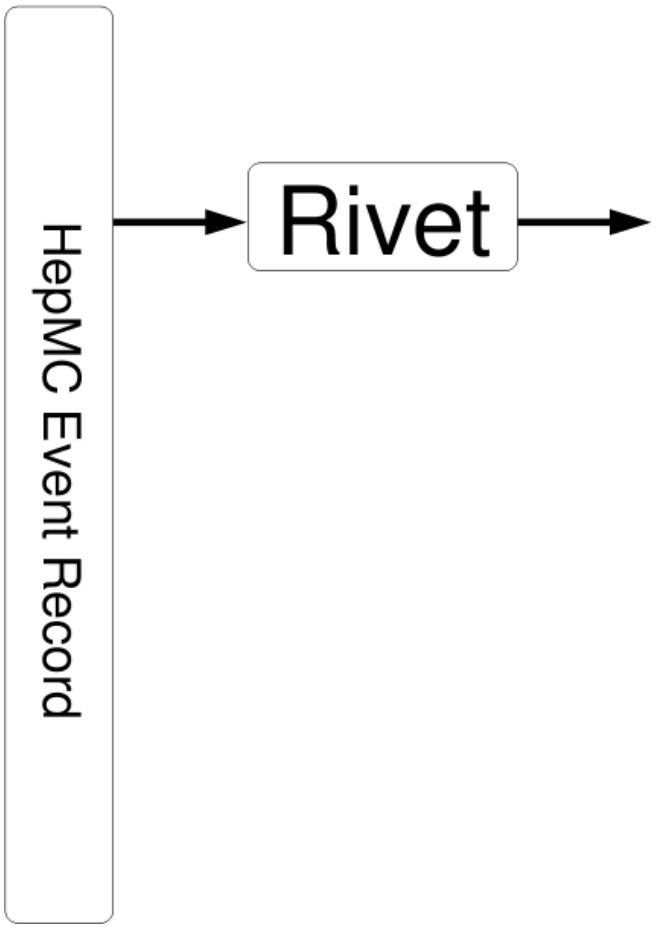
New developments in event generator tuning techniques

Holger Schulz, Heiko Lacker, Jan Eike von Seggern (HU Berlin),
Andy Buckley (Edinburgh), Hendrik Hoeth (Durham)

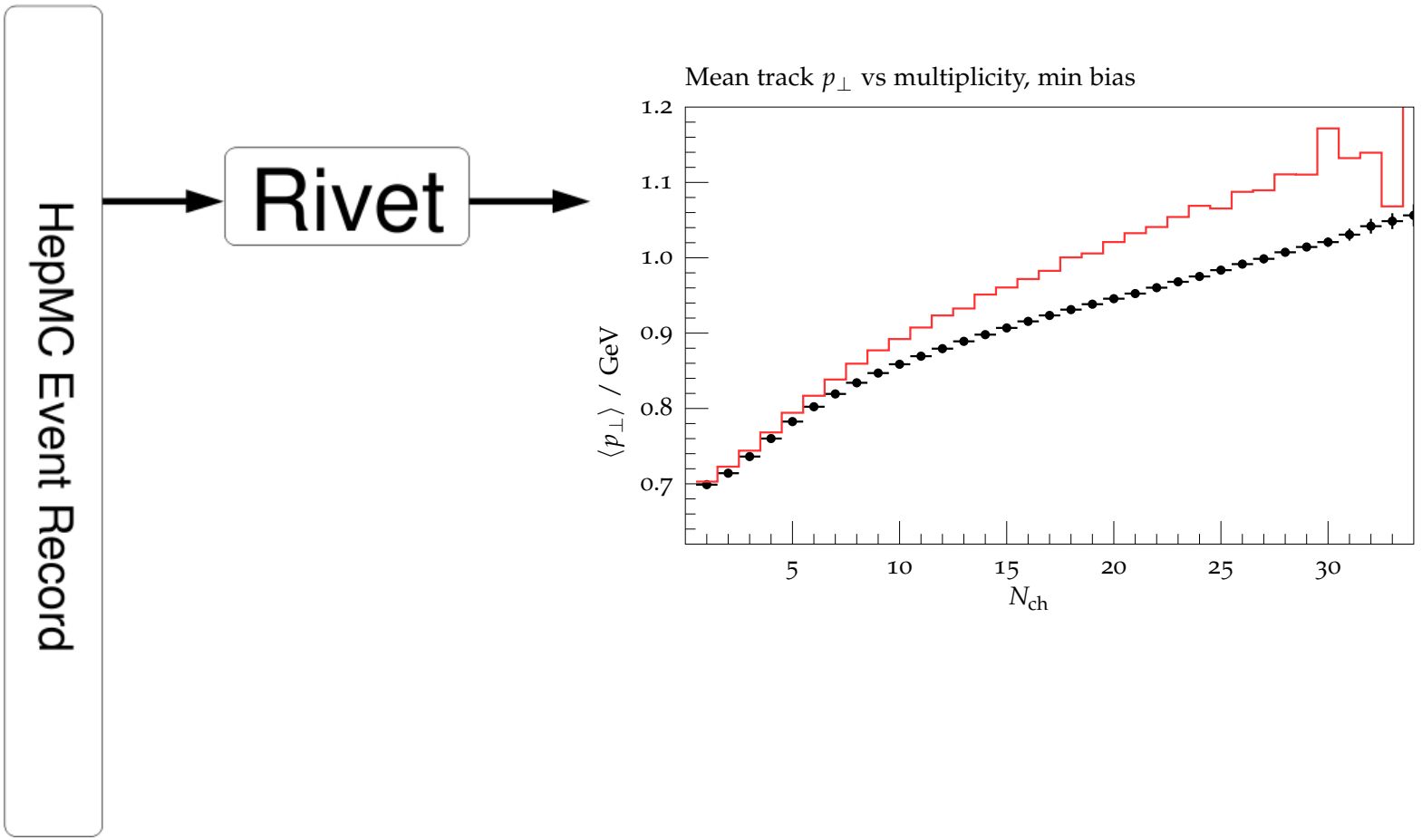
ACAT Jaipur, 22-27 February 2010

Introduction



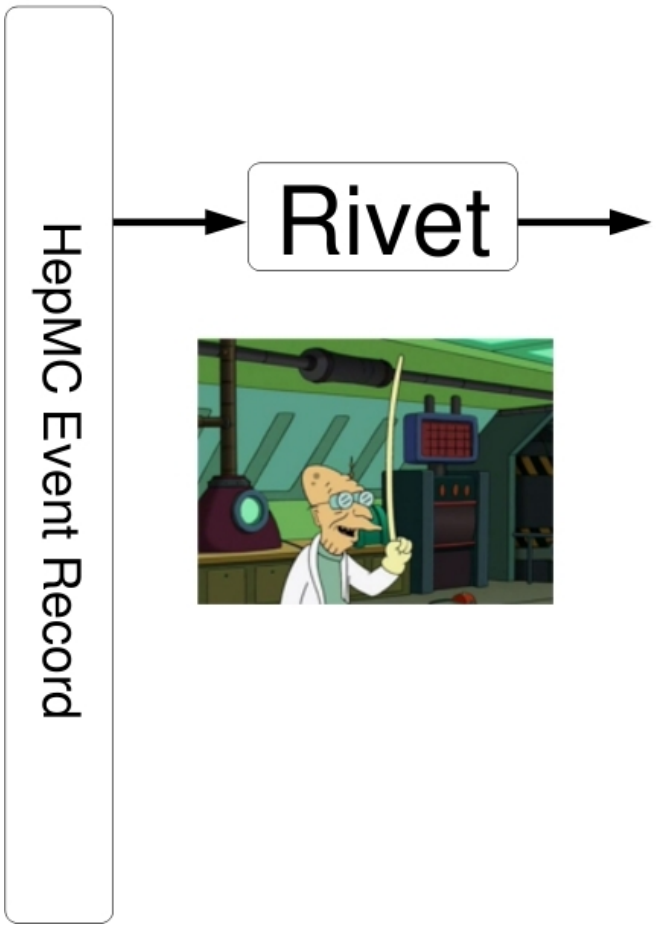
Rivet uses generator independent HepMC events as input

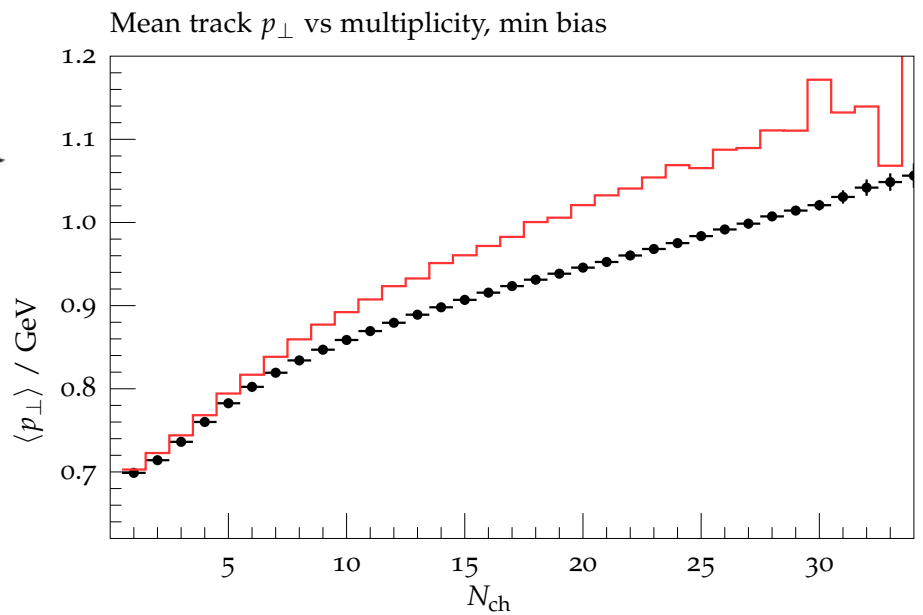
Introduction



The output can be compared with data, but may be untuned

Introduction



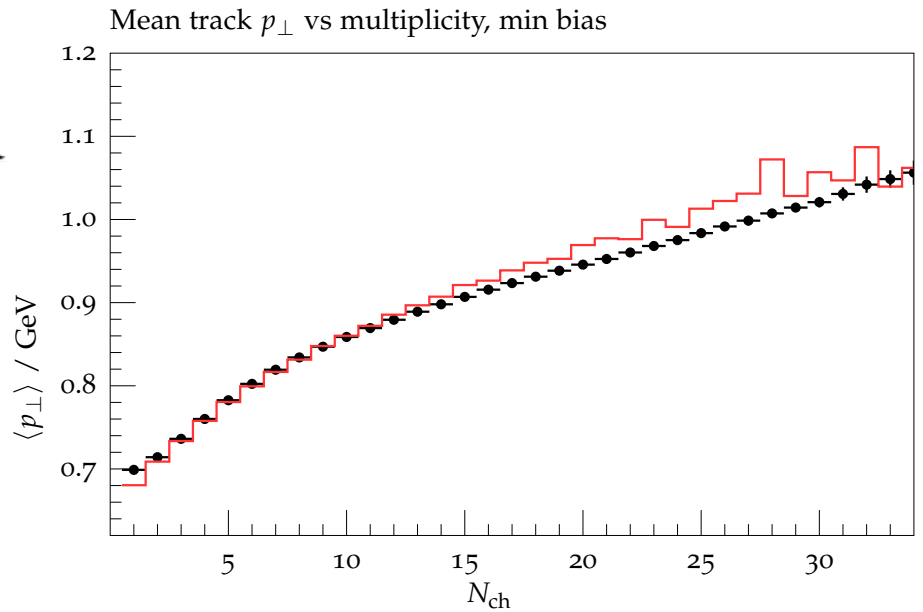


Luckily, there is Professor

Introduction

HepMC Event Record

Rivet



Professor tunes generators based on Rivet output

Rivet

- Version 1.2.0 to be released this month
- More than 60 validated analyses from experiments like LEP, TEVATRON, RHIC, SPS, HERA are now implemented
- Rivet is used by ATLAS, CMS, LHCb
- Output histograms are used by Professor for generator tuning
- HepData (`http://durpdg.dur.ac.uk`) now offers AIDA data export

Tools for systematic generator tuning

DELPHI 1995, Hamacher et al.: bin-wise interpolation of MC generator response and χ^2 minimization

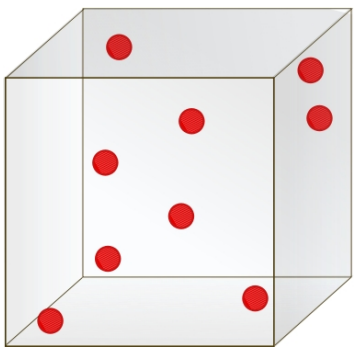
Professor 2008 (arXiv:0907.2973, arXiv:0906.0075, arXiv:0902.4403):

"PROcedure For ESTimating Systematic errORrs"

- Pick up DELPHI idea
- Use flexible `python` interface
- Use quadratic or cubic interpolations
- Interpolation storing with python.cPickle
- Validation of results possible in many ways
- NEW: GUI tool, uncertainty estimates

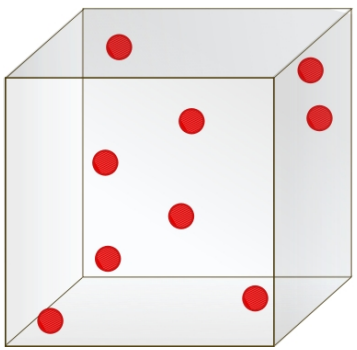
Tuning procedure in Professor (1D, 1Bin)

1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`



Tuning procedure in Professor (1D, 1Bin)

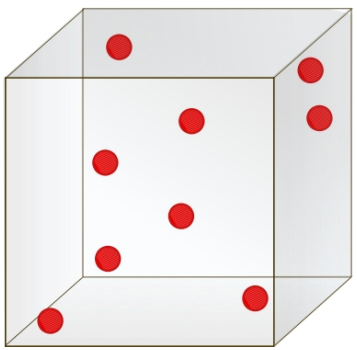
1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`



p

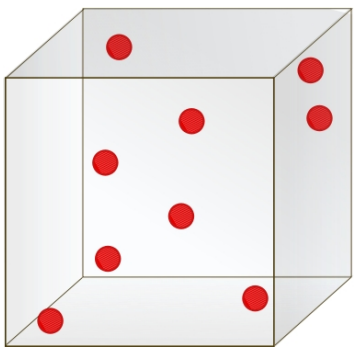
Tuning procedure in Professor (1D, 1Bin)

1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`



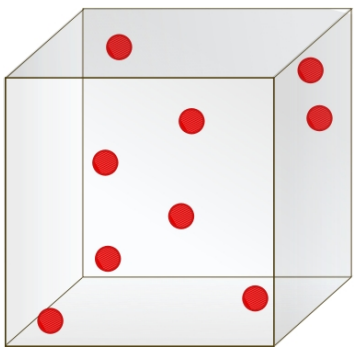
Tuning procedure in Professor (1D, 1Bin)

1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`



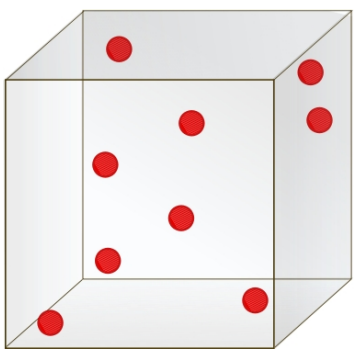
Tuning procedure in Professor (1D, 1Bin)

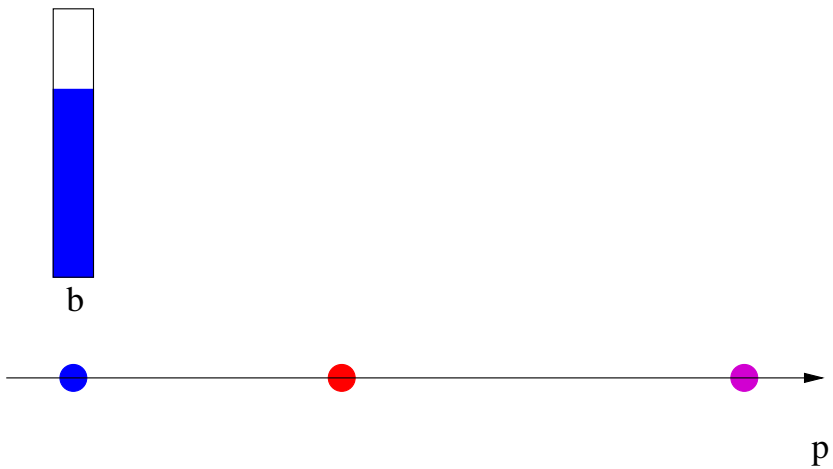
1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`



Tuning procedure in Professor (1D, 1Bin)

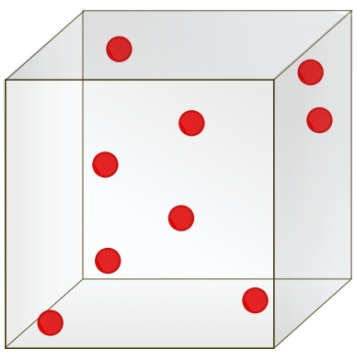
1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`

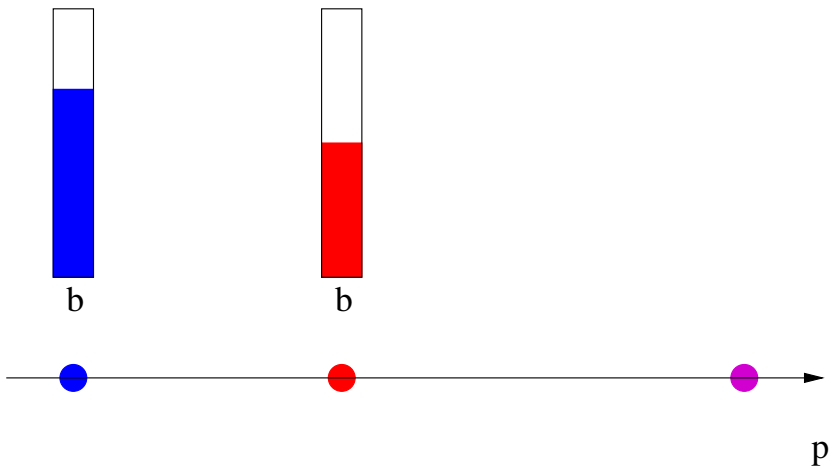




Tuning procedure in Professor (1D, 1Bin)

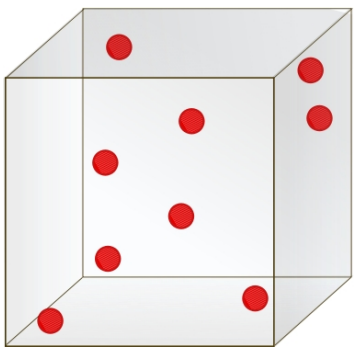
1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`

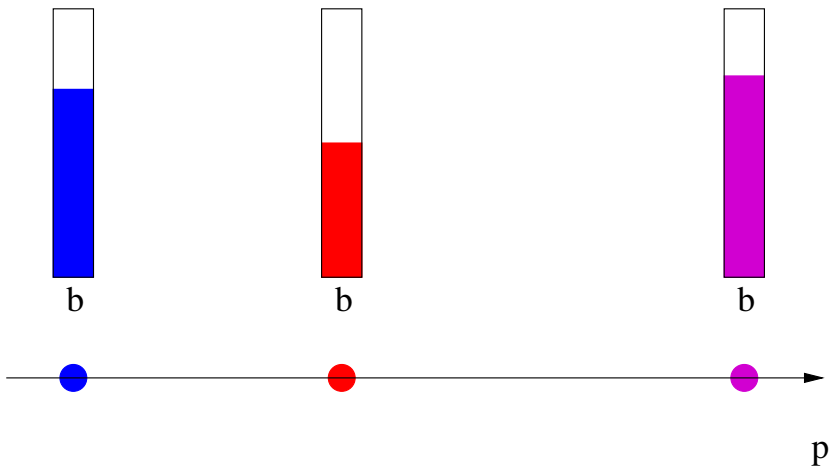




Tuning procedure in Professor (1D, 1Bin)

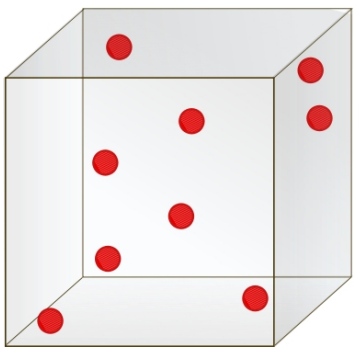
1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`

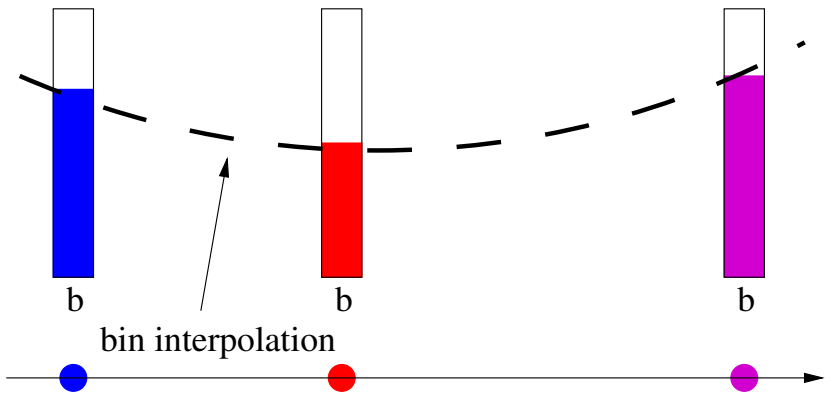




Tuning procedure in Professor (1D, 1Bin)

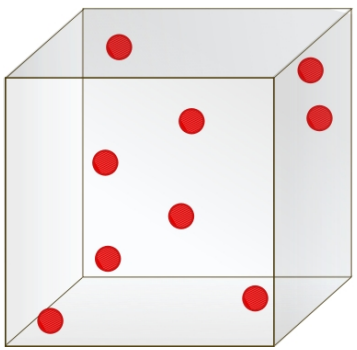
1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`

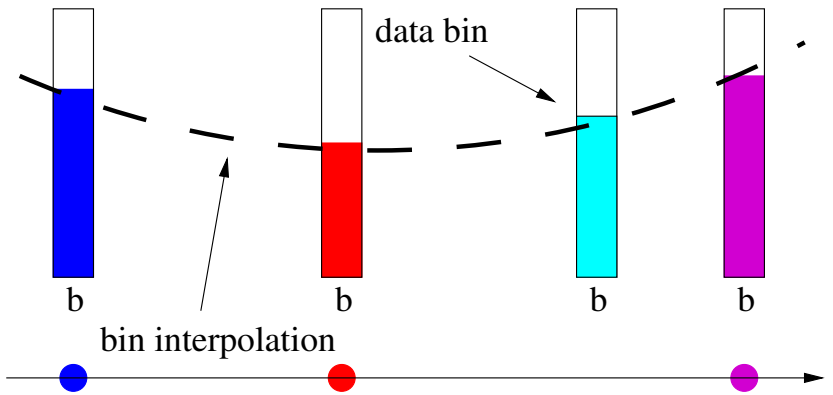




Tuning procedure in Professor (1D, 1Bin)

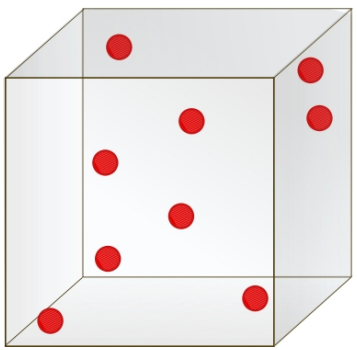
1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`

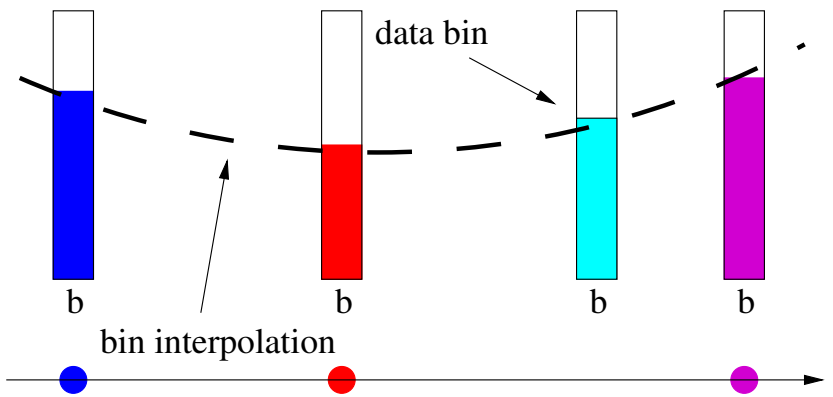




Tuning procedure in Professor (1D, 1Bin)

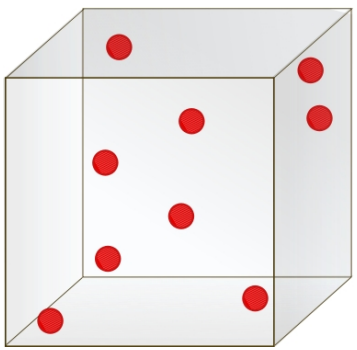
1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit`, `SciPy`

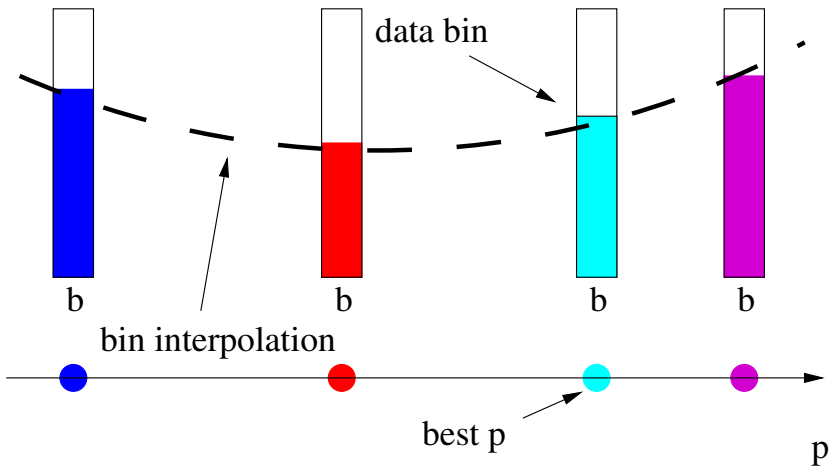




Tuning procedure in Professor (1D, 1Bin)

1. Random sampling: N parameter points in n-dimensional space
2. Run generator and fill histograms
3. For each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. Construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. Numerically minimize using `pyMinuit, SciPy`





Oversampling and interpolation quality

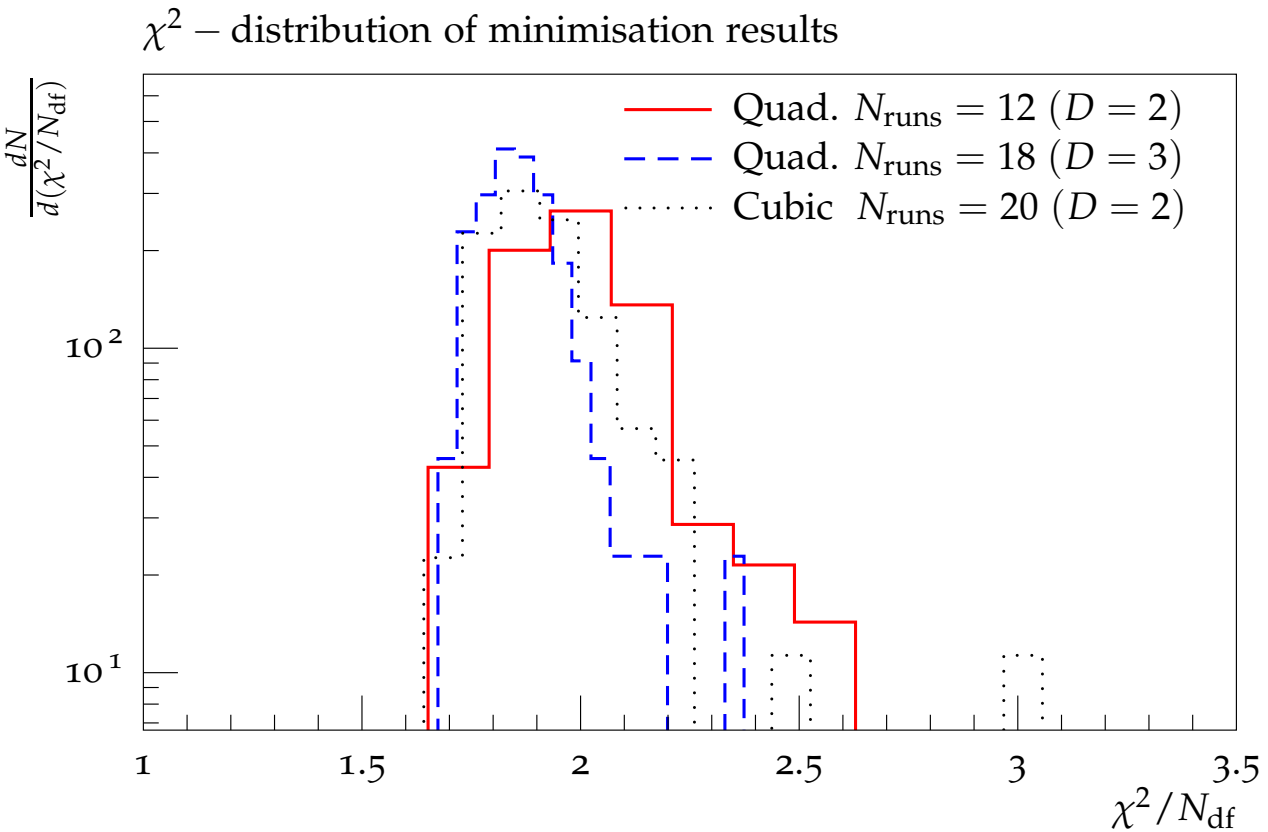
- In n dimensions: Singular Value Decomposition (SVD) requires $N_{\text{min}}^{(n)}$ generator runs:

$$N_{\text{min}}^{(n)} = 1 + n + n(n+1)/2 + \underbrace{(n+1)(n+2)/6}_{\text{cubic only}}$$

- SVD allows for oversampling.
- Degree of oversampling: $D = N_{\text{runs}} / N_{\text{min}}^{(n)}$
- What is a sensible D?
- $\rightarrow$ use $\mathcal{O}(1000)$ different interpolations with different N_{runs}
- Perform minimisations, investigate g.o.f. measures
- Examples shown are from a two-dimensional Tuning of Jimmy

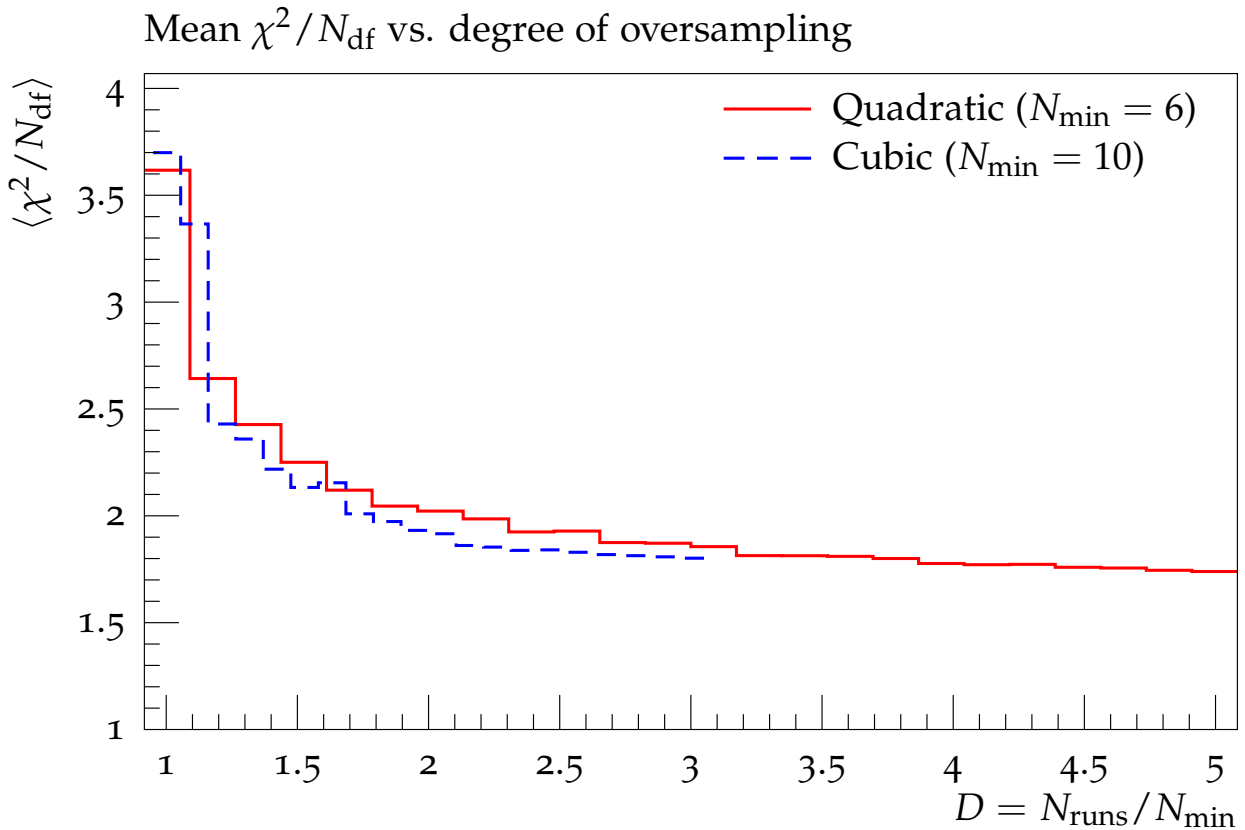
Distribution of Goodness of Fit values

- Spread of results decreases with increasing D, polynomial degree
- Observe lower χ^2 / N_{df}-boundary



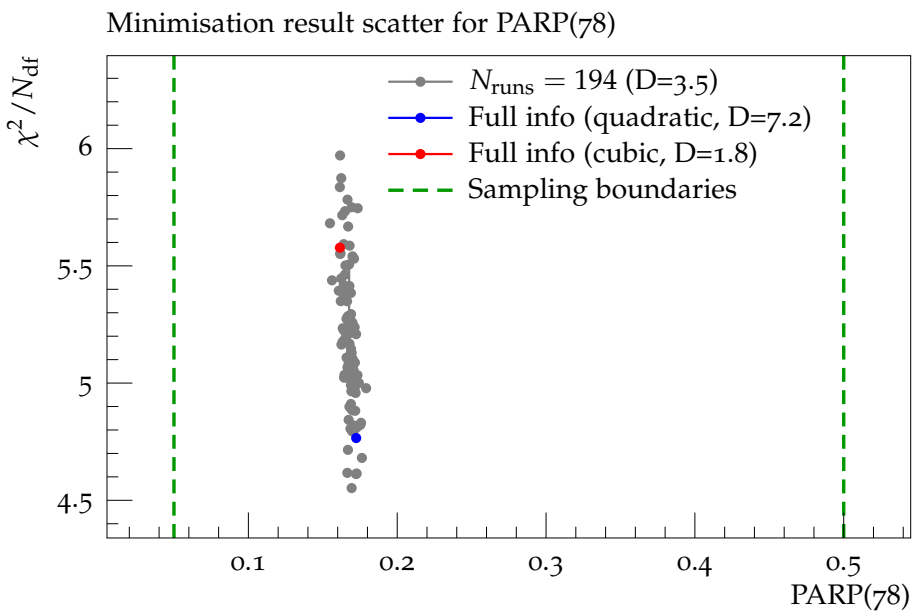
Goodness of Fit vs. degree of oversampling

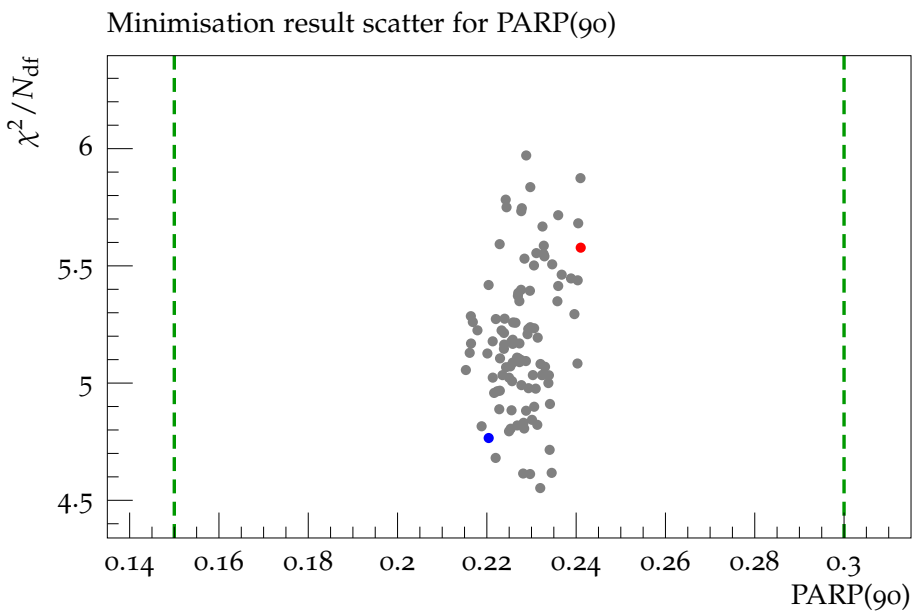
- Oversampling is neccesary, $D > 2 \ldots 3$ seems sensible
- Hower, g.o.f. improves slowly for $D > 4$, almost saturates



Combination tuning uncertainty

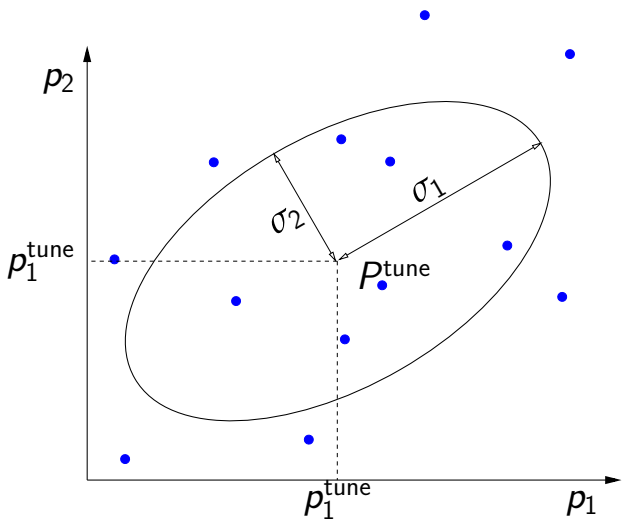
- "Kebap plots": investigate minimisation results parameterwise
- Used as rough estimate on how well a parameter is constrained
- Goal: translate spread into tuning uncertainty, *combination* error





Statistical tuning uncertainty

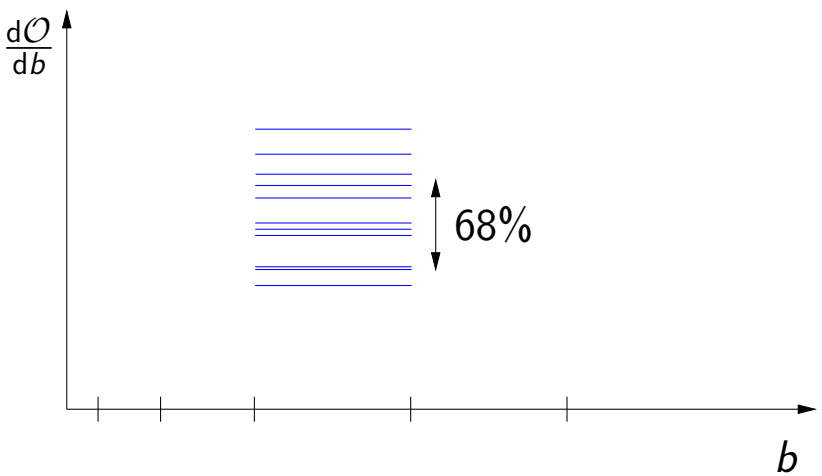
- Exploit covariance matrix returned by minimiser
- Sample points from the valley of the minimum
- Translate these points into *statistical* tuning uncertainty

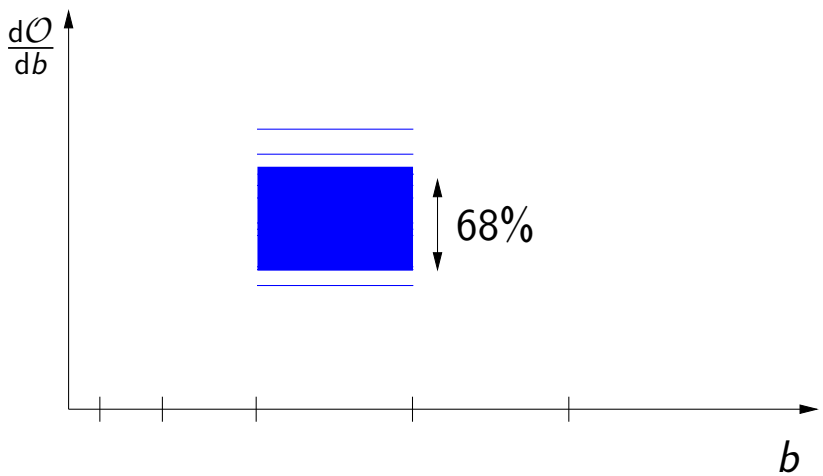


Ultimately: combine *statistical* and *combination* error to a *combined* error

Confidence belt construction

1. Use points sampled from ellipsis or different minimisation results
2. Use parameterisation to get bin-content predictions
3. For each bin b and each observable $\mathcal{O}$: determine central 68, 95 pct.



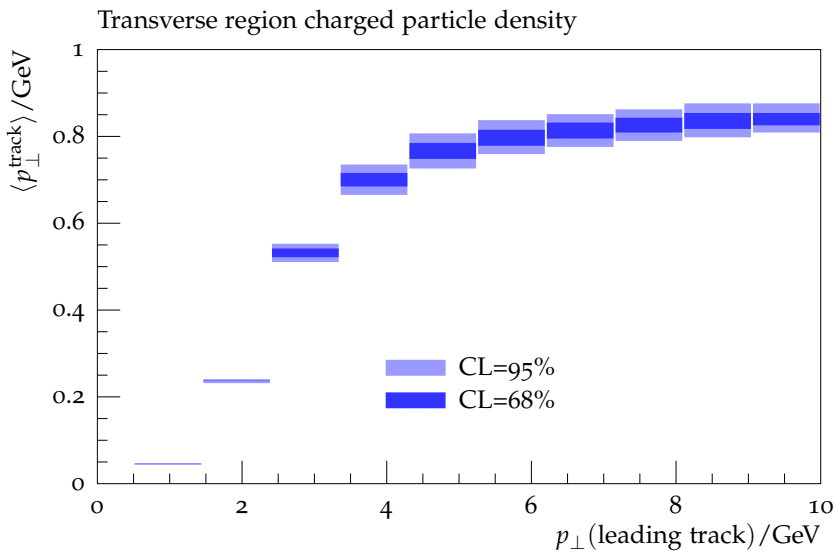


In the following: study behaviour of tuning uncertainty estimates when (pseudo-) data is added

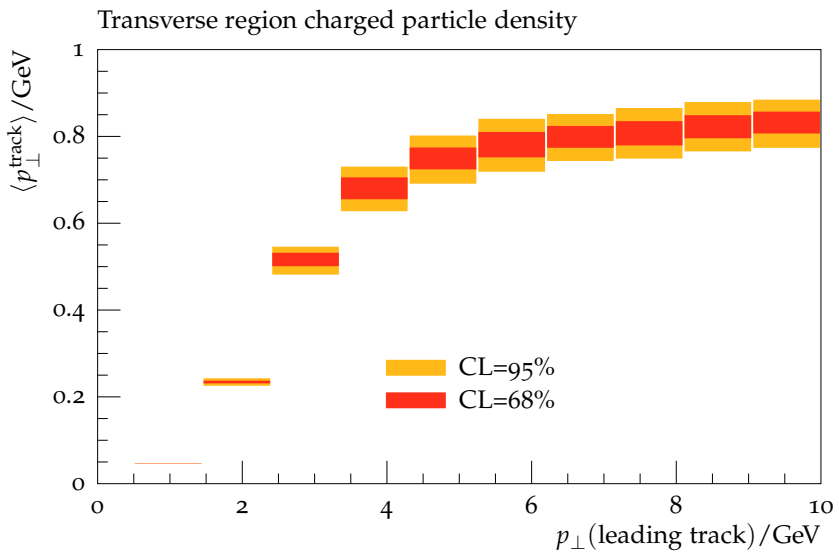
Confidence belt - without pseudodata

These plots show the tuning uncertainties of Jimmy if tuned with Professor to Tevatron data for an observable at 7 TeV

statistical uncertainties



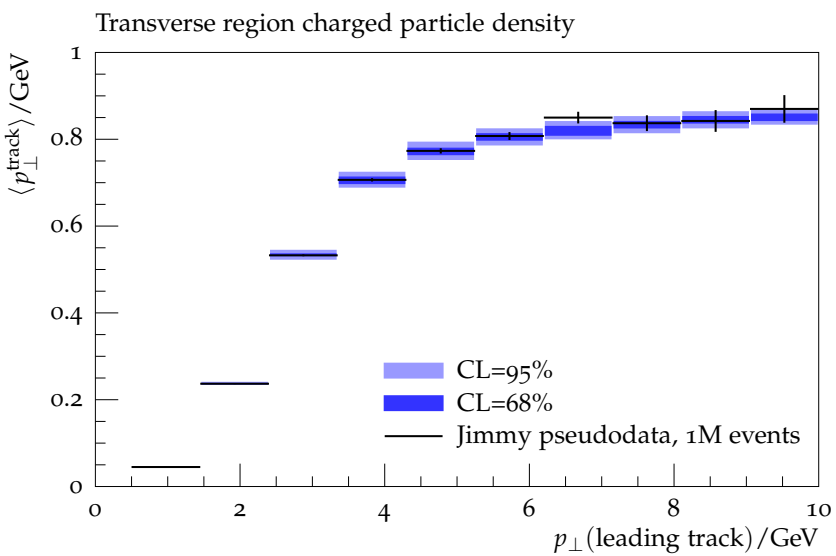
combination uncertainties



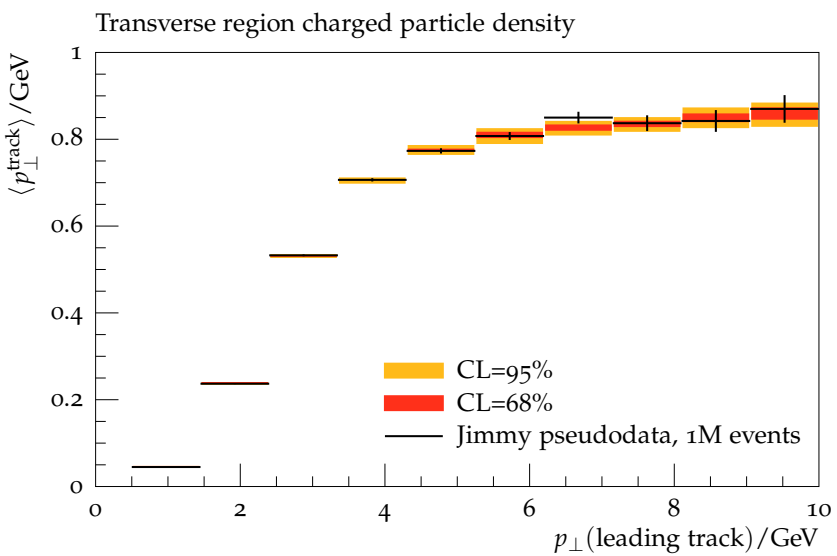
Confidence belt - adding pseudodata

And this is what happens if we add 1 million events of 7 TeV pseudo-data to the Tevatron data in the tuning

statistical uncertainties

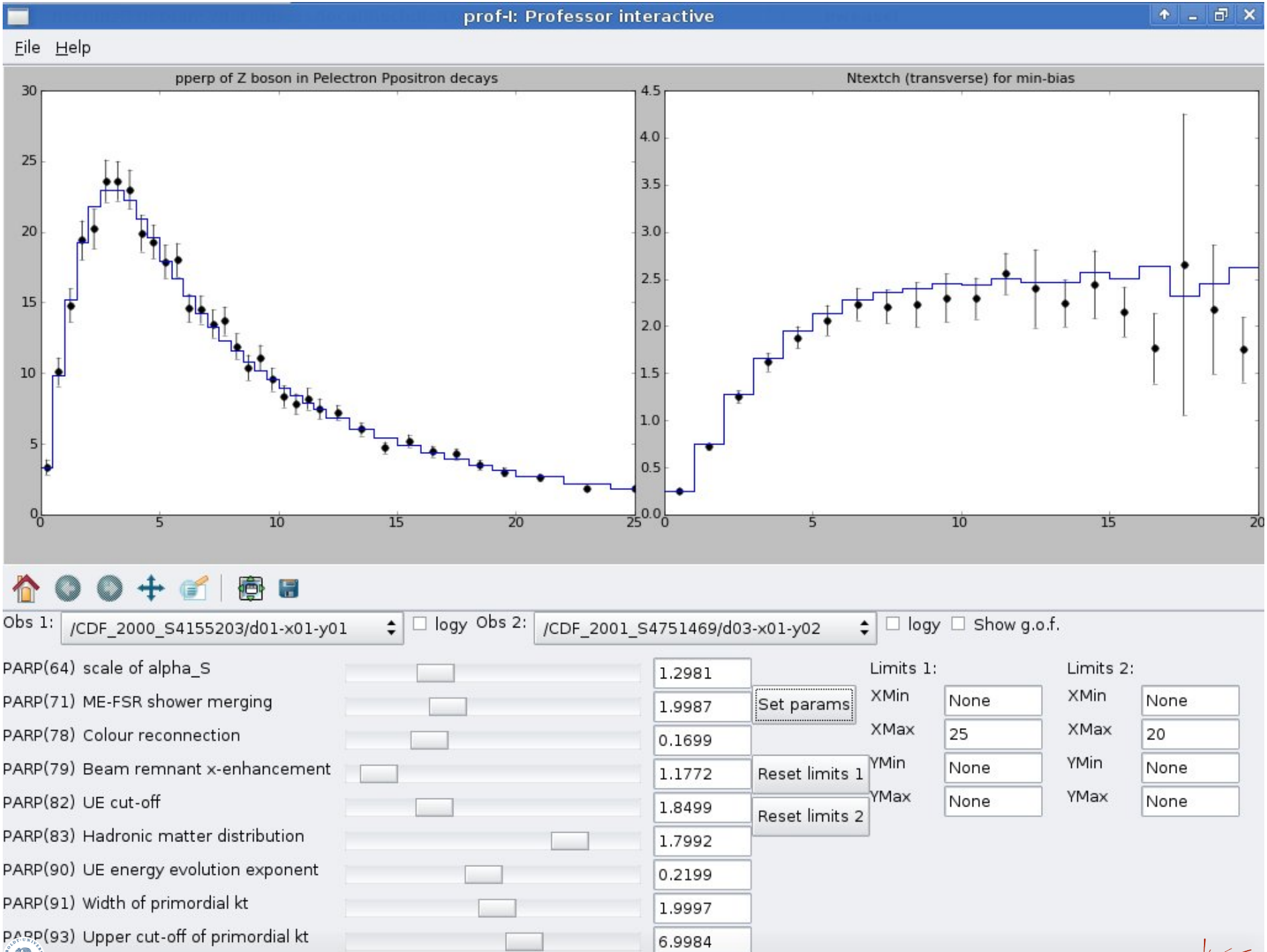


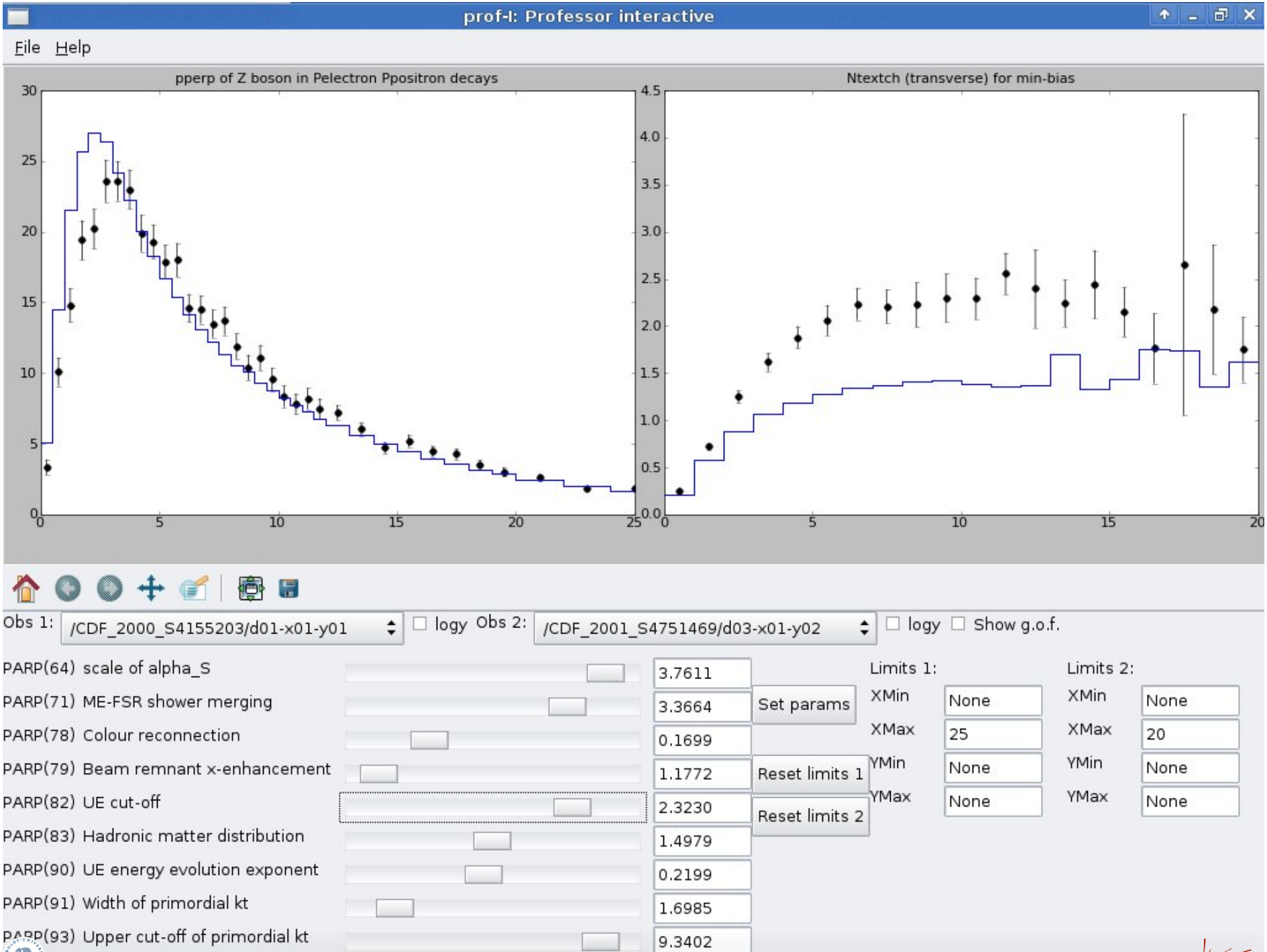
combination uncertainties

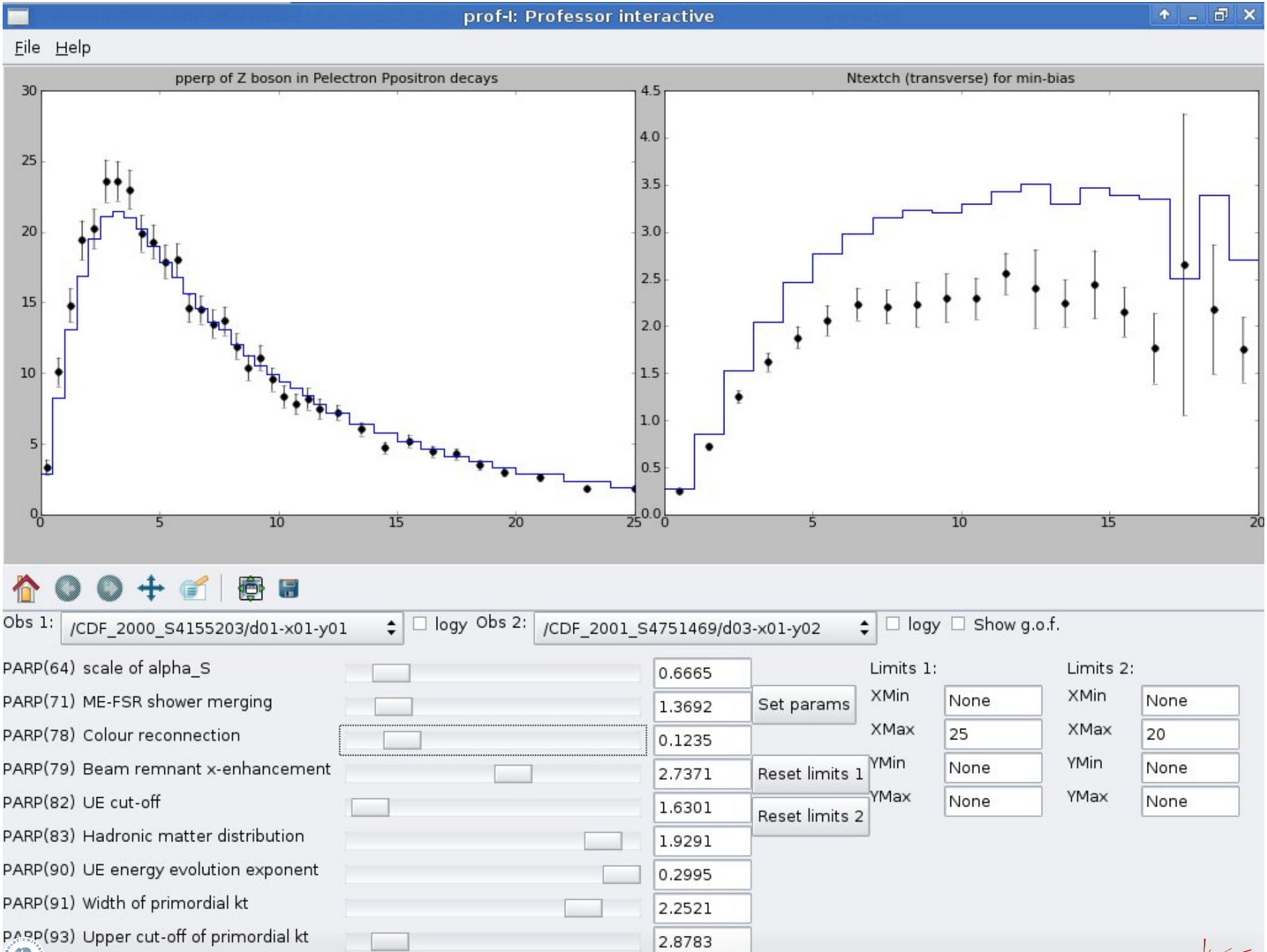


Interactive parameter exploration

- Event generation usually very time consuming
- Hard to get a feeling how parameters influence observables
- Luckily, we have the generator parameterisation handy
- Prediction for bin content can be approximately calculated
- Why not put this into a nice GUI (wxPython), add a vertical slider for each parameter and play a bit?
- $\rightarrow$ real-time response to shifts in parameter space


prof-I: Professor interactive
File Help
pperp of Z boson in Pelectron Ppositron decays
Ntextch (transverse) for min-bias
Obs 1: /CDF_2000_S4155203/d01-x01-y01
logy
Obs 2: /CDF_2001_S4751469/d03-x01-y02
logy
Show g.o.f.
PARP(64) scale of alpha_S 1.2981
PARP(71) ME-FSR shower merging 1.9987
PARP(78) Colour reconnection 0.1699
PARP(79) Beam remnant x-enhancement 1.1772
PARP(82) UE cut-off 1.8499
PARP(83) Hadronic matter distribution 1.7992
PARP(90) UE energy evolution exponent 0.2199
PARP(91) Width of primordial kt 1.9997
PARP(93) Upper cut-off of primordial kt 6.9984
Set params
Reset limits 1
Reset limits 2
Limits 1:
XMin None
XMax 25
YMin None
YMax None
Limits 2:
XMin None
XMax 20
YMin None
YMax None


prof-I: Professor interactive
File Help
pperp of Z boson in Pelectron Ppositron decays
Ntextch (transverse) for min-bias
Obs 1: /CDF_2000_S4155203/d01-x01-y01
logy
Obs 2: /CDF_2001_S4751469/d03-x01-y02
logy
Show g.o.f.
PARP(64) scale of alpha_S 3.7611
PARP(71) ME-FSR shower merging 3.3664
PARP(78) Colour reconnection 0.1699
PARP(79) Beam remnant x-enhancement 1.1772
PARP(82) UE cut-off 2.3230
PARP(83) Hadronic matter distribution 1.4979
PARP(90) UE energy evolution exponent 0.2199
PARP(91) Width of primordial kt 1.6985
PARP(93) Upper cut-off of primordial kt 9.3402
Set params
Reset limits 1
Reset limits 2
Limits 1:
XMin None
XMax 25
YMin None
YMax None
Limits 2:
XMin None
XMax 20
YMin None
YMax None


prof-I: Professor interactive
File Help
pperp of Z boson in Pelectron Ppositron decays
Ntextch (transverse) for min-bias
Obs 1: /CDF_2000_S4155203/d01-x01-y01
logy
Obs 2: /CDF_2001_S4751469/d03-x01-y02
logy
Show g.o.f.
PARP(64) scale of alpha_S 0.6665
PARP(71) ME-FSR shower merging 1.3692
PARP(78) Colour reconnection 0.1235
PARP(79) Beam remnant x-enhancement 2.7371
PARP(82) UE cut-off 1.6301
PARP(83) Hadronic matter distribution 1.9291
PARP(90) UE energy evolution exponent 0.2995
PARP(91) Width of primordial kt 2.2521
PARP(93) Upper cut-off of primordial kt 2.8783
Set params
Reset limits 1
Reset limits 2
Limits 1:
XMin None
XMax 25
YMin None
YMax None
Limits 2:
XMin None
XMax 20
YMin None
YMax None

Summary

- We studied how the interpolation benefits from oversampling
- $N_{\text{runs}} / N_{\text{min}}^{(n)} > 2 \ldots 3$ is advisable
- Working on quantification of tuning uncertainties (to be submitted to Les Houches proceedings)
- We have designed a GUI tool to explore behaviour of observables interactively

Thank you!

Backup

2nd order polynomial includes lowest-order correlations between parameters

$$MC_b(\vec{p}\,) \approx f^{(b)}(\vec{p}\,) = \alpha_0^{(b)} + \sum_i \beta_i^{(b)}\, p'_i + \sum_{i \le j} \gamma_{ij}^{(b)}\, p'_i\, p'_j$$

Now use N generator runs, i.e. N different parameter sets x,y:

$$\underbrace{\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}}_{\vec{v}\ \text{(N values, i.e. N bin contents)}} = \underbrace{\begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 \\ & & & \vdots & & \\ 1 & x_N & y_N & x_N^2 & x_N y_N & y_N^2 \end{pmatrix}}_{\tilde{\mathbf{P}}\ \text{(N sampled parameter sets)}} \underbrace{\begin{pmatrix} \alpha_0 \\ \beta_x \\ \beta_y \\ \gamma_{xx} \\ \gamma_{xy} \\ \gamma_{yy} \end{pmatrix}}_{\vec{c}\ \text{(coeffs)}}$$

Therefore: $\boxed{\vec{c}_b = \tilde{\mathcal{I}}[\tilde{\mathbf{P}}]\vec{v}}$ where $\tilde{\mathcal{I}}$ is the pseudoinverse operator.

$$\vec{c}_b = \tilde{\mathcal{I}}[\tilde{\mathbf{P}}]\vec{v}$$

- Use Singular Value Decomposition (SVD), a general diagonalisation for all normal matrices M: $M = U\Sigma V^*$
- Method available in SciPy.linalg
- Minimal number of runs = number of coefficients in $\vec{c}_b$:
  $N_{\text{min}}^{(n)} = 1 + n + n(n+1)/2 + \underbrace{(n+1)(n+2)/6}_{\text{cubic only}}$
- Oversampling by a factor of three has proven to be much better

Num params, P	$N_2^{(P)}$ (2nd order)	$N_3^{(P)}$ (3rd order)
1	3	4
2	6	10
4	15	35
6	28	84
8	45	165
9	55	220

$$\vec{c}_b = \tilde{\mathcal{I}}[\tilde{\mathbf{P}}]\vec{v}$$

- Use Singular Value Decomposition (SVD), a general diagonalisation for all normal matrices M: $M = U\Sigma V^*$
- Method available in SciPy.linalg
- Minimal number of runs = number of coefficients in $\vec{c}_b$:
  $N_{\text{min}}^{(n)} = 1 + n + n(n+1)/2 + \underbrace{(n+1)(n+2)/6}_{\text{cubic only}}$
- Oversampling by a factor of three has proven to be much better

Num params, P	$N_2^{(P)}$ (2nd order)	$N_3^{(P)}$ (3rd order)
1	3	4
2	6	10
4	15	35
6	28	84
8	45	165
9	55	220

$$\vec{c}_b = \tilde{\mathcal{I}}[\tilde{\mathbf{P}}]\vec{v}$$

- Use Singular Value Decomposition (SVD), a general diagonalisation for all normal matrices M: $M = U\Sigma V^*$
- Method available in SciPy.linalg
- Minimal number of runs = number of coefficients in $\vec{c}_b$:
  $N_{\text{min}}^{(n)} = 1 + n + n(n+1)/2 + \underbrace{(n+1)(n+2)/6}_{\text{cubic only}}$
- Oversampling by a factor of three has proven to be much better

Num params, P	$N_2^{(P)}$ (2nd order)	$N_3^{(P)}$ (3rd order)
1	3	4
2	6	10
4	15	35
6	28	84
8	45	165
9	55	220