Underlying Event measurements with first LHC data

Holger Schulz

Graduiertenkolleg Masse, Spektrum, Symmetrie

Berlin, September 29, 2009

Introduction

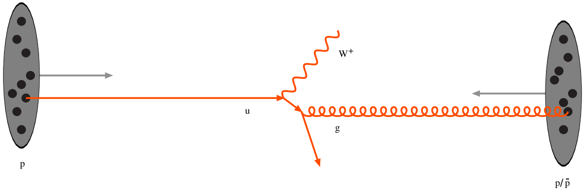
- LHC is a QCD machine $\longrightarrow$ hard to find interesting signals
- QCD perturbatively calculable in hard processes
- Need $\boxed{\text{models for soft physics } (\alpha_s \not\ll 1)}$ to understand background
- Large background at LHC is Underlying Event (UE)
- UE $\approx$ everything except the hard scattering of interest
- Have different models/generators: Herwig, Pythia, Phojet, Sherpa ...
- LHC-predictions differ vastly
- $\longrightarrow$ need measurements to tune generators

Hadron collisions (borrowed from Leif Lönnblad)

Incoming beams, parton density functions (pdfs) & primordial $k_\perp$

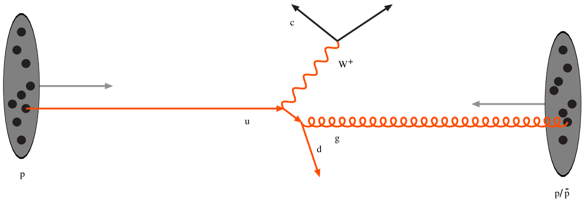
Hadron collisions (borrowed from Leif Lönnblad)

The hard sub-process, the matrix element



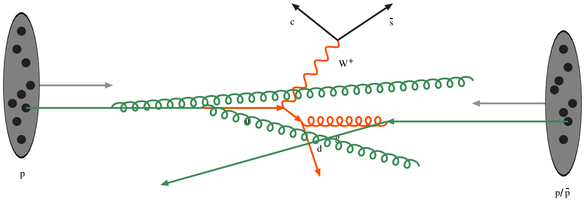
Hadron collisions (borrowed from Leif Lönnblad)

Resonance decays $\longrightarrow$ correlated with the hard sub-process



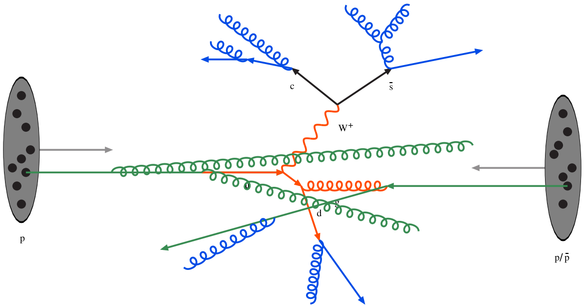
Hadron collisions (borrowed from Leif Lönnblad)

Initial-state radiation (ISR), parton shower (*backward* evolution)



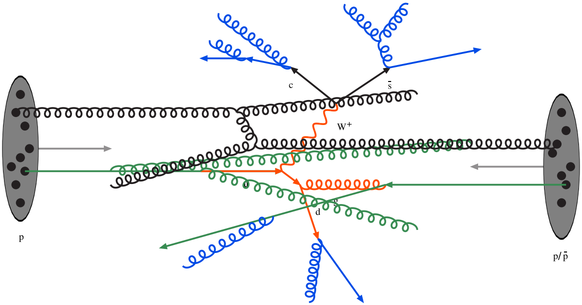
Hadron collisions (borrowed from Leif Lönnblad)

Final-state radiation (FSR), parton shower (*forward* evolution)



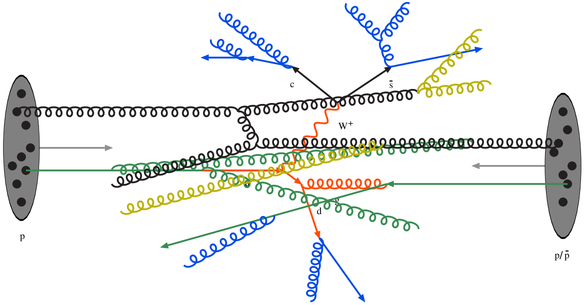
Hadron collisions (borrowed from Leif Lönnblad)

Multiple parton-parton interactions $\longrightarrow$ soft, semi-hard or hard scatterings

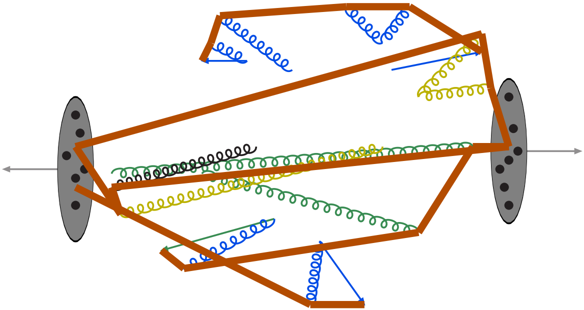


Hadron collisions (borrowed from Leif Lönnblad)

Initial-/Final state showers of ISR-particles

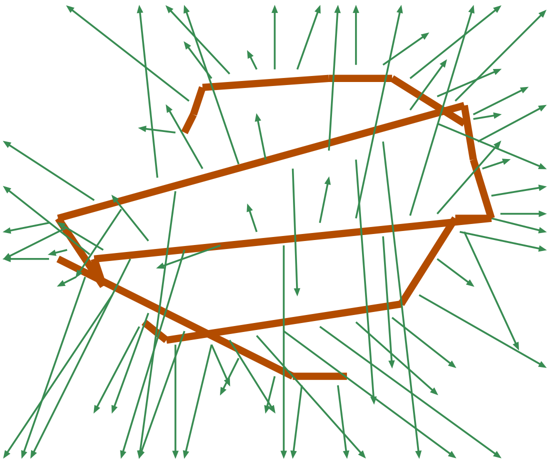


Formation of colour strings, outgoing partons & beam remnants



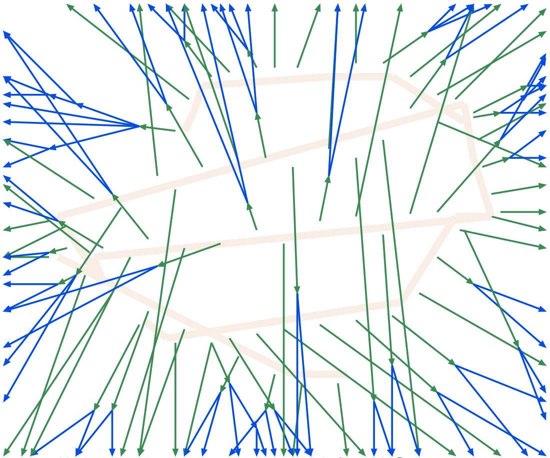
Hadron collisions (borrowed from Leif Lönnblad)

Hadronisation



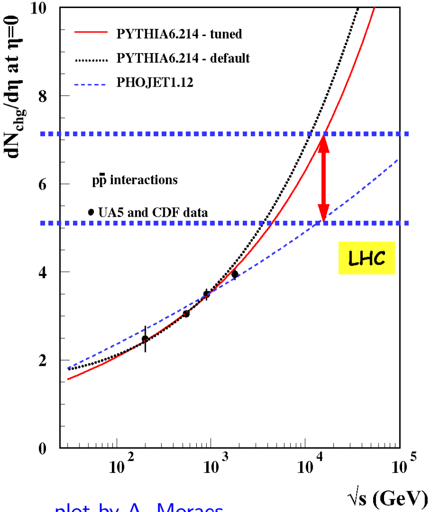
Hadron collisions (borrowed from Leif Lönnblad)

Decay of unstable particles, this is what hits the detector



Extrapolations to the LHC

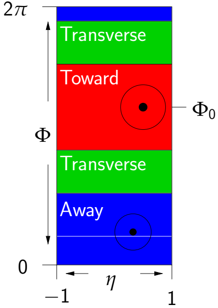
- Drastically different predictions for LHC
- Different UE energy-scaling:
  Phojet $\sim \ln s$
  Pythia $\sim \ln^2 s$
- Generators were tuned to data at different $\sqrt{s}$
- $\rightarrow$ Will need retuning of UE-parameters to LHC data

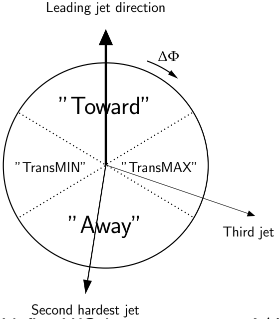


plot by A. Moraes

UE measurements at the Tevatron

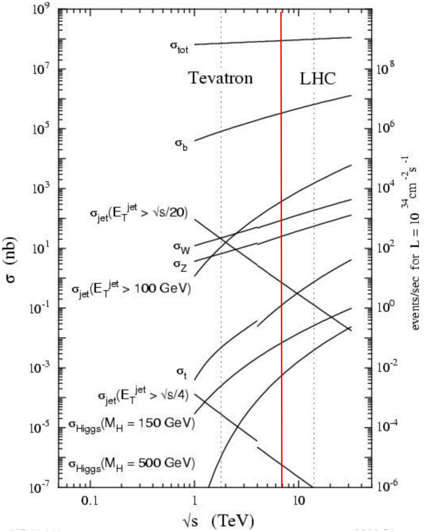
- Z $p_\perp$ from $q\bar{q} \rightarrow$Z: α_S in ISR, primordial $k_\perp$
- Multiplicity distributions: number of particles produced
- $\langle p_\perp \rangle$ vs. N_{ch}: number and $p_\perp$ of particles produced
- Exploiting the event topology - $p_\perp^{\mathrm{sum}}$, N_{ch} vs. $p_{\perp,\mathrm{leading\ jet}}$ in jet events: almost everything





Disadvantages of first LHC data

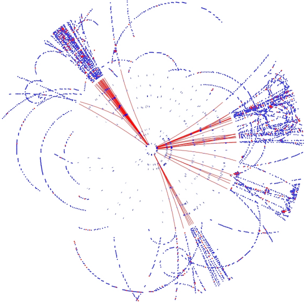
- Jet-energy calibration not very precise in the beginning
- $\rightarrow$ rather use tracks and lepton-ID
- Cross-section at $\sqrt{s} = 7$ TeV smaller than at 10 or 14 TeV
- Expect integrated luminosity of $\mathcal{O}(100\ \mathrm{pb}^{-1})$



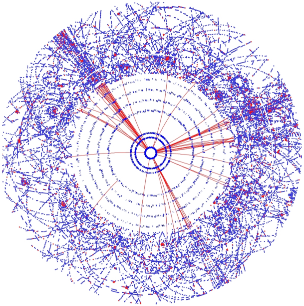
Advantages of first LHC data

- Measurements at $\sqrt{s} = 7$ TeV give another energy point for extrapolations to 10, 14 TeV
- Lower luminosity means reduced pile-up

$\mathcal{L} = 10^{33}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}$



$\mathcal{L} = 10^{34}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}$

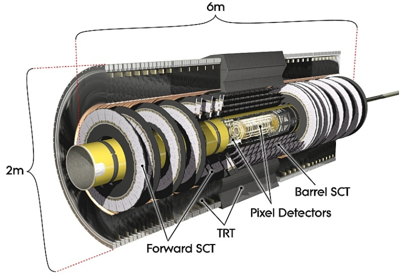


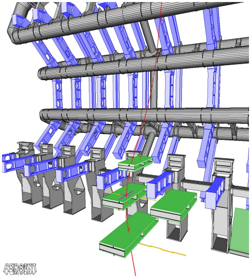
Measurement strategy with ATLAS

Use inner detector for track-$p_\perp$ measurements

+ electron-ID from ECAL

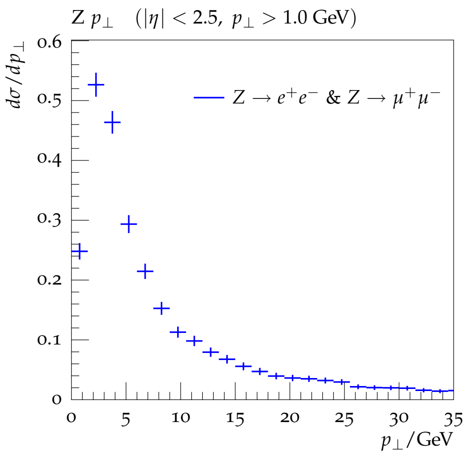
+ muon-ID from muon chambers





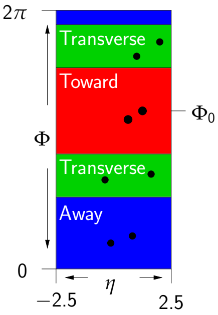
Measuring $p_\perp$ of Z-Bosons

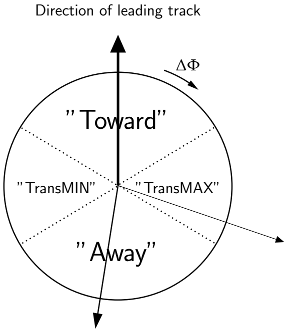
- Use only tracks from leptonic Z-decays
- Clean signal, look for two leptons of opposite sign within a Z-mass window
- 100 pb $^{-1}$ after detector cuts: 14990 events remain on generator level
- Statistics might be too low for a tuning

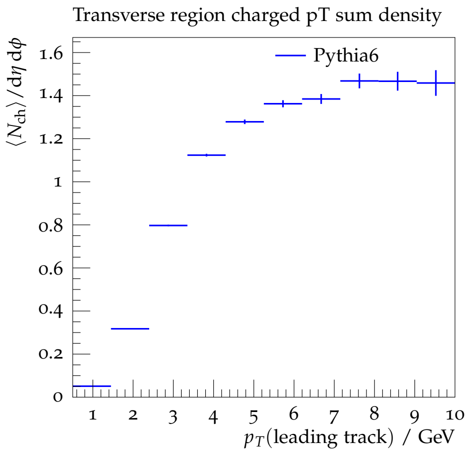


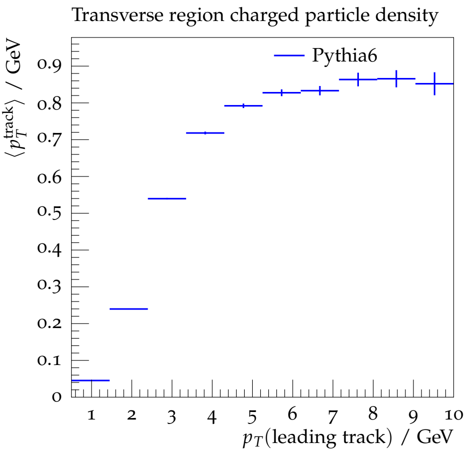
Leading track

- Measure track-$p_\perp$ using only inner detector
- Identify leading track = largest $p_\perp$ in event → defines ϕ_0
- Define "transverse" region, measure N_{tracks}, scalar $p_\perp$-sum as function of $p_{\perp\text{, leading track}}$









- Plateau is a measure for Underlying Event activity
- Data will be taken with Minimum Bias trigger $\rightarrow$ no statistics problem

Does early data improve generator tuning?

Tool for systematic generator tuning: Professor (arXiv:0907.2973)

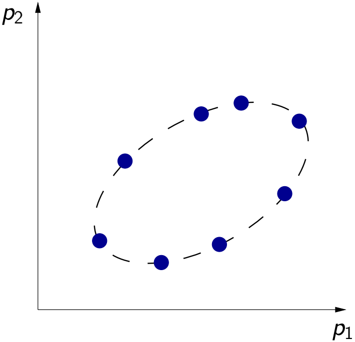
Professor in three lines

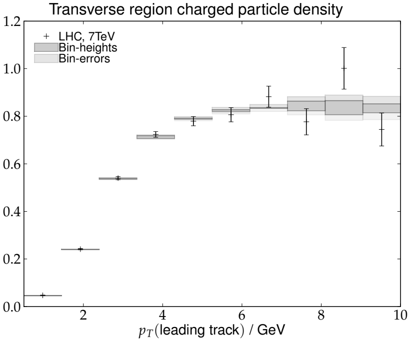
1. Parameterisation of generator response to shifts in parameter space
2. Add experimental data $\longrightarrow$ construct goodness of fit (g.o.f.)
3. Minimise g.o.f. to get best parameter setting (tuning)

- Question: If we add data corresponding to 50, 60, 70 ... 100 pb $^{-1}$ does this improve the tuning?
- $\longrightarrow$ need meaningful error-definition on tuned parameters $\longrightarrow$ use covariance matrix (work in progress) to get error-bands
- $\longrightarrow$ measurement worthwile, if error decreases

Errorbands for generator tuning

- Sample points from contour of hyper (error) ellipsis
- Run generator with these points, construct envelope
- Add fake Monte Carlo "data" $\rightarrow$ see if e.g. plateau is constrained





Summary and Outlook

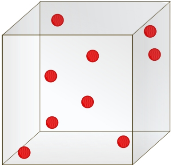
- UE measurements at LHC essential for proper generator tuning
- Need to identify reasonable observables for first data
- UE as function of leading track $p_\perp$ looks promising
- Probably not enough statistics for Z-bosons
- W-bosons might be an option
- ATLAS CMS co-operation on Minimum Bias & UE
- Include UA5 data at $\sqrt{s} = 200$ and 900 GeV

Thank you!

Backup

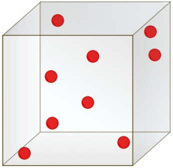
Tuning procedure in Professor

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 (2nd or 3rd order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. and numerically *minimize* `pyMinuit`, `SciPy`



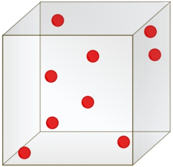
Tuning procedure in Professor

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. and numerically *minimize* `pyMinuit`, `SciPy`



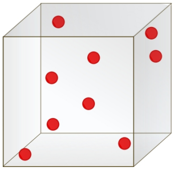
Tuning procedure in Professor

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 (2nd or 3rd order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. and numerically *minimize* `pyMinuit`, `SciPy`



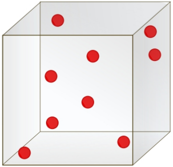
Tuning procedure in Professor

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. and numerically *minimize* `pyMinuit`, `SciPy`



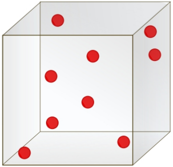
Tuning procedure in Professor

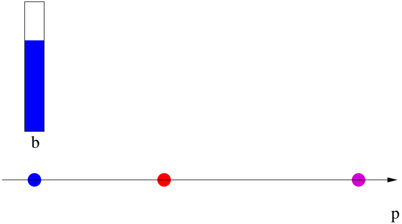
1. random sampling: N parameter points in n-dimensional space
2. run generator and fill histograms
3. for each bin: use N points to fit interpolation (2$^{\text{nd}}$ or 3$^{\text{rd}}$ order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$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



Tuning procedure in Professor

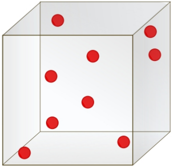
1. random sampling: N parameter points in n-dimensional space
2. run generator and fill histograms
3. for each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. and numerically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

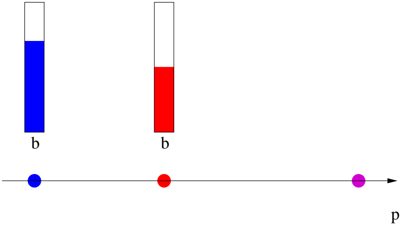




Tuning procedure in Professor

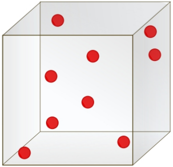
1. random sampling: N parameter points in n-dimensional space
2. run generator and fill histograms
3. for each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. and numerically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

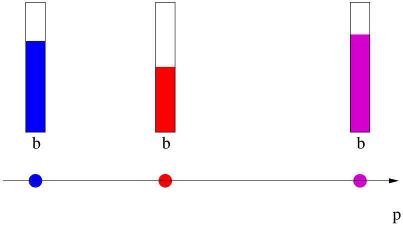




Tuning procedure in Professor

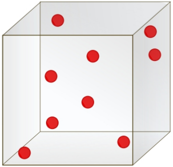
1. random sampling: N parameter points in n-dimensional space
2. run generator and fill histograms
3. for each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. and numerically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

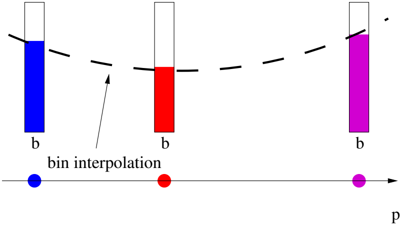




Tuning procedure in Professor

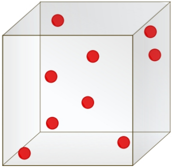
1. random sampling: N parameter points in n-dimensional space
2. run generator and fill histograms
3. for each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. and numerically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

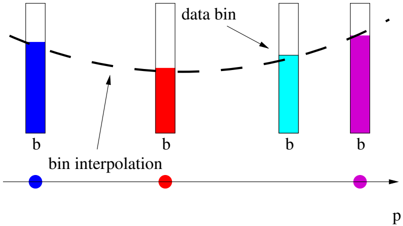




Tuning procedure in Professor

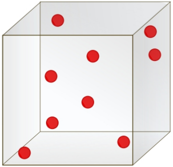
1. random sampling: N parameter points in n-dimensional space
2. run generator and fill histograms
3. for each bin: use N points to fit interpolation (2nd or 3rd order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. and numerically *minimize*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

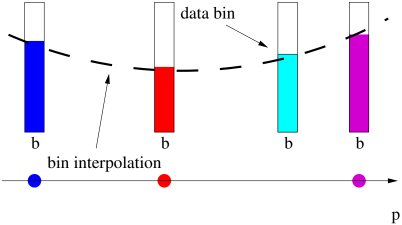




Tuning procedure in Professor

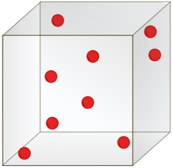
1. random sampling: N parameter points in n-dimensional space
2. run generator and fill histograms
3. for each bin: use N points to fit interpolation (2^{nd} or 3^{rd} order polynomial)
4. construct overall (now trivial) $\chi^2 = \sum_{bins} \frac{(interpolation - data)^2}{error^2}$
5. and numerically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

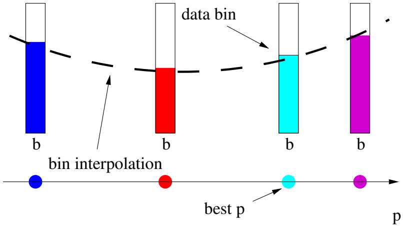




Tuning procedure in Professor

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. and numerically *minimize* `pyMinuit, SciPy`





2nd order polynomial includes lowest-order correlations between parameters

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

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.

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

- 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

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

- 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

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

- 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 $\vec{c}_b$:
  $N_{\mathrm{min}}^{(n)} = 1 + n + n(n+1)/2 + \underbrace{(n+1)(n+2)/6}_{\text{cubic only}}$
- 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