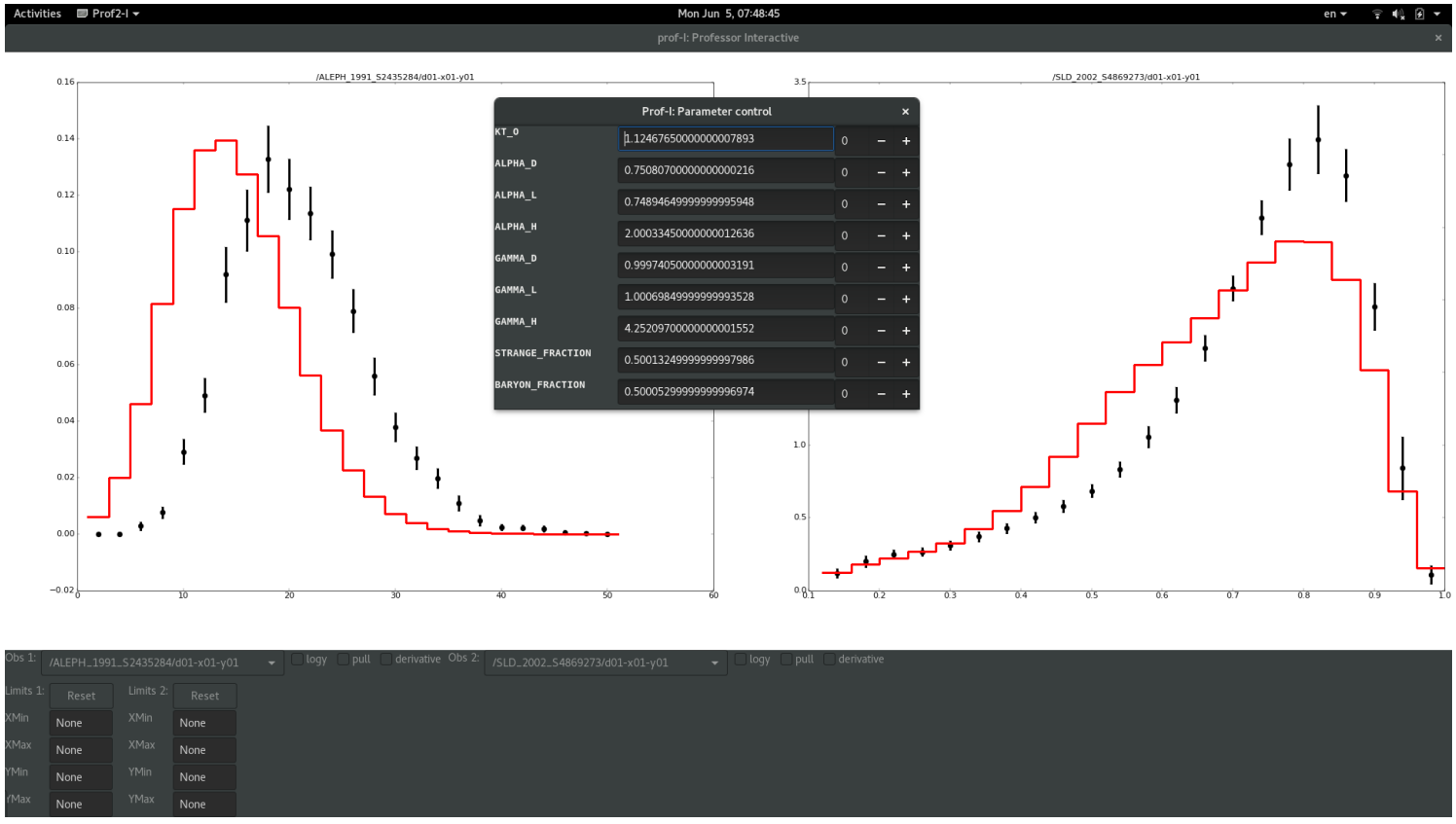
- Weights w_b neccessary because models not perfect:
  - Exclude regions of observables (e.g. bad coverage, breakdown of polynomial approximation)
  - Force good description of certain observables (at the cost of others)





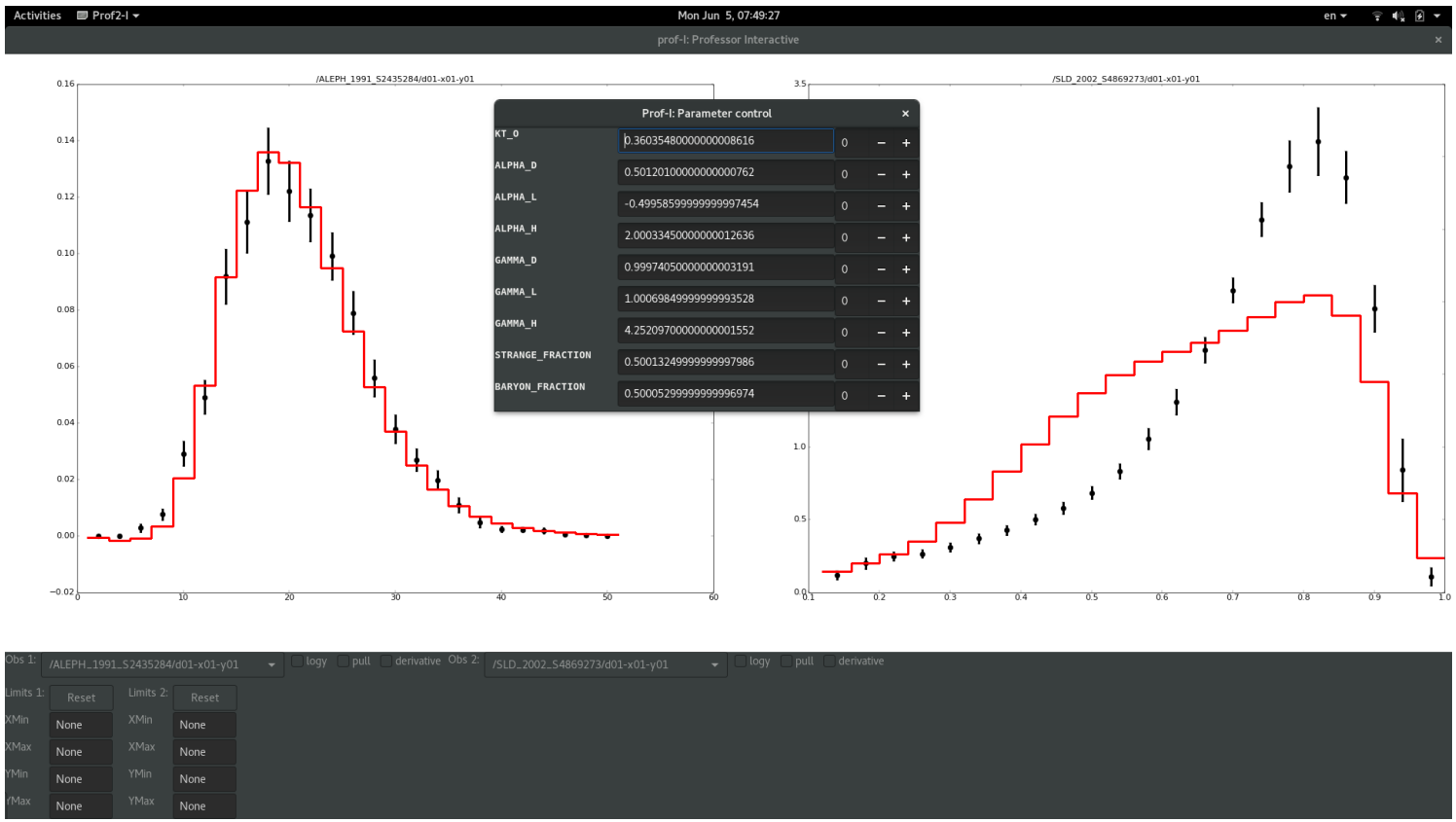
Interactive explorer

- GTK application to interactively “play” with parameterisation
- One slider per parameter, moving them redraws histograms
- Good for intuition building
- Running the event generator would require a few hours wait



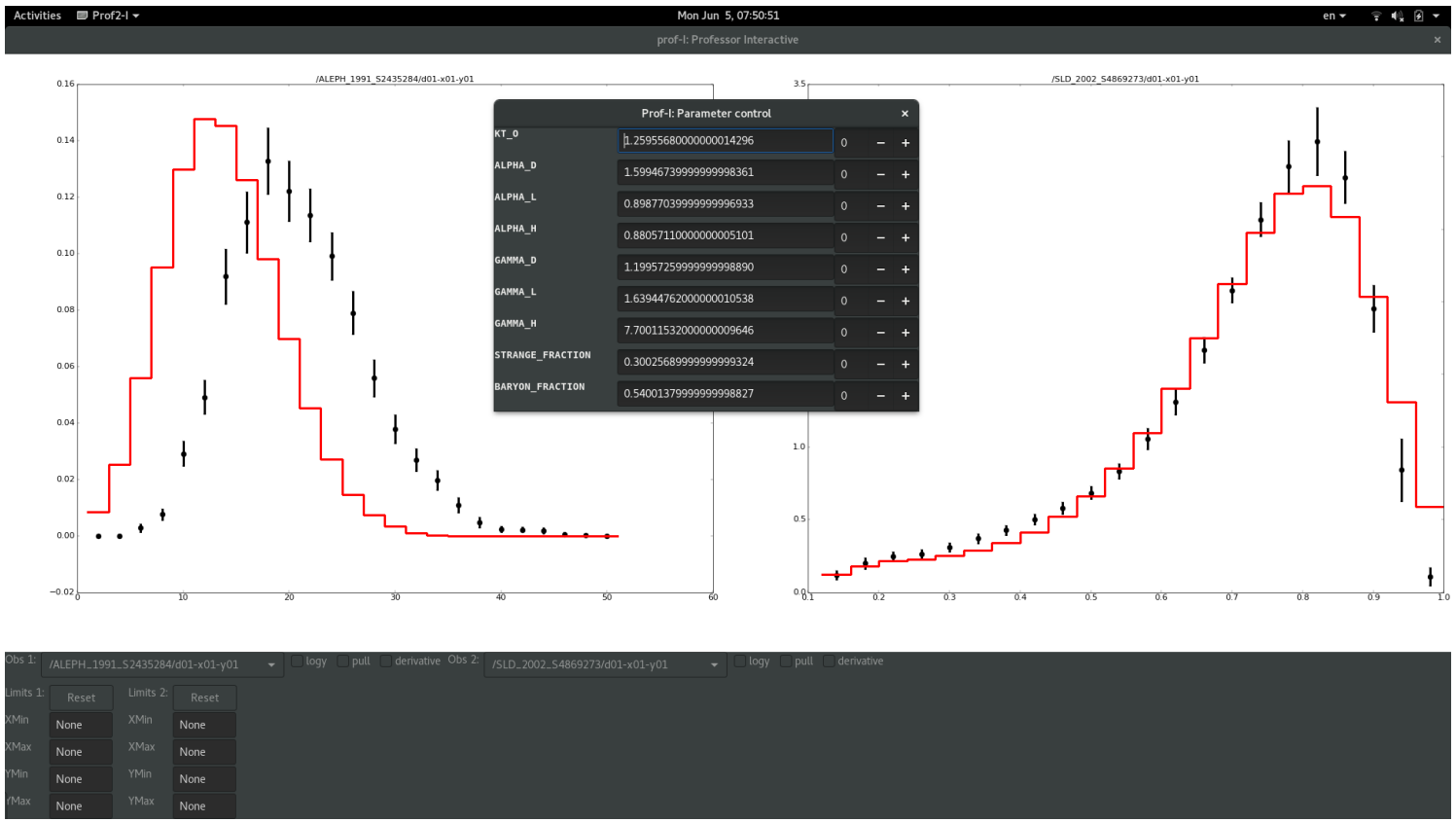
Interactive explorer

- GTK application to interactively “play” with parameterisation
- One slider per parameter, moving them redraws histograms
- Good for intuition building
- Running the event generator would require a few hours wait



Interactive explorer

- GTK application to interactively “play” with parameterisation
- One slider per parameter, moving them redraws histograms
- Good for intuition building
- Running the event generator would require a few hours wait



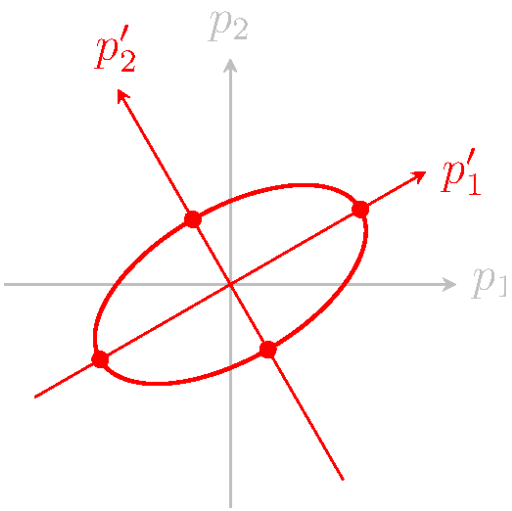
Minimisation

- Tuning is numerical minimisation of goodness-of-fit measure
- We use **`iminuit`** as it is a flexible python wrapper for Minuit
- Input to tuning stage is parameterisation file, text file with weights, directory with data files

```
/ATLAS_2010_S8918562/d03-x01-y01 1            # Set weight to 1   for  each bin of this histo
/ATLAS_2010_S8918562/d05-x01-y01 100          # Set weight to 100 for  each bin of this histo
/ATLAS_2010_S8918562/d07-x01-y01@0:20  10   # Set weight to  10 for  bins with binEDGES in [0,20)
/ATLAS_2010_S8918562/d07-x01-y01@20:40 50   # Set weight to  10 for  bins with binEDGES in [20,50)
/TOTEM_2012_I1115294/d01-x01-y01#0:20  10    # Set weight to  10 for  bins with binINCIDES in [0,20)
/TOTEM_2012_I1115294/d01-x01-y01#20:40 50    # Set weight to  10 for  bins with binINDICES in [20,50)
```

- Output:
  - Text file with minimisation result, covariance matrix etc.
  - File with histograms calculated from parameterisation at this minimum
- Quick turnaround, minimisation seconds to minutes, plots comparing with data seconds
- Usually iterative procedure, look at plots, adjust weights

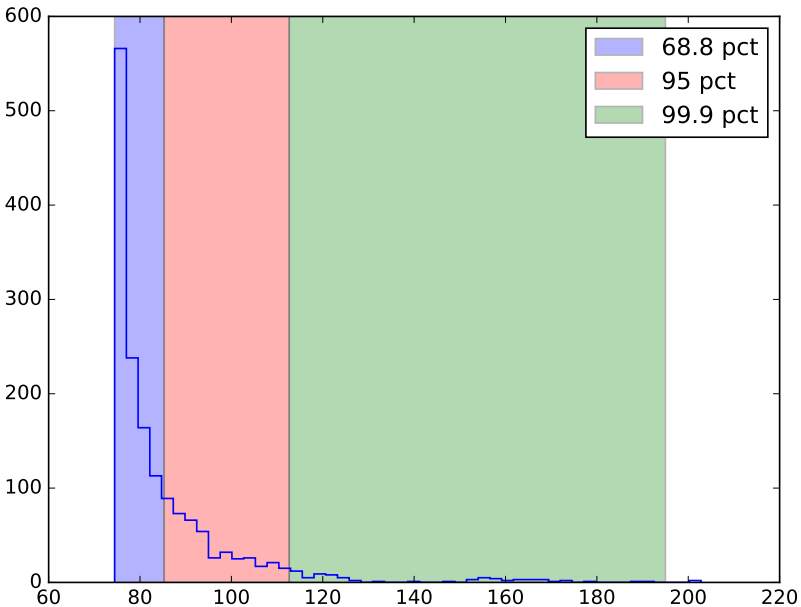
Uncertainties

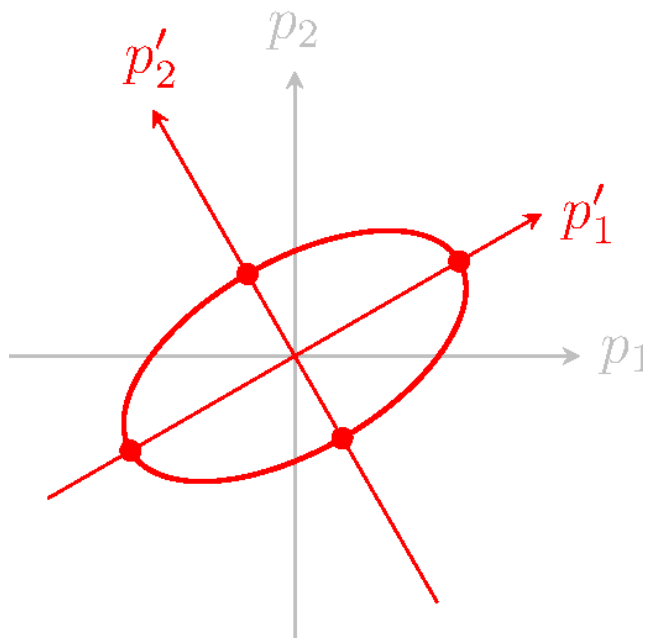


- Can exploit minimiser covariance of parameters, C_{xy}
- Get principal directions, similar to PDF fit uncertainties
- "Eigentunes"
- But: what Δ goodness-of-fit since not really $\chi 2$?

Goodness-of-fit distribution

- Need to obtain distribution of goodness-of fit measure to find e.g. 68% percentile
- We use the parameterisation of the input uncertainties and central values to smear inputs MC($\vec{p}$)
- Say 1000 smeared clones of the input data $\rightarrow$ 1000 slightly different parameterisations
- Do 1000 minimisations using the *same* weights (+ smear data)
- Histogram resulting goodness-of-fit values $\rightarrow \Delta$gof
- Plots: 3 parameters, 20 bins $\rightarrow$ clearly no χ^2! (less than 10 minutes)





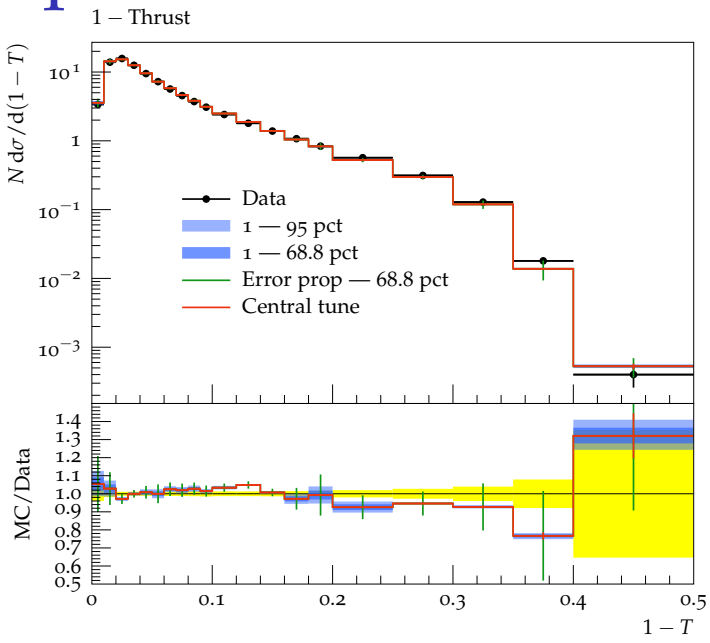
Error propagation

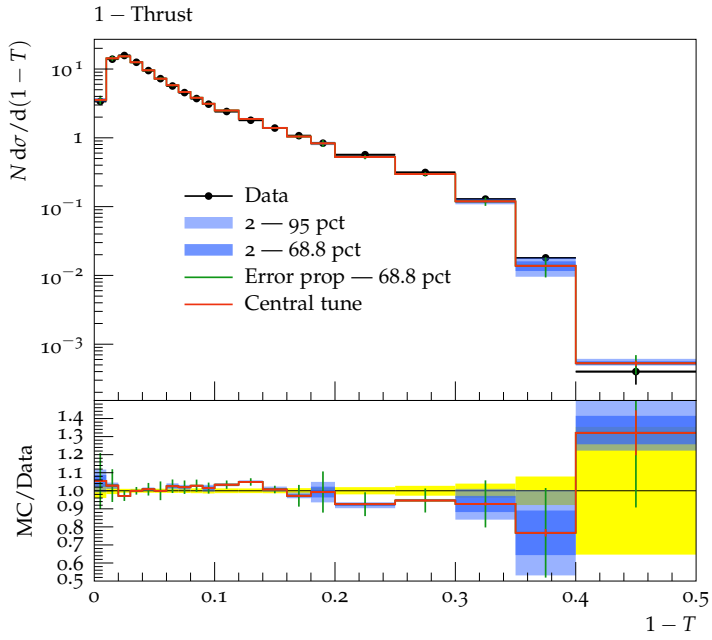
- The Eigentune points allow us to determine a modified covariance matrix $\tilde{C}_{xy}$
- Gradients of polynomials $\vec{\nabla} I(\vec{p})$ always known in Professor
- Can do error propagation of $\tilde{C}_{xy}$:

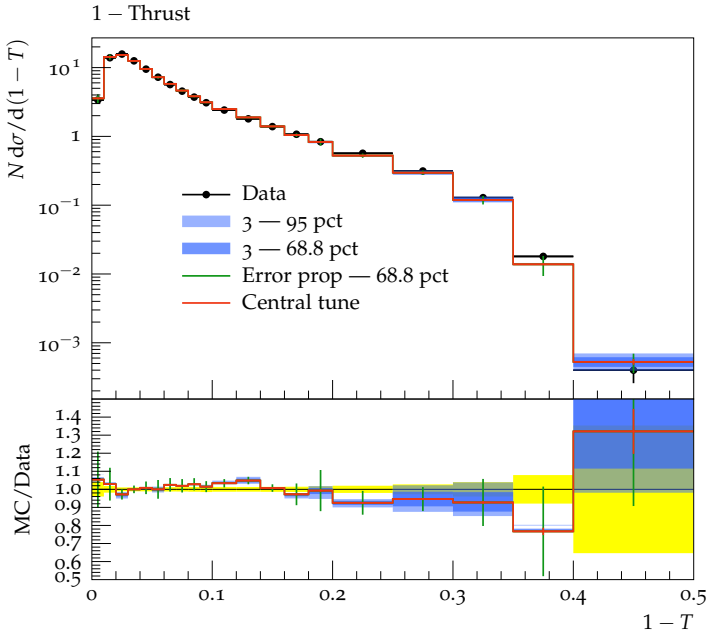
$$C_{ij} = \vec{\nabla} I(\vec{p}_0)\ \tilde{C}_{xy}\ (\vec{\nabla} I(\vec{p}_0))^T$$

- Resulting in uncertainty for bin i as $\sqrt{C_{ii}}$

Example uncertainties







- Eigentunes for all three principal directions
- Contributions differ over observable range
- Propagated uncertainties shown as well

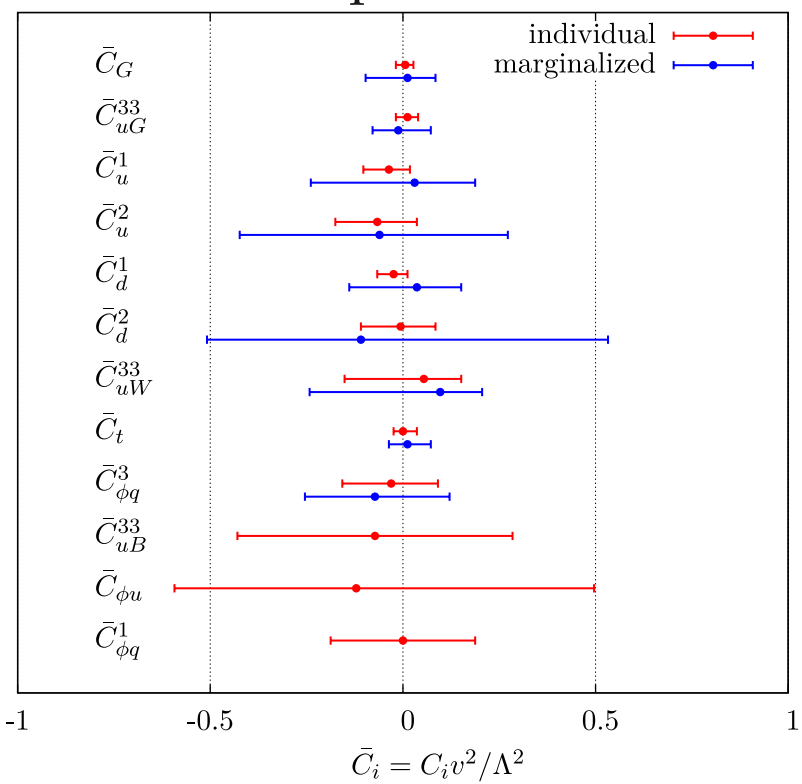
Getting Professor

- Prerequisits: Eigen3 headers, C++ 11 compiler, Python 2.7
- **professor.hepforge.org**
- Docker image: `docker pull iamholger/professor:2.2.1`
- Try out at **http://mybinder.org/repo/iamholger/professor**

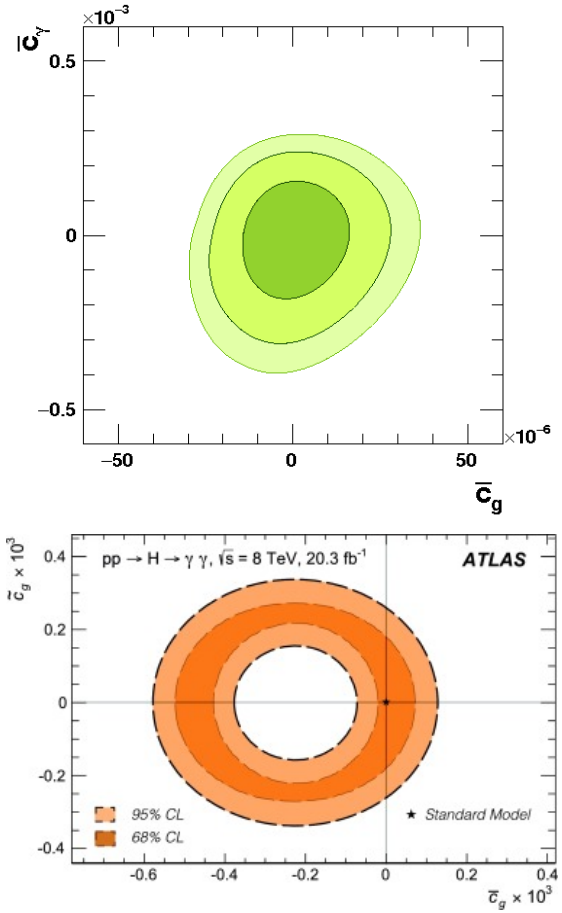
Professor beyond tuning

- Instead of fiddling with say hadronisation model parameters, explore BSM parameter space
- Lots of experience can be transferred from tuning to BSM

TopFitter

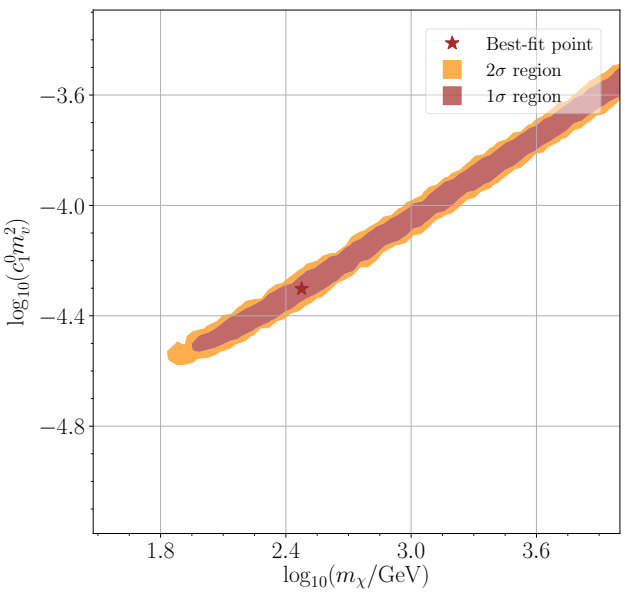


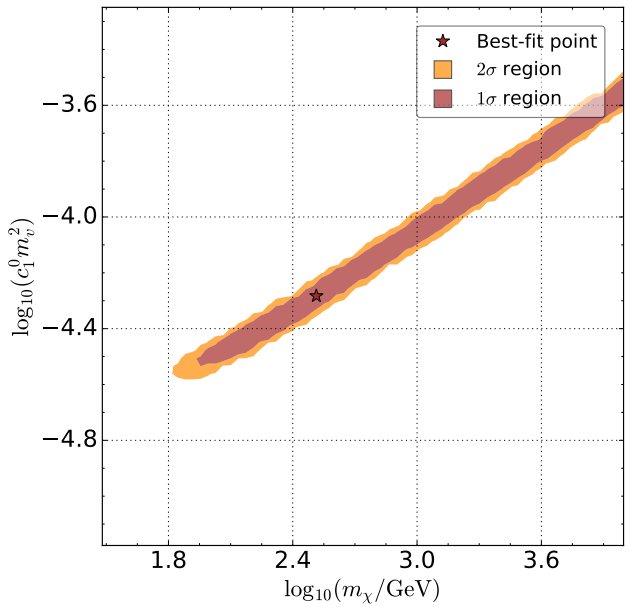
HEFT



Professor beyond collider physics

- Started collaboration with neutrino MC community, **`Genie`**
- Triggered containerisation of Professor with Docker
- Similarly, Dark Matter direct detection codes: Professor in likelihood evaluation (MultiNest)





Next design goals for Professor

- Automatic checks of validity of polynomial approximation.
  - Partitioning of parameter space in case parameter space too big
  - Avoid overfitting
  - Need to allow to drop inputs in case of vanishing cross-sections (avoid discontinuities in polynomial fit)
  - Move away from histogram picture in storage
- Usage of other parameterisations, e.g. Gaussian Processes for $\frac{1}{x}$, *Pade'* approximation
- Finalise work on uncertainties, usage of `pymultinest`

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: impact beyond "get a paper out"
- Also a very useful cross-check: quite a few ATLAS analysis bugs have been found via Rivet!
- Integrated with ATLAS and CMS software
- Now supports detector simulation for BSM search preservation

- Professor:
  - Parametrisation of computationally expensive functions
  - Inputs can always be parallelised in a trivial way
  - Seamless integration into numerical tools `iminuit`, `pymultinest` through python bindings $\rightarrow$ tuning and BSM applications
  - Robust estimation and propagation of tuning uncertainties available very soon
  - Immediately available through `docker`