

November 20, 2019

The TUnfold package: user manual

Stefan Schmitt, DESY, Notkestraße 85, 22607 Hamburg

Email: `Stefan.Schmitt@desy.de`

Abstract

TUnfold is a package with provides functionality for correcting migration and background effects for multi-dimensional distributions. This document gives a user-oriented technical description of the package, valid for the version number 17.9.

1 Package overview

The TUnfold package provides algorithms to correct measured distributions for migration effects. The algorithm is based on least-square fitting and Tikhonov regularisation, it is described in [1]. In this document, details of the technical implementation and of the user interface are described. It is assumed that the reader is familiar with the algorithm [1].

The package is written in the C++ programming language. It consists of the six classes `TUnfold`, `TUnfoldSys`, `TUnfoldDensity`, `TUnfoldIterativeEM`, `TUnfoldBinning` and `TUnfoldBinningXML`. The package is tied to the ROOT analysis framework [2].

1.1 Root versions and TUnfold versions

As of root version 5.22, some version of the TUnfold package is distributed together with the root software. Table 1 summarizes the connection between TUnfold versions and distributed root versions. The most recent Root version typically does not include the full functionality of TUnfold. However, it is possible to download the latest TUnfold version 17.9 and use it together with older ROOT releases, even if that ROOT release already includes another version of TUnfold. In order to achieve this, in the distributed TUnfold

ROOT	TUnfold	Supported TUnfold classes
5.21 and earlier	—	—
5.22	V6	<code>TUnfold</code>
5.23-5.25	V13	+ <code>TUnfoldSys</code>
5.27	V15	
5.28-5.36	V16.0	
—	V17.1	+ <code>TUnfoldDensity</code> , <code>TUnfoldBinning</code>
—	V17.2	+ <code>TUnfoldBinningXML</code>
6.00	V17.3	
—	V17.4	
—	V17.5	
6.10	V17.6	
—	V17.7	
—	V17.8	
—	V17.9	+ <code>TUnfoldIterativeEM</code>

Table 1: correspondence of distributed ROOT versions and TUnfold versions.

17.9 package, the classes have been renamed: the class `TUnfold` is named `TUnfoldV17`, the class `TUnfoldSys` is named `TUnfoldSysV17`, etc. In the header files, statements like

```
#define TUnfold TUnfoldV17
```

have been added, such that the renamed classes are accessible under their usual name.

1.2 TUnfold distribution

The TUnfold package is available for download here [3]. The package comes as a gzipped tar archive. The archive should contain the files given in table 2. TUnfold is free software:

README	notes on compiling
COPYING	licence file
tunfold_manual.tex	LaTeX source of this manual
fig/tunfold_manual_fig1.eps	Figure 1 of this manual
fig/tunfold_manual_fig2.eps	Figure 2 of this manual
Makefile	default makefile for linux systems
altercodeversion.sh	auxillary script
TUnfold.h	header file providing the class TUnfoldV17
TUnfoldSys.h	header file providing the class TUnfoldSysV17
TUnfoldDensity.h	header file providing the class TUnfoldDensityV17
TUnfoldIterativeEM.h	header file providing the class TUnfoldIterativeEMV17
TUnfoldBinning.h	header file providing the class TUnfoldBinningV17
TUnfoldBinningXML.h	header file providing the class TUnfoldBinningXMLV17
TUnfoldV17.cxx	implementation of the class TUnfoldV17
TUnfoldSysV17.cxx	implementation of the class TUnfoldSysV17
TUnfoldDensityV17.cxx	implementation of the class TUnfoldDensityV17
TUnfoldIterativeEMV17.cxx	implementation of the class TUnfoldIterativeEMV17
TUnfoldBinningV17.cxx	implementation of the class TUnfoldBinningV17
TUnfoldBinningXMLV17.cxx	implementation of the class TUnfoldBinningXMLV17
testUnfoldXX.C	example macros where XX=1, 2a, 2b, 2c, 3, 4, 5a, 5b, 5c, 5d, 6, 7a, 7b, 7c

Table 2: files distributed with the TUnfold package version 17.9.

you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

TUnfold is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with TUnfold. If not, see <http://www.gnu.org/licenses/>.

1.3 Makefile

For many unix systems, the Makefile provided with this distribution is suitable for compiling the examples and the library. Note however, compilation only has been tested on selected systems. In general, modifications to the Makefile may be needed in order to compile the TUnfold package. The main commands from the Makefile are

make lib creates a shared library `libtunfold.so`.

make bin creates wrapper code to call the example macros and compiles them as stand-alone executables. For example the file `testunfold1.C` is created and compiled as executable `testunfold1`.

For using the TUnfold package, it is probably best to work through the example given by the three macros `testUnfold2a.C`, `testUnfold2b.C` and `testUnfold2c.C`.

1.4 Class overview

The classes distributed with TUnfold are described briefly in the following. For most applications, the proper class to use is `TUnfoldDensity` and possibly also the class `TUnfoldBinning` to set up the analysis bins. The class `TUnfoldIterativeEM` can be used to compare with the iterative EM unfolding method [4, 5, 6, 7], also known as D’Agostini or “Bayesian iterative” unfolding [8].

class `TUnfold` provides the core unfolding algorithm, matrix operations and methods to import from histograms or to export to histograms.

class `TUnfoldSys` adds functionality to the class `TUnfold` to treat background and systematic uncertainties.

class `TUnfoldDensity` adds functionality to the class `TUnfoldSys` to properly take into account bin widths and multi-dimensional distributions.

class `TUnfoldIterativeEM` comes with an interface similar to `TUnfoldDensity`, so one can compare directly between iterative and Tikhonov type unfolding, using an equivalent methods to choose the regularisation strength or the number of iterations.

class `TUnfoldBinning` is used to tell the class `TUnfoldDensity` how the bins in complex binning schemes are arranged.

class `TUnfoldBinningXML` provides an interface to read and write complex binning schemes as XML files.

Table 3 gives a summary of the most important methods available with the TUnfold package.

2 Getting started with an example

In this section, the steps to run the example implemented in the macros 2a, 2b, 2c is explained.

Run the unfolding	
Method	Description
constructor	define matrix of migrations and basic regularisation scheme
SetInput()	define measurement
AddSysError()	set a systematic uncertainty
SubtractBackground()	set a background source
DoUnfold()	unfold once, with fixed tau
ScanSURE()	scan the SURE variable (unfold multiple times) and determine the optimal tau or the optimal number of iterations
ScanLcurve()	scan L-curve (unfold multiple times) and determine tau
ScanTau()	scan correlations (unfold multiple times) and determine tau
Retreive unfolding results	
Method	Description
GetOutput()	unfolding result
GetEmatrixTotal()	total error matrix
GetRhoItotal()	total global corelations
GetDeltaSysSource()	systematic shifts from one systematic error
GetDeltaSysBackgroundScale()	systematic shifts from one background scale error
GetEmatrixSysUncorr()	error matrix from uncorrelated uncertainty on migration matrix
GetEmatrixSysBackgroundUncorr	error matrix from uncorrelated uncertainty on one background source
GetEmatrixInput	error matrix from input errors
Retreive unfolding error matrix (only when using class TUnfold)	
Method	Description
GetEmatrix	(deprecated) get error matrix
GetRhoI	(deprecated) get global corelations

Table 3: basic methods required to use the unfolding package. The table lists the name of the method and a short description. Not all methods are implemented for the iterative EM unfolding.

2.1 Example compilation

This describes the typical steps to compile the examples on a linux computer.

1. Set up ROOT on your computer
2. Check that root version is greater or equal 6 and class TXMLNode is available

```
> root-config --version
```

```
6.18/04
> ls $(root-config --incdir)/TXMLNode.h
/cvmfs/sft.cern.ch/lcg/releases/LCG_96b/ROOT/6.18.04/x86_64-centos7-gcc8-opt/include
```

3. Download, unpack and compile the TUnfold package

```
> mkdir tunfold_v17.9
> cd tunfold_v17.9
> wget https://www.desy.de/~sschmitt/TUnfold/TUnfold_V17.9.tgz
...
> tar xvfz TUnfold_V17.9.tgz
> make bin
...
```

2.2 Run the example

The example consists of three parts

1. Generate signal, background and “data”

```
> ./testunfoldmain2a
...
```

This produces the file “testUnfold2a.data.root” (data events) and the file “testUnfold2a_MC.root” (signal and background events with truth information).

2. Fill signal, background and data into histograms

```
> ./testunfoldmain2b
...
```

This produces the root file “testunfold2b_histograms.root” (input histograms for the unfolding) and two sets of plots “testunfold2b_controlPlots.eps” (Figure 1) and “testunfold2b_migrations.eps” (Figure 2).

3. Unfold the data

```
> ./textunfoldmain2c
...
```

This produces the root file “testunfold2c_unfolded.root” with unfolding results, a set of plots “testunfold2c_scan.eps” (Figure 3) and another set of plots “testunfold2c_coverageFromData.eps” (Figure 4)

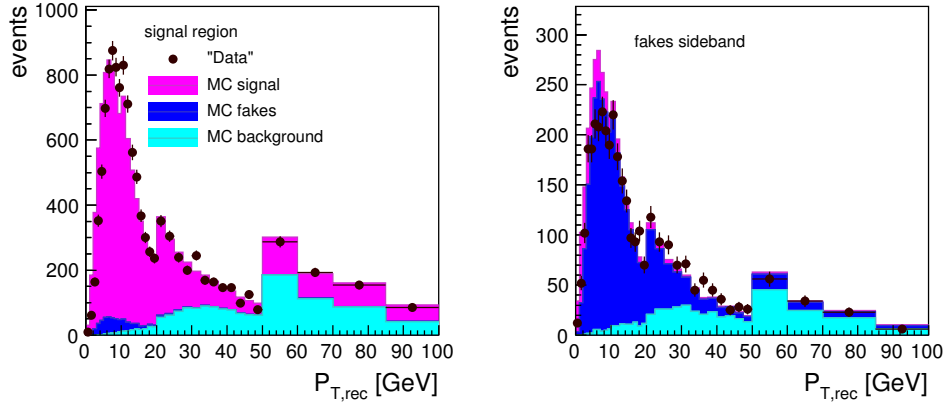


Figure 1: Control plots produced by the example 2b, in the “signal region” and the “fakes sideband”.

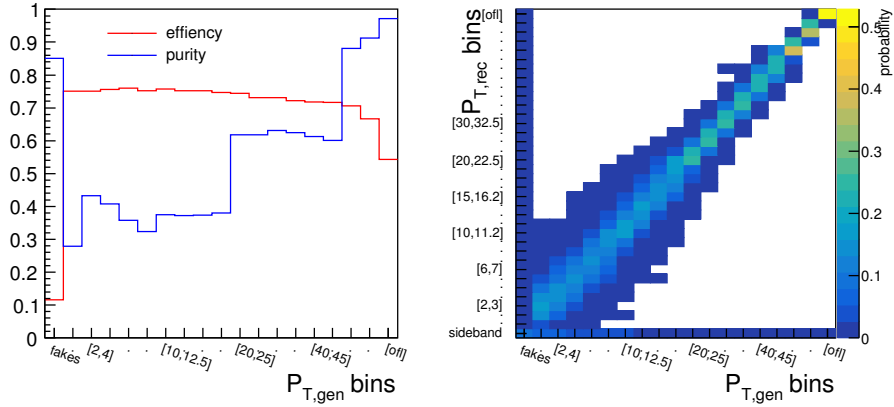


Figure 2: Bin purity, efficiency and matrix of probabilities, for the example 2b.

2.3 Discussion of the plots

The plots produced by the example 2a, 2b, 2c are discussed in the following. Figure 1 shows the input to the unfolding. The property $P_{T,rec}$ is measured in two regions: the “signal region” and the “fakes sideband”. In this example there is background mainly at high $P_{T,rec}$. At low $P_{T,rec}$ there is some background from “fakes”. The normalisation of the “fakes” background is determined in the unfolding. There is an extra reconstructed bin for measuring the total event count in the “fakes sideband”. In addition there is an extra truth bin to determine the fakes normalisation from data.

Figure 2 shows the efficiency, purity and matrix of probabilities for the unfolding. The purities (shown here after subtracting background but including fakes) are rather low; however it is still possible to get reasonable results in the unfolding step.

Figure 3 shows the unfolding results from scanning the L-curve (Tikhonov only) and from scanning the “SURE” variable (Tikhonov and iterative EM method). The optimal

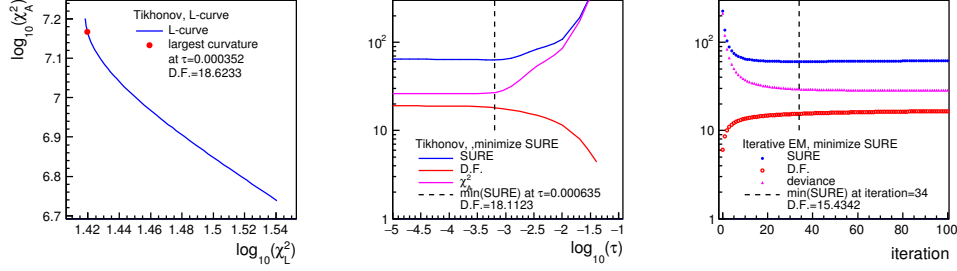


Figure 3: Scan of the Tikhonov regularisation parameter (left,middle) and of the number of EM iterations (right). For the left plot, the L-curve method is used. For the middle and right plot, the SURE variable is minimized.

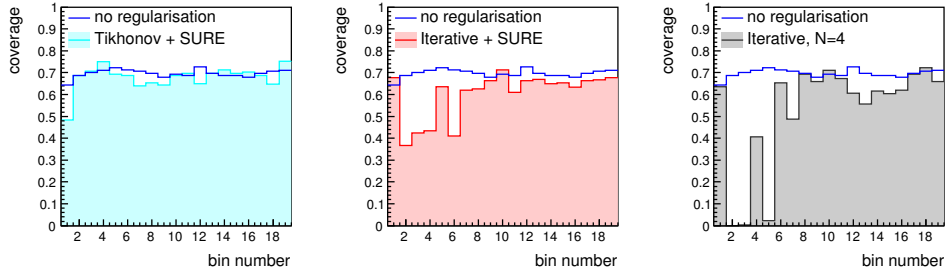


Figure 4: Data-driven determination of coverage intervals for each analysis bin and three choices of Regularisation: (left) Tikhonov with SURE minimisation, (middle) iterative EM method with SURE minimisation (right) iterative EM method with four iterations.

Tikhonov parameter τ is found to be of order 0.5×10^{-4} , while the optimal number of EM iterations is 34 in this example. The effective number of degrees of freedom is larger than 18 for the Tikhonov case but only 15.4 when using EM iterations, as compared to the number of unfolded bins which is 19.

Figure 4 shows a determination of the data-driven coverage for each of the analysis bins. Ideally, the coverage is expected to be 68%. One can see that the Tikhonov method with SURE minimisation performs reasonably well, while the iterative EM method with SURE minimisation has a poor coverage in the low-purity bins and an unacceptably low coverage for a small number of iterations.

3 Histograms and binning schemes

ROOT histograms are used to exchange information between the TUnfold package and the user. Internally, the algorithm works with vectors to store the bins of the input and output distributions. In the following, the relations of histogram bins to vector elements are discussed.

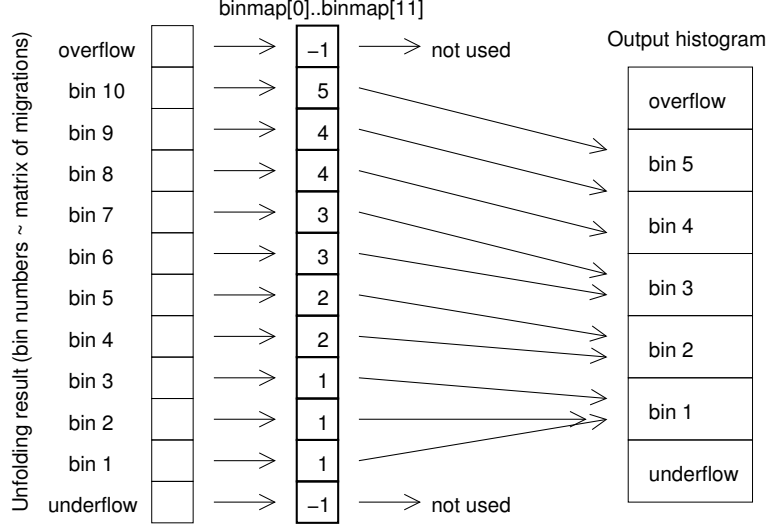


Figure 5: For the classes `TUnfold` and `TunfoldSys`, the bin map defines which bins of the unfolding result are stored in which histogram bin. In the example, 10 bins are mapped to 5 bins.

3.1 Use of bin maps with class `TUnfold` and `TUnfoldSys`

When importing data into the classes `TUnfold` or `TunfoldSys`, only the bin contents and bin errors of the histograms are relevant. The bin edges are not used. When extracting data into an existing histogram, the binning of that histogram is not checked. It is up to the user to book a histogram with the proper binning. It is then possible to change the mapping of the vector components to histogram bins. The mapping function is stored as an array of integer numbers and is denoted “bin map”. Each element of the bin map corresponds to one of the bins in the unfolding result. The value stored in the bin map indicates the destination histogram bin in which the result shall be stored. It is possible to add up several bins of the unfolding result simply by using the same destination bin number for different elements of the bin map. The concept of the bin map is illustrated in figure 5.

3.2 Binning schemes and `TUnfoldDensity`

For the classe `TUnfoldDensity` and `TUnfoldIterativeEM` the bins are structured in a “binning scheme” using the class `TUnfoldBinning`. For one-dimensional unfolding, the binning schemes can be constructed directly from the histogram bins of the matrix of migrations. In this case, the user does not have to use the class `TUnfoldBinning` directly. For more complex problems, involving multi-dimensional distributions, multiple channels or unfolding of background normalisation factors, the corresponding binning schemes have to be defined by the user. The binning scheme information is used when setting up the regularisation scheme. In addition, it is used to create histograms having proper bin widths when extracting data from the classes `TUnfoldDensity` or `TUnfoldIterativeEM`.

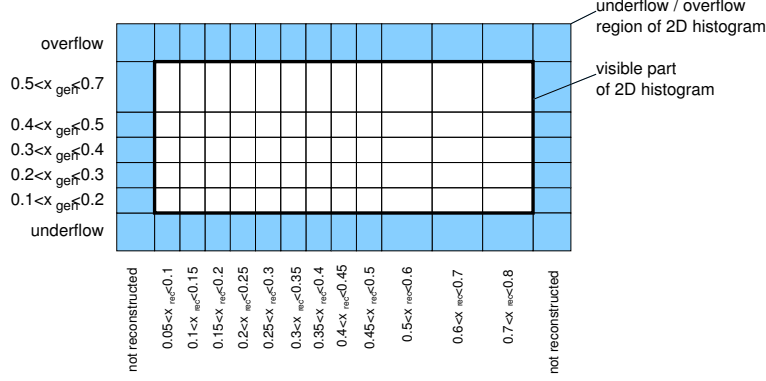


Figure 6: The matrix of migrations in the case of one-dimensional unfolding is illustrated. The truth parameter x_{gen} has five non-equidistant bins, ranging from 0.1 to 0.7 plus underflow and overflow bins (seven bins in total). The reconstructed parameter x_{rec} has twelve bins ranging from 0.05 to 0.8. The underflow and overflow bins in x_{rec} are used to count the non-reconstructed events.

In the event loop, the binning scheme provides functionality to find the proper bin numbers for filling the histogram of migrations or the histogram of measurements.

3.3 Unfolding one-dimensional distributions in TUnfoldDensity

When unfolding one-dimensional distributions, it is most convenient to book and fill the histogram of migrations using the bins as required for the analysis. There is no need to define binning schemes. The matrix of migrations is stored as a two-dimensional histogram, where on one axis the truth bins are arranged. It is possible to have underflow and overflow bins for the truth parameters. If these are present, their content is also unfolded from the data.

On the other axis there is the reconstructed quantity, again with the appropriate bins. Here, it is not possible to use the underflow and overflow bins for measurements. Instead, these bins are used to count events which originate from a specific truth bin but where the reconstructed quantity is not available (inefficiencies). An example is given in figure 6.

3.4 Complex binning schemes

The class `TUnfoldBinning` provides means to map bins originating from multi-dimensional distributions on a one-dimensional histogram axis. The multi-dimensional distributions are arranged in a tree structure.

For the truth parameters, the branches of the tree structure could correspond to different decay channels, signal, background, etc. Each branch then holds several bins, usually in the form of a multi-dimensional histogram.

Similarly, for the reconstructed parameters, the branches of the tree structure could correspond to different reconstructed channels, control distributions, auxillary measurements, etc.

So in general, there are two “binning trees”, a tree of truth bins and a tree of reconstructed bins. When filling the histogram of migrations, the proper bin numbers both in the tree of truth bins and in the tree of reconstructed bins have to be determined.

The bin number i_{gen} on truth level is determined as follows: first, the appropriate branch in the tree is determined, for example by deciding on the event type (signal or background). The method `FindNode()` may be used to locate a branch in the tree using its name.

Next, using the truth parameters, the bin number i_{gen} is calculated using the method `GetGlobalBinNumber()` on the branch. Below this is illustrated in a code fragment, assuming the signal branch contains a three-dimensional histogram with the variables `xTrue`, `yTrue`, `zTrue`.

```
Int_t iGen;
const TUnfoldBinning *signalBranch=generatorBinning->FindNode("signal");
iGen=signalBranch->GetGlobalBinNumber(xTrue,yTrue,zTrue);
```

The bin number i_{rec} in the tree of reconstructed bins is determined in a similar manner, for example by deciding on the reconstructed channel and then using the appropriate reconstructed quantities to calculate the bin number. If the event was not reconstructed, the special bin number $i_{\text{rec}} = 0$ has to be used.

Finally, the event weight w_{gen} is filled in the corresponding bin of the two-dimensional histogram of migrations. Sometimes there is a secondary event weight w_{rec} to account for detector efficiency corrections. If that extra weight is different from unity, the event has to be filled twice into the histogram of migrations. First, the histogram of migration is filled at the position $(i_{\text{gen}}, i_{\text{rec}})$ using the weight $w_{\text{gen}} \times w_{\text{rec}}$. Next, the histogram of migration is filled again, this time at the position $(i_{\text{gen}}, 0)$ using the event weight $w_{\text{gen}} \times (1 - w_{\text{rec}})$.

For data, the procedure to determine the bin number i_{rec} is applied for the reconstructed quantities only, and a one-dimensional histogram is filled. There is no need to fill the special bin number $i_{\text{rec}} = 0$ for data.

Setting up binning schemes with `TUnfoldBinning` is illustrated in the example macro `testUnfold5b.C`. How to use the binning scheme to fill histograms is illustrated in `testUnfold5c.C`. Unfolding and extracting distributions using the binning scheme is illustrated in `testUnfold5d.C`.

There is also a newer example, `testUnfold2a.C`, `testUnfold2b.C` and `testUnfold2c.C`. In that example, Tikhonov methods and the iterative EM method are compared, in each case using the equivalent “SURE” method to define the regularisation strength (or number of iterations).

Another example `testUnfold7a.C`, `testUnfold7b.C`, `testUnfold7c.C` also compares different unfolding methods. It has been presented at a conference and a writeup is available [9].

3.5 XML interface to binning schemes

An XML interface is provided for the binning schemes. The DTD definition is repeated here. There is a method `TUnfoldBinningXML::WriteDTD()` which saves the DTD to a file.

```
<!-- TUnfold Version V17.3 -->
<!ELEMENT TUnfoldBinning (BinningNode)+ >
<!ELEMENT BinningNode (BinningNode+|(Binfactorlist?,Axis)|Bins) >
<!ATTLIST BinningNode name ID #REQUIRED firstbin CDATA "-1"
        factor CDATA "1.">
<!ELEMENT Axis ((Bin+,Axis?)|(Axis)) >
<!ATTLIST Axis name CDATA #REQUIRED lowEdge CDATA #REQUIRED>
<!ELEMENT Binfactorlist (#PCDATA)>
<!ATTLIST Binfactorlist length CDATA #REQUIRED>
<!ELEMENT Bin EMPTY>
<!ATTLIST Bin width CDATA #REQUIRED location CDATA #IMPLIED
        center CDATA #IMPLIED repeat CDATA #IMPLIED>
<!ELEMENT Bins (BinLabel)* >
<!ATTLIST Bins nbins CDATA #REQUIRED>
<!ELEMENT BinLabel EMPTY>
<!ATTLIST BinLabel index CDATA #REQUIRED name CDATA #REQUIRED>
```

There are methods `ExportXML` and `ImportXML` to write or read binning trees in XML format. One XML file may contain several binning schemes.

It is probably best to study the example macros `testUnfold5a.C`–`testUnfold5d.C` to find out how the XML interface works. In the example `testUnfold5a.C`, pseudo events are written to root files “testUnfold5_data.root”, “testUnfold5_signal.root”, “testUnfold5_background.root”. In `testUnfold5b.C`, binning schemes are set up and then stored as XML in a file named “testUnfold5binning.xml”. In `testUnfold5c.C` the XML file is read and the binning schemes are used when looping over the data, signal and background events. Histograms required for the unfolding are filled. The histograms and the binning schemes are then stored in another root file “testUnfold5_histograms.root”. Finally, the macro `testUnfold5d.C` reads back that root file and runs the unfolding. One could try to edit the XML file and run `testUnfold5c.C` and `testUnfold5d.C` repeatedly to see the effects.

There is also a macro `testUnfold6.C` which reads a binning scheme from an xml file and prints the binning scheme and various bin maps extracted from the binning scheme.

4 Regularisation

For unfolding, regularisation conditions are imposed. For the Tikhonov methods `TUnfold`, `TUnfoldSys` or `TUnfoldDensity`, the regularisation is given by the scalar product $(\tau Lz, \tau Lz)$,

where z is the difference of the unfolding result to a bias vector and L is a matrix describing the regularisation scheme. The parameter τ gives the strength of the regularisation. The number of columns of L is identical to the number of unfolded bins. The number of rows reflects the number of regularisation conditions, it may be different from the number of columns.

When using the iterative EM method `TUnfoldIterativeEM`, the regularisation strength is given by the number of iterations. As opposed to Tikhonov methods, where the special point $\tau = 0$ corresponds to the absence of regularisation, for the iterative method the special point (start value) corresponds to maximum regularisation.

4.1 Basic Tikhonov regularisation types

For the Tikhonov methods, three basic types of regularisation are supported: `kRegModeSize`, `kRegModeDerivative`, `kRegModeCurvature`. The type of regularisation may be specified with the constructor of either of the classes `TUnfold`, `TUnfoldSys`, `TUnfoldDensity` as the third argument. In that case, the given basic regularisation is applied to all bins.

The simplest regularisation condition is given by `kRegModeSize`, corresponding to the case where L is the unity matrix. The matrix L is diagonal and does not mix different bins. The regularisation is given by $\tau^2 \sum z_i^2$.

For the condition `kRegModeDerivative`, the matrix L calculates differences $x_j - x_i$, thus approximating first derivatives. In that case, the structure of the input bins matters, because differences should be calculated between adjacent bins only. For one-dimensional distributions this is done by simply setting $j = i + 1$. For two-dimensional distributions, derivatives may be defined along both dimensions and the relation is getting more complicated. When using the classes `TUnfoldDensity` and `TUnfoldBinning`, the relation of the bins is known and appropriate regularisation schemes are defined automatically.

For the condition `kRegModeCurvature`, the matrix L approximates second derivatives $(x_k - x_j) - (x_j - x_i)$. Similar to the case of `kRegModeDerivative` the corresponding matrix structure may get rather complicated in case of multi-dimensional distributions, and is most conveniently handled through the use of the classes `TUnfoldDensity` and `TUnfoldBinning`.

4.2 Non-standard Tikhonov regularisation schemes

Sometimes it is useful to set up non-standard regularisation schemes. When using the class `TUnfoldDensity` with user-defined binning schemes, there is additional control over the regularisation scheme. One may select modifications of the calculation of L such that the components of z are normalized to the corresponding bin widths prior to calculating the regularisation conditions. Furthermore, it is possible to take into account the bin widths for the calculation of the first or second derivatives. One may also set specific normalisation factors or normalisation functions with the binning scheme and use those to modify the normalisation of z in the calculation of the regularisation.

For binning schemes based on trees with several branches it is possible to restrict the regularisation to one of the branches or to set up dedicated regularisation schemes for each of the branches (method `RegularizeDistribution()`). For multi-dimensional distributions it is possible to exclude underflow or overflow bins or to exclude derivatives calculated along specific axes from the regularisation. Ultimately, it is also possible to define arbitrary regularisation conditions by adding single rows to the matrix L (method `AddRegularisationCondition()`).

5 Determination of the regularisation strength

One of the frequent questions related to regularized unfolding methods is the choice of the regularisation parameter τ for Tikhonov methods or the number of iterations for iterative methods.

If τ is too small, there is no regularisation. If τ is too large, the unfolding result is biased strongly by the regularisation condition. Similarly, for iterative methods there is danger to have a low number of iterations and thus “unfold” the start value decorated with small data statistical uncertainties.

In the TUnfold package, three basic methods to determine the regularisation strength have been implemented. The SURE method is implemented both for Tikhonov and iterative unfolding, while the other two methods, L-curve scan and minimisation of correlations, are available only for the Tikhonov case.

5.1 Scan of the normalized SURE variable

The SURE variable is constructed from the “training error” and the estimated effective number of degrees of freedom. The method is aiming to minimise a risk estimator. It is described in [10] and the references therein. The normalized SURE variable is given by $[-2 \ln \mathcal{L}(y) + 2\text{DF}(y)]$, where both the likelihood \mathcal{L} and the effective number of degrees of freedom DF are estimated from the data y . For the Tikhonov methods, the likelihood is given by a least-square with the inverse weight matrix taken from the covariance or the histogram errors as provided with the `SetInput()` method. For the iterative EM method, the likelihood is constructed using Poisson probability distributions in each bin. The effective number of degrees of freedom is given by $\text{DF} = \sum_{i,j} A_{ij} \frac{\partial \hat{x}_j}{\partial y_i}$, where A_{ij} is the matrix of probabilities. The partial derivative describe how the unfolding result in bin j depends on the data observed in bin i .

5.2 L-curve scan

The L-Curve scan is available with the classes `TUnfold`, `TUnfoldSys` and `TUnfoldDensity`. The method is named `ScanLcurve`. It works as follows: the unfolding is repeated for a number of points with different τ , for example $n_p = 30$. A parametric curve of two

variables $X(\tau)$ and $Y(\tau)$ is calculated. The exact definition of these variables is given in [1]. The optimal choice of τ is determined as the position having the largest curvature (“kink”) in the (X, Y) plane. For scanning the L-curve, the following parameters may be set: number of points n_p , minimum (τ_{\min}) and maximum (τ_{\max}) value of τ to scan. If $\tau_{\min} = \tau_{\max}$, the interval is chosen automatically. When running the scan, the following three curves are produced: $X(\tau)$, $Y(\tau)$ and $Y(X)$.

The scan proceeds as follows: Given a τ interval to scan, first the unfolding is performed for $\tau = \tau_{\min}$ and $\tau = \tau_{\max}$. Intermediate points are then inserted such that a most uniform population along the curve $X(\tau), Y(\tau)$ is achieved. Given two or more points (X_i, Y_i) , ordered in the corresponding τ_i , a new point is inserted into the interval which has the largest size $S^2 = (X_{i+1} - X_i)^2 + (Y_{i+1} - Y_i)^2$ until $n_p - 1$ points have been calculated. The last point of the scan is inserted at the best choice of tau, determined from the set of $n_p - 1$ points.

5.3 Minimisation of correlation coefficients or other quantities

With the class `TUnfoldDensity` another method of determining τ is implemented. The method `ScanTau()` repeats the unfolding n_p times for different choices of τ . During that scan, the minimum of a function $Z(\tau)$ is determined. The possible choices of the function Z are summarized in table 4. They all depend on the calculation of global correlation coefficients ρ_i , which is described in [1]. When using the method `ScanTau()`, the following

Mode	definition of Z
<code>kEScanTauRhoAvg</code>	$Z = \frac{1}{n_{\text{bin}}} \sum_i \rho_i$ (average global correlation)
<code>kEScanTauRhoMax</code>	$Z = \max_i \rho_i$ (maximum global correlation)
<code>kEScanTauRhoAvgSys</code>	$Z = \frac{1}{n_{\text{bin}}} \sum_i \rho_{i,\text{sys}}$ (average global correlation, including systematic errors)
<code>kEScanTauRhoMaxSys</code>	$Z = \max_i \rho_{i,\text{sys}}$ (maximum global correlation, including systematic errors)
<code>kEScanTauRhoSquareAvg</code>	$Z = \frac{1}{n_{\text{bin}}} \sum_i (\rho_i)^2$ (average of squares of global correlation coefficients)
<code>kEScanTauRhoSquareAvgSys</code>	$Z = \frac{1}{n_{\text{bin}}} \sum_i (\rho_{i,\text{sys}})^2$ (average of squares of global correlation coefficients including systematic errors)

Table 4: Choices of the function Z for implemented with the method `ScanTau()`.

parameters have to be set: the number of points n_p , the minimum and maximum value of τ to scan and the mode (table 4). If the minimum and maximum value of τ agree, the scan range is determined automatically. In addition one may change the way the correlation coefficients ρ_i are calculated. The calculation may be restricted to one branch in the binning tree or may use all branches. Within the distributions it is possible to exclude underflow and overflow bins or to integrate over bins. The scan returns four curves: the curve $Z(\tau)$ and in addition the three curves also returned by `ScanLcurve()`. For a given interval in τ , $n_p - 1$ points are inserted such that large τ intervals are split into

two. Finally, using the set of $n_p - 1$ points, the position of the minimum is determined and the unfolding is repeated at the position of the minimum.

The scan of correlation coefficients has the desired property that correlations in the result are minimized. Ideally, the correlation coefficients are small and can be neglected. However, this has to be checked carefully.

A drawback of the method is that it often fails. In particular, this method can not be used with the `kRegModeSize` regularisation condition. For the regularisation methods `kRegModeDerivative` and `kRegModeCurvature`, the method is expected to work more reliably.

References

- [1] S. Schmitt, JINST **7** (2012) T10003 [arXiv:1205.6201].
- [2] R. Brun and F. Rademakers, Nucl. Instrum. Meth. A **389** (1997) 81.
- [3] S. Schmitt, TUnfold version 17.9, <http://www.desy.de/~sschmitt/tunfold.html>.
- [4] Richardson, W. H.", Opt. Soc. Amer. A **62** (1972) 55.
- [5] Lucy, L. B., Astron. J. **79** (1974) 745.
- [6] Vardi, Y. and Shepp, L. A. and Kaufman, L., J. Amer. Stat. Assoc. **80** (1985) 8.
- [7] Multhei, H. N. and Schorr, B., Nucl. Instrum. Meth. **A257** (1987) 371.
- [8] D'Agostini, G., Nucl. Instrum. Meth. **A362** (1995) 487.
- [9] S. Schmitt, XII Quark Confinement and the Hadron Spectrum conference, proceedings [arXiv:1611.01927].
- [10] Tibshirani, R. J. and Rosset, S., J. Amer. Stat. Assoc. **114** (2019) 697 [arXiv:1612.09415].