Unfolding

Philippe Gras
May 13-17, 24
Université Paris-Saclay, CEA/IRFU

Overview

Unfolding introduction

What is unfolding?

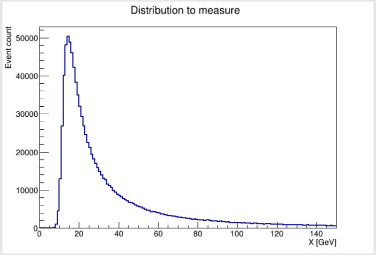
Distribution measured in HEP are distorted by the limited detector acceptance and resolution. We call *unfolding* the inference of the distribution before the distortion.

In disciplines analysing images, it would be called *deconvolution*.

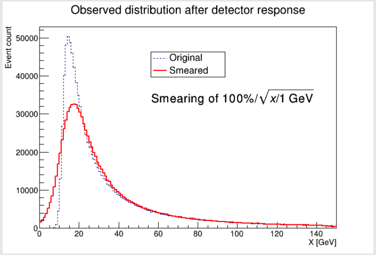
Problem to address

When measuring a distribution or a differential cross section the measured distribution is distorted by the finite detector resolution.

$$\frac{\mathrm{d}\sigma}{\mathrm{d}X^{\text{truth}}} \neq \frac{\mathrm{d}\sigma}{\mathrm{d}X^{\text{reco}}}$$

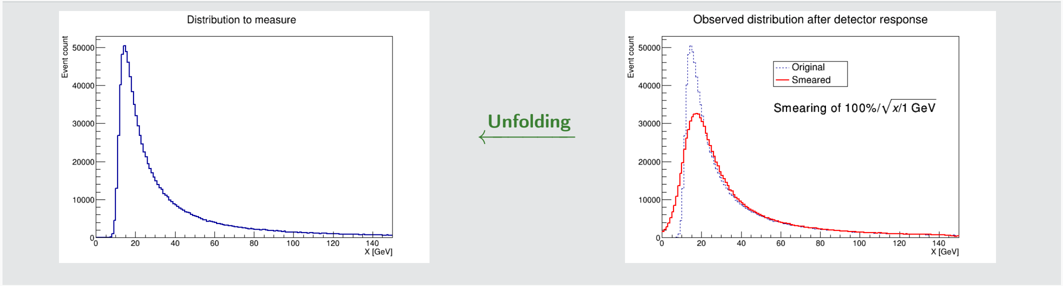


$$\xrightarrow[\text{effect}]{\text{Detector resolution}}$$



Unfolding

- Aims to infer the distribution before the detector distortion from observed event yields.



Other detector effects to correct

Other effects than smearing

- acceptance: limited $|\eta|$ and p_{T} ranges.

  When measurement phase space is matched to the detector acceptance, there is still an effect due to event migration through the boundaries;
- Identification efficiency;
- Background.

The unfolding covers corrections of all above effects.

Cross section measurement in practice

Unfolding is typically used for differential cross-section measurements.

To measure a differential cross section we typically define a histogram and count the number of event in each bin of the histogram.

$$\frac{\mathrm{d}\sigma}{\mathrm{d}X} \rightsquigarrow \frac{\delta_k \sigma}{\delta_k X} = \frac{(N_k^{\text{data}} - N_k^{\text{bkg}}) \cdot \mathcal{C}_k}{\delta_k X \cdot \mathcal{L}}$$

With $\delta_k X$ the bins of a histogram, N_k^{data} the measured event yield in the bin, N^{bkg} the estimated background contribution, $\delta_k \sigma$ the cross section integrated over the bin, $\mathcal{L}$ the integrated luminosity, $\mathcal{C}_k$ the correction for efficiency, bin-to-bin migration, and acceptance.

Note: everything we will address in this lecture applies also to multi-dimension cross sections: X is then a surface, volume or hypervolume depending on the dimension.

Ill-posed problem

- Because of the detector finite resolution, we cannot infer $\mathrm{d}\sigma/\mathrm{d}X$ from the measurement (= measurement of event yields) without regularity assumptions: cannot see variations below the resolution.
  - $\rightarrow$ Needs to add a hypothesis on the regularity of the distribution to infer: Regularization
- Nevertheless, actually measuring $\delta\sigma/\delta X$
  - reduced to a ill-conditioned problem i.e. solutions with large variance or well-conditioned.
  - If bin width, δX, $\approx$ resolution, then regularization is often not needed.
  - The case for many analyses.
  - Regularization is not needed if the condition number of the response matrix is small ($\lesssim 10$).

To unfold or not to unfold

Why unfolding a measurement ?

- Obtain a more fundamental result that does not depend on the apparatus.
- Ease comparison with results from other experiments.
- Ease comparison with other theoretical comparison: no need to simulate the detector response.

Why not unfolding ?

- Unfolding is an ill-posed problem and regularization that it may require can bias the result.
- Unfolding can only reduce the information contents.

Classical methods

Naive approach, aka the bin-by-bin correction

Measure independently the cross section for each bin.

- Count the event yield in the bin;
- Subtract the background contribution;
- Correct for efficiency;
- Divide by the bin size and the integrated luminosity.

Issue with the naive approach

The number of events that migrate from one bin to another depends on the shape of distribution to measure: a steeper distribution will lead to large migrations.

$\Rightarrow$ Measurement biased by the model used in the simulations

Bin-by-bin correction must be reserved in cases where bin-to-bin migration can be neglected (large bins wrt to resolution and/or flat distributions).

A better approach

Extract from the Simulation the probability that an event in a bin i before the detector response (i.e. at generator level) ends up in the bin j after the detector response (i.e. at reconstruction level):

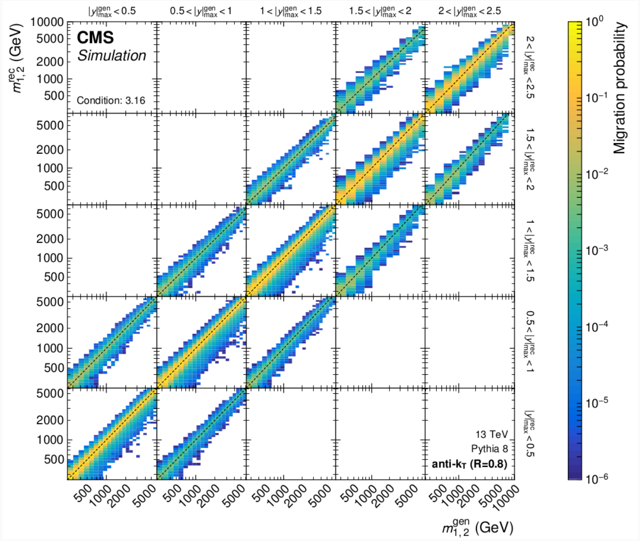
- Fill a 2D histogram of reco vs gen (*migration histogram*)
  - x-axis the generator level (gen) quantity (i.e. before the detector response).
  - y-axis: the reconstruction level (reco) quantity (i.e. after detector response simulation)
- Normalize the histogram such that the sum along the reco axis is equal to one (or to the efficiency) to obtain the probabilities.

This matrix, called **Response Matrix**, is then used to unfold the data. We need to solve,

$$\boldsymbol{N}_{\text{data}} = R\,\boldsymbol{N}_{\text{unfold}} + \boldsymbol{N}_{\text{bkg}}$$

$\boldsymbol{N}$: histograms, i.e. vectors of bin contents

Response matrix



Plot from doi:10.48550/arXiv.2312.16669

$$\mathcal{P}(\mathrm{E}_i|\mathrm{C}_j) \approx \frac{N_{i,j}}{\sum_j N_{i,j}}$$

E_i event is in reco bin i (effect)

C_j event is in gen bin j (cause)

Tip: to unfold a multidimensional distribution, map the bins to a 1-D axis.

Boundaries

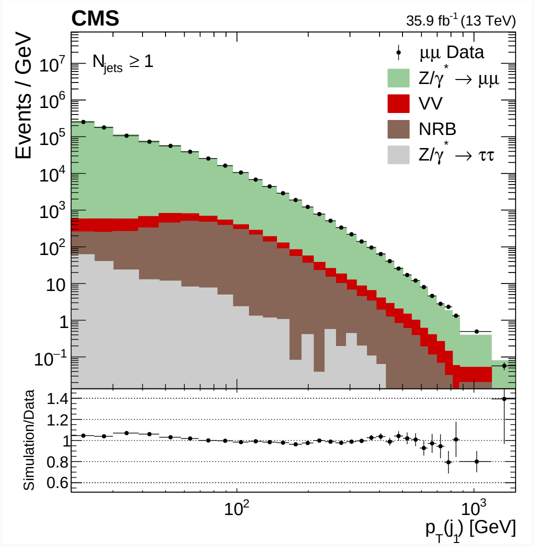
Migration through boundaries

- Events migrating out of boundaries are treated as inefficiencies.
- Event migrating into the boundaries are treated as "fakes".

Steep slope

Recommend adding extra bins beyond boundaries with a steep slope, dropped from the unfolded distribution result.

E.g., in PRD 108 (2023) 05204 two extra bins are added at low value to unfold the distribution on the right.



Unfolding classical methods

Three main methods

- Response matrix (pseudo-)inversion $\equiv$ least-square method
- D'Agostini iterative method: converge to Maximum likelihood estimate (MLE)
- MLE

Least-square	**MLE**
$(R\boldsymbol{x} - \boldsymbol{N}_{\text{data}})^{\mathrm{T}}\Sigma^{-1}(R\boldsymbol{x} - \boldsymbol{N}_{\text{data}})$	$-\sum_j \ln(\text{Poiss}(\boldsymbol{N}_{\text{data},j} \vert [\mathcal{R}\boldsymbol{x}]_j + b_j))$
Gaussian approx.	*Unc. can be profiled*
$\rightarrow$ *Linear algebra*	

$$\boldsymbol{x} \equiv \boldsymbol{N}_{\mathrm{unf}}$$

Least-square method

- Invert the response matrix by minimizing,

$$\chi^2_{\text{unf}} = (R\boldsymbol{x} - N_{\text{data}})^{\text{T}}\Sigma^{-1}(R\boldsymbol{x} - N_{\text{data}}) + \tau^2\chi^2_{\text{reg}}$$

  with Σ the data covariance matrix and $\boldsymbol{x} \equiv \boldsymbol{N}_{\text{unf}}$.
- $\chi^2_{\text{reg}} = (x - f * x_0)^T L^T L(x - f * x_0)$ used to favor regular solutions: Tikhnonov regularization.
- matrix L used to select type of regularization: on the amplitude, the derivative or curvature.

Implemented by TUnfold from S. Schmitt, JINST 7 (2012) T10003, included in ROOT.

D’Agostini iterative method

doi:10.1016/0168-9002(95)00274-X.

Also known as Lucy–Richardson deconvolution (doi:10.1364/JOSA.62.000055, doi:10.1086/111605)

An iterative method using the Bayes theorem ($\rightarrow$ also called D’Agostini Bayes method)

$$\mathcal{P}(\mathrm{C}_i|\mathrm{E}_j) = \frac{\mathcal{P}(\mathrm{E}_j|\mathrm{C}_i)\,\mathcal{P}(\mathrm{C}_i)}{\sum_{l=1}^{n_\text{gen}} \mathcal{P}(\mathrm{E}_j|\mathrm{C}_l)\,\mathcal{P}(\mathrm{C}_l)} \qquad (1)$$

$$\hat{N}_i^\text{gen} = \frac{1}{\epsilon_i} \sum_j \mathcal{P}(\mathrm{C}_i|\mathrm{E}_j) N_j^\text{reco} \qquad (2)$$

$\mathcal{P}(\mathrm{E}_j|\mathrm{C}_i) \equiv R_{ji}$: response matrix

$\epsilon_i = \sum_j \mathcal{P}(\mathrm{E}_j|\mathrm{C}_i)$: reco efficiency.

D'Agostini iterative method

$$\mathcal{P}(\mathrm{C}_i|\mathrm{E}_j) = \frac{\mathcal{P}(\mathrm{E}_j|\mathrm{C}_i)\,\mathcal{P}(\mathrm{C}_i)}{\sum_{l=1}^{n_\mathrm{gen}} \mathcal{P}(\mathrm{E}_j|\mathrm{C}_l)\,\mathcal{P}(\mathrm{C}_l)} \quad (1)$$

$$\hat{N}_i^\mathrm{gen} = \frac{1}{\epsilon_i} \sum_j \mathcal{P}(\mathrm{C}_i|\mathrm{E}_j) N_j^\mathrm{reco} \quad (2)$$

1. Start with some priors $\mathcal{P}(\mathrm{C}_i) = \mathcal{P}_0(\mathrm{C}_i)$: distribution from MC, flat prior, or some other choice;
2. Compute $\hat{\mathcal{P}}(\mathrm{C}_i|\mathrm{E}_j)$ using eq. (1) with the priors;
3. Estimate $\hat{\boldsymbol{N}}^\mathrm{gen}$ by injecting step-2 $\hat{\mathcal{P}}(\mathrm{C}_i|\mathrm{E}_j)$ in eq. (2);
4. Estimate new priors $\mathcal{P}_1(\mathrm{C}_i) = \hat{N}_i^\mathrm{gen} / \sum_k \hat{N}_k^\mathrm{gen}$ and repeat from step 2.

D'Agostini iterative method

Properties

- Converges to the MLE, although convergence can be slow in some cases.
- It runs fast.
- Regularization is obtained by stopping the iterations before convergence.
- N_i^{unf} can be written as a linear combination of N_j^{reco}, $\boldsymbol{N}^{\mathrm{unf}} = U \cdot \boldsymbol{N}^{\mathrm{reco}}$.

Maximum likelihood

Principle

- Maximize a likelihood, e.g. using Minuit.
- If the measurement already uses a likelihood to extract reco-level event yields, use a single likelihood.
- See doi:10.1007/JHEP03(2021)003 measurement that uses this approach.

Pros

- Simultaneous fit of signal, background and unfolding;
- Profiling of systematics;
- Poisson statistics.

Cons

- Slow compared to the other methods that use linear algebra.
- Number of bin limit due to both computation time and fit stability: ok up to $\mathcal{O}(100)$. Use of ML fitting as in doi:10.1103/PhysRevD.102.092012 may leverage this limitation.

Regularization

Three regularization methods encountered in LHC data analyses

- Tikhonov regularization we saw before (Tikhonov, Soviet Math Dokl 4, 1035-1038). Can be used with both χ^2 and MLE methods.
- Early stopping in the D'Agostini iterative method
- SVD: smooth rejection of the smallest single values (doi:10.1016/0168-9002(95)01478-0)

Use regularization only when needed

- Regularization introduce a bias and must be avoided when not needed;
- With a choice of $\delta X \approx$ resolution, regularization is often not needed.

Choice of regularization strength

- To minimize bias it is important to make an objective selection of the regularization strength.
- Many methods on the market.
- Most used methods in LHC data analyses:
    - L-curve scan;
    - Minimization of global correlation;
    - Minimization of unfolding mean square error (MSE) using simulation;
    - Minimization of error on reunfolded data.

Regularization strength choice: L-curve

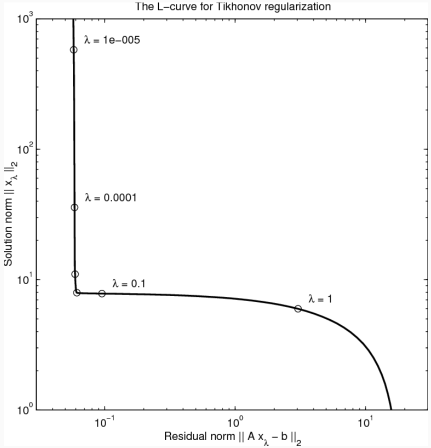
L-curve

- Applies to minimization using Tikhonov regularization.
- Goal: find a compromise between fit residual minimization and solution regularity. P. C. Hansen 2000, WIT Press

Method

- Draw the curve $\log \chi^2_{\text{unf.}}$ vs $\log \frac{\chi^2_{\text{reg.},\tau}}{\tau^2}$, with τ as parameter.
- Select the τ value of the point with the maximum curvature.

Specific to Tikhonov regularization. *Implemented in TUnfold*



from P. C. Hansen 2000

Regularization strength choice: Global correlation

Principle

Minimize the correlation between bins of the unfolded histogram.

Implementation

- Scan the regularization strength values and select the value that minimizes the global correlation of the bins, $\rho_i = \sqrt{1 - \frac{1}{(\Sigma_{ii} * \Sigma^{-1}_{ii})}}$
- Two options: use the mean or max of ρ_i. If mean is used, check that the max has a reasonable value.

Implemented in TUnfold

Regularization strength choice: MSE

Method recommended by the RooUnfold manual

Principle

- Minimize the error: difference between truth and estimation, including both bias and variance. Used the mean squared error (MSE), with average done over the bins.
- Use the simulation for which truth is known.

Implementation

- Make replicas of the simulation reco histogram using a Poisson law for the bin content.
- Unfold each replica and compute the MSE with respect to the simulation gen histogram.
- Average MSE over the replicas.
- Select the regularization strength that minimizes the averaged MSE.
- Check that the error is small enough in every bin and uncertainty coverage is sufficient.

Regularization strength choice: MSE

Limitations

- Based on the simulation.
- If the shape of the truth distribution differs from the model used in the simulation, then the unfolding can behave differently.
- Especially, it seems important to use a flat prior for the D'Agostini iterative method and a flat bias for least square.
- Limitation can be alleviated by testing with different event generators, reweight the simulation to match with the observed reco histogram, or by distorting the distribution used for the test.

Regularization strength choice: reunfold

used e.g. in doi:10.1140/epjc/s10052-018-6373-0

Variation of MSE using data as template.

- Draw N replicas of the unfolded distribution using a Poisson law for the bin contents;
- "Fold" each replica by applying the response matrix and resample it.
- Unfold the N folded replicas and for each of them compute $T = \sum_i (N_i^{\text{unf}} - N_i^{\text{gen}})^2 / N_i^{\text{gen}}$.
- With a D'Agostini iterative unfolding, T will typically decrease in absolute value with the number of iterations (approaching to the solution) and then increases (because of the fluctuations added by the unfolding). Select the minimum as working point.
- Check that the $(N_i^{\text{unf}} - N_i^{\text{gen}})/N_i^{\text{gen}}$ is small enough in every bin and uncertainty coverage is sufficient.

Cross-checks

It is essential to check the unfolding process in the simulation.

Cross-checks: closure tests

Closure test I

- Use MC to generate pseudo-data ($\rightarrow$ gen- and reco- level distributions) and response matrix;
- Unfold the reco-level distribution
- Check that the unfolded distribution matches with the gen-level distribution

Closure test II: sensitivity to the MC model

- Same as test I but using a different event generator for the MC sample used to extract the response matrix

Cross-checks: bottom-line test

Bottom-line test

- The bottom line: unfolding should not enhance the measurement discrimination power between two models.
- The test:
  - Pick up a model for the true distribution $\rightarrow \lambda_{\text{gen}}$;
  - Smear the model to obtain the reconstruction level distribution $\rightarrow \lambda_{\text{reco}} = R\lambda_{\text{gen}}$
  - Compare the p-value of the χ^2-tests of backgound-subtracted data vs λ_{reco} and of unfolded data vs λ_{gen}: the p-values must be similar and the one in the unfolded space should not be smaller than the one in the reco space.
- Beware the test is not valid in case of large regularization because the ndof for the unfold-space test is no more equal to the number of bins. `http://arxiv.org/pdf/1408.6500` provides a method to estimate ndof in such a case.

Cross-checks: coverage test

coverage test

If the result is biased, then the uncertainty coverage will be too small.

$$\text{coverage} = \Phi(\frac{\text{bias}}{\sigma} + 1) - \Phi(\frac{\text{bias}}{\sigma} + 1)$$

with Φ, the normal cumulative distribution function.

- Coverage can be checked using toy experiments.

Uncertainties

Correlation

- Because an unfolded bin content is a linear combination of several measured bin contents, measurement uncertainties which are not correlated between the bins at reco level leads to **correlated uncertainties** on the unfolded histogram.
- It is important to estimate and provide the full covariance matrix.

Reco-level measurement uncertainties

- Data statistical uncertainties are uncorrelated between bins at reco level and must be propagated analytically or by toy MC to the unfolded distribution.
- Two ways to propagate measurement systematic uncertainties:
    - Estimate the uncertainties on the reco-level data histogram and propagate them similarly to the data statistical uncertainties.
    - Estimate the uncertainties on the response matrix elements (vary the uncertainty source by $\pm 1\,\sigma$) and propagate them to unfolded histogram.
- The second option that uses simulation is more flexible and can be used for all types of systematics. In particular, resolution can be varied in both direction in the simulation, where it can only be worsened in data.

Unfolding statistical uncertainties

- The response matrix is build from a finite size simulation sample and each element has a statistical uncertainties.
- This statistical uncertainty must be propagated to unfolded histogram analytically or using toy MC.
- **Note:** it is often considered as a systematic uncertainty of the measurement, because it does not depend on the measurement data sample size.

Unfolding model uncertainties

Limitations of the response matrix approach

- Sensitive to the modelling of the distribution within the bins;
- Dependency of event migration on other observables than the unfolded one(s) ignored
  $\rightarrow$ e.g. unfolding of a p_T distribution sensitive to MC η distribution accuracy

Unfolding model uncertainties

Because of this limitation the result depends on the accuracy of the event generator, and we should account for model uncertainties.

Model uncertainties

Different methods used in LHC data analyses, based on computing alternative response matrices from:

- Gen. parameter variations (using weights produced by generators): energy scales (renormalization, factorization, parton showering), PDF, α_S variations
  More variations can be included. E.g., for analyses with top quarks, colour reconnection, top mass, B-fragm., h_{damp}.
- One (or more) alternative generator(s)
  $\rightarrow$ used to derive an unc. or as cross check.
- Reweighted MC: variation based on the difference of data/MC reco distributions

In all methods, it is important to check at reco-level that the variations cover differences between data and simulation (by construction for the last one).

None of the methods is perfect. Ideally we would like to minimize this uncertainty to limit the sensitivity of the measurement to it.

Software libraries for classical-method unfolding

Software libraries: TUnfold

Perfect tool for the least-square method. Very complete and well documented; including regularization, methods to choose the regularization strength, mapping of multi-D to 1-D, support variable bin widths, error propagation and covariance matrix evaluation, etc.

Included in ROOT.

https://root.cern/doc/master/group__Unfold.html
doi:10.1088/1748-0221/7/10/T10003

Software libraries: TUnfold

Usage example

Inputs

```
//Migration histogram (filled with simulation)
TH2D resp;

//Data histogram
TH1D hdata;

//Background reco histogram
TH1D hbkg;
```

Efficiency

In order to correct for efficiency with TUnfold, put the missed events in the overflow bin of the relevant gen row.

Software libraries: TUnfold

Run unfolding

```
auto tunfold = TUnfoldDensity(resp, TUnfold::kHistMapOutputVert);
tunfold.SubtractBackground(hbkg, "backgound");
auto tau = 0.; //no regularization
tunfold.DoUnfold(tau, hdata)
TH1* hunfold = tunfold.GetOutput("hunfolded");
```

See also the ROOT reference manual and dedicated tutorials.

Software libraries: TUnfold L-curve scan

```
auto nScan = 20;
auto tauMin = 0.;
auto tauMax = 0.; //tauMin = tauMax = 0 => auto select
TGraph* lCurve;
TSpline *logTauX;
TSpline *logTauY;

auto iBest = tunfold.ScanLcurve(nScan, tauMin, tauMax, &lCurve,
                                &logTauX, &logTauY);

Double_t tau,x,y;
logTauX->GetKnot(iBest,&tau,&x);
logTauY->GetKnot(iBest,&tau,&y);

//output contains the result unfolded with the best tau
TH1* hunfold = tunfold.GetOutput("hunfolded");
```

Software libraries: RooUnfold

Provides a unified interface to several methods and includes methods not provided in other common software, like the D'Agostini iterative method. For χ^2 and SVD, it acts as an interface to TUnfold and TSVDUnfold. Fully integrated with ROOT, although shipped as a separate package.

Support error propagation, performed with a toy MC. Beware that covariance matrix elements far from the diagonal can require a very large number of toy experiments.

Note: Current version supports custom priors for the D'Agostini iterative unfolding instead of the default MC distribution.

https://gitlab.cern.ch/RooUnfold/RooUnfold
doi:https://doi.org/10.48550/arXiv.1910.14654

Software libraries: RooUnfold

Usage example

Inputs

```
//Migration matrix. Beware RooUnfold expects
 reco on x-axis
TH2D hresp;

//MC reco histogram
TH1D hreco;

//MC gen histogram
TH1D hgen;

//Data histogram
TH1D hdata;

//Background reco histogram
TH1D hbkg;
```

Efficiency

Efficiency is obtained by comparing hgen and hresp: to correct for efficiency, when an event is dropped by the reconstruction, fill hgen but not resp.

Software libraries: RooUnfold

Unfolding

```
auto rooResp = new RooUnfoldResponse(hreco, hgen, hresp, "hresp",
                                     "Response␣matrix");
auto niters = 10; //to be adjusted after regularization studies
auto unfold_bayes = RooUnfoldBayes(rooResp, hdata, niters);

//Perform the unfolding
auto hunfold_bayes = unfold_bayes.Hunfold();
```

Software libraries: Combine

doi:10.48550/arXiv.2404.06614

Combine: RooStat/RooFit-based profile-likelihood framework originally developed for Higgs boson search and measurements.

- A tutorial of MLE unfolding using Combine can be found here.
- Other profile likelihood frameworks can be used for MLE unfolding.
- Pay attention to remove the theory uncertainty that affects the signal process rates from the nuisance parameters.

Machine learning based unfolding

A rich literature

- OmniFold: A Method to Simultaneously Unfold All Observables
- Unfolding with Generative Adversarial Networks
- How to GAN away Detector Effects
- Machine learning approach to inverse problem and unfolding procedure
- Machine learning as an instrument for data unfolding
- Advanced event reweighting using multivariate analysis
- Unfolding by weighting Monte Carlo events
- Binning-Free Unfolding Based on Monte Carlo Migration
- Invertible Networks or Partons to Detector and Back Again
- Neural Empirical Bayes: Source Distribution Estimation and its Applications to Simulation-Based Inference
- Foundations of a Fast, Data-Driven, Machine-Learned Simulator
- Comparison of Machine Learning Approach to other Unfolding Methods
- Scaffolding Simulations with Deep Learning for High-dimensional Deconvolution
- Preserving New Physics while Simultaneously Unfolding All Observables
- Measurement of lepton-jet correlation in deep-inelastic scattering with the H1 detector using machine learning for unfolding
- Presenting Unbinned Differential Cross Section Results
- Feed-forward neural network unfolding
- Optimizing Observables with Machine Learning for Better Unfolding
- Unbinned profiled unfolding

Two approaches

- Iterative unfolding (Omnifold)
- Generative unfolding

List from the HEPML Living Review

Omnifold

Principle

Exploit the following properties of binary classifiers: for two probability distributions of events, it approximates the likelihood ratio.

E.g. with a NN $f(x)$ trained with a cross-entropy loss function,

$$\mathrm{loss}(f(x)) = -\sum_{i\in\mathrm{Cat.}0}\log f(x_i) - \sum_{i\in\mathrm{Cat.}1}\log(1-f(x_i))$$

we have[1],

$$\frac{f(x)}{1-f(x)} \approx \frac{p_0(x)}{p_1(x)}$$

with p_i the probability to be in Category i.

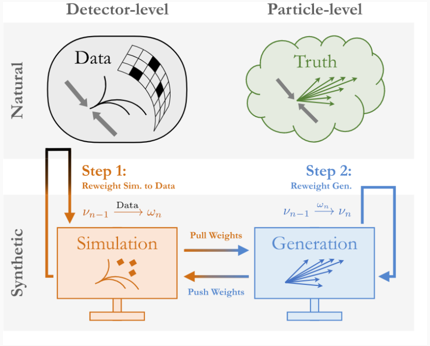
[1] assuming the same number of events in both categories for the training

Ommifold

Generalizes the iterative D'Agostini method to unbinned unfolding of the full phase space

1. Train a classifier to distinguish data from simulation events
   $\Rightarrow$ Probability an event is from data,
   $\mathcal{P}(\text{Data}|x_{\text{det}}) = 1 - \mathcal{P}(\text{Simu}|x_{\text{det}})$
2. Reweight Simulated event with $\mathcal{P}(\text{Data}|x_{\text{det}})/\mathcal{P}(\text{Simu}|x_{\text{det}})$
3. Train a second classifier to distinguish at gen. level original from reweighted events
   $\Rightarrow$ Probability an event is from reweighted simulation,
   $\mathcal{P}(\text{Reweighted}|x_{\text{part}}) = 1 - \mathcal{P}(\text{Original}|x_{\text{part}})$
4. Reweight simulation with
   $\mathcal{P}(\text{Reweighted}|x_{\text{part}})/\mathcal{P}(\text{Original}|x_{\text{part}})$
5. Repeat from 1

PRL 124 182001 (2020), PRD 104 076027 (2021)

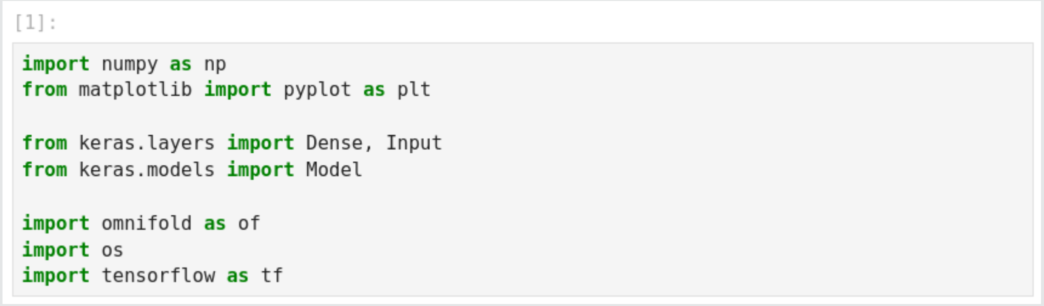


Can also be used on a limited number of observables: called Unifold for 1 observables and Multifold for more.

Omnifold example: step 1

Example from https://github.com/hep-lbdl/OmniFold (GaussianExample_minimal)

Modules to load



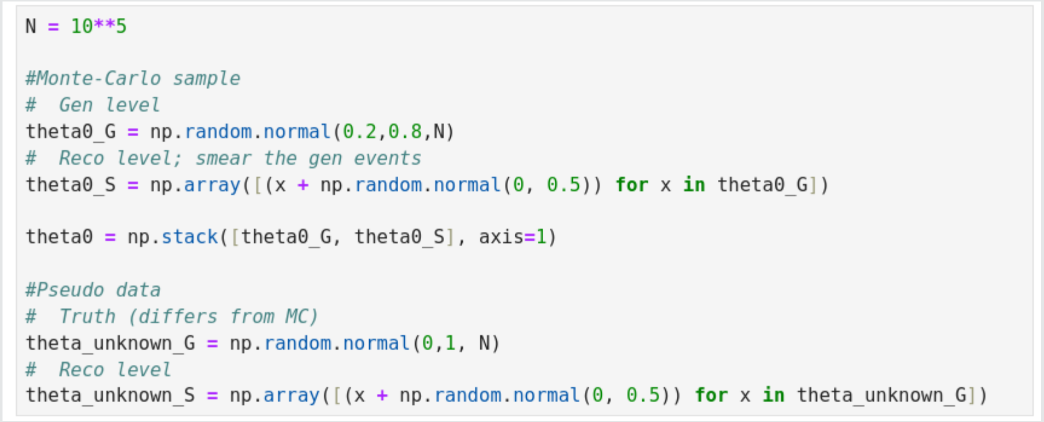
[1]:

```
import numpy as np
from matplotlib import pyplot as plt

from keras.layers import Dense, Input
from keras.models import Model

import omnifold as of
import os
import tensorflow as tf
```

Omnifold example: step 2 (1/2)

Example from https://github.com/hep-lbdl/OmniFold (GaussianExample_minimal)

Produce the mockup samples

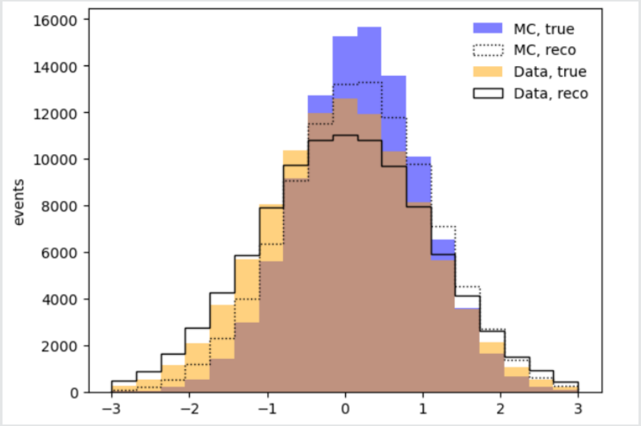


```
N = 10**5

#Monte-Carlo sample
#  Gen level
theta0_G = np.random.normal(0.2,0.8,N)
#  Reco level; smear the gen events
theta0_S = np.array([(x + np.random.normal(0, 0.5)) for x in theta0_G])

theta0 = np.stack([theta0_G, theta0_S], axis=1)

#Pseudo data
#  Truth (differs from MC)
theta_unknown_G = np.random.normal(0,1, N)
#  Reco level
theta_unknown_S = np.array([(x + np.random.normal(0, 0.5)) for x in theta_unknown_G])
```

Omnifold example: step 2 (2/2)

Example from https://github.com/hep-lbdl/OmniFold (GaussianExample_minimal)

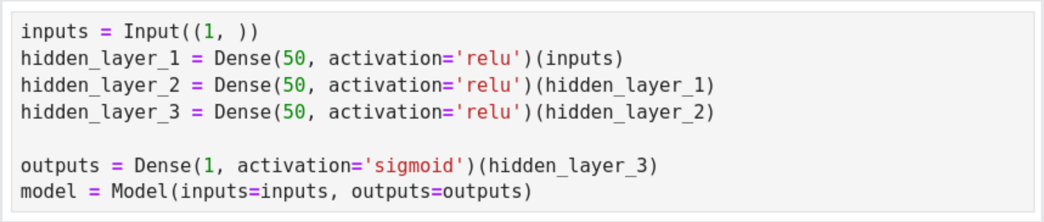
The input distributions



Omnifold example: step 3

Example from https://github.com/hep-lbdl/OmniFold (GaussianExample_minimal)

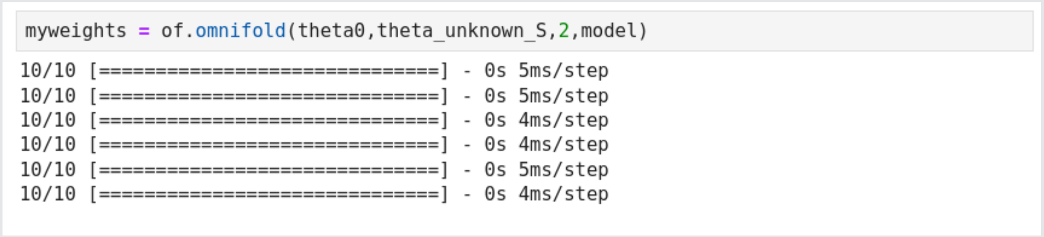
Define the neural network



```
inputs = Input((1, ))
hidden_layer_1 = Dense(50, activation='relu')(inputs)
hidden_layer_2 = Dense(50, activation='relu')(hidden_layer_1)
hidden_layer_3 = Dense(50, activation='relu')(hidden_layer_2)

outputs = Dense(1, activation='sigmoid')(hidden_layer_3)
model = Model(inputs=inputs, outputs=outputs)
```

Omnifold example: step 4

Perform the unfolding

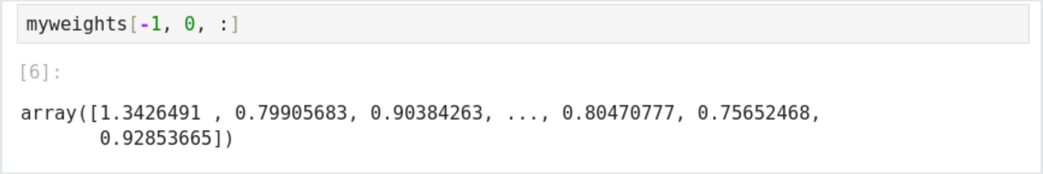


```
myweights = of.omnifold(theta0,theta_unknown_S,2,model)
```

```
10/10 [==============================] - 0s 5ms/step
10/10 [==============================] - 0s 5ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 5ms/step
10/10 [==============================] - 0s 4ms/step
```

Omnifold example: result (1/2)

Ominfold produced weights to correct the MC to infer the unfolded distribution

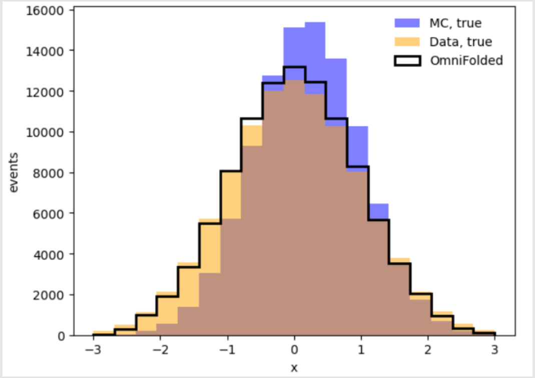


```
myweights[-1, 0, :]
```

```
[6]:

array([1.3426491 , 0.79905683, 0.90384263, ..., 0.80470777, 0.75652468,
       0.92853665])
```

Omnifold example: result (2/2)

Result compared with inputs



What does of.omnifold() do? (1/3)

```
def omnifold(theta0,theta_unknown_S,iterations,model,verbose=0):
  weights = np.empty(shape=(iterations, 2, len(theta0)))
  # shape = (iteration, step, event)

  theta0_G = theta0[:,0]
  theta0_S = theta0[:,1]

  labels0 = np.zeros(len(theta0))
  labels_unknown = np.ones(len(theta_unknown_S))
  labels_unknown_step2 = np.ones(len(theta0_G))

  xvals_1 = np.concatenate((theta0_S, theta_unknown_S))
  yvals_1 = np.concatenate((labels0, labels_unknown))

  xvals_2 = np.concatenate((theta0_G, theta0_G))
  yvals_2 = np.concatenate((labels0, labels_unknown_step2))

  # initial iterative weights are ones
  weights_pull = np.ones(len(theta0_S))
  weights_push = np.ones(len(theta0_S))
```

What does `of.omnifold()` do? (2/3)

```
for i in range(iterations):

  # STEP 1: classify Sim. (which is reweighted by weights_push) to Data
  # weights reweighted Sim. --> Data

  weights_1 = np.concatenate((weights_push, np.ones(len(theta_unknown_S))))

  X_train_1, X_test_1, Y_train_1, Y_test_1, w_train_1, w_test_1 \
    = train_test_split(xvals_1, yvals_1, weights_1)

  # zip ("hide") the weights with the labels
  Y_train_1 = np.stack((Y_train_1, w_train_1), axis=1)
  Y_test_1 = np.stack((Y_test_1, w_test_1), axis=1)

  model.compile(loss=weighted_binary_crossentropy,
  optimizer='Adam',
  metrics=['accuracy'])

  model.fit(X_train_1, Y_train_1, epochs=20, batch_size=10000,
            validation_data=(X_test_1, Y_test_1), verbose=verbose)

  weights_pull = weights_push * reweight(theta0_S,model)
  weights[i, :1, :] = weights_pull
```

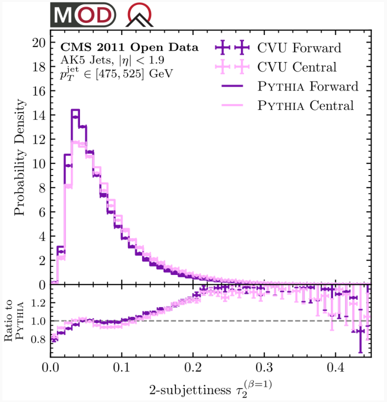
What does of.omnifold() do? (3/3)

```
    # STEP 2: classify Gen. to reweighted Gen. (which is reweighted by
    # weights_pull). weights Gen. --> reweighted Gen.

    weights_2 = np.concatenate((np.ones(len(theta0_G)), weights_pull))
    # ones for Gen. (not MC weights), actual weights for (reweighted) Gen.

    X_train_2, X_test_2, Y_train_2, Y_test_2, w_train_2, w_test_2 = train_test_split

    # zip ("hide") the weights with the labels
    Y_train_2 = np.stack((Y_train_2, w_train_2), axis=1)
    Y_test_2 = np.stack((Y_test_2, w_test_2), axis=1)

    model.compile(loss=weighted_binary_crossentropy,
    optimizer='Adam',
    metrics=['accuracy'])
    model.fit(X_train_2, Y_train_2, epochs=20, batch_size=2000,
              validation_data=(X_test_2, Y_test_2), verbose=verbose)

    weights_push = reweight(theta0_G,model)
    weights[i, 1:2, :] = weights_push
    #next iteration i

  return weights
```

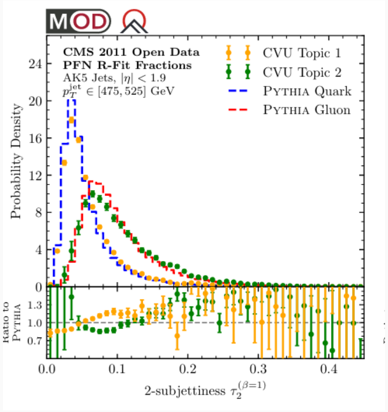
Omnifold on LHC data

Disentangling quarks and gluons in **CMS open data**

PRD 106 (2022) 9, P. T. Komiske, S. Kryhin, J. Thaler



Unfolded τ_2 distributions of the two jet categories compared to Pythia8.

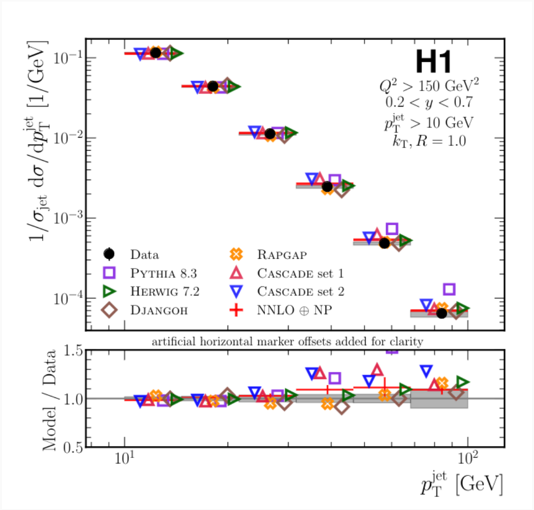


Distributions extracted for quark and gluons

Omnifold on Hera data

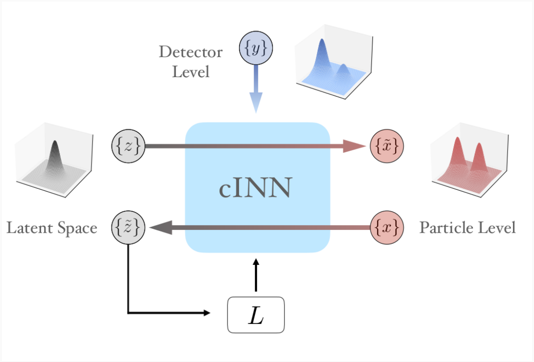
Measurement of lepton-jet correlation in deep-inelastic scattering with the H1 detector using machine learning for unfolding, doi:10.1103/PhysRevLett.128.132002

MultiFold of 8 observables, $p_{\mathrm{T}}^{\mathrm{e}}$, $p_{\mathrm{z}}^{\mathrm{e}}$, $p_{\mathrm{T}}^{\mathrm{jet}}$, $\eta_{\mathrm{T}}^{\mathrm{jet}}$, φ^{jet}, $q_{\mathrm{T}}^{\mathrm{jet}}/Q$, $\Delta\varphi^{\mathrm{jet}}$



Generative unfolding

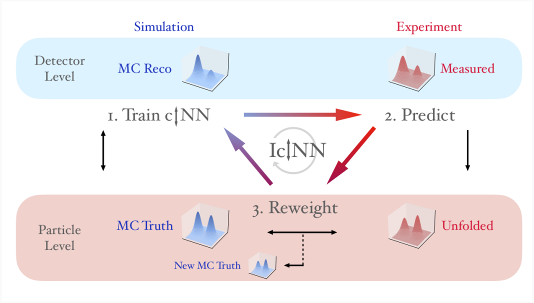
- Uses a conditional invertible neural network (cINN)
- Trained to generate a gen-level event on the condition of a reco-level event
- Apply to data to generate the unfolded distribution



arXiv:2006.06685, arXiv:1806.00433, arXiv:1912.00477, arXiv:2212.08674

Iterative generative unfolding

- Mitigate MC bias using iterations
- After the generative unfolding, use a classifier to learn ratio of unfolded to truth-level distribution and extract weights for the simulation
- Repeat the generative unfolding with the reweighted simulation



arXiv:2212.08674

There are so many methods. Which one should I use for my analysis?

Choosing a method

- If you want to take the simple path, use TUnfold. Try first without regularization and apply regularization only if you get a too large variance.
- In case the signal yield is already extracted with a likelihood, consider using the MLE method with a single likelihood. If its computing-wise affordable, it is the best approach.
- If you want to be innovative and explore new areas, go for a machine learning unfolding.

Conclusions

- Unfolding used to unroll detector effects in differential cross section measurements.
- Two different approaches: as a last step (TUnfold) or included in the signal extraction fit.
- Model unfolding uncertainties are difficult to estimate and different approaches used.
- Machine Learning opens new horizons with high-dimensional unfolding.