Unfolding algorithms and tests using RooUnfold

Tim Adye
Rutherford Appleton Laboratory, Science and Technology Facilities Council,
Harwell Science and Innovation Campus, Didcot OX11 0QX, United Kingdom.

Abstract
The RooUnfold package provides a common framework to evaluate and use different unfolding algorithms, side-by-side. It currently provides implementations or interfaces for the Iterative Bayes, SVD, and TUnfold methods, as well as bin-by-bin and matrix inversion reference methods. Common tools provide covariance matrix evaluation and multi-dimensional unfolding. A test suite allows comparisons of the performance of the algorithms under different truth and measurement models. Here I outline the package, the unfolding methods, and some experience of their use.

1 RooUnfold package aims and features

The RooUnfold package [1] was designed to provide a framework for different unfolding algorithms. This approach simplifies the comparison between algorithms and has allowed common utilities to be written.

Currently RooUnfold implements or interfaces to the Iterative Bayes [2, 3], Singular Value Decomposition (SVD) [4–6], TUnfold [7], bin-by-bin correction factors, and unregularized matrix inversion methods.

The package is designed around a simple object-oriented approach, implemented in C++, and using existing ROOT [8] classes. RooUnfold defines classes for the different unfolding algorithms, which inherit from a common base class, and a class for the response matrix. The response matrix object is independent of the unfolding, so can be filled in a separate 'training' program.

RooUnfold can be linked into a stand-alone program, run from a ROOT/CINT script, or executed interactively from the ROOT prompt. The response matrix can be initialized using existing histograms or matrices, or filled with built-in methods (these can take care of the normalization when inefficiencies are to be considered). The results can be returned as a histogram with errors, or a vector with full covariance matrix. The framework also takes care of handling multi-dimensional distributions (with ROOT support for 1–, 2–, and 3–dimensional histograms), different binning for measured and truth distributions, variable binning, and the option to include or exclude under- and over-flows.

It also supports different methods for calculating the errors that can be selected with a simple switch: bin-by-bin errors with no correlations, the full covariance matrix from the propagation of measurement errors in the unfolding, or the covariance matrix calculated using Monte Carlo (MC) toys.

All these details are handled by the framework, so don't have to be implemented for each algorithm. However different bin layouts may not produce good results for algorithms that rely on the global shape of the distribution (SVD).

A toy MC test framework is provided, allowing selection of different MC probability density functions (PDF) and parameters, comparing different binning, and performing the unfolding with the different algorithms and varying the unfolding regularization parameters. Tests can be performed with 1D, 2D, and 3D distributions. The results of a few such tests are presented in section 4.

2 C++ classes

Figure 1 summarizes how the ROOT and RooUnfold classes are used together. The RooUnfoldResponse object can be constructed using a 2D response histogram (TH2D) and 1D truth and measured projections

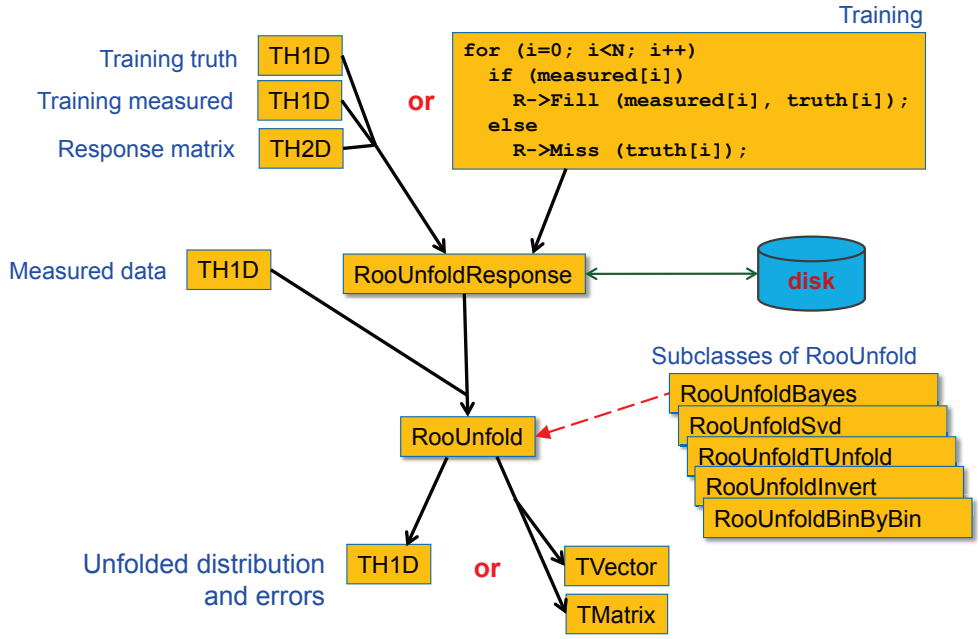

Fig. 1: The RooUnfold classes. The training truth, training measured, measured data, and unfolded distributions can also be given as TH2D or TH3D histograms.

(these are required to determine the effect of inefficiencies). Alternatively, RooUnfoldResponse can be filled directly with the `Fill`(x_{measured}, x_{true}) and `Miss`(x_{true}) methods, where the `Miss` method is used to count an event that was not measured and should be counted towards the inefficiency.

The RooUnfoldResponse object can be saved to disk using the usual ROOT input/output streamers. This allows the easy separation in separate programs of MC training from the unfolding step.

A RooUnfold object is constructed using a RooUnfoldResponse object and the measured data. It can be constructed as a RooUnfoldBayes, RooUnfoldSvd, RooUnfoldTUnfold, (etc) object, depending on the algorithm required.

The results of the unfolding can be obtained as ROOT histograms (TH1D, TH2D, or TH3D) or as a ROOT vector (TVectorD) and covariance matrix (TMatrixD). The histogram will include just the diagonal elements of the error matrix. This should be used with care, given the significant correlations that can occur if there is much bin-to-bin migration.

3 Unfolding algorithms

3.1 Iterative Bayes' theorem

The RooUnfoldBayes algorithm uses the method described by D'Agostini in [2]. Repeated application of Bayes' theorem is used to invert the response matrix. Regularization is achieved by stopping iterations before reaching the 'true' (but wildly fluctuating) inverse. The regularization parameter is just the number of iterations. In principle, this has to be tuned according to the sample statistics and binning. In practice, the results are fairly insensitive to the precise setting used.

RooUnfoldBayes takes the training truth as its initial prior, rather than a flat distribution, as described by D'Agostini. This should not bias the result once we have iterated, but could reach an optimum after fewer iterations.

This implementation takes account of errors on the data sample but not, by default, uncertainties in the response matrix due to finite MC statistics. That calculation can be very slow, and usually the training sample is much larger than the data sample.

RooUnfoldBayes does not normally do smoothing, since this has not been found to be necessary and can, in principle, bias the distribution. Smoothing can be enabled with an option.

3.2 Singular Value Decomposition

RooUnfoldSvd provides an interface to the TSVDUnfold class implemented in ROOT by Tackmann [6], which uses the method of Höcker and Kartvelishvili [4]. The response matrix is inverted using singular value decomposition, which allows for a linear implementation of the unfolding algorithm. The normalization to the number of events is retained in order to minimize uncertainties due to the size of the training sample. Regularization is performed using a smooth cut-off on small singular value contributions ($s_i^2 \rightarrow s_i^2/(s_i^2 + s_k^2)$, where the kth singular value defines the cut-off), which correspond to high-frequency fluctuations.

The regularization needs to be tuned according to the distribution, binning, and sample statistics in order minimize the bias due to the choice of the training sample (which dominates at small k) while retaining small statistical fluctuations in the unfolding result (which grow at large k).

The unfolded error matrix includes the contribution of uncertainties on the response matrix due to finite MC training statistics.

3.3 TUnfold

RooUnfoldTUnfold provides an interface to the TUnfold method implemented in ROOT by Schmitt [7]. TUnfold performs a matrix inversion with 0-, 1-, or 2-order polynomial regularization of neighbouring bins. RooUnfold automatically takes care of packing 2D and 3D distributions and creating the appropriate regularization matrix required by TUnfold.

TUnfold can automatically determine an optimal regularization parameter (τ) by scanning the 'L-curve' of $\log_{10}\chi^2$ vs $\log_{10}\tau$.

3.4 Unregularized algorithms

Two simple algorithms, RooUnfoldBinByBin, which applies MC correction factors with no inter-bin migration, and RooUnfoldInvert, which performs unregularized matrix inversion with singular value removal (TDecompSVD) are included for reference.

4 Examples

Examples of toy MC tests generated by RooUnfoldTest are shown in Figs. 2–4. These provide a challenging test of the procedure. Completely different training and test MC models are used: a single wide Gaussian PDF for training and a double Breit-Wigner for testing. In both cases these are smeared, shifted, and a variable inefficiency applied to produce the 'measured' distributions.

5 Unfolding errors

Regularization introduces inevitable correlations between bins in the unfolded distribution. To calculate a correct χ^2, one has to invert the covariance matrix:

$$\chi^2 = (\mathbf{x}_{\mathrm{measured}} - \mathbf{x}_{\mathrm{true}})^{\mathrm{T}}\mathbf{V}^{-1}(\mathbf{x}_{\mathrm{measured}} - \mathbf{x}_{\mathrm{true}}) \tag{1}$$

However, in many cases, the covariance matrix is poorly conditioned, which makes calculating the inverse problematic. Inverting a poorly conditioned matrix involves subtracting large, but very similar numbers, leading to significant effects due to the machine precision.

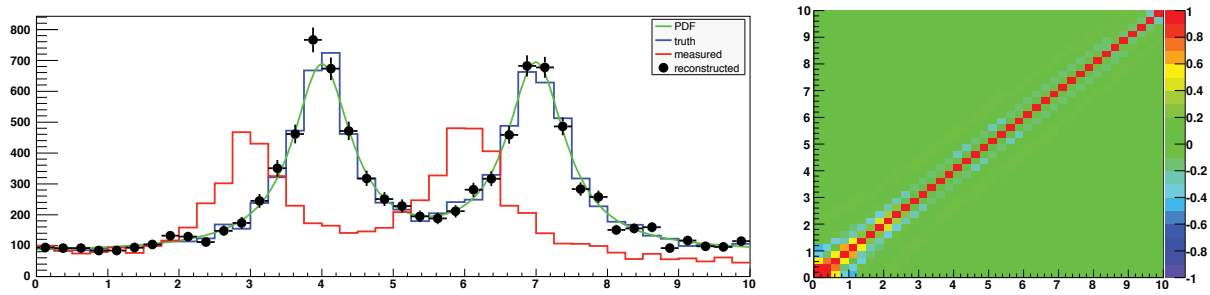

Fig. 2: Unfolding with the Bayes algorithm. On the left, a double Breit-Wigner PDF on a flat background (green curve) is used to generate a test 'truth' sample (upper histogram in blue). This is then smeared, shifted, and a variable inefficiency applied to produce the 'measured' distribution (lower histogram in red). Applying the Bayes algorithm with 4 iterations on this latter gave the unfolded result (black points), shown with errors from the diagonal elements of the error matrix. The bin-to-bin correlations from the error matrix are shown on the right.

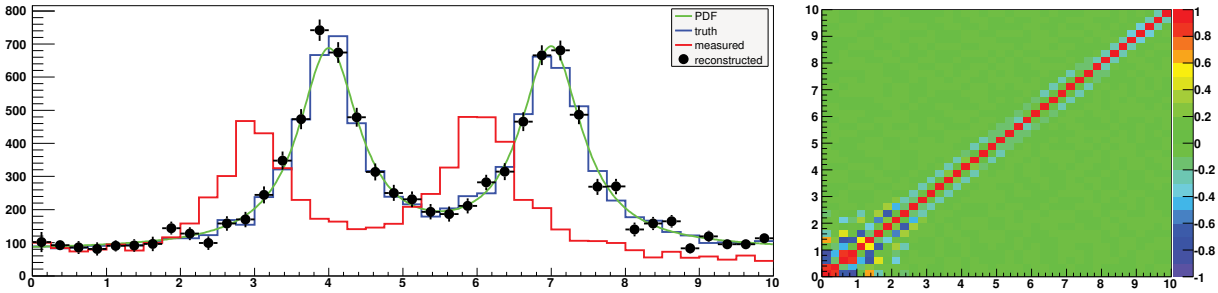

Fig. 3: Unfolding with the SVD algorithm ($k = 30$) on the same training and test samples as described in Fig. 2.

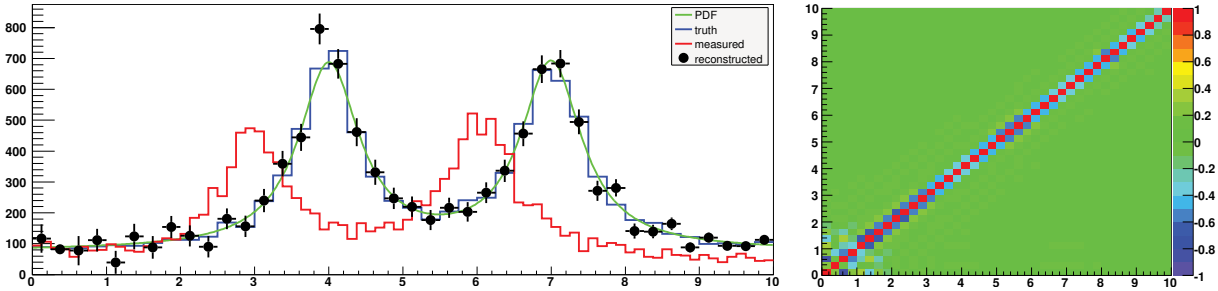

Fig. 4: Unfolding with the TUnfold algorithm ($\tau = 0.004$) on the same training and test samples as described in Fig. 2. Here we use two measurement bins for each truth bin.

5.1 Unfolding errors with the Bayes method

As shown on the left-hand side of Fig. 5, the uncertainties calculated by propagation of errors in the Bayes method were found to be significantly underestimated compared to those given by the toy MC. This was found to be due to an omission in the original method outlined by D'Agostini ([2] section 4).

The Bayes method gives the unfolded distribution ('estimated causes'), $\hat{n}(\mathrm{C}_i)$, as the result of applying the unfolding matrix, M_{ij}, to the measurements ('effects'), $n(\mathrm{E}_j)$:

$$\hat{n}(\mathrm{C}_i) = \sum_{j=1}^{n_\mathrm{E}} M_{ij} n(\mathrm{E}_j) \quad \text{where} \quad M_{ij} = \frac{P(\mathrm{E}_j|\mathrm{C}_i) n_0(\mathrm{C}_i)}{\epsilon_i f_j} \tag{2}$$

$P(\mathrm{E}_j|\mathrm{C}_i)$ is the response matrix, $\epsilon_i \equiv \sum_{j=1}^{n_\mathrm{E}} P(\mathrm{E}_j|\mathrm{C}_i)$ are efficiencies, and $f_j \equiv \sum_{l=1}^{n_\mathrm{C}} P(\mathrm{E}_j|\mathrm{C}_l) n_0(C_l)$ is the folded prior distribution, $n_0(C_l)$ — initially arbitrary (eg. flat or MC model), but updated on subsequent iterations.

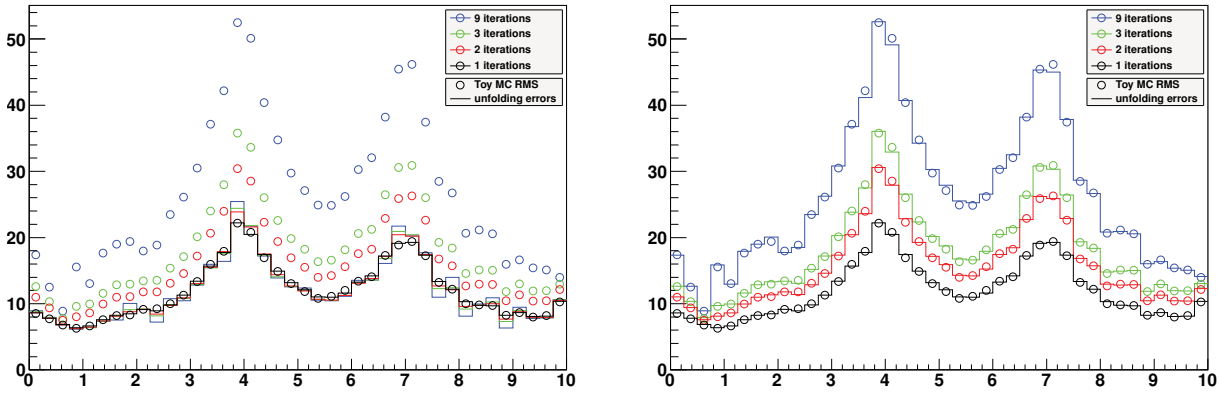

Fig. 5: Bayesian unfolding errors (lines) compared to toy MC RMS (points) for 1, 2, 3, and 9 iterations. The left-hand plot shows the errors using D'Agostini's original method, ignoring any dependence on previous iterations (only the M_{ij} term in Eq. (3)). The right-hand plot shows the full error propagation.

The covariance matrix, which here we call $V(\hat{n}(\mathrm{C}_k), \hat{n}(\mathrm{C}_l))$, is calculated by error propagation from $n(\mathrm{E}_j)$, but M_{ij} is assumed to be itself independent of $n(\mathrm{E}_j)$. That is only true for the first iteration. For subsequent iterations, $n_0(\mathrm{C}_i)$ is replaced by $\hat{n}(\mathrm{C}_i)$ from the previous iteration, and $\hat{n}(\mathrm{C}_i)$ depends on $n(\mathrm{E}_j)$ (Eq. (2)).

To take this into account, we compute the error propagation matrix

$$\frac{\partial \hat{n}(\mathrm{C}_i)}{\partial n(\mathrm{E}_j)} = M_{ij} + \sum_{k=1}^{n_\mathrm{E}} M_{ik} n(\mathrm{E}_k) \left(\frac{1}{n_0(\mathrm{C}_i)} \frac{\partial n_0(\mathrm{C}_i)}{\partial n(\mathrm{E}_j)} - \sum_{l=1}^{n_\mathrm{C}} \frac{\epsilon_l}{n_0(\mathrm{C}_l)} \frac{\partial n_0(\mathrm{C}_l)}{\partial n(\mathrm{E}_j)} M_{lk} \right) \tag{3}$$

This depends upon the matrix $\frac{\partial n_0(\mathrm{C}_i)}{\partial n(\mathrm{E}_j)}$, which is $\frac{\partial \hat{n}(\mathrm{C}_i)}{\partial n(\mathrm{E}_j)}$ from the previous iteration. In the first iteration, the second term vanishes ($\frac{\partial n_0(\mathrm{C}_i)}{\partial n(\mathrm{E}_j)} = 0$) and we get $\frac{\partial \hat{n}(\mathrm{C}_i)}{\partial n(\mathrm{E}_j)} = M_{ij}$.

The error propagation matrix can be used to obtain the covariance matrix on the unfolded distribution

$$V(\hat{n}(\mathrm{C}_k), \hat{n}(\mathrm{C}_l)) = \sum_{i,j=1}^{n_\mathrm{E}} \frac{\partial \hat{n}(\mathrm{C}_k)}{\partial n(\mathrm{E}_i)} V(n(\mathrm{E}_i), n(\mathrm{E}_j)) \frac{\partial \hat{n}(\mathrm{C}_l)}{\partial n(\mathrm{E}_j)} \tag{4}$$

from the covariance matrix of the measurements, $V(n(\mathrm{E}_i), n(\mathrm{E}_j))$.

Without the new second term in Eq. (3), the error is underestimated if more than one iteration is used, but agrees well with toy MC tests if the full error propagation is used, as shown in Fig. 5.

6 Status and plans

RooUnfold was first developed in the *BABAR* software environment and released stand-alone in 2007. Since then, it has been used by physicists from many different particle physics, particle-astrophysics, and nuclear physics groups. Questions, suggestions, and bug reports from users have prompted new versions with fixes and improvements.

Last year I started working with a small group hosted by the Helmholtz Alliance, the Unfolding Framework Project [9]. The project is developing unfolding experience, software, algorithms, and performance tests. It has adopted RooUnfold as a framework for development.

Development and improvement of RooUnfold is continuing. In particular, determination of the systematic errors due to uncertainties on the response matrix, and due to correlated measurement bins will be added. The RooUnfold package will be incorporated into the ROOT distribution, alongside the existing TUnfold and TSVDUnfold classes.

References

[1] The RooUnfold package and documentation are available from
http://hepunx.rl.ac.uk/~adye/software/unfold/RooUnfold.html

[2] G. D'Agostini, "A Multidimensional unfolding method based on Bayes' theorem," Nucl. Instrum. Meth. A **362** (1995) 487.

[3] K. Bierwagen, "Bayesian Unfolding," these proceedings.

[4] A. Hocker and V. Kartvelishvili, "SVD Approach to Data Unfolding," Nucl. Instrum. Meth. A **372** (1996) 469.

[5] V. Kartvelishvili, "Unfolding with SVD," these proceedings.

[6] K. Tackmann, "SVD-based unfolding: implementation and experience," these proceedings.

[7] The TUnfold package is available in ROOT [8] and documented in
http://www.desy.de/~sschmitt/tunfold.html

[8] R. Brun and F. Rademakers, "ROOT: An object oriented data analysis framework," Nucl. Instrum. Meth. A **389** (1997) 81. See also http://root.cern.ch/.

[9] For details of the Unfolding Framework Project, see
https://www.wiki.terascale.de/index.php/Unfolding_Framework_Project