Professor

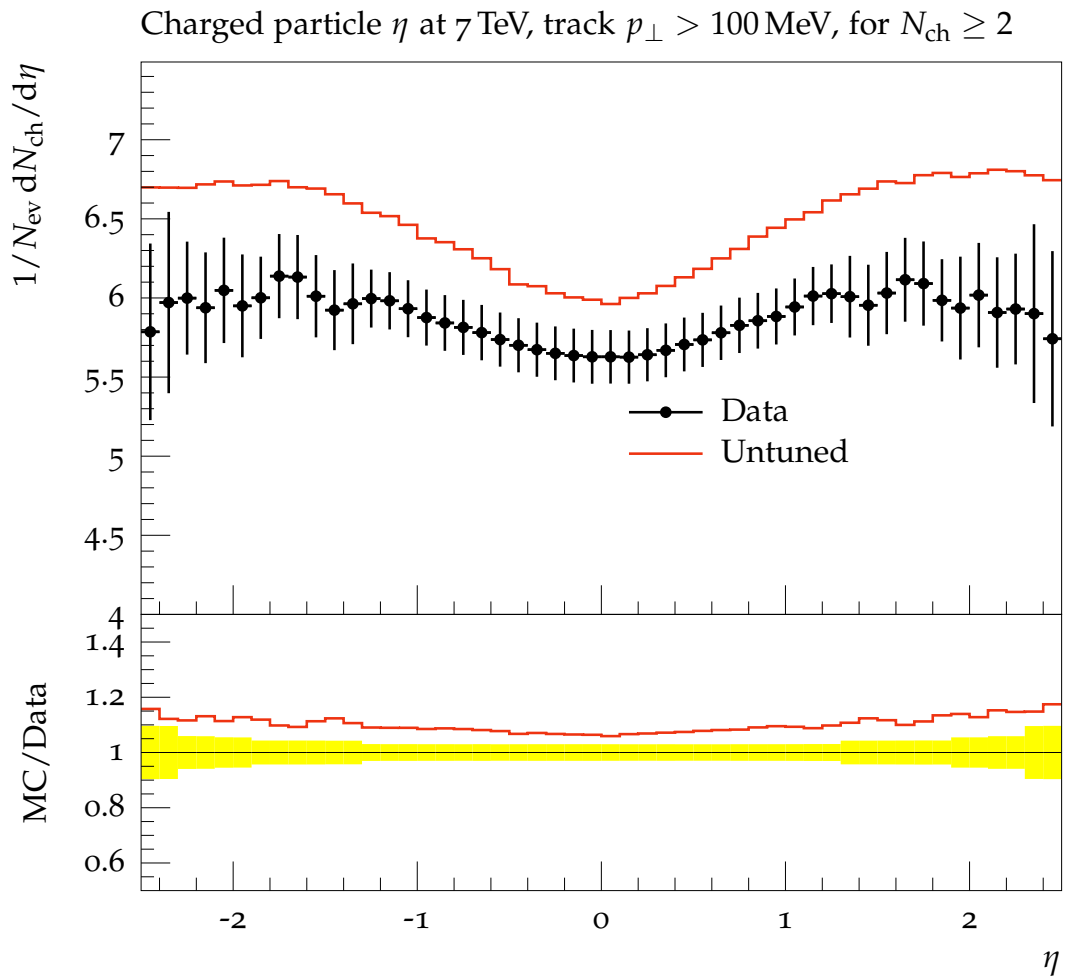
Holger Schulz (IPPP Durham)

October 21, 2016

Berlin

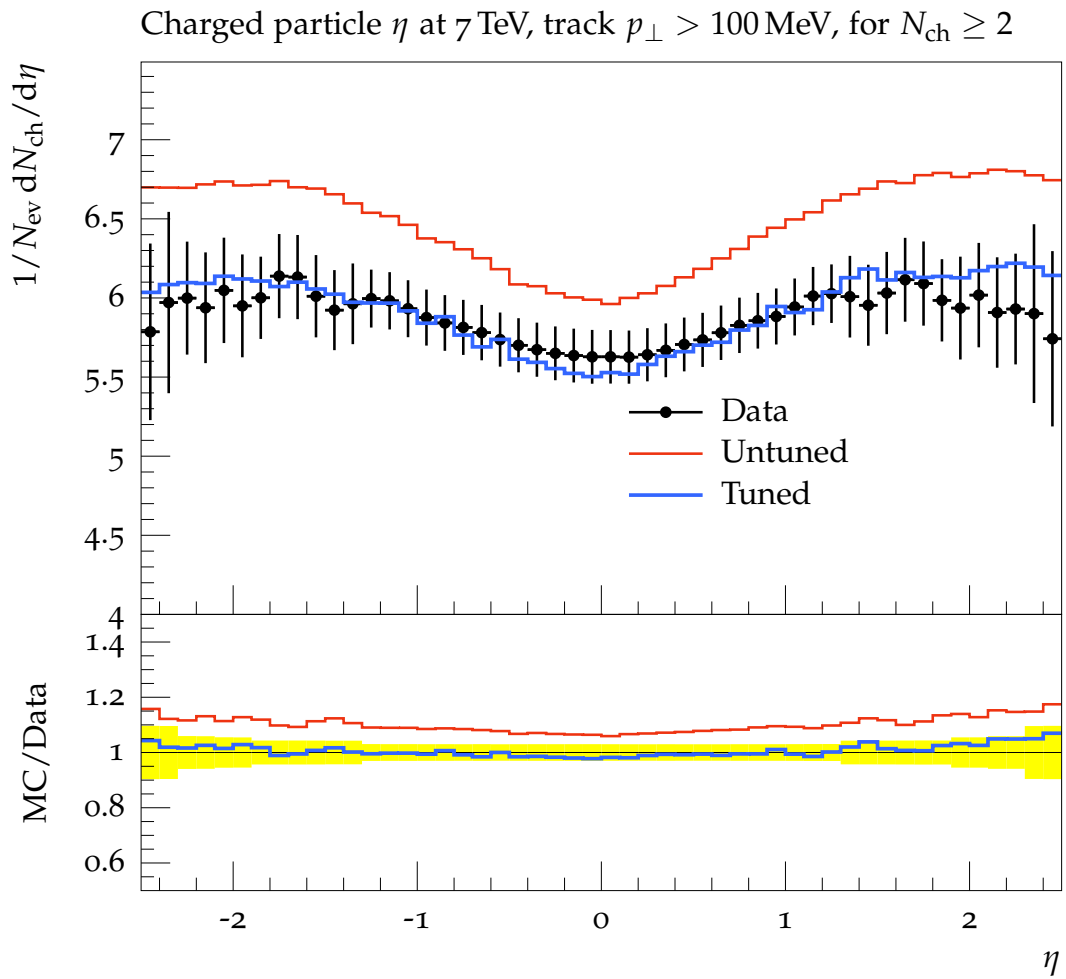
Tuning

Comparison of SGH predictions might be incompatible with data



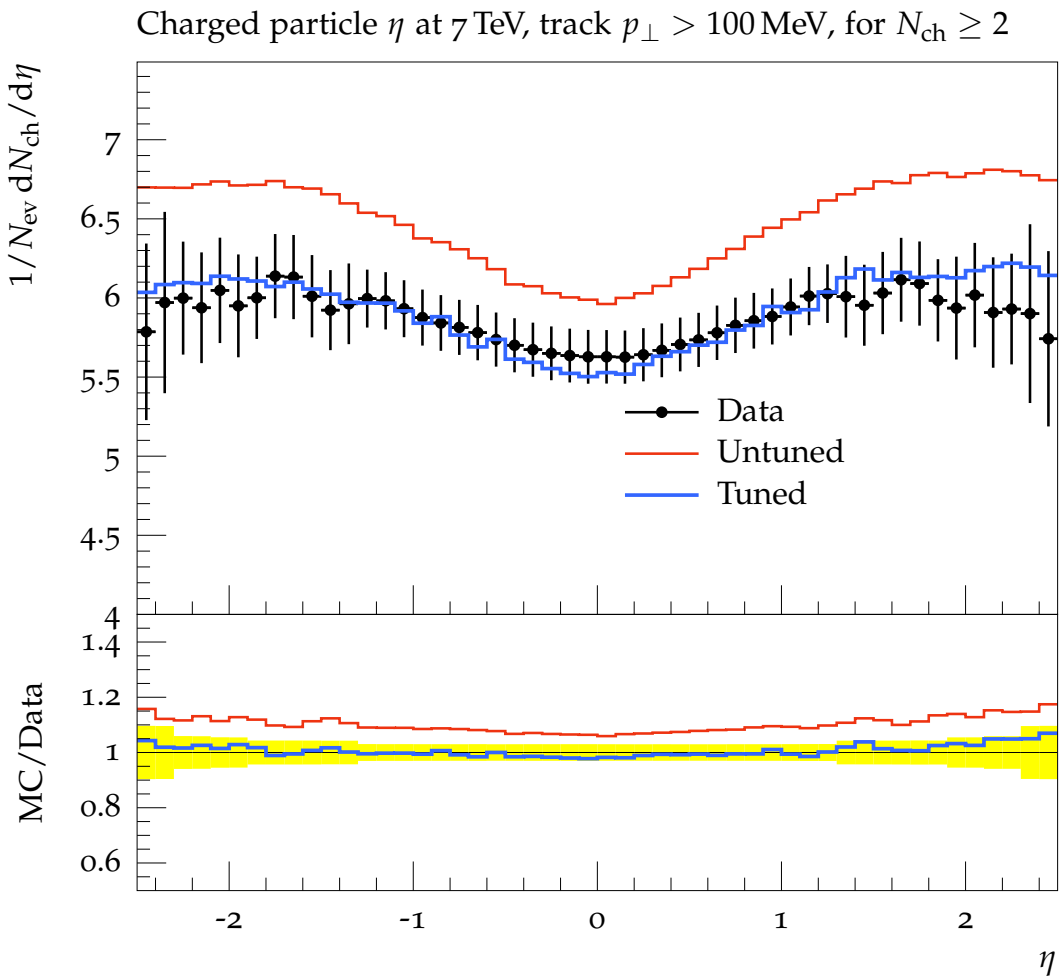
Tuning

Comparison of SGH predictions might be incompatible with data



Tuning

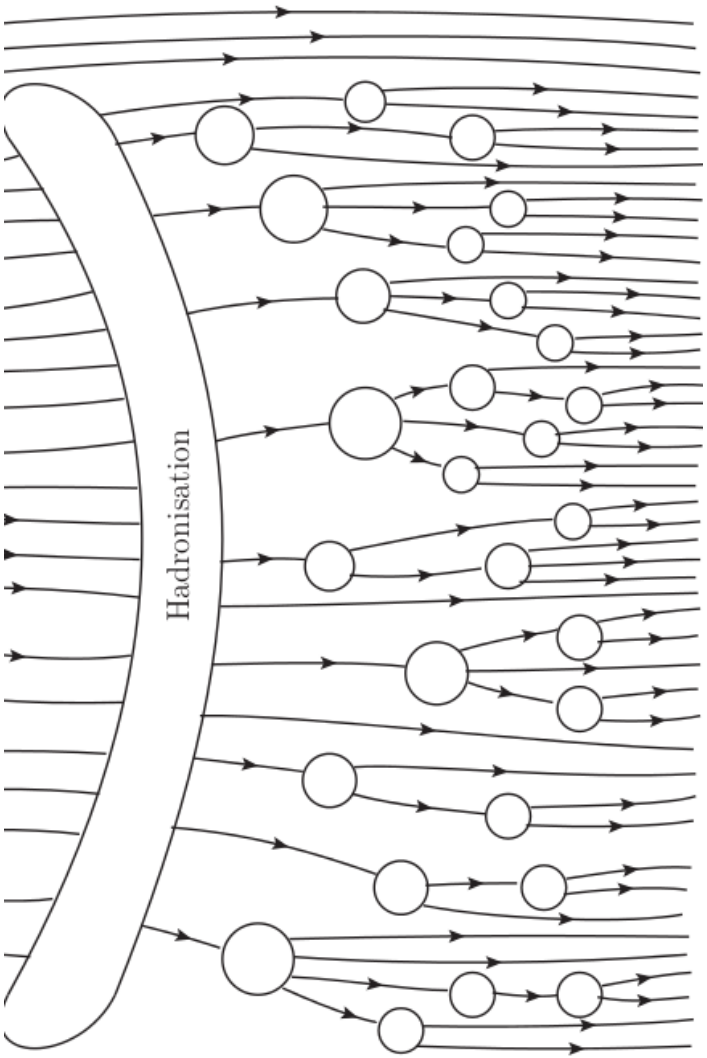
Comparison of SGH predictions might be incompatible with data



In the following: how to compare MC with data, how to get from red to blue

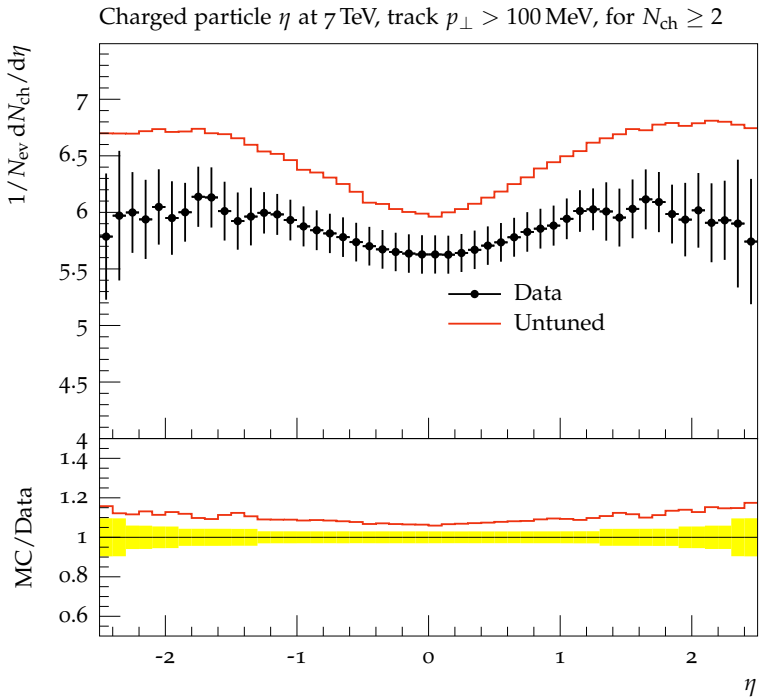
Rivet

- 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)
- Strong ties to HepData
- Implementation and validation of new (data) analyses provided by experiment
- Easy to write new analyses for signal and background estimates from MC



Tuning

- Set of tuneable parameters $\vec{p}$
- For bins b have data D_b and $MC_b(\vec{p})$
- Looks like a χ^2 minimisation problem:
  $\chi^2(\vec{p}) = \sum_b (D_b - MC_b(\vec{p}))^2$
- However, $MC_b(\vec{p})$ is an expensive function
- Professor: replace $MC_b(\vec{p})$ with fast approximation $I_b(\vec{p})$

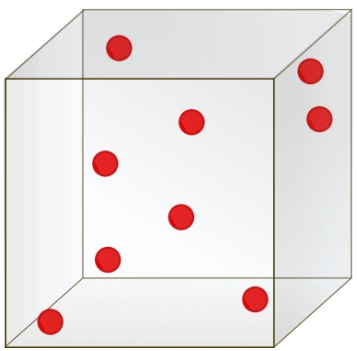


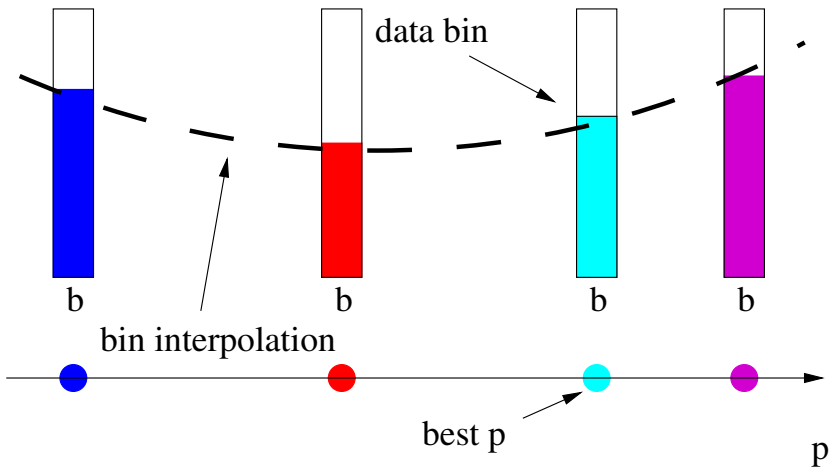
Tuning

- Set of tuneable parameters $\vec{p}$
- For bins b have data D_b and $MC_b(\vec{p})$
- Looks like a χ^2 minimisation problem:
  $\chi^2(\vec{p}) = \sum_b (D_b - MC_b(\vec{p}))^2$
- However, $MC_b(\vec{p})$ is an expensive function
- Professor: replace $MC_b(\vec{p})$ with fast approximation $I_b(\vec{p})$

Tuning with Professor

- Random sampling: N parameter points in n-dimensional space
- Run generator and fill histograms (e.g. Rivet)
- For each bin:
  - Don't care about actual dependence on parameters
  - Polynomial approximation
- 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`





Professor 2

- `http://professor.hepforge.org`, release 2.1.3
- Complete rewrite
- Parametrisation now in C++ (Eigen)
  - Usage in other codes (`arXiv:1511.05170[hep-ph]`, `arXiv:1506.08845[hep-ph]`)
- Python bindings (through cython) for flexibility:

```
import professor2 as prof
# X ... parameter points, e.g. 3-dimensional
# Y ... corrsponding values
I=prof2.Ipol(X,Y, order=5)
print I.val([0, -.5, 13])
```

- HepMC to Rivet to YODA to Professor tool chain of course still supported with set of scripts
- Much improved command line
- Parametrisations stored in text files
- Added derivatives

Professor approach

- Replace exakt $f(\vec{p})$ by **analytic** approximation $I(\vec{p})$
- Thus replace CPU time for evaluation from hours . . . days to milliseconds

Professor approach

- Replace exakt $f(\vec{p})$ by **analytic** approximation $I(\vec{p})$
- Thus replace CPU time for evaluation from hours . . . days to milliseconds

Basic work cycle

1. Define and sample M-times from d-dimensional parameter space $\mathcal{P}$
2. For each of the M points $\vec{p}_i$: evaluate exact $f(\vec{p}_i)$

   N.b. this step is trivially parallelisable
3. Fit **polynomial** $I(\vec{p})$ through
   $[(\vec{p}_1, f(\vec{p}_1)), (\vec{p}_2, f(\vec{p}_2)), \cdots, (\vec{p}_M, f(\vec{p}_M))\ ]$

   e.g. $$I(p_1, p_2) = \alpha_0 + \beta_1 p_1 + \beta_2 p_2 + \gamma_{11} p_1^2 + \gamma_{12} p_1 \cdot p_2 + \gamma_{22} p_2^2$$

   Store coefficients in text file

Parametrisation validation

- E.g. histogram H with N bins:
  - $I(\vec{p}) \rightarrow \{I_b(\vec{p})\}_{b=1\ldots N}$

Parametrisation validation

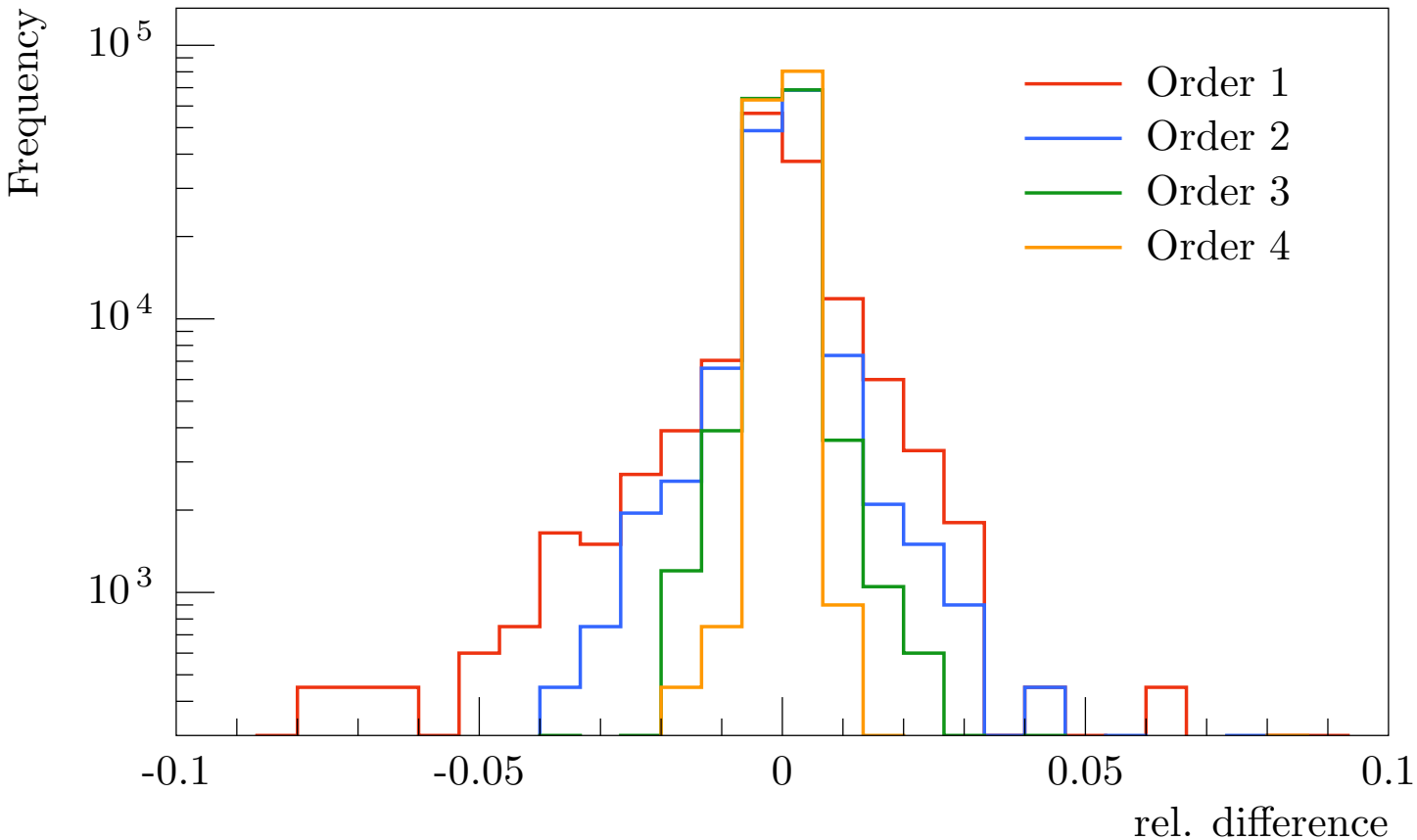
- E.g. histogram H with N bins:
  - $I(\vec{p}) \rightarrow \{I_b(\vec{p})\}_{b=1\ldots N}$
  - Thus $\vec{H}(\vec{p}) = (\ I_1(\vec{p}), I_2(\vec{p}), \ldots, I_N(\vec{p})\ )^T$

Parametrisation validation

- E.g. histogram H with N bins:
  - $I(\vec{p}) \rightarrow \{I_b(\vec{p})\}_{b=1\ldots N}$
  - Thus $\vec{H}(\vec{p}) = (\ I_1(\vec{p}), I_2(\vec{p}), \ldots, I_N(\vec{p})\ )^T$
  - Calculate difference of $f_b(\vec{p_i})$ and $I_b(\vec{p_i})$

Parametrisation validation

- E.g. histogram H with N bins:
  - $I(\vec{p}) \rightarrow \{I_b(\vec{p})\}_{b=1\ldots N}$
  - Thus $\vec{H}(\vec{p}) = (\ I_1(\vec{p}), I_2(\vec{p}), \ldots, I_N(\vec{p})\ )^T$
  - Calculate difference of $f_b(\vec{p_i})$ and $I_b(\vec{p_i})$



Backup

Advantages

- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications

Advantages

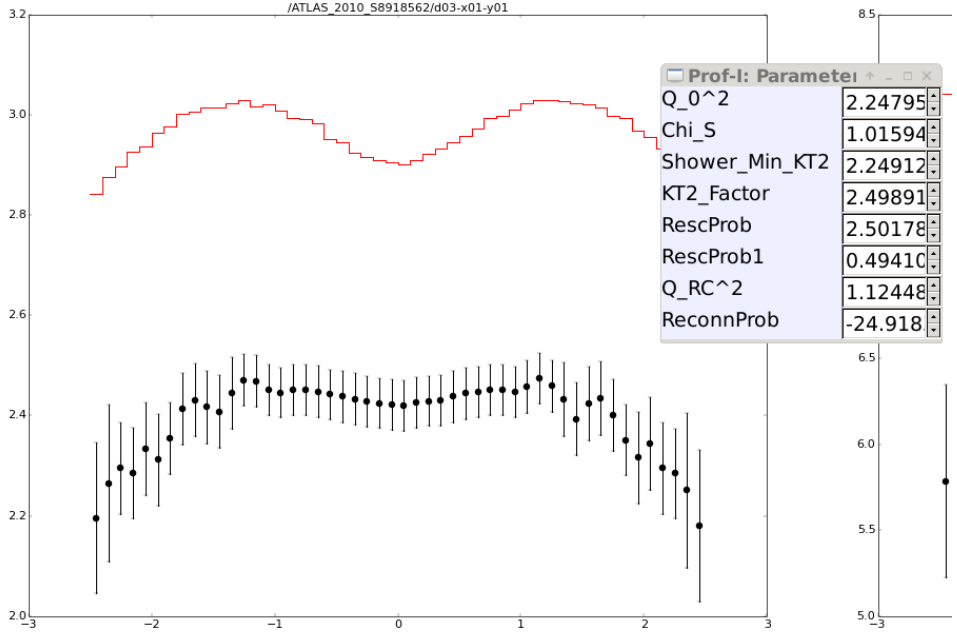
- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications
- Fitting against data cheap, can bias minimisation e.g. if $f(\vec{p})$ known to be imperfect (AMBT2 and AUET2)

$$\chi^2(\vec{p}) = \sum_{b}^{\text{Nbins}} w_b \cdot \left(\frac{I_b(\vec{p}) - D_b}{\Delta(\vec{p})} \right)^2$$

Advantages

- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications

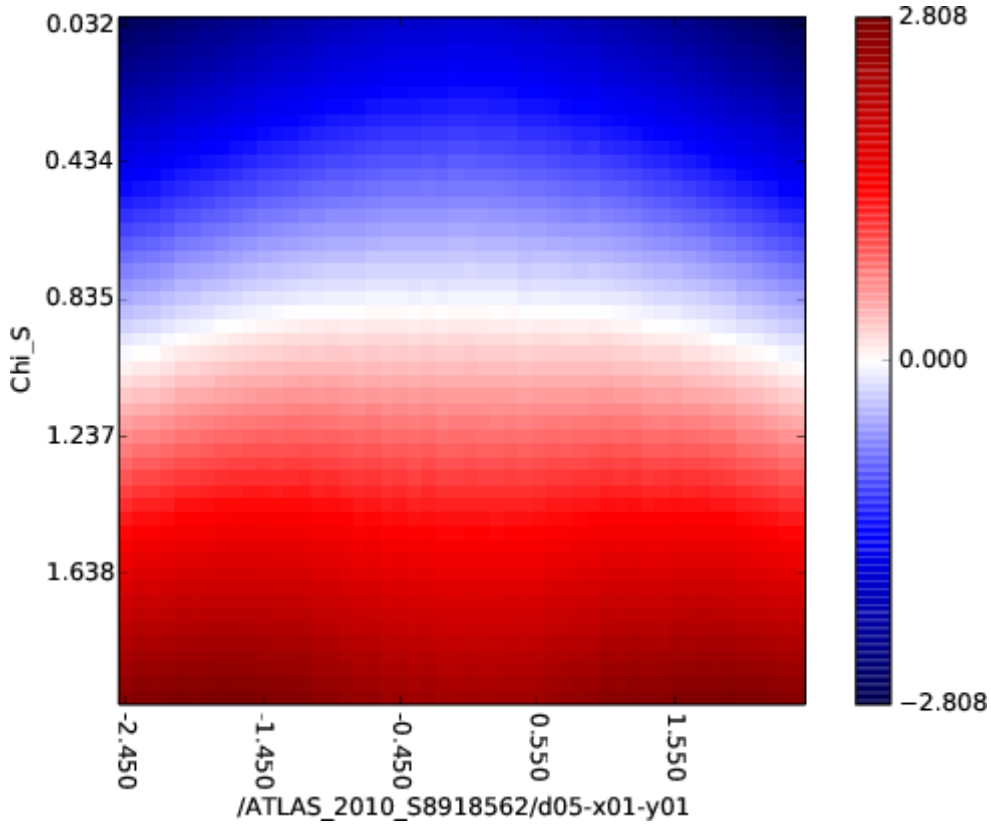
- Interactive parametrisation explorer



Advantages

- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications

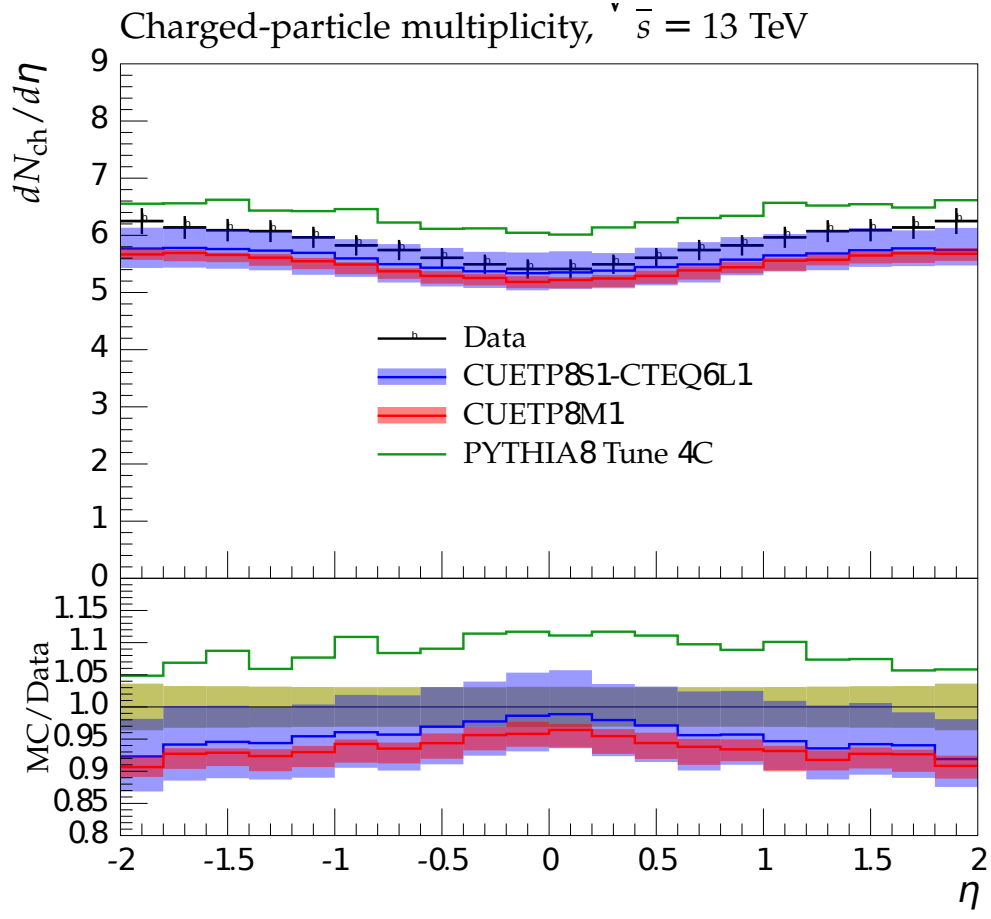
- Sensitivity and correlation analysis cheap $\rightarrow$ find parameters that do nothing $\rightarrow$ reduce dimensionality



Advantages

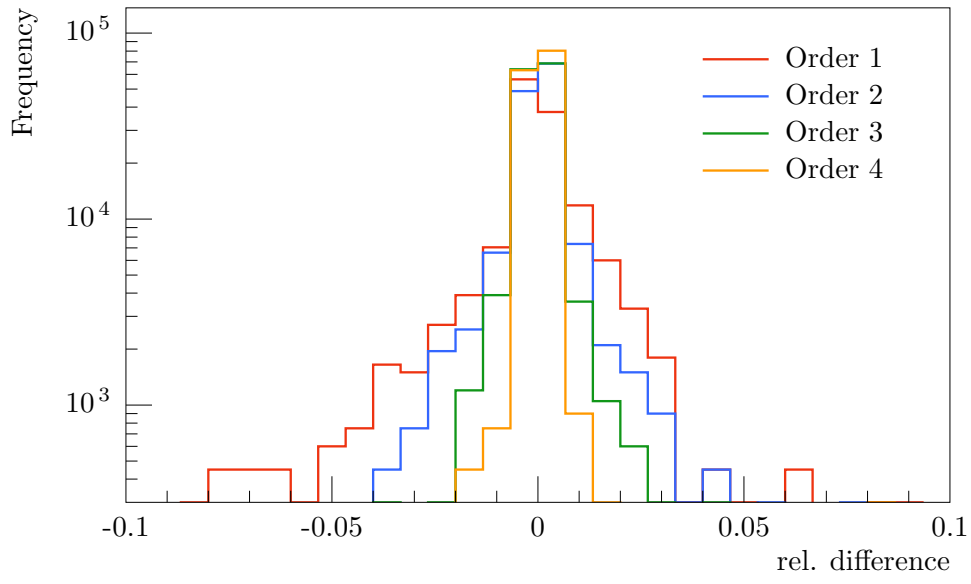
- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications

- Can exploit χ^2 valley to get error-tunes `http://inspirehep.net/record/1407839`



Advantages

- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications
- Validation of parametrisation allows to catch errors early on
- Improve quality by
  - Throwing and exact evaluation for more points
  - Using higher order polynomials



Technicalities

- Core functionality (parametrisation) written in `C++`
- Allows for usage in programs such as `GFitter`
- Arbitrary polynomial order and first derivative automatically
- Dependency: Eigen3 ($\geq v2.6$)
- Platform independent storage of parametrisation (ASCII)
- Tuning system: set of factorised python scripts (via cython)
- Minimisation done using `iminuit` `https://github.com/iminuit/`
- ROOT support via YODA

`PROFESSOR.HEPFORGE.ORG`

- Current version: Professor 2.1.3
- Bootstrap script
- Exhaustive documentation with videos