The Unfolding Program in the Standard Analysis Chain Part 1. How To Use It

E. Aliu, W. Wittek
aliu@ifae.es, wittek@mppmu.mpg.de

10th April 2006

(Revised version)

Abstract

In the Standard Analysis Chain of MAGIC the last step, after the *flux step*, is the *unfolding step*. In the unfolding step the distribution of excess events (gammas) in the estimated energy, as determined in the flux step, is converted into a distribution of excess events in the true energy, from which absolute corrected gamma fluxes are computed. In addition, the unfolding step includes an iteration over an idealized flux shape, in order to make sure that the flux shape used to calculate average collection areas and migration matrices is in good agreement with the gamma flux obtained after the unfolding.

This note is divided into two parts. The present one is about the general unfolding program implemented in MARS, which includes the unfolding algorithms explained in [1]. It describes the classes, options and output plots and explains how the program is to be used. A second note deals with the performance of the unfolding program, which was studied by means of simulated data and of real data.

Contents

1 Introduction **3**

2 The unfolding program **4**
2.1 **The class MCombineDataForUnfolding** 4
2.2 The classes **MCallUnfold** and **MUnfold** 5
2.3 Calling the Unfolding program 5
2.4 Steering the unfolding 6
2.5 The classes **MCorrelatedFit** and **MCalcXvalues** 7

3 The format of the input files for the unfolding program **8**

4 The format of the file containing the combined input data **9**

5 Options and parameters for the unfolding **9**

6 Plots produced by the Unfolding program **13**
6.1 Plots produced by *MCombineDataForUnfolding* 13
6.2 Plots produced by *MCallUnfold* and *MUnfold* 13

7 Judging the unfolding **16**

1 Introduction

Measurements of a physical quantity are often systematically distorted due to the fact that the detectors are not ideal. These distortions are due to limited acceptance, to biases in the measurements and especially to the finite resolution of the detectors. Limited acceptance is referred to the fact that the probability of observing an event is less than one, which will depend on the trigger and the cuts applied. The second effect, the biases in the measurements, transforms the measurements, which means that we are not measuring exactly the quantity we want. And finally, the finite resolution of the apparatus produces a smearing in the measurement. The effect of limited acceptance, which is usually known as a function of the true quantity, can be treated separately, and the actual unfolding deals only with the biases and the finite resolution. The limited acceptance is taken into account when converting the unfolded distribution of excess events (gammas) into absolute gamma fluxes.

The distortions due to biases and finite resolution can be written in the form

$$Y(y) = \int M(y,x)S(x)dx \qquad or \qquad Y_i = \sum_j M_{ij}S_j \qquad or \qquad Y = M \cdot S \tag{1}$$

where M describes the detector response (determined from Monte Carlo), Y is the expected measured and S the true distribution, the one without distortions.

A standard task of experiments with Cherenkov telescopes is to determine the energy spectrum of gammas. This is done by measuring the number of gammas in bins of the measured (estimated) energy. The aim is to derive the number of gammas in bins of the true energy, which one would obtain with an ideal detector.

There are various approaches to solve this problem. One approach consists in a deconvolution of the different effects affecting the measurements. This is done by essentially inverting a matrix representing the detector response. This procedure, called **Deconvolution or Unfolding**, often leads to unsatisfactory results: The matrix inversion gives a solution which is in a sense technically correct, but which often is totally useless due to the large correlations between adjacent bins, which imply large fluctuations of their contents. This fact is the basis of the unfolding methods with **regularization**. In these methods one considers two terms: one term, χ_0^2, expressing the degree of agreement between the prediction $M \cdot S$ and the measurement Y, and another term , Reg, which is a function of S expressing the smoothness of the distribution. The different unfolding methods differ by the choice of the regularization term Reg. A solution for S is obtained by minimizing the expression

$$\chi^2 \;=\; \frac{w}{2} \cdot \chi_0^2 \;+\; Reg \tag{2}$$

for a fixed weight w, also called regularization parameter. Very large values of w, corresponding to an unfolding without regularization, often produce noisy unfolded distributions that fit the data perfectly. Moderate values of w will result in smoother distributions, although they will show small deviations from the measurements, and very small w will overemphasize the regularization, leading to larger deviations from the measurements. So, the proper choice of the weight is very important. Some criteria for choosing the "best" weight will be discussed later in this note.

Another approach consists in determining the parameters of an assumed parametrization of the true distribution and to check how well this parametrization is consistent with the experimental distribution. This is called **Forward Unfolding**. The basic difference between this and the previous methods is that in the Forward Unfolding an explicit hypothesis is made on the true distribution, involving only a few free parameters. Moreover, no explicit regularization is done in the Forward Unfolding.

2 The unfolding program

At present an unfolding can be performed by running the macro **CombUnfold.C**. In the macro objects of the classes *MCombineDataForUnfolding* and *MCallUnfold* are created and member functions of them are called. The macro also contains the iteration loop for the flux spectrum. The macro is being converted into an executable.

2.1 The class MCombineDataForUnfolding

With the class **MCombineDataForUnfolding** the input data, produced in the *flux step* of the Standard Analysis, are read in. The input data consist of the distribution Y to be unfolded, the migration matrix M, the effective collection area A and the effective observation time T. For the actual unfolding only Y and M are needed. However, also A and T are provided in order to convert the unfolded distribution into an absolute flux spectrum. A and T are also necessary for the Forward unfolding.

For a given source there may be more than 1 set of input data, corresponding to different conditions of data taking (ON/OFF, wobble, low/high zenith angles, moon/no moon data, ...). These data will be combined before they are further processed by the classes MCallUnfold anf MUnfold. The formulas used when combining the different data sets can be found in [1].

The classs is prepared to be called in an iteration loop, in which the flux shape is iterated, which is used to recalculate the effective collection area and the migration matrix.

The member function*ReadEnv()* reads the steering file *combunfold.rc*, in which all options and parameters are set (see Section 5).

CombineData() steers the combination of different data samples. It calls *GetInputData(), CombineAtFixedTheta(), CombineThetaBins()* and *WriteCombinedData()*.

GetInputData() reads the data from the input files, specified in the *combunfold.rc* file, and recalculates the effective collection area (*RecalcAeff()*) and the migration matrix (*RecalcMigMatrix()*) using the current flux shape (*fSpectrum*).

CombineAtFixedTheta() combines the data of a selected Θ bin, as defined in the *combunfold.rc* file.

CombineThetaBins() combines the data of all Θ bins.

WriteCombinedData() writes the combined data onto a file, whose name is defined in the *combunfold.rc* file. This file can be read by MCallUnfold.

MCombineDataForUnfolding also produces 2 sets of plots. On set shows the input data and the combined data, for each iteration step separately. Another set displays the flux shapes used in the different iteration steps and compares the combined data of the different iteration steps (see Section 6).

2.2 The classes MCallUnfold and MUnfold

For applying the unfolding two C^{++} classes **MCallUnfold.[h,cc]** and **MUnfold.[h,cc]** are needed. In *MCallUnfold* the unfolding is prepared whereas the actual unfolding is done in *MUnfold*.

Detailed explanations of all the formulas implemented in **MUnfold** are given in [1]. The structure of *MUnfold* is very flexible and allows the user to add his own code, both new options and new unfolding algorithms. In the constructor the input data is transformed to the format of root matrices (TMatrix), in which the unfolding program internally works. The different unfolding parameters and options are set in the steering file *combunfold.rc*. The options and parameters are explained in Section 5.

For each of the unfolding methods, except the Forward Unfolding, the unfolding is performed for 45 different values of the regularization parameter (weight or iteration number). The calculations include the storage of the values of all quantities needed for judging the result for a given weight or iteration step. After the loop over the 45 weights, the function *SelectBestWeight()* is called which selects the "best" weight according to some criteria (see *FlagCriterion* in Section 5). A final unfolding is done using the selected weight to produce the final unfolded distribution.

As the Forward Unfolding doesn't involve a weight it is performed only once. The method yields the parameters of the assumed parametrization of the true energy spectrum. Using these parameters a kind of "unfolded" distribution is calculated, which is then transformed into an absolute flux spectrum, like it is done for the other unfolding methods. Fitting this flux by the same function as used in the Forward unfolding should yield a χ^2 of 0.0, demonstrating the consistency of the calculations in the Forward unfolding and in the flux calculation.

Almost all the information involved in the unfolding, such as the input data, the quantities used to judge the quality of the unfolding, the unfolded distribution and the resulting flux spectrum are dumped into histograms in a method in charge of drawing such plots, which results in the output of the unfolding package.

The program is very fast, making life easy when playing with the large number of options to study the performance of the unfolding. The output plots provide all the information necessary to judge whether the result is meaningful, as will be discussed later in this section.

2.3 Calling the Unfolding program

The call to the unfolding program is performed with the class **MCallUnfold** :

```
MCallUnfold *callunfold = new MCallUnfold();
callunfold → InitializeUnfold();
callunfold → SteerUnfold();
```

The member function *InitializeUnfold()* calls *ReadEnv(), PrepareInput(), ProposeRanges()* and *Check-Input()*.

ReadEnv() reads the steering file *combunfold.rc*, in which all options and parameters are set.

PrepareInput() reads the input file, whose name is given in the *combunfold.rc* file, and stores various histograms, which contain the distribution to be unfolded, the migration matrix, the effective collection areas and the effective observation time.

On the basis of these histograms, *ProposeRanges()* determines the ranges in the estimated (Eest) and true quantity (Etrue), which may be used in the unfolding. The final choice of the ranges depends on the flags *RangeAutoSelectA, RangeAutoSelectB, nminA, nmaxA, nminB* and *nmaxB*, set in the *combunfold.rc* file.

CheckInput() checks the validity and consistency of the options and parameters set in the *combunfold.rc* file.

2.4 Steering the unfolding

The actual unfolding is performed by the method *SteerUnfold()* which calls the methods of the unfolding package *MUnfold.[h,cc]*, arranged according to the tasks they perform in the unfolding. The most important methods are :

- *MUnfold()* : The distribution to be unfolded, its covariance matrix and the migration matrix are passed to *MUnfold* as arguments of the constructor.
- *Setter functions* : Options and parameters for the unfolding, which were read from the *combunfold.rc* file, are transmitted to *MUnfold* by various Setter functions.
- *SetAcceptance()* : This function transfers to *MUnfold* the acceptance, defined as the fraction of the migration matrix contained in the selected range of the estimated quantity. There is one acceptance value for each bin of the true quantity.
- *SmoothMigrationMatrix()* : If *FlagSmoothing* is set to 1 in the *combunfold.rc* file this member function is called to smooth the migration matrix. Attention : the function is not yet in its final form and should not be used.
- *Definition of the prior distribution* : Depending on the value of *FlagPrior* in the *combunfold.rc* file different functions are called to define the prior distribution.
- *PreventFitting()* : Changes the prior distribution to be consistent with the selected bins in the estimated quantity and with the migration matrix.
- *CalculateG()* : Calculates Gram's matrix $G = M \cdot M^T$, its eigen values and vectors, to be used in the iterative unfolding methods.
- *CalculateGW()* : Calculates the matrix $GW = M^T \cdot K^{-1} \cdot M$, its eigen values and vectors, to be used in the iterative unfolding methods. K is the covariance matrix of the distribution to be unfolded.
- *Do the unfolding* : Depending on the value of *FlagUnfold* in the *combunfold.rc* file the corresponding unfolding method is called. After this step the unfolded distribution is stored in *fVb*, with the covariance matrix *fVbcov*.
- *CorrectAcceptance()* : Corrects the unfolded distribution for the losses due to the selection of bins in the estimated quantity. The resulting corrected unfolded distribution is stored in *fVbCorr*, with the covariance matrix *fVbCorrcov*.

- *DrawPlots()* : Produces the 3 sets of plots : *input, iter, results.* See Section 6 for further details.

Further operations in *SteerUnfold()* are :

- A call to *MCallUnfold::CalculateFluxes()*, which converts the unfolded distribution into absolute gamma fluxes.
- A call to *MCallUnfold::DrawOrigPlots()*, which produces a set of plots (*orig*) in the original binnings, before the selection of ranges. See Section 6 for further details. The gamma flux is fitted by a function, ignoring the correlations between the bins.
- A call to *MCallUnfold::JudgeUnfolding()*. It checks the conditions and results of the unfolding, giving some recommendations or warnings.
- Calls to the class *MCorrelatedFit*. *MCorrelatedFit* performs a fit to the gamma flux ($\Phi(E)$),taking into account all correlations. The class also produces plots of $\Phi(E)$ and of $E^2 \cdot \Phi(E)$ (see Section 6).

2.5 The classes MCorrelatedFit and MCalcXvalues

The class **MCorrelatedFit** is independent of the classes *MCallUnfold* and *MUnfold* and can thus be used also outside of these classes. It performs a fit of a function to data points, taking all correlations between the data points into account. For calculating the x positions at which the data points are to be plotted the class *MCalcXvalues* is called.
The input to *MCorrelatedFit* is defined via the functions :

- **DefineInput()** The data points Y, the covariance matrix K of Y, the overall relative uncertainty r of Y, the lower bin edges and other quantities are passed to the class in the argument list.
- **SetFunction()** Details about the function to be fitted to the data points are given as arguments of SetFunction().

The actual fit is performed by the call *FitFunction()*. The results of the fit are drawn by *DrawFitResults()*, and printed by *PrintFitResults()*.
The results of the fit consists of :

- the fitted values of the parameters of the function,
- the covariance matrix of the fitted parameters.
- a plot displaying the data points and their errors, the χ^2, the number of degrees of freedom and the χ^2-probabilty of the fit and a graph of the fitted function,
- the χ^2 contributions for each data point.

Often data points are given in certain bins of a variable, and the data points have the meaning of the average of some quantity in this bin. In the case of MAGIC the data points are gamma fluxes in bins

of the true energy (E), and the gamma fluxes Φ_i are to be understood as the average gamma fluxes in the bins i :

$$\Phi_i \;=\; \frac{1}{ElowEdge_{i+1} - ElowEdge_i} \cdot \int_{ElowEdge_i}^{ElowEdge_{i+1}} \Phi(E)\; dE \tag{3}$$

When drawing the gamma fluxes the question arises at which E_i they should be plotted. If the gamma fluxes are described by a function $f(E)$ a possible choice of E_i is that energy at which $f(E_i)$ agrees with the average of $f(E)$ in bin i :

$$f(E_i) \;=\; \frac{1}{ElowEdge_{i+1} - ElowEdge_i} \cdot \int_{ElowEdge_i}^{ElowEdge_{i+1}} f(E)\; dE \tag{4}$$

The class **MCalcXvalues** calculates for a given function $f(E)$ and for a given binning $ElowEdge_i$ the values E_i. *MCalcXvalues* also computes the corresponding $\overline{E_i}$ for the function $g(E) \;=\; E^2 \cdot f(E)$:

$$g(\overline{E_i}) \;=\; \frac{1}{ElowEdge_{i+1} - ElowEdge_i} \cdot \int_{ElowEdge_i}^{ElowEdge_{i+1}} g(E)\; dE \tag{5}$$

The ratio of the averages of $f(E)$ and $g(E)$ in each bin of E is given by $factorE2_i \;=\; g(\overline{E_i})/f(E_i)$. This factor is used to calcultate from the flux measurements Φ_i the corresponding data points for $E^2 \cdot \Phi_i$:

$$E^2 \cdot \Phi_i \;=\; factorE2_i \cdot \Phi_i \tag{6}$$

3 The format of the input files for the unfolding program

In the following a list is given of the objects contained in each of the input files for the unfolding program. Also the member functions of the respective classes, used in the unfolding program, are listed.

The input files are read by the member functions *GetDimensions(), GetInputData()* and *BookCombinedHistograms()* of the class *MCombineDataForUnfolding.*

- **The energy spectrum**
  The energy spectrum used in the flux macro when averaging the effective collection area and the migration matrix :
  TF1 *fSpectrum; name = "Spectrum"

- **The effective collection area**
  MHMcCollecxtionArea *collareaclass; name = "MHMcCollectionAreaEtrue"

  Member functions :

  - Calc(); to recalculate Aeff in coarse bins using the current energy spectrum.
  - GetHistCoarse(); to get Aeff in coarse bins of Etrue and Theta.
  - Get Hist(); to get Aeff in fine bins of Etrue and Theta.

- **The migration matrix**
  MHMcEnergyMigration *migclass; name = "MHMcEnergyMigration"

  Member functions :

  - SetHistCol(); to set Aeff to be used in Calc()
  - SetSpectrum(); to set the energy spectrum to be used in Calc().
  - Calc(); to recalculate the migration matrix in coarse bins using the current energy spectrum and the current Aeff.
  - GetHistMigCoarse(); to get the migration matrix in coarse bins.

- **The effective observation time**
  MHEffectiveOnTime effontimeclass; name = "MHEffectiveOnTime"

  Member functions :

  - GetTimeInCoarseZaBin(); to get Teff for the specified Theta bin.

- **The distribution to be unfolded**
  MHExcessEnergyTheta *excessclass; name = "MHExcessEnergyTheta"

  Member functions :

  - GetHist(); to get the distribution to be unfolded.

4 The format of the file containing the combined input data

The data which were combined by the class *MCombineDataForUnfolding* are written by the member function *WriteCombinedData()* onto a file, which is then read in by the class *MCallUnfold*.
This file contains the objects

- TH1D *fExcessEnergy : distriburtion to be unfolded (no.of excess events vs. Eest)
- TH2D *fMigrMatrix : migration matrix (vs. Eest and Etrue)
- TH1D *fColArea : effective collection area (Aeff vs. Etrue)
- TH1D *fEffOnTime : effective observation time (Teff vs. sample number)

5 Options and parameters for the unfolding

The options and parameters for the unfolding have to be set in the file *combunfold.rc*, which is read by *MCombineDataForUnfolding::ReadEnv()* and by *ReadEnv()* in *MCallUnfold::InitializeUnfold()*. In the *combunfold.rc* file each option or parameter is recognized by its identifier. Below is a listing of all identifiers, with explanations of the corresponding options and parameters.

1. **MCombineDataForUnfolding.NumFiles** number of input files

2. **MCombineDataForUnfolding.InputFiles[0]** path of 1st input file
3. **MCombineDataForUnfolding.InputFiles[1]** path of 2nd input file, etc.
4. **MCombineDataForUnfolding.OutputFile** path of output file
5. **MCombineDataForUnfolding.OutputLevel** output level
6. **MCombineDataForUnfolding.NSpectrumIterations** number of spectrum iterations to be done
7. **MCombineDataForUnfolding.ThetaBin** if = 0 the data of all Θ bins will be combined
   if > 0 the number of the Θ bin for which the data should be combined

8. **MCallUnfold.InputFile** The full path of the file containing the input histograms.
9. **MCallUnfold.IsData** This flag is 1 for experimnental data and 0 for MC data.
10. **MCallUnfold.OutputLevel** Possible values of the ouput level are
    - -1 suppress most of the output
    - 0 get errors and warnings
    - 1 normal output
    - 2 more output (dumps of arrays)
    - 3 maximum output (details of the MINUIT fit)
11. **MCallUnfold.ThetaBin** The input file usually contains the histograms for several bins of the zenith angle Theta. *Thetabin* indicates the number of the Theta bin to be treated in the unfolding.
12. **MCallUnfold.FlagSmoothing** Sometimes it can be useful to smooth the migration matrix in order to remove statistical fluctuations. The smoothing is activated by setting *FlagSmoothing* equal to 1. Several algorithms have been implemented to perform the smoothing like a smoothing with a fit (default one), box average, and gaussian (or flat) smearing.
13. **MCallUnfold.FlagPrior** The unfolding algorithms with regularization or iteration need some initial guess of the unfolded distribution to begin with. This somehow "a priori knowledge" can be set as a constant function (*FlagPrior = 1*), as a power law (*FlagPrior = 2*), as a user-specified distribution (*FlagPrior = 3*) or as the rebinned original measured distribution (*FlagPrior = 4*).
14. **MCallUnfold.Gamma** When the prior distribution is chosen to be a power law (*FlagPrior = 2*), the user has to specify the value of the power (*Gamma*).
15. **MCallUnfold.nminAnmaxA** Bins with missing or bad measurements can be removed by selecting ranges in the estimated quantity Eest, in terms of the first (nminA) and last (nmaxA) Eest bin, to be considered in the unfolding.

16. **MCallUnfold.nminBnmaxB** Also for the true quantity Etrue a range of bins can be selected (nminB, nmaxB). This is useful and necessary if some bins are expected to have no or only little influence on the unfolding, implying that the unfolded distribution cannot be determined in these bins. This is for example the case if the corresponding columns of the migration matrix contribute only very little to the selected bins of the estimated quantity, or if the acceptance (effective collection area) in these bins is extremely low. Keeping these bins in the unfolding can be the reason for non-convergence of the minimization.

17. **MCallUnfold.RangeAutoSelectA** Should be set to 1 if the Eest range, to be used in the unfolding, should be selected automatically.

18. **MCallUnfold.RangeAutoSelectB** Should be set to 1 if the Etrue range, to be used in the unfolding, should be selected automatically.

19. **MCallUnfold.MinAcc** The automatic selection of bins in Etrue is made such that all bins beyond the range have an acceptance $<$ MinAcc.

20. **MCallUnfold.FlagUnfold** This flag selects the unfolding method to be used. Currently, the unfolding program offers 6 different algorithms (see [1]).

    - *FlagUnfold = 1* The method of Reduced Cross Entropy by M. Schmelling [3]. The regularization term is the reduced cross entropy and the χ^2 is minimized using the Gauss-Newton method.
    - *FlagUnfold = 2* Tikhonov's method [4]. The regularization term is a sum of "second derivatives" of the unfolded distribution. The χ^2 is minimized using MINUIT.
    - *FlagUnfold = 3* Bertero's method [5] consists in calculating a solution iteratively and stopping the iteration at some point. In the limit of infinite iterations the solution tends to the special solution S_0 ([1]).
    - *FlagUnfold = 4* The Forward Unfolding. A certain parametrization is assumed for the true energy distribution and the free parameters are determined using MINUIT.
    - *FlagUnfold = 5* Schmelling's method where the χ^2 is minimized using MINUIT.
    - *FlagUnfold = 6* A modified Bertero method in which the solution tends to the LSQ solution ([1]).

21. **MCallUnfold.FlagStart** When Bertero's algorithm is selected (FlagUnfold = 3 or 6) two choices for the initial solution are foreseen: a vanishing distribution (*FlagStart = 1*); the measured distribution Y (*FlagStart = 2*).

22. **MCallUnfold.FlagCriterion** In the literature various criteria are proposed for choosing the "best" weight, or, in the case of an iterative method, the "optimal" number of iterations. None of these criteria is equally good for all methods, as the optimal weight depends on the shape of the solution and/or also on the prior distribution. The different criteria are

    - Choose that weight at which the noise component (*Trace(T)*) of the solution changes most when changing the weight (*FlagCriterion = 1*).
    - Choose the Least Squares Solution (LSQ), which is equivalent to taking the highest weight (*FlagCriterion = 2*).

- Choose that weight for which χ_0^2 is close to the number of significant measurements (*FlagCriterion = 3*).
- Choose that weight for which χ_0^2 is close to the rank of the matrix G [1] (*FlagCriterion = 4*).
- Choose that weight for which $Trace(T) = Trace(K)$ is closest to 1 (*FlagCriterion = 5*).
- Choose that weight for which the squared difference between the unfolding result and the ideal distribution is minimal (only for MC, or when the true distribution is known from somewhere else) (*FlagCriterion = 6*).
- (*FlagCriterion = 7*) Not used.
- (*FlagCriterion = 8*) Choose the weight given by the bin number IterBin.

23. **MCallUnfold.IterBin** If FlagCriterion is equal to 8, a specific weight can be selected by specifying the bin number (*IterBin*) of the weight out of the 45 trial weights.

24. **MCallUnfold.F1Type** By *F1Type* one specifies the type of the function to be fitted to the unfolded gamma flux or to be used in the Forward Unfolding :

- *F1Type = 1* selects a simple power law:
  $dN_\gamma/(dt\ dA\ dE) = f_0 \cdot (E/r)^\alpha$, with the parameters f_0, α and r. By a proper choice of the parameter r the correlation between f_0 and α can be reduced. Its value can be set by the user, however, it must not be varied in the fit.
- *F1Type = 2* selects a power law with a cutoff:
  $dN_\gamma/(dt\ dA\ dE) = f_0 \cdot (E/r)^\alpha \cdot \exp(-E/E_{cut})$, with the parameters f_0, α, E_{cut} and r.
- *F1Type = 3* selects a power law with a variable power index:
  $dN_\gamma/(dt\ dA\ dE) = f_0 \cdot (E/r)^\alpha$, $\alpha = a + b \cdot \log 10(E/r)$, with the parameters f_0, a, b and r.
- *F1Type = 4* selects a power law with a variable power index and a cutoff:
  $dN_\gamma/(dt\ dA\ dE) = f_0 \cdot (E/r)^\alpha \cdot \exp(-E/E_{cut})$, $\alpha = a + b \cdot \log 10(E/r)$, with the parameters f_0, a, b, E_{cut} and r.
- *F1Type = 5* selects a broken power law:
  $dN_\gamma/(dt\ dA\ dE) = f_0 \cdot (E/r)^{\alpha 1} \cdot [1 + (E/E_0)^\beta]^{(\alpha 2 - \alpha 1)/\beta}$, with the parameters f_0, $\alpha 1$, $\alpha 2$, E_0, β and r.
- *F1Type = 6* selects a broken power law with a variable power index $\alpha 1$:
  $dN_\gamma/(dt\ dA\ dE) = f_0 \cdot (E/r)^{\alpha 1} \cdot [1 + (E/E_0)^\beta]^{(\alpha 2 - \alpha 1)/\beta}$, $\alpha 1 = a + b \cdot \log 10(E/r)$, with the parameters f_0, a, $\alpha 2$, E_0, β, b and r.
- *F1Type = 7* selects a broken power law with a cutoff:
  $dN_\gamma/(dt\ dA\ dE) = f_0 \cdot (E/r)^{\alpha 1} \cdot [1 + (E/E_0)^\beta]^{(\alpha 2 - \alpha 1)/\beta} \cdot \exp(-E/E_{cut})$, with the parameters f_0, $\alpha 1$, $\alpha 2$, E_0, β, E_{cut} and r.
- *F1Type = 8* selects a broken power law with a variable power index $\alpha 1$ and a cutoff:
  $dN_\gamma/(dt\ dA\ dE) = f_0 \cdot (E/r)^{\alpha 1} \cdot [1 + (E/E_0)^\beta]^{(\alpha 2 - \alpha 1)/\beta} \cdot \exp(-E/E_{cut})$, $\alpha 1 = a + b \cdot \log 10(E/r)$, with the parameters f_0, a, $\alpha 2$, E_0, β, E_{cut}, b and r.

25. **MCallUnfold.Npar** Npar is the number of values to be read for the subsequent identifiers.

26. **MCallUnfold.ParamVinit** Starting values of the parameters for the function specified by F1Type.
27. **MCallUnfold.ParamStep** Initial step sizes of the parameters for the function specified by F1Type.
28. **MCallUnfold.ParamLimlo** Lower limits of the parameters for the function specified by F1Type.
29. **MCallUnfold.ParamLimup** Upper limits of the parameters for the function specified by F1Type.
30. **MCallUnfold.ParamFix** Has to be set to 1 for those parameters which should be kept fixed in the fit. Otherwise it has to be set to 0. In all functions, the parameter r has to be kept fixed in the fit.
31. **MCallUnfold.RelErrorMax** In the correlated fit to the unfolded gamma flux only those data points should be considered which have a relative error $<$ RelErrorMax.

6 Plots produced by the Unfolding program

6.1 Plots produced by *MCombineDataForUnfolding*

Two sets of plots are produced by *MCombineDataForUnfolding::PlotCombinedData(), SuperimposeSample()* and by *MCombineDataForUnfolding::PlotSpectrum(), PlotSpectrumIteration()* respectively.

Combination Category: There is one canvas of this type for each flux iteration step (see Figure 1). The first plot of a canvas shows the individual distributions to be unfolded, and the sum of these distributions. The second plot displays the individual migration matrices and the combined one. In the third plot the individual effective collection areas are compared with the combined one. In the last plot the effective observation time is plotted for the individual data sets and for the combined one.

Flux iteration Category: This canvas (see Figure 2)shows as the first plot a comparison of the flux shapes used in the different iteration steps. In the second plot the combined migration matrices and in the third plot the combined effective collection areas are compared for the different iteration steps.

6.2 Plots produced by *MCallUnfold* and *MUnfold*

The method *MUnfold::DrawPlots()* draws all the histograms which are of interest in the unfolding. Each histogram belongs to one of the three categories: *input, iteration* or *results*. Additional categories, *orig* and *corrfit*, are produced by the methods *MCallUnfold::DrawOrigPlots()* and *MCorrelatedFit::DrawFitResults()* respectively.

Input Category: The data used to perform the unfolding are put into this category. A canvas (see Figure 3) with the following eight plots appears at the end of the program execution:

- Distribution Y of the measurements (open circles) compared to the distribution $M \cdot S$ (red bars
- Prior distribution
- Migration Matrix M

- Smoothed Migration Matrix
- Eigen values of the matrix $G = M \cdot M^T$
- Eigen values of the matrix $GW = M^T \cdot K^{-1} \cdot M$
- Covariance matrix K of the measurements Y
- Acceptance due to the bin selection in the estimated quantity. For a given bin of the true quantity, the acceptance is defined as the fraction of the migration matrix contained in the selected bins of the estimated quantity.

Iteration Category:
The solution of the unfolding procedure depends on the weight w, or in the case of Bertero's method, on the number of iterations. Low w or a small number of iterations correspond to strong regularization. High w or a large number of iterations correspond to little or no regularization.

In this category, different quantities are plotted as a function of the weight, to see the effect of the regularization. The full circles indicate the values of the various quantities obtained when doing the unfolding with the "best" weight.

- χ_0^2 : One observes a decrease of χ_0^2 as the weight increases, which means that the convoluted solution $M * S$ gets more similar to the measured distribution Y as the degree of regularization decreases. A solution to be acceptable must not have a too large χ_0^2.
- Squared difference between the unfolded and the true distribution : Obviously this quantity can only be calculated if the true solution is known, like in Monte Carlo studies. This quantity is very useful for getting an idea about the choice of the optimal weight w.
- *Trace*(A^R) (black) and *Trace*(R) (blue) : These quantities characterize the resolution of the result S. The higher their value, the higher is the resolution and the lower is the bias of S. Low values indicate a bad resolution of S or a strong bias.
- *Trace*(T)/ *Trace*(K) : The ratio of the noise component *Trace(T)* of the solution S and the noise component *Trace(K)* of the measurements Y. A large value would indicate a solution with a large noise component.
- Second derivative of S: Small values are expected for a smooth solution, whereas strongly fluctuating solutions will give large values. Usually, with increasing strength of the regularization (corresponding to decreasing weights) the solution becomes smoother and the second derivative decreases.
- Cross Entropy : This quantity measures the deviation of the solution S from the prior distribution. Small entropies mean good agreement.

Results Category:
Within this category one finds the final results of the unfolding obtained with the "best" weight, as indicated by the full circles in Figure 4:

- Unfolded distribution S before correcting for the acceptance (green points) compared to the measured distribution Y (open circles). Note that the measured distribution has been rescaled to take the different bin sizes in Eest and Etrue into account. It should be noted that similarity of the two distributions is only expected if the measured (estimated) quantity differs only little from the true quantity. Similarity is by no means a condition for a good unfolding.
- Covariance matrix T of the unfolded distribution S: This matrix shows the error of the unfolded distribution and also the correlation between different bins. A good unfolding result is characterized by small correlations.
- Contributions to χ_0^2 from the different bins of the measured distribution : These are the χ^2 values for the comparison shown in the first figure of the Input Category.
- Unfolded distribution S, before (green points) and after (red points) correcting for the acceptance. If the ideal distribution is known it is also superimposed. For a good unfolding the corrected unfolded distribution (red points) should be similar to the ideal one.

Original Category:

This is a special category containing histograms in the original binning, that is before any bin selection.

- Measured distribution Y of excess events. The lines indicate the bins in the estimated variable selected for the unfolding
- Migration matrix M and lines indicating the bins selected in both, the estimated (green) and the true variable (blue)
- Unfolded distribution S of excess events : green points before and red points after correcting for the acceptance
- Effective collection area as a function of the true variable
- Unfolded flux with a power-law fit
- Acceptance due to the selection of bins in the estimated quantity

Correlated fit Category:

In this category the result of the correlated fit are displayed :

- Flux data points as a function of E_{true} with the fit result superimposed as a solid line. If some data points were not used in the fit (because of too large errors) they are drawn as dashed crosses
- Data points for E^2 times the flux as a function of E_{true}. The solid line represents E^2 times the function fitted to the flux data points.
- χ^2 contribution in each bin of E_{true}
- Residual (signed sqrt of χ^2) in each bin of E_{true}

7 Judging the unfolding

- **Selection of ranges in Eest and Etrue** A proper selection of ranges in Eest and Etrue to be used in the unfolding is necessary for several reasons :

  - It makes no sense to include bins in Eest which contain a bad or no measurement. However, it is not recommended to exclude such a bin if it is surrounded by selected bins.
  - It makes no sense to include a bin in Etrue whose content cannot be determined. This is the case if the acceptance for this bin is very small or zero. Including such a bin in the unfolding may would unnecessarily increase the number of unknows and may cause non-convergence in the minimization.
  - If the effective collection area is very small or zero in an Etrue bin, this bin should be excluded in the unfolding because in this case the contributions from this bin to the measured Eest distribution is expected to be negligible.

- **Bin sizes for Eest and Etrue** The number of constraints in the unfolding is equal to the number of selected bins in Eest (fNa). For the standard methods of unfolding, the number of unknowns is given by the number of selected bins in Etrue (fNb). Using the constraint that the total number of events is not changed by the unfolding the number of unknowns is reduced by 1. The system is thus underconstrained if fNa < fNb-1, and overconstrained if fNa > fNb-1. The unfolding can also handle underconstrained systems. However, in this case stronger regularization is needed which may lead to biases. Therefore it is advisable to make the system overconstrained. This can be achieved by making the bin size in Etrue sufficiently large. The bin size in Eest is usually defined by the requirement that the number of excess events in this bin can be determined sufficiently well. For an overconstrained system only little regularization is needed, making the unfolding result more stable and reliable.

  For the Forward unfolding the number of unknowns is equal to the number of free parameters (fNpar) of the function assumed to describe the gamma flux. Forward Unfolding is only reasonable if the system is overconstrained, i.e. fNa > fNpar.

- **Unfolding method** In principle any unfolding method should give an acceptable unfolding result. However, due to technical problems like non-convergence of the minimization some methods may fail. In this case the result from another method may be accepted. Another reason for a failure of the unfolding may be inconsistencies in the measurements like biases in the measurements which are not (or badly) described by the migration matrix. In this case a reliable unfolding result cannot be expected.

- **Optimum choice of the weight** On the basis of certain criteria, the program determines the "best" weight (or "best" iteration number) automatically. The plots of the **Iteration category** can help to decide whether the selected weigth, and thus the selected solution, is the proper one. The main criteria are : the χ_0^2 must not be too high and the noise component of the unfolded distribution ($Trace(T)$) must not be much higher than the noise component of the measurements ($Trace(K)$). A situation $Trace(T) >> Trace(K)$ either points to inconsistencies in the measurements or may indicate that the migration matrix does not describe the energy migration properly. Note that the user can choose his own preferred weight using the option *MCallUnfold.FlagCriterion: 8* and *MCallUnfold.IterBin*.

- **Criteria for an acceptable unfolding result** A good check of the quality of an unfolding result is a comparison of the results from different methods. Agreement of the different results within the errors is a necessary condition for a result to be acceptable. Note that exact agreement between the different results is not expected because the regularization conditions are different for the different methods.

  Very useful is also a comparison with the result from the Forward Unfolding, provided the Forward Unfolding was successful. If the result from one of the standard methods (which make no assumption about the shape of the unfolded distribution) is compatible with the result from the Forward Unfolding (which makes an explicit asumption about the shape of the unfolded distribution), this can be regarded as a kind of confirmation of the Forward Unfolding result.

References

[1] W. Wittek, *Unfolding*, MAGIC-TDAS 05-05 (2005).

[2] V. Blobel, *Unfolding methods in high-energy physics experiments*, DESY 84-118 (1984).

[3] M. Schmelling, NIM A 340 (1994) 400.

[4] A.N. Tikhonov and V.Ja. Arsenin, *Methods of Solution of Ill-posed Problem - M* (Nauka, 1979).

[5] M. Bertero, INFN/TC-88/2 (1988).

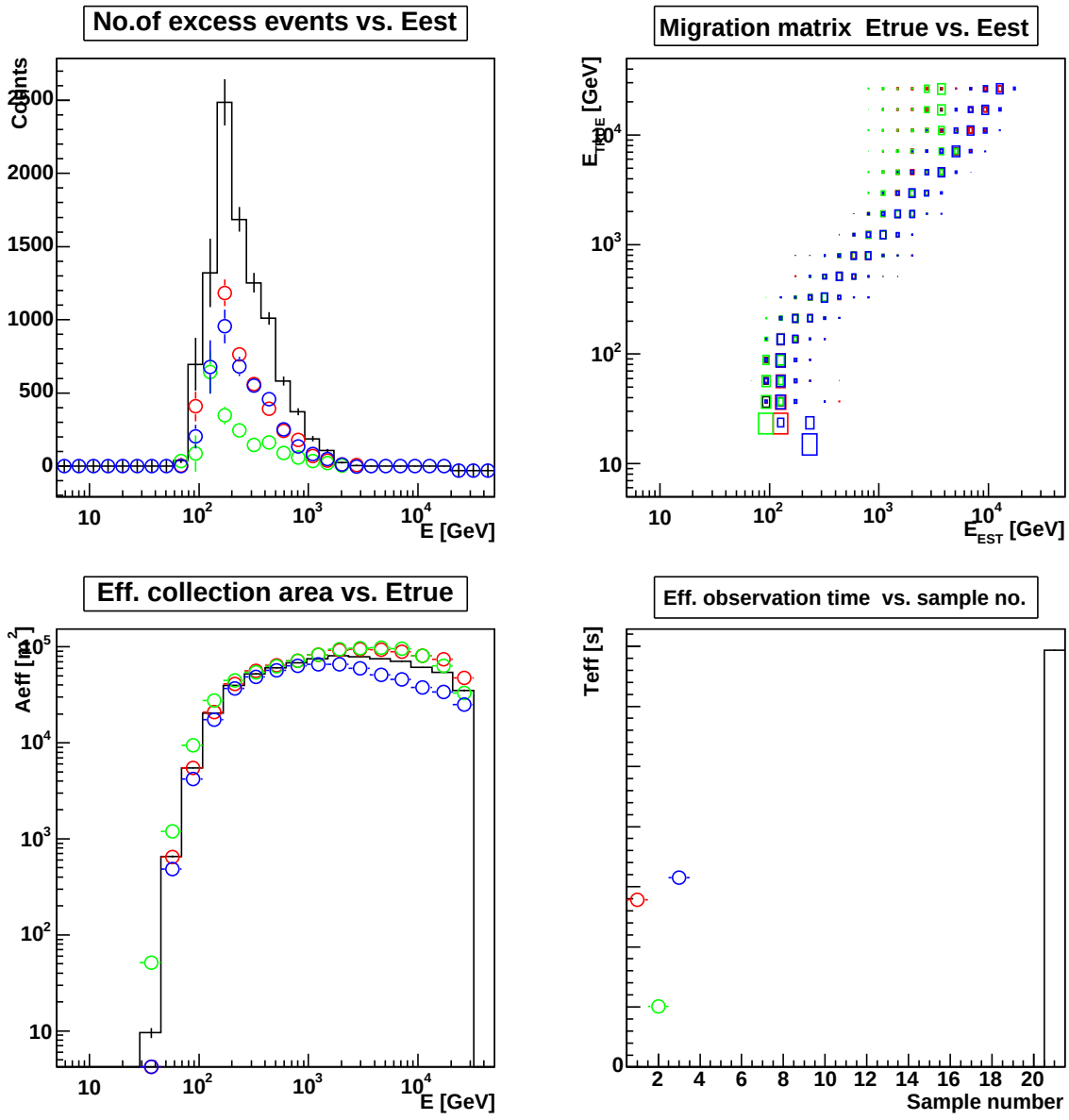

Figure 1: Combination category plots

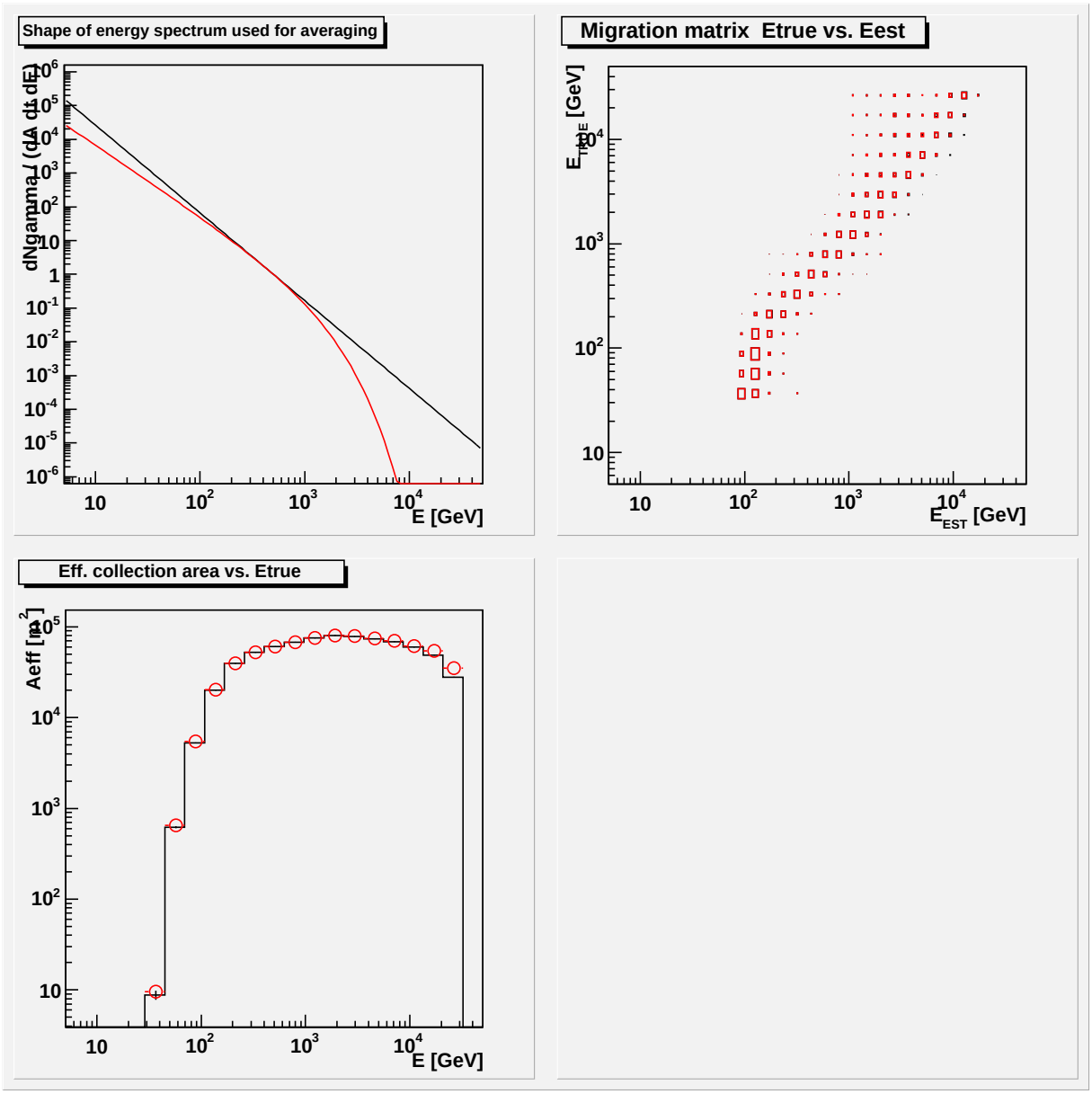

Figure 2: Flux iteration plots

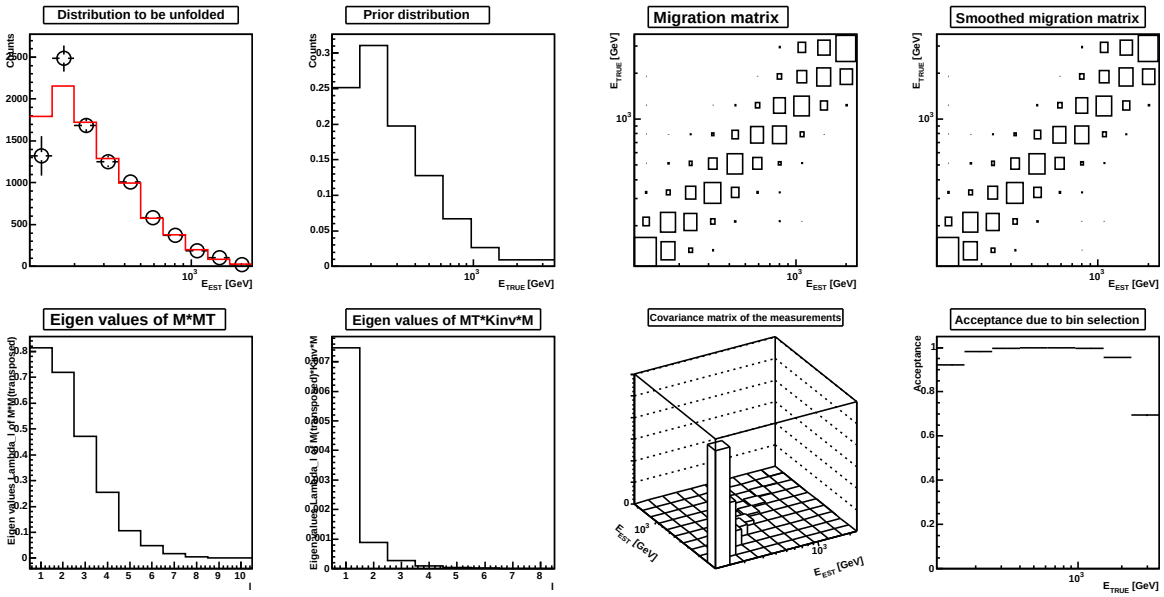

Figure 3: Input category plots

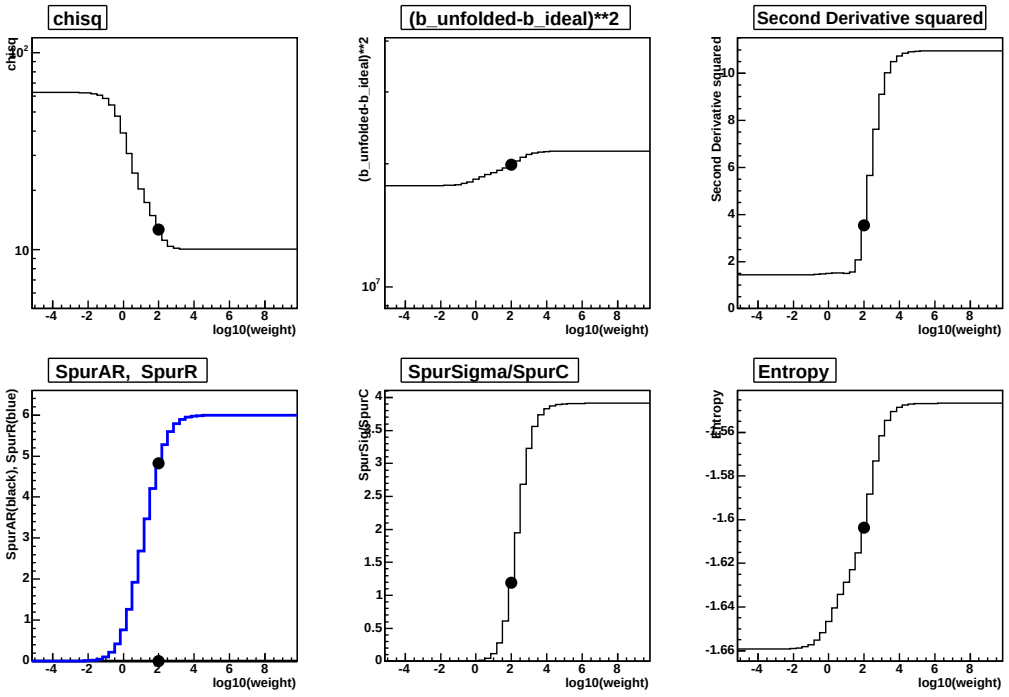

Figure 4: Iteration category plots

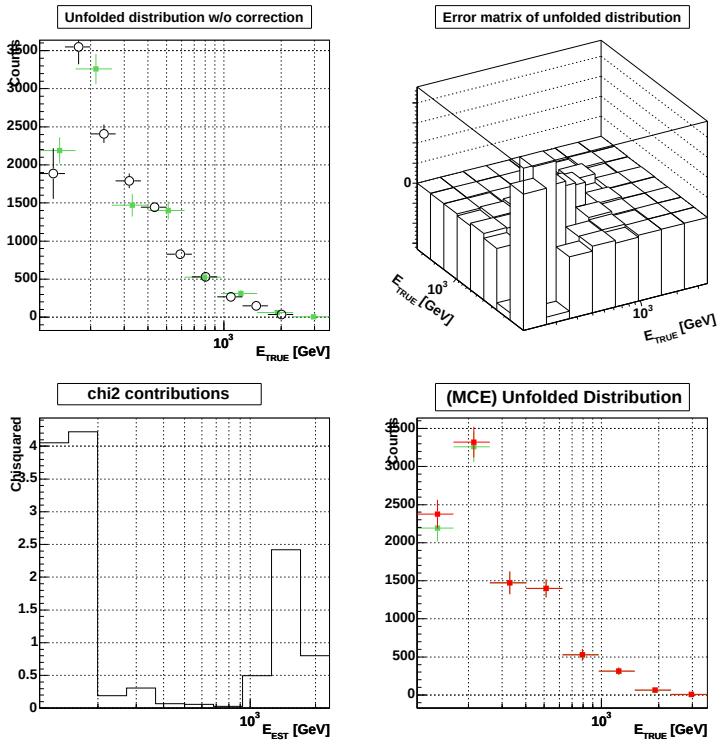

Figure 5: Result category plots

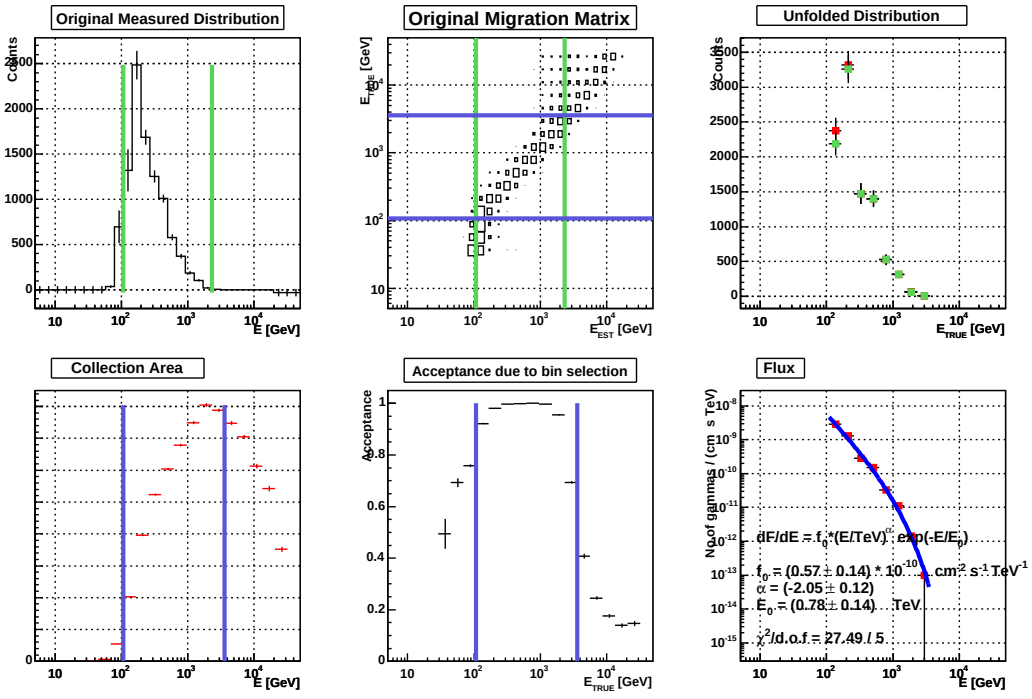

Figure 6: Original category plots

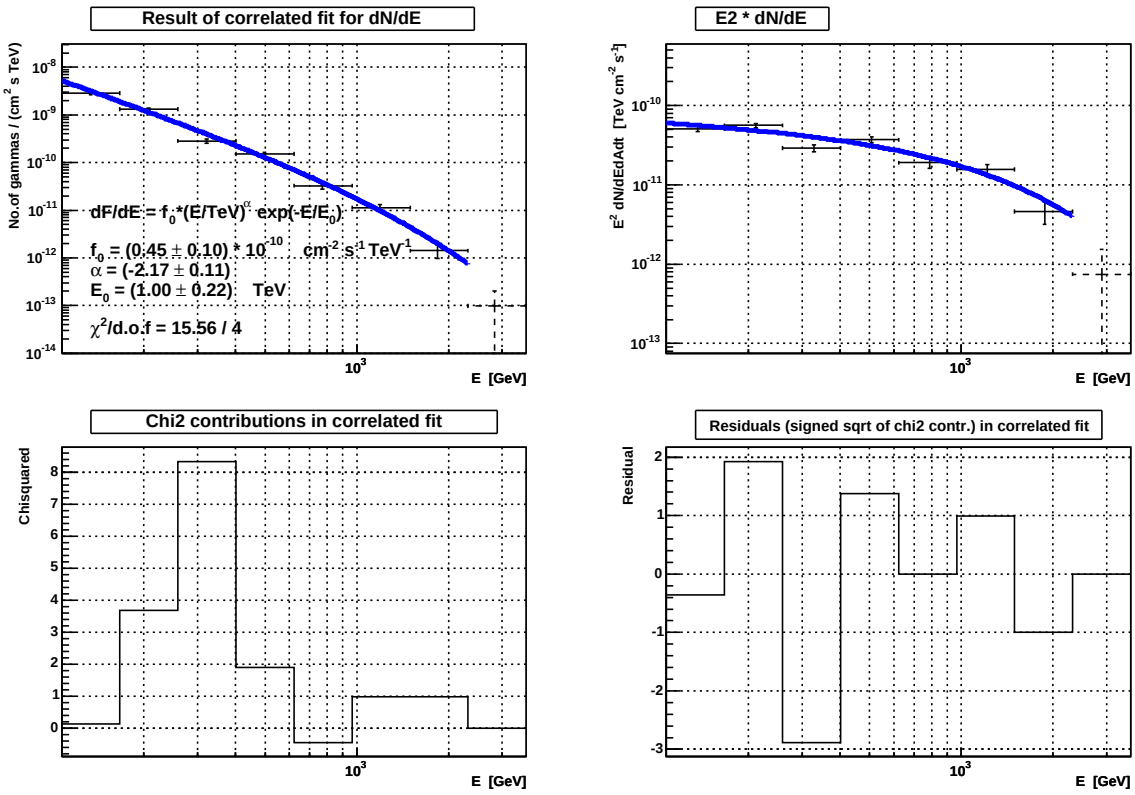

Figure 7: Correlated fit plots