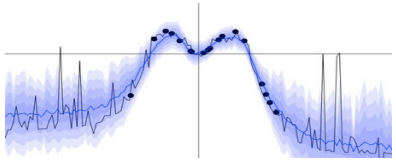
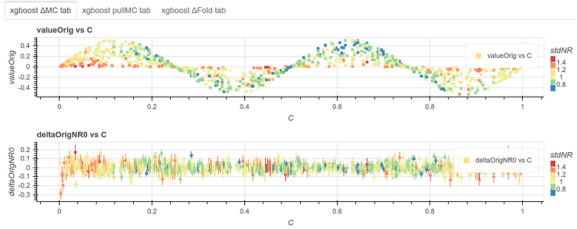
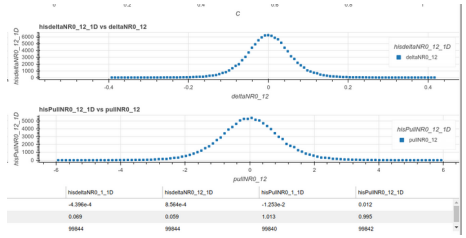
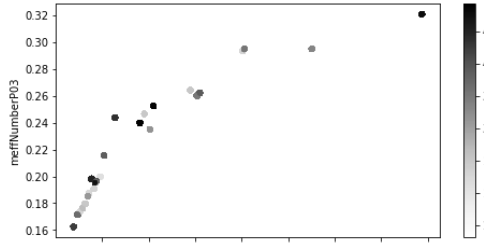
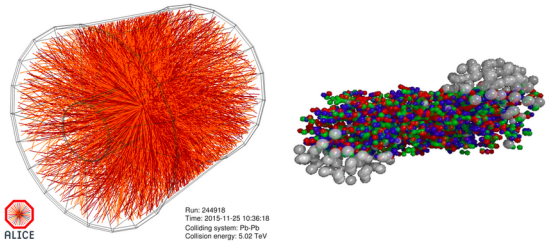








ALICE - Non-parametric and parametric models, dealing with uncertainty.



Alice Experiment at CERN

Multidimensional analysis pipeline

- non parametric and parametric representation (physics models and effective parameterization)

RootInteractive

- for interactive visualization, functional composition and data aggregation

Machine learning uncertainty wrappers, adaptive kernel method

Example use case of data driven space charge calibration - advantage of hybrid (physics &ML) approach

- Minimising dependence on varying calibration parameters (TPC electron transparency, gain, drift velocity).
- Significantly less statistical data is required for distortion calibration, or more frequent calibration is possible

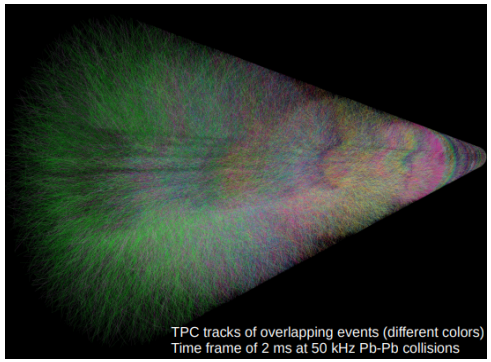
ALICE is one of eight detector experiments at the Large Hadron Collider at CERN

Time Projection Chamber (TPC) is main tracking detector

- gas detector, particle detection via gas ionization

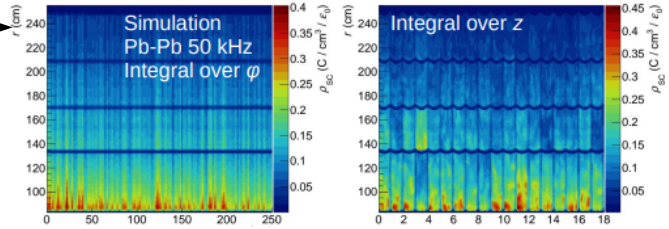
TPC continuous readout at 50 kHz interaction rate in Pb-Pb collisions

- Drift chamber - unknown event time → unknown z position
- Events overlapping in TPC → substantial higher occupancy (~5 event)



Space charge in TPC inside the drift volume **distorting** trajectories

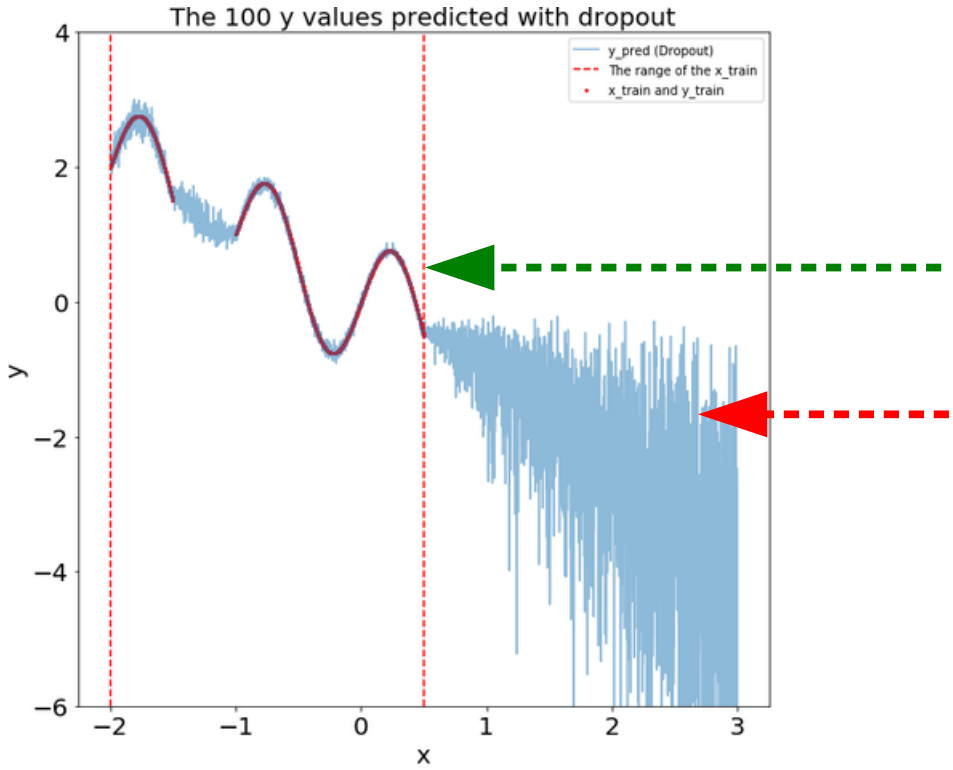
- Non-uniform space-charge density ρ_{SC} ➔ space points distortion O(5 cm)
- ➔ Space-charge density and distortion fluctuations O(5 %) ~ 0.2 cm
- To be calibrated/corrected to σ ~100 μm with granularity O(10^6) in space O(5 ms) in time



A high interaction rate environment, pile-up, distortions, etc. … necessitates the use of advanced methods of data analysis. Differential understanding of the processes in ND needed

Why Should we Care About Uncertainty?

https://fairyonice.github.io/Measure-the-uncertainty-in-deep-learning-models-using-dropout.html



Knowledge of errors and PDF crucial for data interpretation

- irreducible error intrinsic data fluctuation
- reducible error
- model error

ML non-parametric (non-constrained) models good for interpolation
bad for extrapolation
Errors and PDF to be extracted locally

Combination of physical model and ML non parametric models preferable

What is the prediction error for non seen data ?

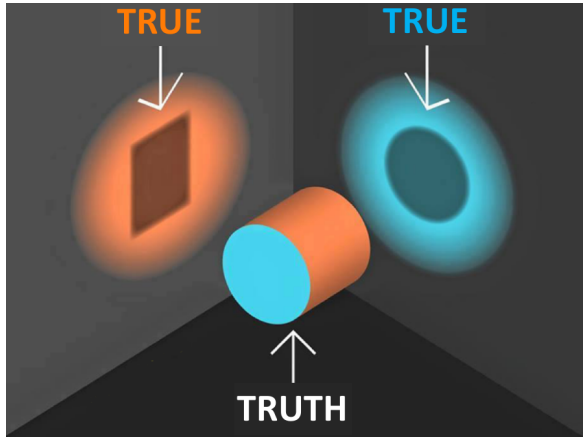
RootInteractive+ N dimensional interactive analysis:

Seeing is believing

RootInteractive + N dimensional analysis - Seeing is believing

https://en.wikipedia.org/wiki/Occam%27s_razor

*“Occam's razor is the problem-solving principle that "entities should not be multiplied without necessity",[1][2] or more simply, **the simplest explanation is usually the right one**.”*



By oversimplifying in analysis level, the explanations tends to be more complex resp. wrong

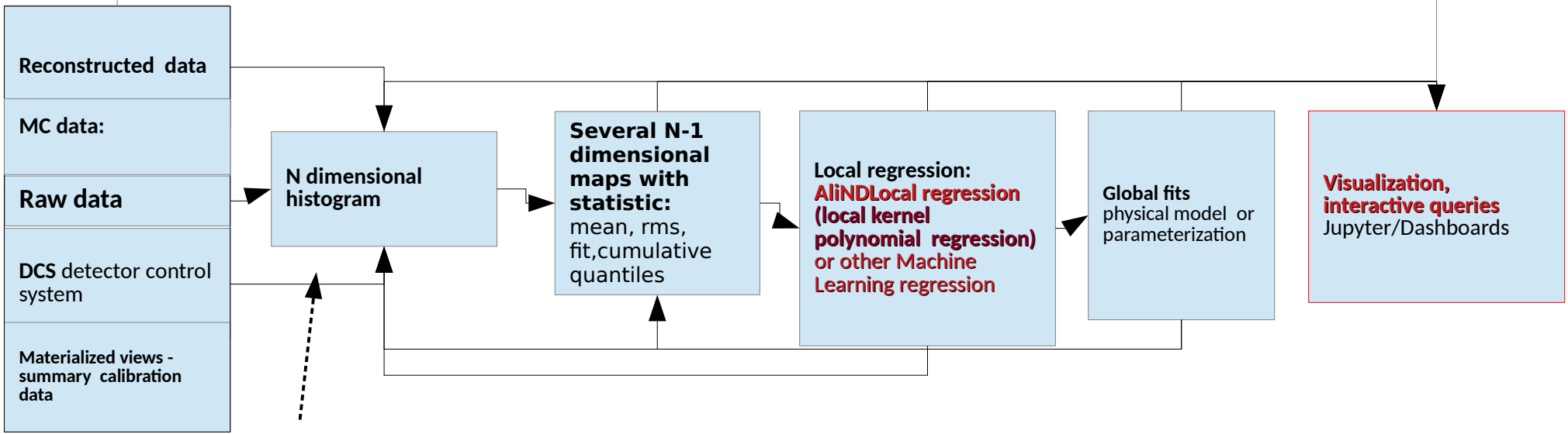
Our goal to provide a tool to deal with ND problems

- simplify data analysis in many (optimally all relevant) dimensions
- fit (ML regression) and visualise N-dimensional functions **including their uncertainties and biases**
- validate assumptions, approximations
- enable simple **functional composition for (non-parametric, parametric) functions and error propagation**
- aimed for **standard users** (Masters, PhD), not just computer experts for educational purposes
- **very fast feedback** from day one (seconds instead of weeks), to allow **interactive expert communication**
- for **multidimensional parameter optimisation** with fast convergence

Multidimensional analysis pipeline & RootInteractive

- ND pipeline - libStat library written in C++98 (AliRoot+ROOT5)
- RootInteractive (Python+TypeScript+C++)
- Usage examples
  - TPC calibration, Tracking performance parameterization, MC/data parameters tuning
  - Detector and reconstruction QA
  - Toy MC, Digital signal processing optimization …

Standard ND pipeline (0) + RootInteractive



$$f(p_0, p_1, p_2, \ldots) \neq f_0(p_0) \oplus f_1(p_1) \oplus f_2(p_2) \oplus \ldots.$$

Standard calibration/performance maps and QA done and interpreted in multidimensional space

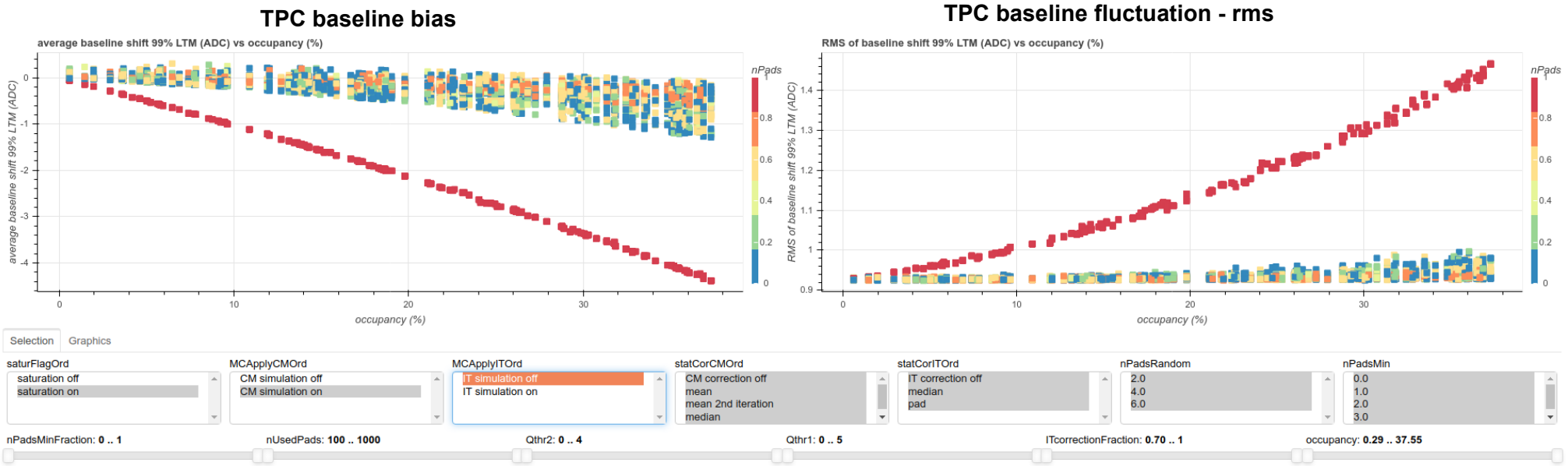
- dimensionality depends on the problem to study (and on available resources)
- **skimmed version of input data usually used in interactive or semi-interactive analysis**
- Data →Histogram → set of ND maps → set of NDlocal regression/TMVA → Global fits (physical model)
- Histogramming in case of non sparse data
- **ML for sparse (going to higher dimensions)**

- **Generic "interactive" code. Minimizing amount of custom macros.**
- **"Declarative" programming - simple queries**
- **Non parametrical and parametrical functions physics models**

N dimensional parameter optimization example - digital signal processing

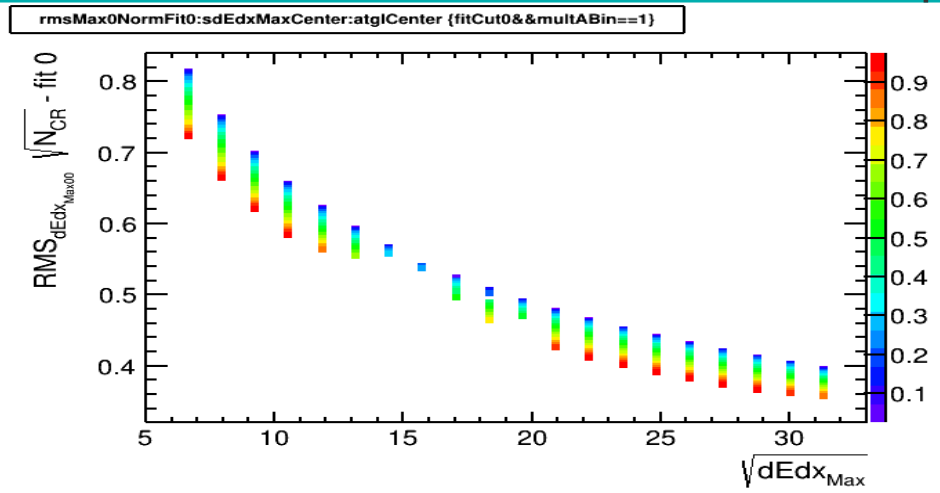
Optimization of the digital signal processing (13 parameters in example) needed for particle identification and data volume optimization O(200000) settings simulated/generated

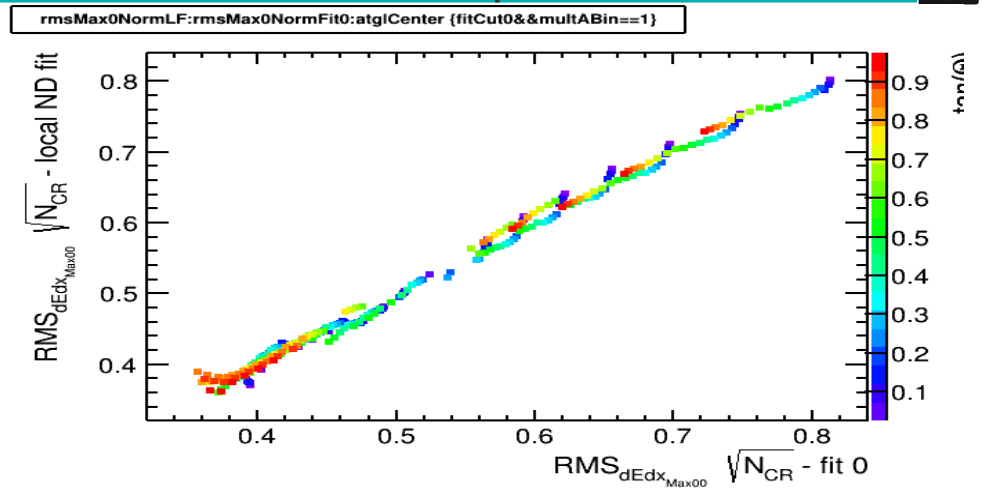
- parameters: effects (On/Off), algorithm (different version), parameters of individual algorithms
- simulation and visualization job done by master student, very effective for education
- enabling very constructive interactive discussion within expert group, quickly converging to “expert” decision, generating new ideas
- standalone dashboards as a support material for internal/public notes



https://gitlab.cern.ch/alice-tpc-offline/alice-tpc-notes/-/blob/master/JIRA/ATO-559/parameterScan.ipynb
https://indico.cern.ch/event/1073883/contributions/4588170/attachments/2334149/3986420/simulScan_02112021.html

TPC dEdx resolution parameterization example





Physical model:

dEdx resolution depends on 3 main variable
dEdx, track length (tan(θ)) and number of measurement (N_{CR})
3 measurement in regions (i,j,k)

$$RMS_{Qi} = \sqrt{RMS^2_{Qi/Qj} + RMS^2_{Qi/Qk} - RMS^2_{Qj/Qk}/2}$$

$$RMS_{ROC} \times \sqrt{N_{CR}} \approx p_0 \left(dEdx^{p_1} \times \sqrt{(1+\tan(\theta))^{2^{p_2}}} \right)$$

Input data pipeline:

skimmed data → 6x4D histograms of dEdx ratios in regions → 6x3D resolution maps (non parameteric) → local fits → global fit of physical model

At low IR agreement between dEdx intrinsic resolution and power low model as expected

Reducible, irreducible error and **P**robability **d**ensity **f**unction

RootInteractive ML wrappers

For data taken from a completely unknown distribution, a CI and errors can be calculated using a bootstrapping method (Efron, 1992; Johnson, 2001).

Bootstrappig CPU consuming

To speed up - to use the internal dispersion of the prediction in ensamble learning methods (random forest, xgboost)

Machine learning based regression algorithm for the non-parametric description of an unknown function:

- N-dimensional calibration, tracking performance parameterization ($\chi 2$, $N_{Clusters}$, σ_{DCA}), conditional PDF distribution

Provides wrappers for the standard ensemble learning method (Random forest, xgboost)

- Local error (reducible, irreducible) parameterization
- Automatic parameters adjustment to minimize reducible error
- Robust local estimator
- Conditional probability density function and quantiles
- Linear Regression Forest - to reduce model error (Work in progress)

For the Neural net, error estimated using dropout prediction

- only prototype, not used yet in real use cases, model dependent

For the RandomForest - error estimated using decision trees RMS, for trees with and without max_depth

- ~irreducible error estimated using RMS of unbound trees
- ~reducible error estimated using RMS of prediction for trees with max_depth limitted

Irreducible local error could be strongly parameter dependent e.g.:

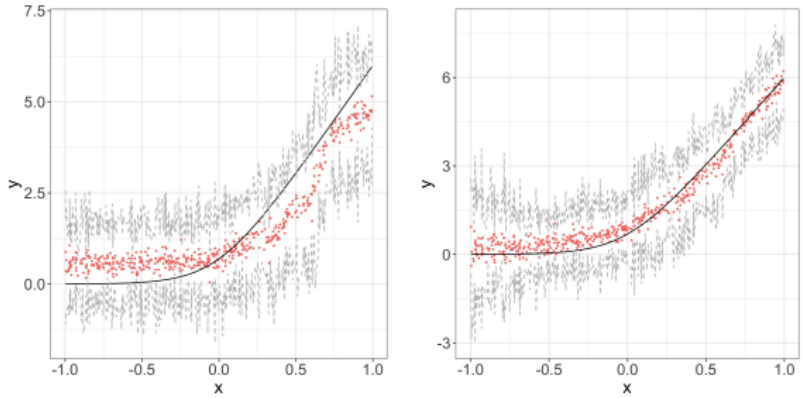
- e.g. bigger relative error of the ion tail for more noisy pads with smaller signal (signal/noise), multiplicity error proportional to sqrt(multiplicity), tracking relative pt resolution ~ (dEdx,L_{arm},)

Reducible error strongly depends on the granularity and on the function derivative and local density of points. Error of the extrapolation explodes.

https://arxiv.org/pdf/1807.11408.pdf

Random forests are a powerful method for non-parametric regression, but are limited in their ability to fit smooth signals. Taking the perspective of random forests as ***an adaptive kernel method,*** *we pair the forest kernel with a local linear regression adjustment* ***to better capture smoothness****. The resulting procedure, local linear forests, enables us to* ***improve on asymptotic rates of convergence for random forests with smooth signals****, and provides substantial gains in accuracy on both real and simulated data.*

https://grf-labs.github.io/grf/articles/llf.html



Random forest Local linear forest

An Adaptive kernel method

where the forest weight $\alpha_i(x_0)$ is the fraction of trees in which an observation appears in the same leaf as the target value of the covariate vector.

$$\alpha_i(x_0) = \frac{1}{B}\sum_{b=1}^{B}\frac{1\{x_i \in L_b(x_0)\}}{|L_b(x_0)|}$$

Local linear forests take this one step further: now, instead of using the weights to fit a local average at x_0, we use them to fit a local linear regression, with a ridge penalty for regularization. This amounts to solving the minimization problem below, with parameters: $\mu(x)$ for the local average, and $\theta(x)$ for the slope of the local line.

$$\begin{pmatrix}\hat{\mu}(x_0)\\ \hat{\theta}(x_0)\end{pmatrix} = \text{argmin}_{\mu,\theta}\left\{\sum_{i=1}^{n}\alpha_i(x_0)(Y_i - \mu(x_0) - (x_i - x_0)\theta(x_0))^2 + \lambda||\theta(x_0)||_2^2\right\}$$

R package integrated within GeneralizedRandomForest (https://grf-labs.github.io/grf/)

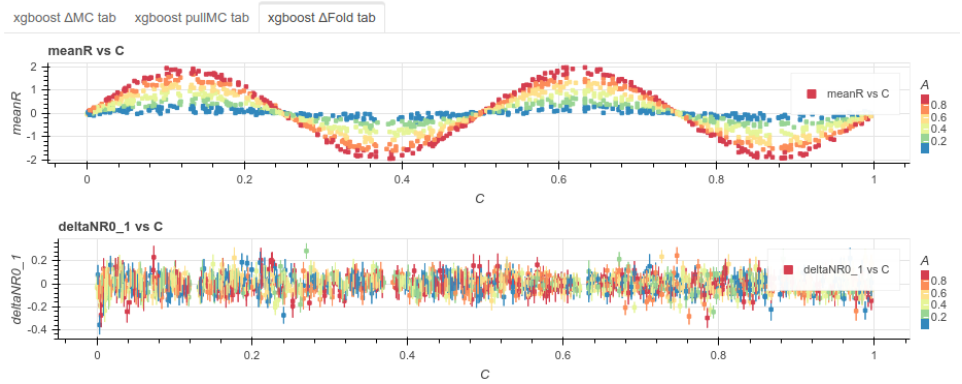
RootInteractive python implementation planned to be ready for this workshop

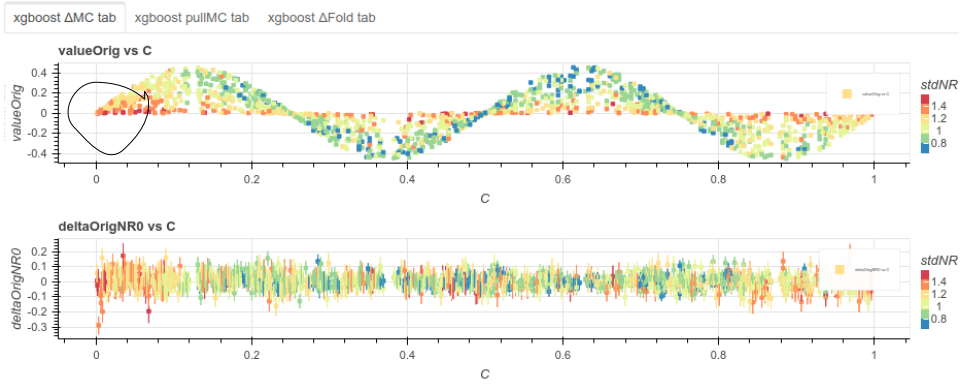
- too slow (similar as in R package)

Cached version with approximation as used in our previous C++ implementation → fir the moment postponed

xgboost wrapper - Toy MCbenchmark use case (0)

$$f(A,B,C,D) = norm*A*\sin(n*2*pi*C) + B*noise$$





4D Uniform input

```
df = pd.DataFrame(np.random.random_sample(size=(nPoints, 4)), columns=list('ABCD'))
df["B"]=df["B"]+0.5
df["noise"] = np.random.normal(0, stdIn, nPoints)
df["noise"] += (np.random.random(nPoints)<outFraction)*np.random.normal(0, 2, nPoints)
```

Local reducible (color code) error increased at the boundaries

Reducible error estimated using spread of the xgboost in iterations after “early_stop”.
Keeping all parameters - reducing subsample and learning_rate

https://indico.cern.ch/event/1147231/contributions/4815612/attachments/2424564/4150687/MIxgboostErrPDF_n2_stdIn0.2_nPoints200000.html
https://indico.cern.ch/event/1147231/contributions/4815612/attachments/2424564/4150688/MIxgboostErrPDF_back11042022.ipynb

xgboost wrapper - benchmark use case (2)

Fold 0 All data

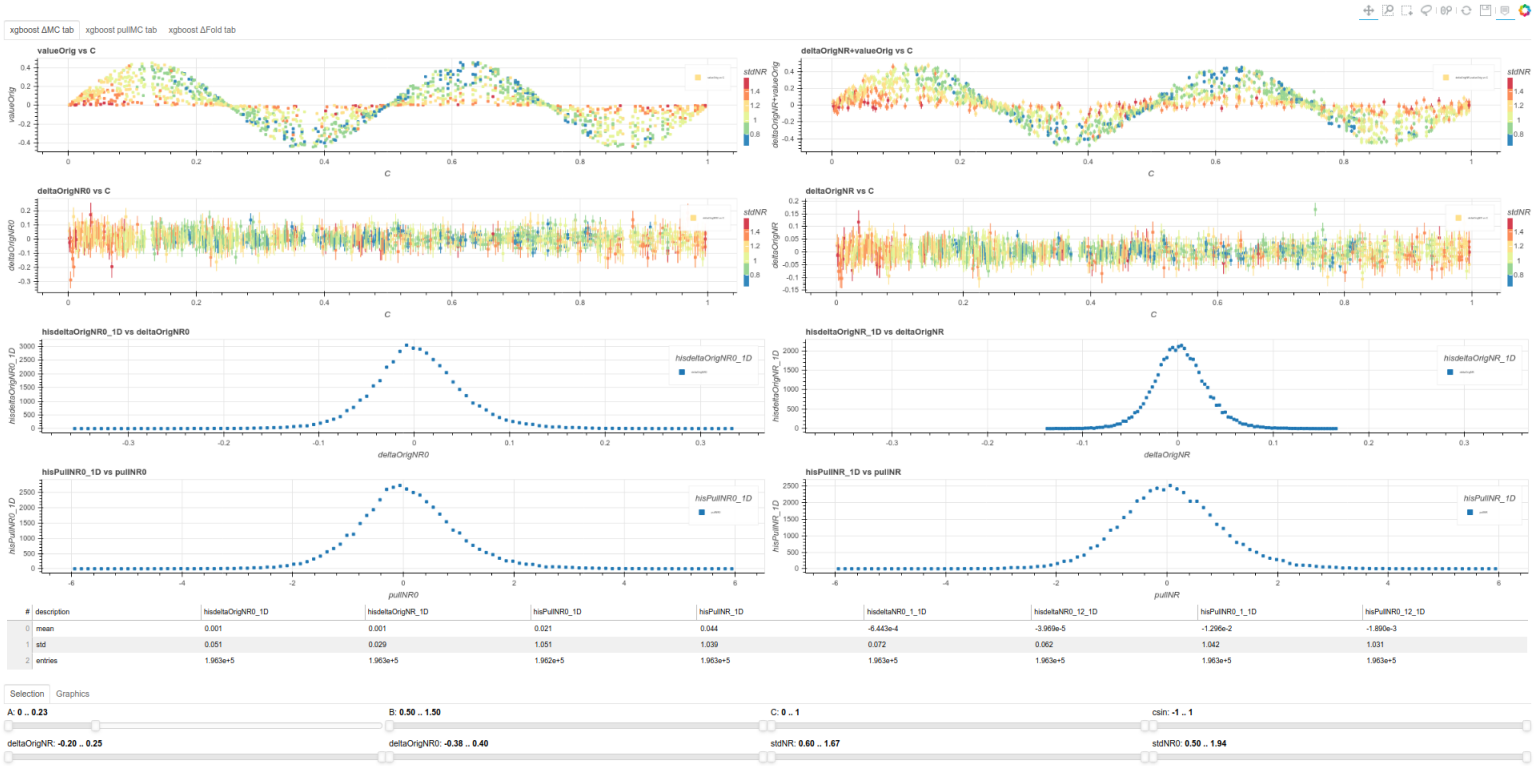
f(A,B,C,D)

Δf

Histogram Δf

Histogram pull f

Histogram stat



#	description	hisdeltaOrigNR0_1D	hisdeltaOrigNR_1D	hisPullNR0_1D	hisPullNR_1D	hisdeltaNR0_1_1D	hisdeltaNR0_12_1D	hisPullNR0_1_1D	hisPullNR0_12_1D
0	mean	0.001	0.001	0.021	0.044	-6.443e-4	-3.969e-5	-1.296e-2	-1.890e-3
1	std	0.051	0.029	1.051	1.039	0.072	0.062	1.042	1.031
2	entries	1.963e+5	1.963e+5	1.962e+5	1.963e+5	1.963e+5	1.963e+5	1.963e+5	1.963e+5

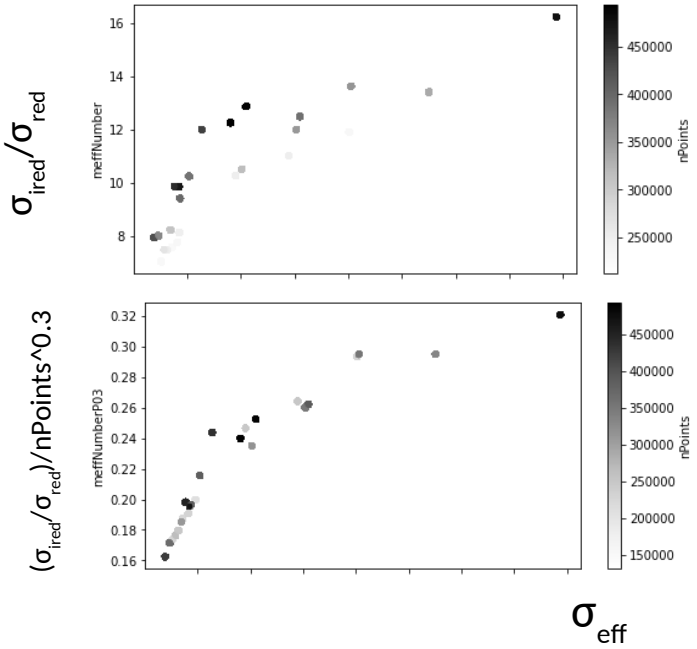
MC tab - Comparison to the MC true

https://indico.cern.ch/event/1147231/contributions/4815612/attachments/2424564/4150687/MIxgboostErrPDF_n2_stdIn0.2_nPoints200000.html
https://indico.cern.ch/event/1147231/contributions/4815612/attachments/2424564/4150688/MIxgboostErrPDF_back11042022.ipynb

$$f(A,B,C,D) = \mathbf{norm}*A*\sin(\mathbf{n}*2*pi*C) + B*\boldsymbol{\sigma}_{noise}$$

$$\sigma_{eff} = \frac{\sigma_{ired}}{norm}$$

$$\sqrt{N_{eff}} = \frac{\sigma_{ired}}{\sigma_{red}}$$



Parameter scan to emulate statistics requirement

- number of points, function normalization, noise (σ_{ired}), n_{Sin}

Making function variation small in respect to intrinsic noise (σ_{ired}), effective number of points increase → reducible error decrease

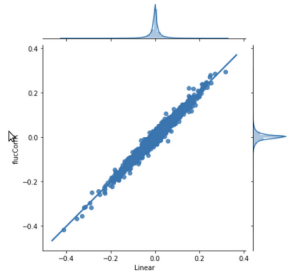
Making regression for delta model (observation - analytical approximation) is preferable

- **Used in many Alice use cases**

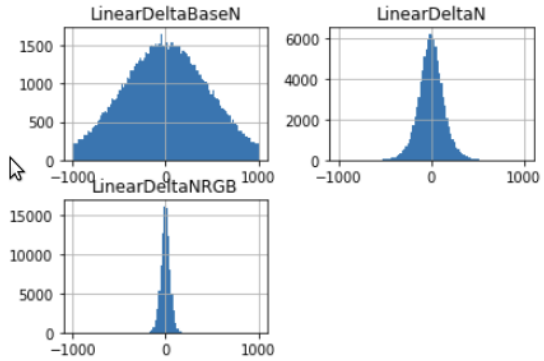
Data driven space charge distortion correction example
1D current fluctuation to 3D distortion fluctuation

Global Linear fit - approximation of the physical model

- Input parameters:
  - local derivative of distortion , current in the TPC (ΔI)
  - ion current as white noise → individual FFT coefficient independent ($\mu=0$, $\sigma_i=\sigma$)
- Output: Δ distortion
- Convolution theorem → approximation response for individual FFT current harmonics
  - convolution in 3D space → multiplication in FFT space
  - Linear fit to approximate convolution kernel
  - 1 FFT as a LinearBase , 20 most important FFT



ΔR at R<95, drift>0.5



flucCorrRN	1264.6
LinearDeltaBaseN	509.1
LinearDeltaN	153.4

Random forest and xgboost used with/without physical model as a prefilter

- Using physics models as prefilter significantly better residual resolution
  - for 10^6 training points ~ 80 microns ~ 40 microns
- Residual distortion after the LinearFit+XGB due 3D current fluctuation not used in the model

Multidimensional analysis pipeline

- non parametric and parametric representation (physics models and effective parameterization)

RootInteractive

- for interactive visualization, functional composition and data aggregation

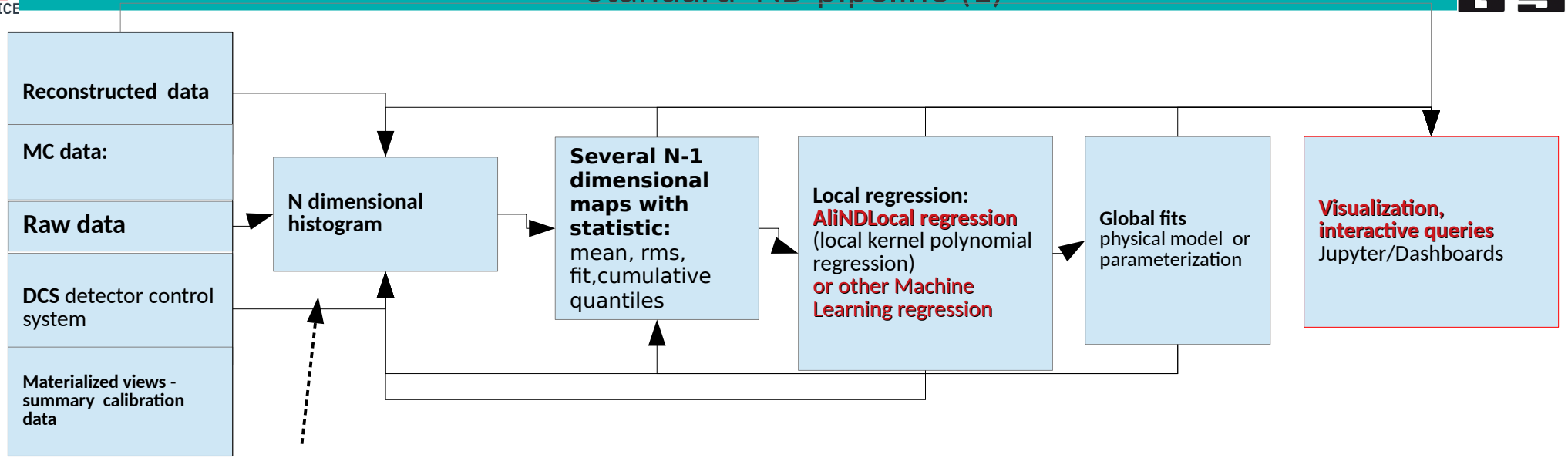
Machine learning uncertainty wrappers, adaptive kernel method

Example use case of data driven space charge calibration - advantage of hybrid (physics & data driven ML) approach

- Minimizing dependence on varying calibration parameters (TPC electron transparency, gain, drift velocity).
- Better precision, significantly smaller statistical data is required for distortion calibration, or more frequent calibration is possible

backup

Standard ND pipeline (1)



Pipeline of standalone C++ tools

- N dimensional histogramming
- Histogram → PDF Map (tree,panda)
  - (default) C++ and (Python) RootInteractive
- Map(Tree) → AliNDLocalRegression (**Local kernel polynomial regression**)
  - parameteric user defined kernel shape, standard error propagation
- Map(Tree) → Global fits (physical models, parameterizations)
  - AliTMinuitToolkit

- **Generic "interactive" code. Minimizing amount of custom macros.**
- **"Declarative" programming - simple queries**
- **Non parametrical and parametrical functions physics models**

xgboost wrapper - benchmark use case (2)

Fold 0

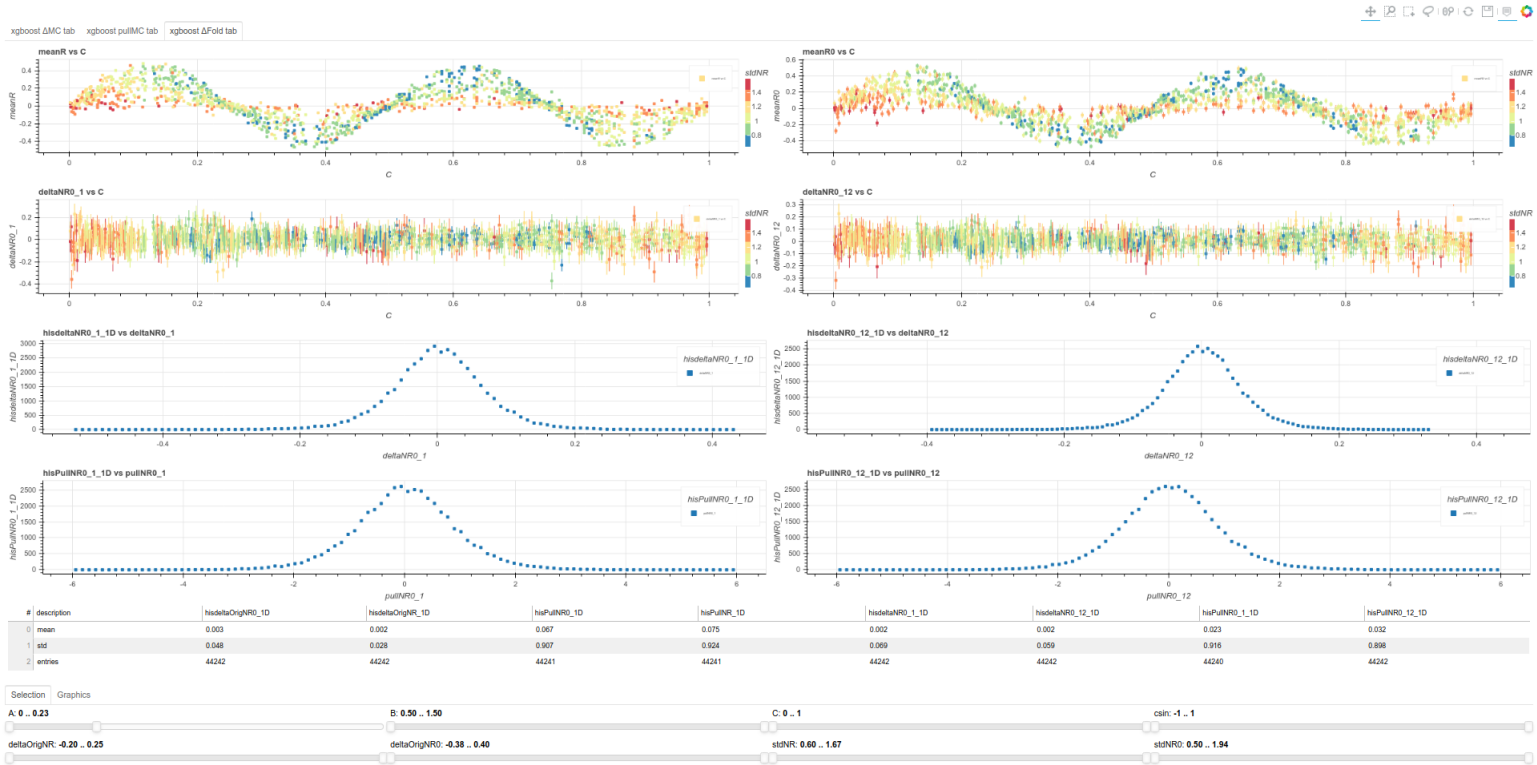
Fold0 to rest

f(A,B,C,D)

Δf

Histogram Δf

Histogram pull f



Comparison of fold predictions

https://indico.cern.ch/event/1147231/contributions/4815612/attachments/2424564/4150687/MIxgboostErrPDF_n2_stdIn0.2_nPoints200000.html
https://indico.cern.ch/event/1147231/contributions/4815612/attachments/2424564/4150688/MIxgboostErrPDF_back11042022.ipynb

```
miErrPDFK0=MIForestErrPDF("Regressor",{"mean_depth":mean_depth, "n_jobs":n_jobs,"niter_max":niter_max })
```

Similar to RandomForest:

- data splitted to 2 individual sets (data ↔ ref. data) batches of trees (DecisionTreeRegressor) for independent data sets
  - niter_max
- max_depth = None

$$\sigma_{\Delta}^{Full} = 0.5 * \sigma_{\Delta}^{fraction} \sqrt{\frac{fraction}{(1 - fraction)}}$$

Local reducible error estimated - std of the prediction using fraction of independent trees

Performance diff - ALICE performance: DCA resolution/bias

http://aliperf0.web.cern.ch/aliperf0/alice/data/2018/LHC18c/kink_3sigma_CENT_pass2/dashboard/LHC16f_lowmult_pass2/fig0/compDefaultV0DCARLHC18c_kink_3sigma_CENT_pass2LHC16f_lowmult_pass2HistComp.html

LHCh18c / LHC16f

Δ norm DCA | Δ DCA pull | σ norm DCA | σ DCA pull

Data

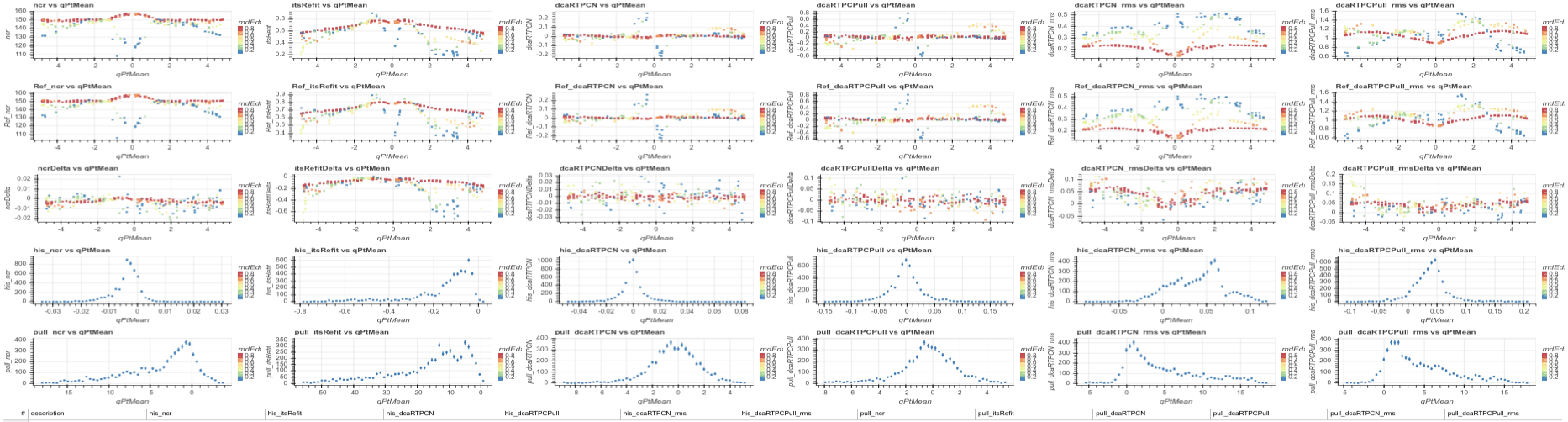
Reference data

Data-Ref.

Histogram: Data-Ref

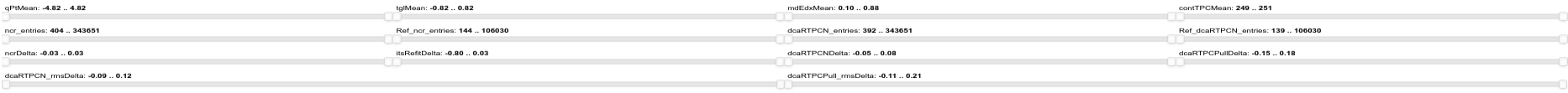
Histogramm: (Data-Ref)/σ

Summary table: Data-Ref



#	description	his_ncr	his_itsRefit	his_dcaRTPCN	his_dcaRTPCPull	his_dcaRTPCN_rms	his_dcaRTPCPull_rms	pull_ncr	pull_itsRefit	pull_dcaRTPCN	pull_dcaRTPCPull	pull_dcaRTPCN_rms	pull_dcaRTPCPull_rms
0	mean	-3.677e-3	-1.498e-1	-5.352e-4	-2.629e-3	0.04	0.038	-3.579e+0	-1.570e+1	-3.248e-1	-3.412e-1	4.618	4.365
1	std	0.005	0.141	0.009	0.031	0.03	0.031	4.085	11.926	1.876	2.035	4.831	4.048
2	entries	4990	4990	4990	4990	4990	4990	4990	4990	4990	4990	4990	4990

Widgets for interactive ND selection



Test data/Data, production/reference production, Period/Period, Data/MC

Production comparison in many dimensions (q/Pt,pz/pt,MIP/dEdx, mult)

Interactive browsing/histograms/aggregation in ND

- Δ and pulls (Δ normalized to error)

Example above used for the low magnetic field (B=0.2T) reconstruction production preparation

Performance diff - ALICE performance MC/data: TPC+ITS QA

http://aliperf0.web.cern.ch/aliperf0/alice/data/2018/LHC18c/pass2_CENT_syst_err/dashboard/LHC21a6_cent_kink5sigma/fig2/compDefaultV2LHC18c_pass2_CENT_syst_errLHC21a6_cent_kink5sigmaHistComp.html

LHC18c / MC LHC21a

TPC ncls | TPC Cross. | TPC cl frac.. | TPC-ITScl refit | TPC chi2 | ITS chi2

Data

Reference data

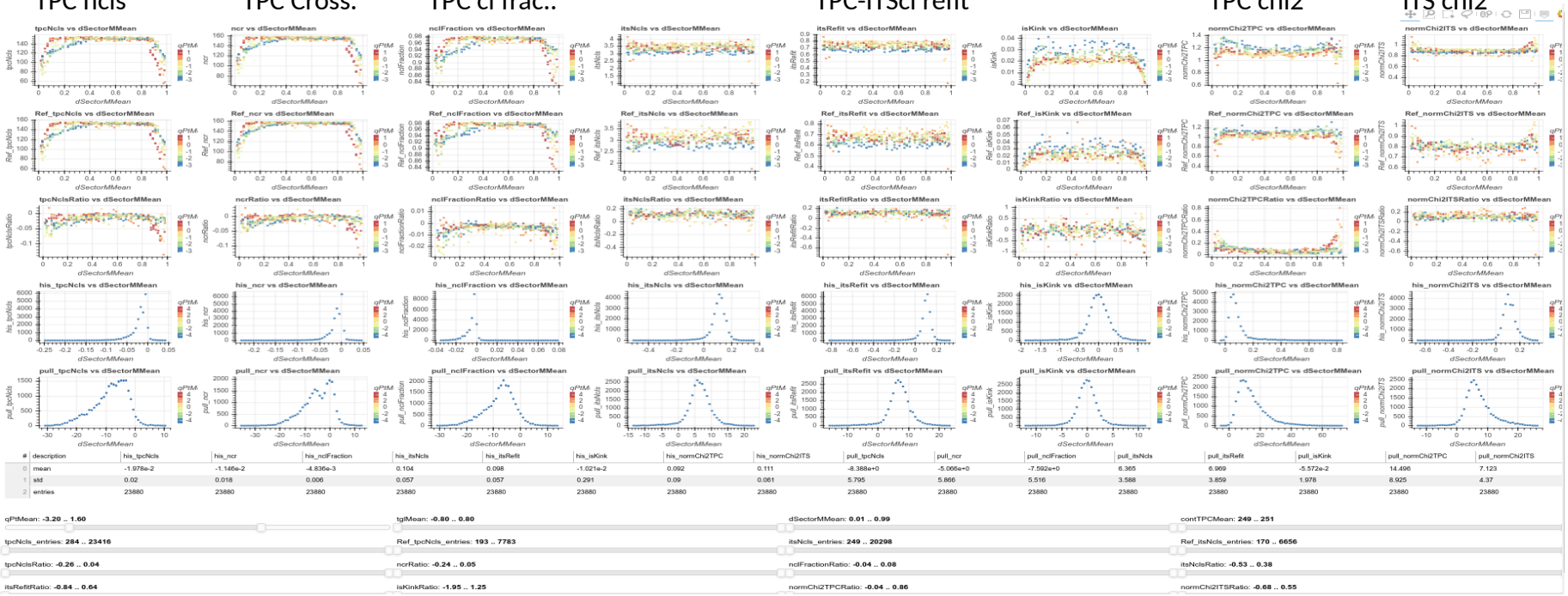
Data-Ref.

Histo: Data-Ref

Histo: (Data-Ref)/σ

Summary table: Data-Ref

Widgets for interactive ND selection



#	description	his_tpcNcls	his_ncr	his_nclFraction	his_itsNcls	his_itsRefit	his_isKink	his_normChi2TPC	his_normChi2ITS	pull_tpcNcls	pull_ncr	pull_nclFraction	pull_itsNcls	pull_itsRefit	pull_isKink	pull_normChi2TPC	pull_normChi2ITS
0	mean	-1.978e-2	-1.145e-2	-4.836e-3	0.104	0.098	-1.021e-2	0.092	0.111	-8.388e+0	-5.066e+0	-7.592e+0	6.365	6.969	-5.572e-2	14.496	7.123
1	std	0.02	0.018	0.006	0.057	0.057	0.291	0.09	0.061	5.795	5.866	5.516	3.588	3.859	1.978	8.925	4.37
2	entries	23880	23880	23880	23880	23880	23880	23880	23880	23880	23880	23880	23880	23880	23880	23880	23880

Test data/Data, production/reference production, Period/Period, Data/MC

Production comparison in many dimensions (q/Pt, pz/pt, sector distance,mult)

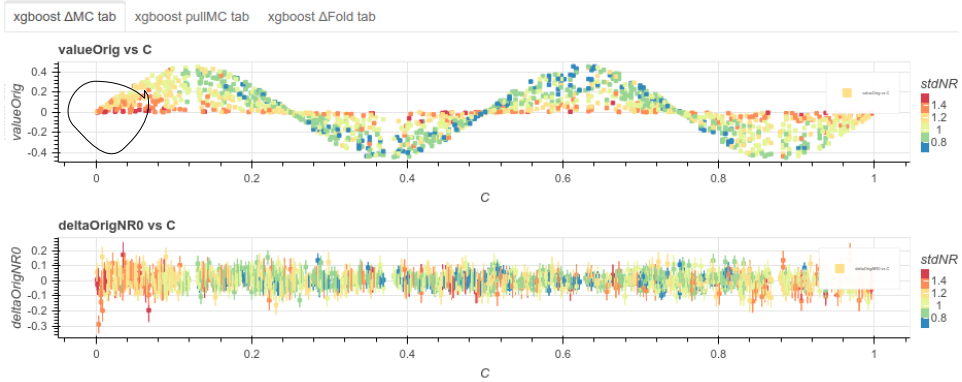
Interactive browsing/histograms/aggregation in ND

- Δ and pulls (Δ normalized to error)

xgboost wrapper - benchmark use case (1)

$$f(A,B,C,D) = 2*A*\sin(n*2*\pi*C) + B*noise$$

```
def fitReducible(self, learning_rate=0.02, subsample=0.05, n_estimators=100) :
    from sklearn.base import clone
    import copy
    for iSample in [0,1,2]:
        # make copy of model and update for error https://stackoverflow.com/questions/62309466/xgboost-how-to-copy-model
        #self.regXGBFacRed[iSample]=clone(self.regXGBFac[iSample])
        self.regXGBFacRed[iSample]=copy.deepcopy(self.regXGBFac[iSample])
        self.regXGBFacRed[iSample].learning_rate=learning_rate
        self.regXGBFacRed[iSample].subsample=subsample
        self.regXGBFacRed[iSample].n_estimators=n_estimators
        self.regXGBFacRed[iSample].fit(self.Xin[iSample], self.Yin[iSample],
xgb_model=self.regXGBFacRed[iSample])
```



4D Uniform input

```
df = pd.DataFrame(np.random.random_sample(size=(nPoints, 4)), columns=list('ABCD'))
df["B"]=df["B"]+0.5
df["noise"] = np.random.normal(0, stdIn, nPoints)
df["noise"] += (np.random.random(nPoints)<outFraction)*np.random.normal(0, 2, nPoints)
```

Local reducible (color code) error increased at the boundaries

Reducible error estimated using spread of the xgboost in iterations after “early_stop”.
Keeping all parameters - reducing subsample and learning_rate

https://indico.cern.ch/event/1147231/contributions/4815612/attachments/2424564/4150687/MIxgboostErrPDF_n2_stdIn0.2_nPoints200000.html
https://indico.cern.ch/event/1147231/contributions/4815612/attachments/2424564/4150688/MIxgboostErrPDF_back11042022.ipynb

V0 invariant mass bias study example
Work in progress

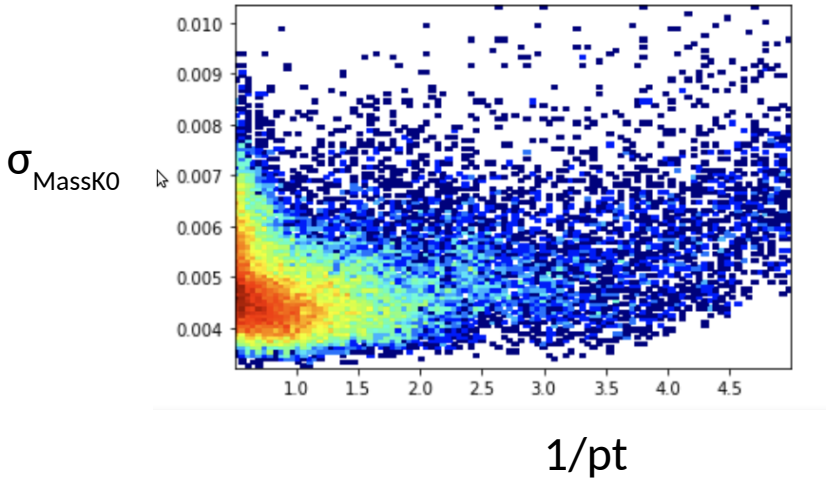
Uncertainty estimated using statistics of individual trees

Local statistics (stat library)

- mean, median, trim_mean
- std, trimmed_std
- quantiles

Interpretation of local statistics settings dependent

- max_depth = None



Signature: predictRFStat(rf, X, statDictionary, n_jobs)
Docstring:
predict statistics from random forest
:param rf: - random forest object
:param X: - input vector
:param statDictionary: - dictionary of statistics to predict
:param n_jobs: - number of parallel jobs for prediction
:return: dictionary with output statisttics
File:
~/github/RootInteractive/RootInteractive/MLpipeline/MIForestErrPDF.py
Type: function

Extract PDF for RF residuals

In [267]:

```
%%time
statDictionary={"mean":0, "std":0, "median":0, "quantile":[0.1,0.5,0.9]}
#dfSample=df.sample(200000).sort_index()
#xxx = predictRFStat(regressorRFF0,dfSample[variablesRF].to_numpy(dtype=np.float32),statDictionary,100)
predictRFF0=predictRFStat(regressorRFF0,df[variablesRF].to_numpy(dtype=np.float32),statDictionary,n_jobs*2)
df["flucCorrR_Mean"]=predictRFF0["mean"]
df["flucCorrR_Median"]=predictRFF0["median"]
df["flucCorrR_std"]=predictRFF0["std"]
df["flucCorrR_F10"]=predictRFF0["quantiles"][0.1]
df["flucCorrR_F90"]=predictRFF0["quantiles"][0.9]
predictRFF1=predictRFStat(regressorRFF1,df[variablesRF].to_numpy(dtype=np.float32),statDictionary,n_jobs*2)
df["LinearDeltaBase_Mean"]=predictRFF1["mean"]
df["LinearDeltaBase_Median"]=predictRFF1["median"]
df["LinearDeltaBase_std"]=predictRFF1["std"]
df["LinearDeltaBase_F10"]=predictRFF1["quantiles"][0.1]
df["LinearDeltaBase_F90"]=predictRFF1["quantiles"][0.9]
predictRFF2=predictRFStat(regressorRFF2,df[variablesRF].to_numpy(dtype=np.float32),statDictionary,n_jobs*2)
df["LinearDelta_Mean"]=predictRFF2["mean"]
df["LinearDelta_Median"]=predictRFF2["median"]
df["LinearDelta_std"]=predictRFF2["std"]
df["LinearDelta_F10"]=predictRFF2["quantiles"][0.1]
df["LinearDelta_F90"]=predictRFF2["quantiles"][0.9]
print(df["flucCorrR_std"].mean(), df["LinearDeltaBase_std"].mean(), df["LinearDelta_std"].mean())
```

```
CPU times: user 5min 1s, sys: 6.57 s, total: 5min 7s
Wall time: 1min 23s
```