The Professor approach to evaluating expensive functions

Holger Schulz (IPPP)
holger.schulz@durham.ac.uk

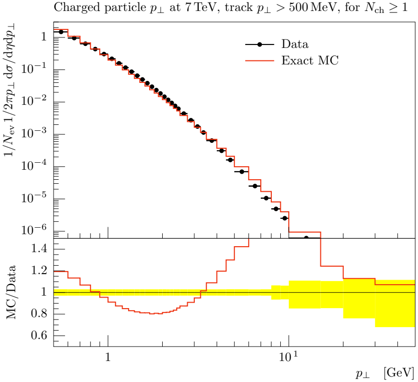
March 30, 2016

Expensive functions

- D-dimensional parameter space, $\mathcal{P}$, with points $\vec{p}$
- Exact but (CPU) expensive function $f(\vec{p})$

Expensive functions

- D-dimensional parameter space, $\mathcal{P}$, with points $\vec{p}$
- Exact but (CPU) expensive function $f(\vec{p})$
- e.g. $\mathcal{P}$: underlying event model parameter space

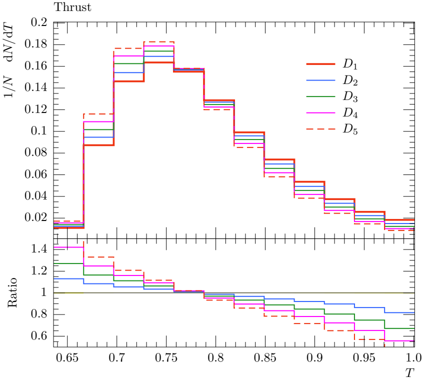


Expensive functions

- D-dimensional parameter space, $\mathcal{P}$, with points $\vec{p}$
- Exact but (CPU) expensive function $f(\vec{p})$

- e.g. $\mathcal{P}$: mean number of pile-up vertices in events

http://inspirehep.net/record/1424838

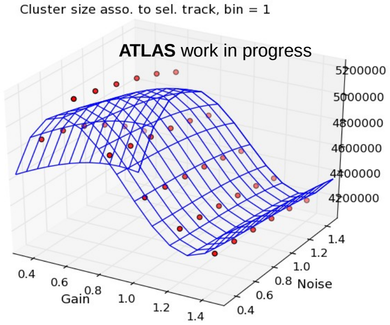


Expensive functions

- D-dimensional parameter space, $\mathcal{P}$, with points $\vec{p}$
- Exact but (CPU) expensive function $f(\vec{p})$

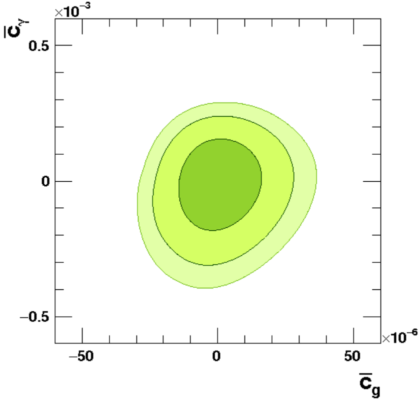
- e.g. $\mathcal{P}$: gain, noise in digitisation of SCT simulation (Akanksha Vishwakarma, DESY Zeuthen)

https://indico.cern.ch/event/485903/session/13/contribution/202/attachments/1227683/1798196/DPGTalk.pdf



Expensive functions

- D-dimensional parameter space, $\mathcal{P}$, with points $\vec{p}$
- Exact but (CPU) expensive function $f(\vec{p})$



- e.g. $\mathcal{P}$: BSM parameter space, e.g. dim 6 operators in HEFT
  http://inspirehep.net/record/1405105

Optimisation

- Typical tasks:
  1. Minimise χ^2 measure between $f(\vec{p})$ and data to find best point $\vec{v}_{\text{best}}$ (e.g. MC tuning)
  2. Limit setting: find collection of points (parameter sub-space) that is not excluded by data (BSM)
- Common problem: CPU time for evaluating $f(\vec{p})$ too large for meaningful processing of higher dimensional parameter spaces

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 $\mathcal{P}$

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

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

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
4. Validate $I(\vec{p})$

Parametrisation validation

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

Parametrisation validation

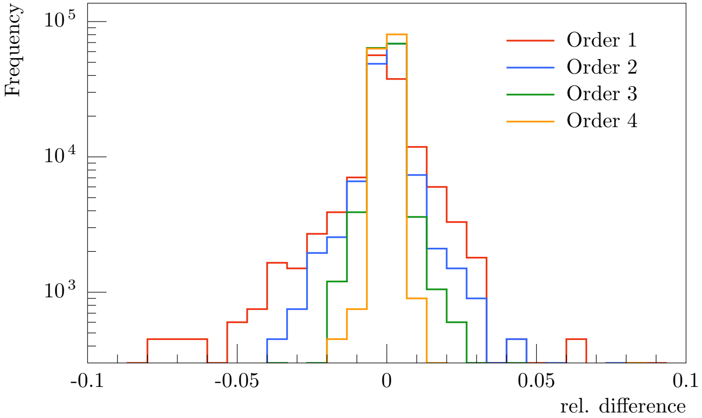
- E.g. histogram H with N bins:
  - $I(\vec{p}) \rightarrow \{I_b(\vec{p})\}_{b=1\ldots N}$
  - Thus $\vec{H}(\vec{p}) = (\ I_1(\vec{p}), I_2(\vec{p}), \ldots, I_N(\vec{p})\ )^T$

Parametrisation validation

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



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

Advantages

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

Advantages

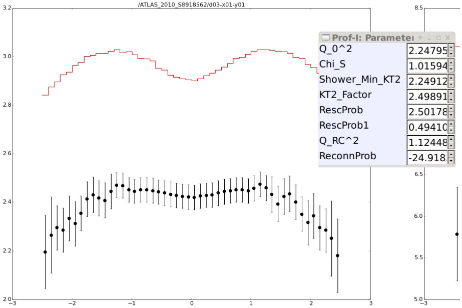
- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications
- Fitting against data cheap, can bias minimisation e.g. if $f(\vec{p})$ known to be imperfect (AMBT2 and AUET2)

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

Advantages

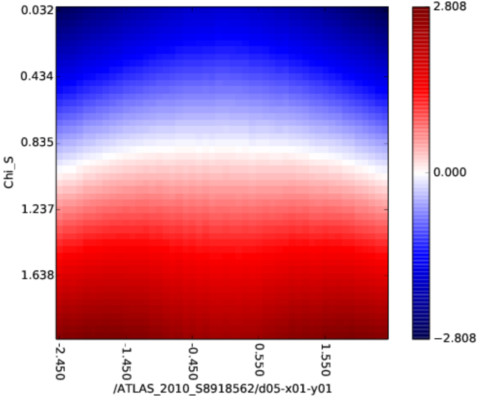
- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications

- Interactive parametrisation explorer



Advantages

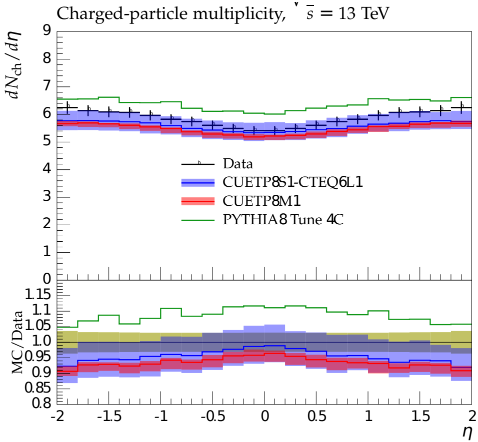
- $I(\vec{p})$ fast, analytical → suitable for numerical applications
- Sensitivity and correlation analysis cheap → find parameters that do nothing → reduce dimensionality



Advantages

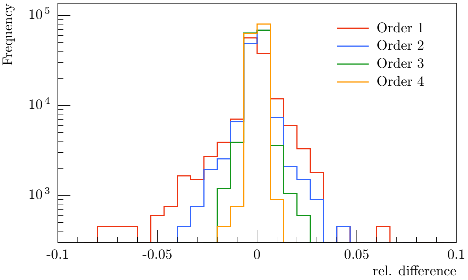
- $I(\vec{p})$ fast, analytical $\rightarrow$ suitable for numerical applications

- 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

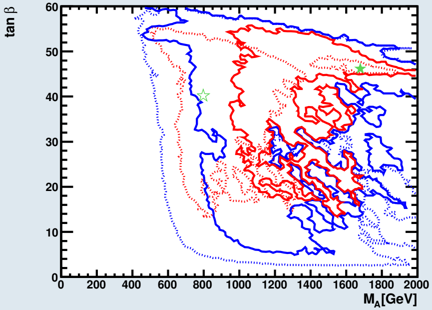


Limitations

- Required number of exact $f(\vec{p})$ grows rapidly with order of polynomial
- Parameter space can be "too big" i.e. polynomial not a meaningful approximation

Next design goals

- Better sampling
- Deal with more complicated parameter spaces
- I.e. how to partition $\mathcal{P}$ cleverly into S sub spaces to automatically evaluate:
  $\vec{I}_b(\vec{p}) = \left(\vec{I}_b^1(\vec{p}), \vec{I}_b^2(\vec{p}), \cdots, \vec{I}_b^S(\vec{p})\right)^T$



http://inspirehep.net/record/1081561