BERKELEY LAB

Bringing Science Solutions to the World

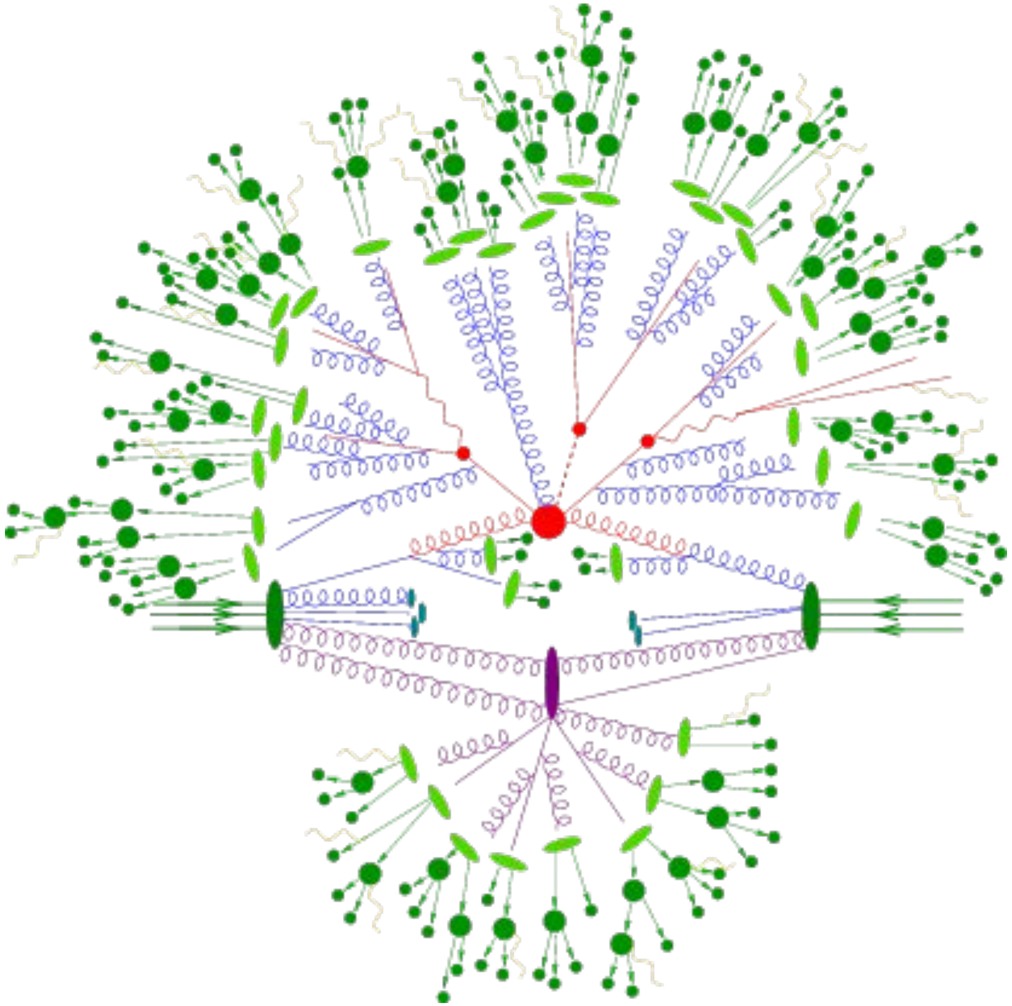
Generator Tuning with MC uncertainties

Xiangyang Ju and Jeffae Schroff

CHEP 2023,
Norfolk Virginia

9 May 2023

Simulation of a proton-proton collision

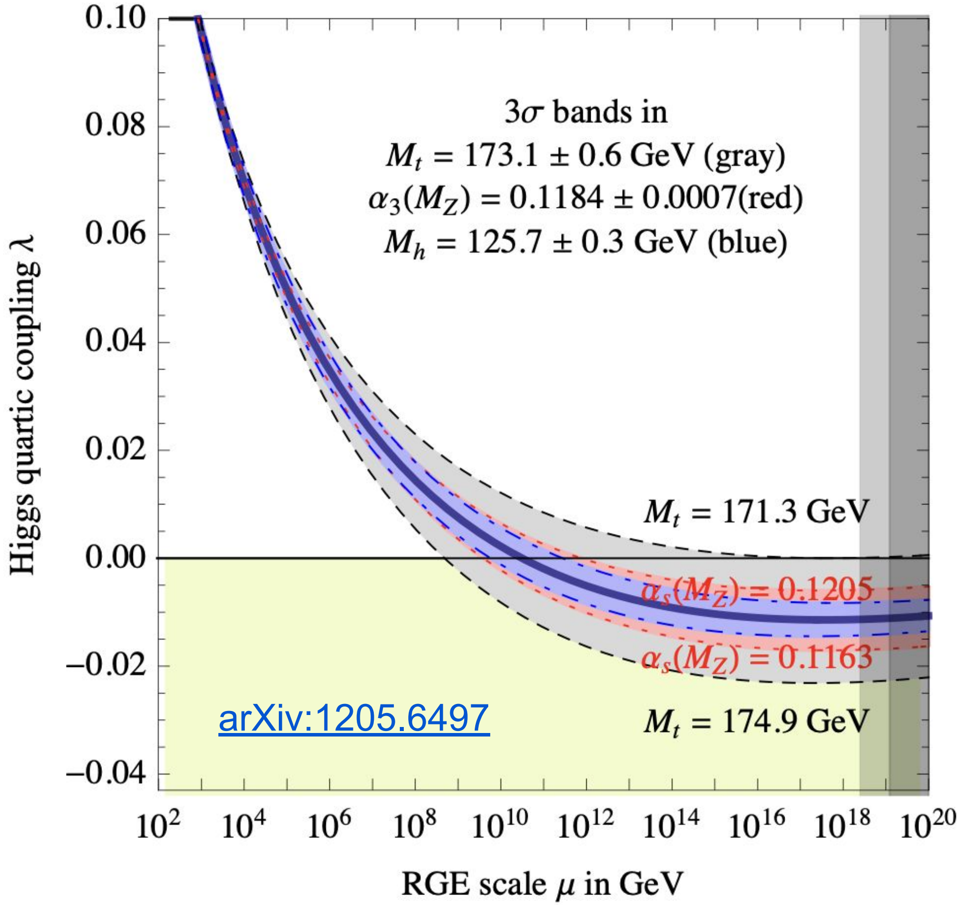


- Hard process
  - Parton interactions, described by Perturbative QCD
- Parton Showering →
  - Partons splitting to many partons
- Multiple Parton Interactions
- Hadronization →
  - Partons to hadrons
  - Simulated by the “string model” (Pythia8 or Sherpa) or the “cluster model” (Herwig)
  - Both models contain free parameters ***that need to be tuned*** so that the model matches data

Motivation

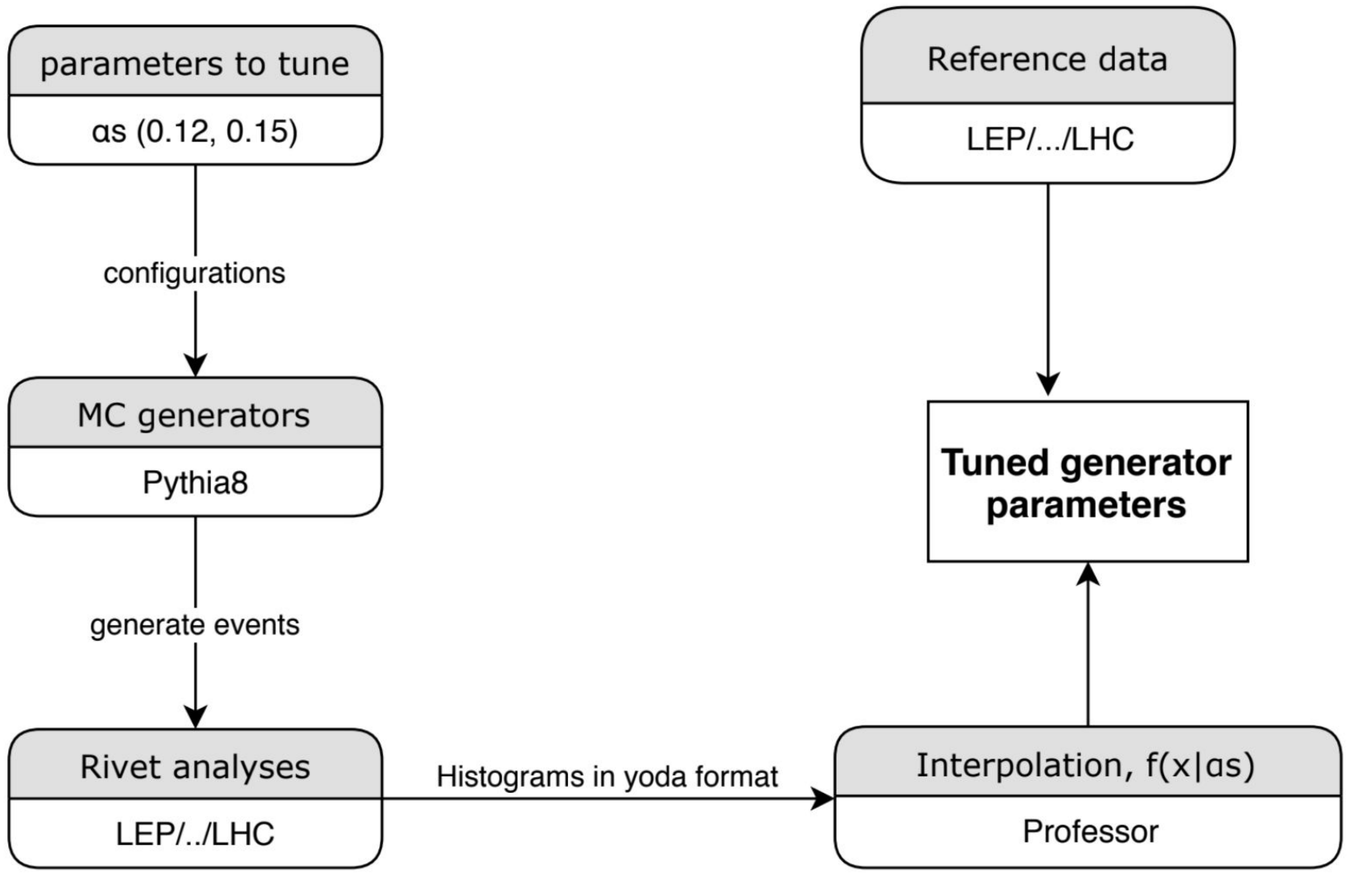
Generator tuning is important for precision measurements

- Generator tuning will play an important role in precision measurement, such as Top mass measurement
- In Top mass from ATLAS combined measurement [TOPQ-2017-03], its precision is entering "uncharted territory", where ***hadronization and color reconnection effects*** become important.
- Strong interactions below ~500 MeV is every difficult, described by empirical models with many parameters that can be tuned to experimental observable



Professor: automated MC tuning

https://professor.hepforge.org/



Key component is to train a "surrogate function" that models the dependence of the observable values on the generator parameters

Then use the surrogate function to find the optimal generator parameters by minimizing the χ^2 function

Apprentice: Python-based automated MC Tuning

https://github.com/HEPonHPC/apprentice

- Rewrite the tools with Python.
  - Data is based on `numpy`, minimizer from `scipy`, parallelism by Message Passing Interface
- Reformulate the tuning procedure as bi-level optimization
  - *Inner loop optimization* and *Outer loop optimization*
- Add HDF5 representation for Histograms
  - Serialize the results in Json files
- Add additional surrogate function → Rational approximation
- Add robust optimization method
- See our paper here, https://arxiv.org/abs/2103.05748

Why yet another tool?

- Minimization of the objective function often require the calculation of gradients of the function w.r.t the parameters to be tuned
  - The gradients are calculated **manually**
- Current objective function does not include MC uncertainties
  - Extending to new objective function requires the calculation of the gradients

- `jax` has features attractive to us
  - supports both GPUs and CPUs
  - **automatic** differentiation → gradients and Hessian matrix
  - support of machine learning models (stochastic gradient descent)

Jax MC Tuner is based on Jax with a python interface for MC Tuning

A dummy data scenario

Two observables (Exponential function), each with 20 bins, x in [0, 3]

$$y_0 = e^{ax_0 + bx_0^2}$$

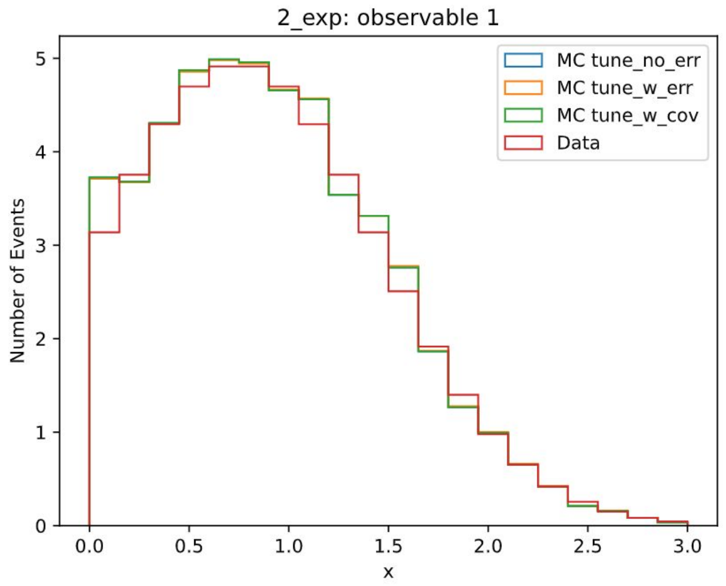
$$y_1 = e^{ax_1 + bx_1^3}$$

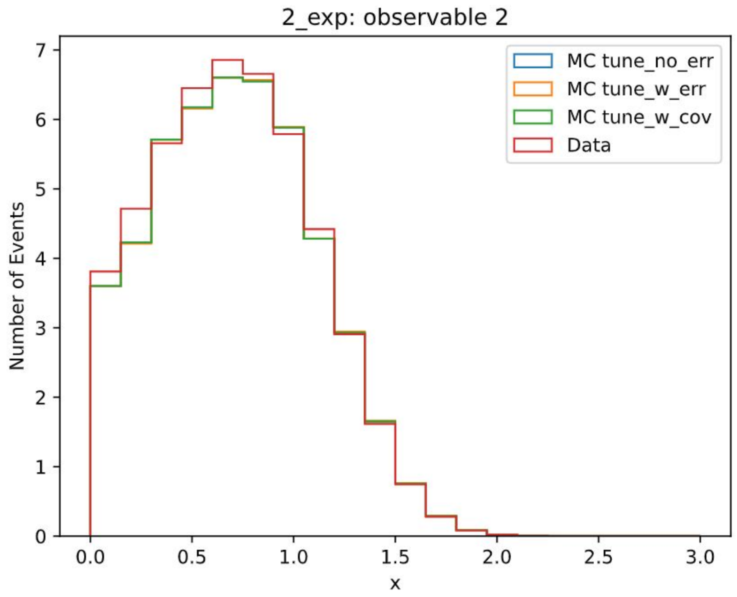
Two “generator” parameters:

- a ∈ [1, 2] and b ∈ [-1.2, -0.8]

MC Runs

- Sample 30 independent pairs of (a, b)
- Generate 100 k events for each pair





Inner loop optimization

We use the *monomial function* of order = 3 as the surrogate function

$$y = \vec{P} \times W$$

y are the observable values in the bin

$$\text{solve: } \min ||y - \vec{P} \times W||^2 \quad \text{obtain } \hat{W}$$

$$\vec{p} = \begin{pmatrix} a_0 & b_0 & a_0^2 & a_0 b_0 & b_0^2 & a_0^3 & a_0^2 b_0 & a_0 b_0^2 & b_0^3 \\ \dots \\ a_n & b_n & a_n^2 & a_n b_n & b_n^2 & a_n^3 & a_n^2 b_n & a_n b_n^2 & b_n^3 \end{pmatrix}$$

$$W = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \end{pmatrix}$$

Inner loop optimization with MC uncertainties

We use the monomial function of order = 3 as the surrogate function

$$y = \vec{P} \times W$$

y are the observable values in the bin

$$\text{solve: } \min || y - \vec{P} \times W ||^2 / y_{\text{error}}^2$$

$$\vec{p} = \begin{pmatrix} a_0 & b_0 & a_0^2 & a_0 b_0 & b_0^2 & a_0^3 & a_0^2 b_0 & a_0 b_0^2 & b_0^3 \\ \dots \\ a_n & b_n & a_n^2 & a_n b_n & b_n^2 & a_n^3 & a_n^2 b_n & a_n b_n^2 & b_n^3 \end{pmatrix}$$

$$W = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \end{pmatrix}$$

$$\text{obtain } \hat{W} \text{ and covariance matrix } V$$

Outer loop optimization

Objective Function:

- *d: experimental data*
- *P: generator parameters to be tuned*
- *W: optimal surrogate function weights*

$$\min \sum\nolimits_{\mathrm{bin}} \frac{d - \vec{P} \times \hat{W}}{\Delta d^2}$$

- By default, no MC uncertainties are added to the objective function (namely, *no_error*)
- The Monash tune, https://arxiv.org/abs/1404.5630, manually add 5% to the objective function
- Optionally, add MC uncertainties as additional term (namely, *with_error*)

$$\min \sum\nolimits_{\mathrm{bin}} \frac{d - \vec{P} \times \hat{W}}{(\Delta d)^2 + \epsilon^2}$$

Outer loop optimization with MC uncertainties

Objective Function:

- *d: experimental data*
- *P: generator parameters to be tuned*
- *W: optimal surrogate function weights*
- *V: covariance matrix of W*

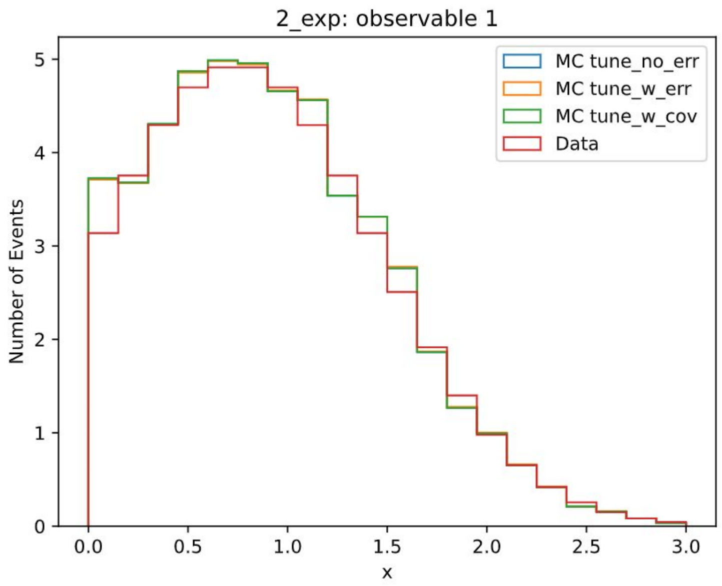
$$\min \sum_{\text{bin}} \frac{d - \vec{P} \times \hat{W}}{\Delta d^2 + \vec{P} V \vec{P}^T}$$

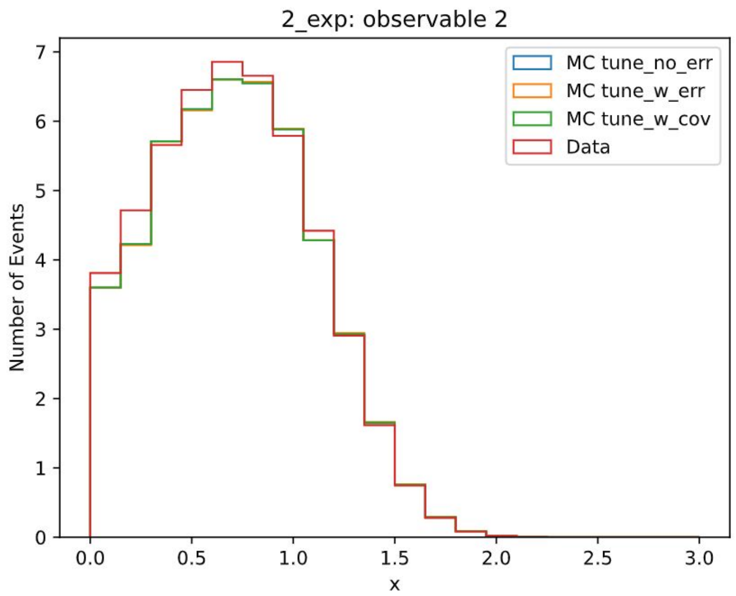
Instead of directly using the MC uncertainty ε in the outer loop optimization, we propagate the error from the inner optimization to the outer optimization (namely ***with_cov***)

With covariance, the objective function becomes difficult to optimize

Dummy data result

Generate dummy experiment data by setting a = 1.5, b = -1.0 (i.e. target params)



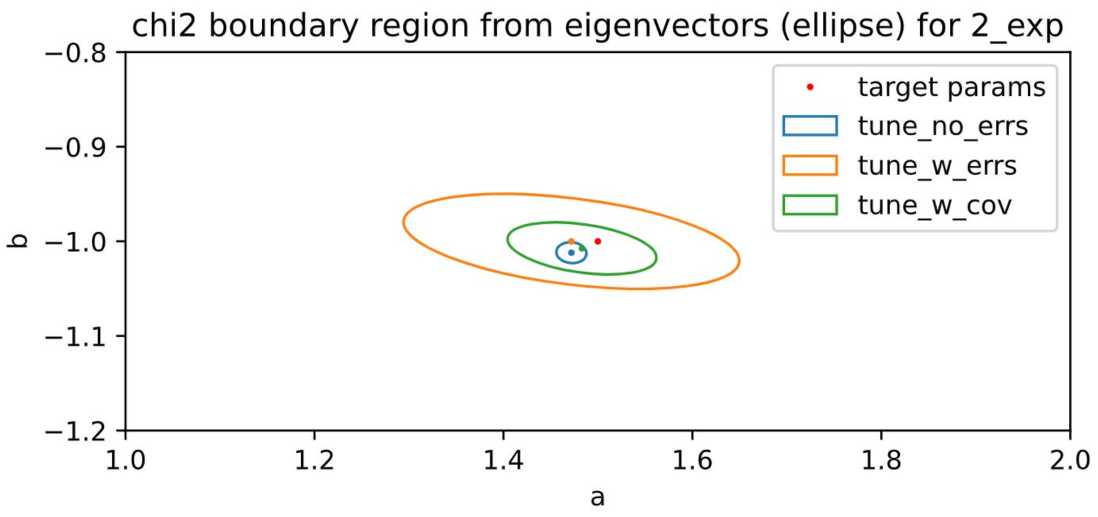


All three methods yield a similar performance

- Similar optimized generator parameters
- Similar agreement between tuned distributions and the true distribution

Dummy data result

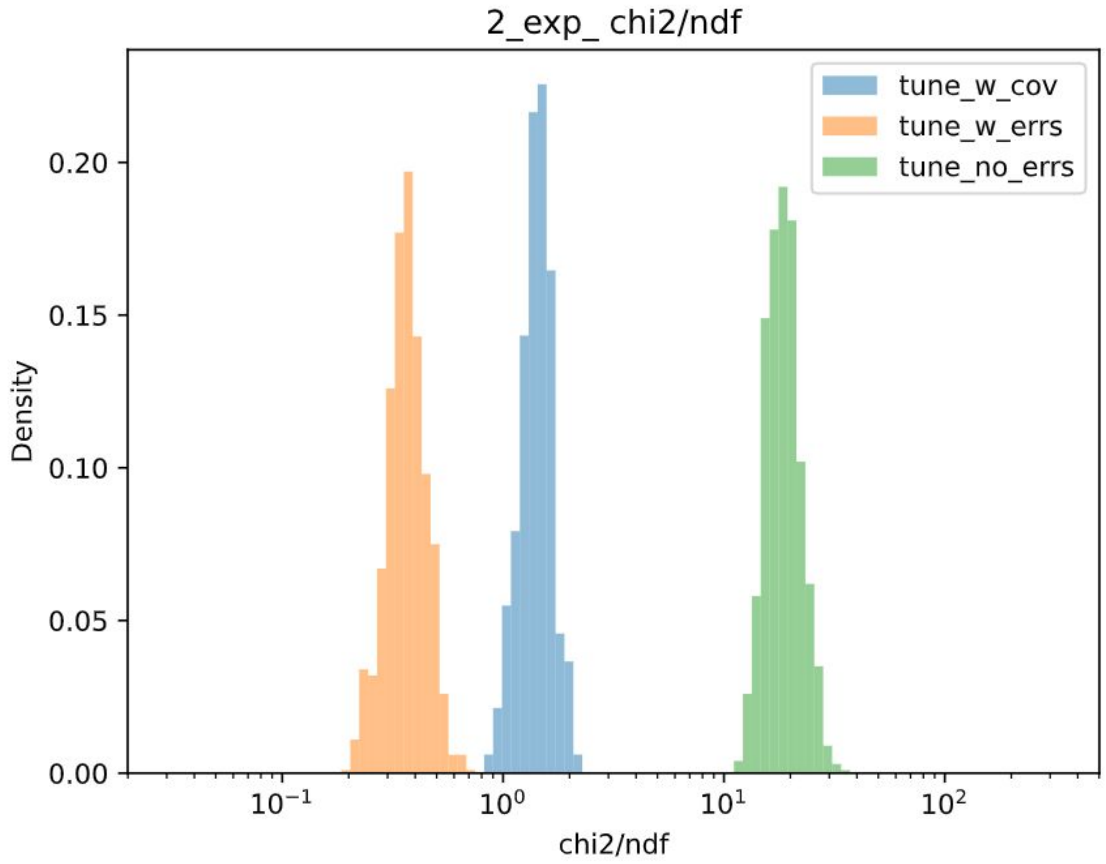
Generate dummy experiment data by setting a = 1.5, b = -1.0 (i.e. target params)



The contour shows the space where the objective function is increase by the number of degree freedom

- *no_errs* yields a very aggressive error estimation (underestimated)
  - In practice, we manually increase the error to an extent that uncertainty band could cover the data
- *with_err*, however, yields a very conservative error estimation (overestimated)
- *with_cov* yields a most reasonable error estimation

Dummy data result



Running the outer optimization 1000 times, and plot the histogram with successful optimization

Again, *with_cov* yields the most reasonable χ2 / nDoF

However, many runs end up to the boundary of the generator parameter space in *with_cov*, partially due to the complication of the objective function

Conclusion

- We examined the impact of adding the MC uncertainties into the Generator tuning
- Propagating the MC uncertainties through the inner loop optimization to the outer loop optimization provides the most sensible generator parameter uncertainty estimation
  - No need of ad-hoc manipulation of the tuning parameters
- Several issues with the integration of the covariance matrix into the outer loop objective function
  - Much more computation, making the tuning hard to scale
  - Harder to find the optimal generator parameters
    - Need multiple starting points,
    - Need better minimization algorithms; stochastic gradient descent?