Document Version: 1.1,
Date:24/01/2008

The Correction Framework

A brief description of the current software and its functionalities

Contributors: S.Arcelli, I.Kraus, A.Mastroserio and R.Vernet

Introduction:

The classes that have been developed so far with the purpose of assisting the ALICE users in deriving the corrections for their analyses can be grouped into two main categories:

- **"Container" Classes**
- **"Selection" Classes**

which reflect the main utilities provided by the Correction Framework (CF):

- The possibility to store, while performing analysis, both real and simulated data over binned N-dimensional grids, to then derive the efficiency correction maps and correct the observed data.

- The coding of general selections which may be common to several analyses, at different stages of the selection process (for example, generator, acceptance, reconstruction, user-specific analysis selection...), with the optional possibility of accumulating control histograms on the selection variables (intended to be the base for the user to perform the correction "QA").

In the following, some information is given on the structure of the code and its functionalities, together with some examples serving as basic guidelines for the usage of the framework classes.

Container Classes:

The general schema of the CF Container classes is shown in Fig.1.

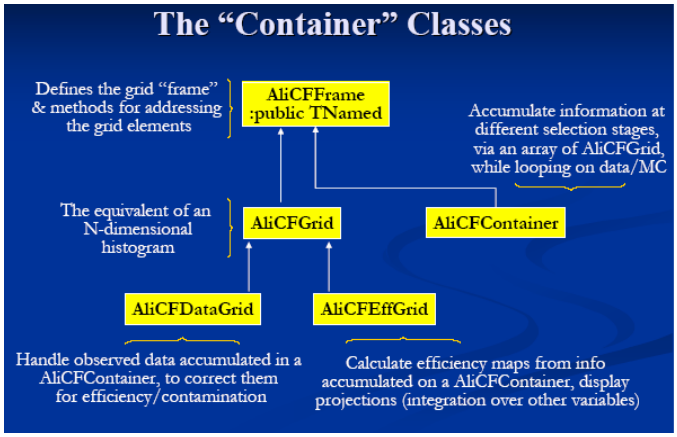

Figure 1. General schema of the CF Container Classes, indicating the main functionalities of each class and the relationships between them.

These classes can be used to store counts (weighted or un-weighted) over N-dimensional grids while looping

on real/simulated event samples at various selection levels. Compared to 'conventional' histograms, the additional functionality provided by these classes is that the dimension of the grid is not restricted to a maximum of three dimensions, but can be any.

The list of Container classes is reported below, together with a brief description. For all of them, additional information is documented in the header/implementation file of the class. All these classes inherit from TNamed, and are all stream-able objects.

AliCFFrame:

This is the base class of the whole set of the CF Container classes: it just defines the structure (the "frame") of the N-dimensional grid used to store the correction maps. This structure is defined by the number of dimensions of the grid (corresponding to the number of sensitive variables), the number of bins and binning (also variable-sized) in each dimension. Public Getters/Setters are implemented to access the grid-defining parameters, for example:

virtual Int_t GetNVar(): number of dimensions on the grid (the number of sensitive variables)
virtual Int_t GetNBins(Int_t ivar): number of bins along variable ivar
virtual void GetBinLimits(Int_t ivar, Double_t * array): array of bin limits along variable ivar
virtual Double_t GetBinCenter(Int_t ivar,Int_t ibin): center of bin # ibin in variable ivar
virtual Double_t GetBinSize(Int_t ivar,Int_t ibin): size of bin # ibin in variable ivar

For additional methods and their description, the user may refer to the header/implementation files.

AliCFGrid:

The N-dimensional grid is practically the equivalent of an N-dimensional histogram. To store the contents, the N-dimensional structure of the grid is converted into a linear array of Floats of dimension $\prod_{i=1,Nvariables} N_i^{bins}$.

The user can configure the dimensionality of the grid, the number of bins and the binning in each dimension. The class allows to perform basic operations as in the case of "conventional" TH1,2,3 histograms, like Add, Multiply, Divide (optionally binomial), Scale, calculate the integral over a range, perform 1,2,3-D projections, etc. For example, the following methods are implemented:

```
//basic operations
  virtual void Add(AliCFGrid* aGrid, Double_t c=1.);
  virtual void Add(AliCFGrid* aGrid1 ,AliCFGrid* aGrid2, Double_t c1=1.,Double_t c2=1.);
  virtual void Multiply(AliCFGrid* aGrid, Double_t c=1.);
  virtual void Multiply(AliCFGrid* aGrid1,AliCFGrid* aGrid2, Double_t c1=1.,Double_t c2=1.);
  virtual void Divide(AliCFGrid* aGrid, Double_t c=1.,Option_t *option=0);
  virtual void Divide(AliCFGrid* aGrid1, AliCFGrid* aGrid2, Double_t c1=1., Double_t c2=1.,Option_t *option=0);
  virtual void Scale(Int_t* bin, Double_t *fact);
//projections
  virtual TH1D* Project( Int_t ivar) const;
  virtual TH2D* Project( Int_t ivar1, Int_t ivar2) const;
  virtual TH3D* Project( Int_t ivar1, Int_t ivar2, Int_t ivar3) const;
  virtual TH1D* Slice( Int_t ivar, Double_t *varMin, Double_t *varMax) const;
//integral
  virtual Float_t GetIntegral() const; // total integral
  virtual Float_t GetIntegral(Int_t *binMin,Int_t *binMax) const; //integral in range
  virtual Float_t GetIntegral(Double_t *varMin,Double_t *varMax) const; // integral in range
```

The filling of the grid, which includes the possibility to handle weighted entries, is performed via the call:

```
  virtual void  Fill(Double_t *var, Double_t weight=1.);
```

A number of Setters/Getters are also available, for example to retrieve/set the value of a certain grid element, or its uncertainty:

```
//get/set content & error in the N-dim cell having bin indeces *bin
  virtual Float_t GetElement(Int_t *bin)const;
```

```
  virtual Float_t GetElementError(Int_t *bin)const;
  virtual void SetElement(Int_t *bin, Float_t val);
  virtual void SetElementError(Int_t *bin, Float_t val);
  //get/set content & error in the  N-dim cell corresponding to the  set of values of the sensitive variables *var
  virtual Float_t GetElement(Double_t *var)const;
  virtual Float_t GetElementError(Double_t *var)const;
  virtual void SetElement(Double_t *var, Float_t val);
  virtual void SetElementError(Double_t *var, Float_t val);
```

Together with simple methods to monitor the statistics accumulated in the cells:

```
// grid statistics
  virtual Int_t GetEmptyBins(Double_t *varMin,Double_t *varMax) const ; // empty bins in a range
  virtual Int_t CheckEfficiencyStats(Float_t thr) const; //number of cells having their content below threshold thr
// total entries, over/underflows in each dimension.
  virtual Float_t GetEntries()const ;
  virtual Float_t GetOverFlows(Int_t var) const;
  virtual Float_t GetUnderFlows(Int_t var)const ;
```

Indeed, the way the contents are stored in this implementation of the N-dimensional grid (the grid size is allocated independently from the fact that a given cell is filled or not) may lead to a significantly inefficient memory usage in the case of a high number of dimensions (sensitive variables), high number of bins and when a high fraction of cells in the grid has a very low occupancy, and this may cause problems when looping over data, especially in the context of organized analyses.

Therefore, a dedicated class (THnSparse) has been developed within the ROOT package, which allows a more efficient memory usage. The N-dimensional grid implementation based on THnSparse (for a stable version of this class, please use root tags starting from v5-18-00 release) , that in the near future is going to replace the current implementation in AliCFGrid, is already available in **AliCFGridSparse.** Its usage in a AliCFContainer can be activated via an input argument of the AliCFContainer constructor.
Finally, **AliCFVGrid** is just an interface class to be able to handle both AliCFGrid and AliCFGridSparse-type objects in a transparent way.

AliCFContainer:
This class is used to book and fill a group of N-Dimensional grids. The user can configure the grids and the number of levels he wants to monitor by accumulating the data (either real or simulated) at different selection steps in his analysis. The efficiency between two different selection steps can then be derived by dividing the contents of the two corresponding grids, to be then applied to the appropriate set of observed data.

The filling of the grid corresponding to selection step *istep* is performed calling the function:

```
 virtual void  Fill(Double_t *var, Int_t istep, Double_t weight=1.);
```

The user may perform 1,2,3-D projections of the content of the grid at selection step istep via:

```
  virtual TH1D* ShowProjection( Int_t ivar, Int_t istep) const;
  virtual TH2D* ShowProjection( Int_t ivar1, Int_t ivar2, Int_t istep) const;
  virtual TH3D* ShowProjection( Int_t ivar1, Int_t ivar2,Int_t ivar3, Int_t istep) const;
  virtual TH1D* ShowSlice( Int_t ivar, Double_t *varMin, Double_t *varMax, Int_t istep) const;
```

Additional methods are documented in the header/implementation file of the class.

AliCFEffGrid and AliCFDataGrid:
Two additional classes, so far including very simple functionalities, have been prepared to handle the efficiency maps and the observed/corrected data.
The first class, AliCFEffGrid, can be used to derive the efficiency correction map based on the information - as stored in an AliCFContainer. To connect the information stored in the AliCFContainer, the user has to

call:
virtual void SetContainer(const AliCFContainer &c);

Then the efficiency corresponding to two different selection steps istep1 (numerator) and istep2 (denominator) can be derived calling the method:
virtual void CalculateEfficiency(Int_t istep1, Int_t istep2)

In the efficiency correction calculation, binomial errors are always assumed, and the number of bins which are found to be empty either in the numerator or in the denominator of the efficiency are also monitored (this may serve as a rough indicator that the statistics used to estimate the correction is not enough).
Dedicated methods to perform 1,2,3-D projections of the efficiency are also implemented: in this case the numerator and the denominator are projected first, then their ratio is taken as the projection of the resulting efficiency. Methods to get the average efficiency:
virtual Double_t GetAverage() const ;
virtual Double_t GetAverage(Double_t *varMin,Double_t *varMax) const ;
over the whole grid, or in a range, are also implemented.

For what concerns class AliCFDataGrid, it is used to deposit the measured data as stored in a AliCFContainer at a given selection step, via calls to methods:
virtual void SetContainer(const AliCFContainer &c);
virtual void SetMeasured(Int_t istep);

Then the observed data are corrected for efficiency (stored in the AliCFEffGrid object calculated above) via the method:
virtual void ApplyEffCorrection(const AliCFEffGrid &eff);

When applying the correction, those cells which have a non-zero content in the observed data, but whose estimated efficiency is zero, are set to zero in the corrected data, and their number is monitored and printed out as an information for the user.

So, a typical flow of the correction process (see Fig.2) may be as follows: the user will first loop over simulated data and accumulate information on the N-dimensional grids using AliCFContainer-type objects, in order to derive the overall efficiency correction and also monitor its components (for example, the acceptance, the reconstruction efficiency, the efficiency of his last-step analysis selection). These correction maps, derived with and stored in AliCFEffGrid-type objects, will then be used to correct the observed data (deposited on a AliCFDataGrid-type object), which were accumulated over an equivalent grid via an AliCFContainer during a separate analysis loop on real events.

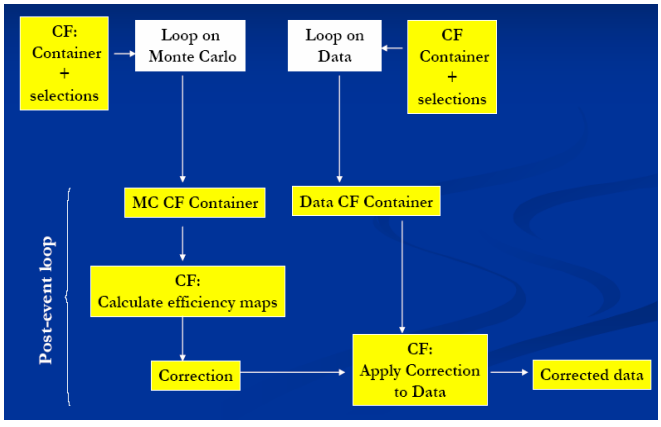

Figure 2. A sketch of the general flow of the correction process.

The N-dimensional grid classes described above have shown to have, for N<=3, a performance comparable

to that of "conventional" histograms; therefore, they may be used in their stead without any significant "penalty". Moreover, they can also be used to accumulate response matrices: indeed, in this case for N sensitive variables, 2N-dimensional grids are needed, and the dimensionality of the response matrix in most cases >=4.

Selection Classes:

A set of classes handling general (configurable) selections, both at Event or at Particle-level, and involving both generator and reconstruction-level quantities, have been prepared. Part of the implemented selections are based on already existing code in PWG0 aliroot repository (in AliPWG0Helper and AliESDtrackCuts). The Selection Classes include:

Event Level Selections:

- **Generator level**: Vertex position, multiplicity, MB Monte Carlo process type
- **Event "Class"**: Trigger (p-p configuration), energy observed in the ZDC
- **Reconstructed level**: Vertex position and resolution, multiplicity

Particle Level Selections:

- **Generator level**: PDG, being a primary/secondary, production/decay vertex
- **Acceptance level**: Basic cuts on the number of track references in barrel tracking detectors, TOF and MUON
- **Kinematic quantities:** Select both generated and reconstructed tracks of a given range in momentum space (total momentum, pt, px, py, pz, eta, rapidity), electric charge (given value or non-neutral) and azimuthal emission angle phi.
- **Track Quality Criteria:** Acts on reconstructed tracks only; selects tracks with more than a given number of clusters in the TPC or ITS, less than the given chi2 / cluster in the TPC or ITS, checks if the refit in TPC or ITS was successful and selects tracks based on covariance matrix diagonal elements
- **Conditions on being a Primary Track:** Acts on reconstructed tracks only; selects tracks with a small impact parameter (smaller than a given value or only successful calculation) and tracks which are (not) daughters of kink decays
- **PID Selection:** Dedicated class to perform Particle Identification at the ESD level (based on Bayesian PID)

and are organized as follows:

- All classes inherit from a base class AliCFCutBase, which in turn inherits from class AliAnalysisCuts in ANALYSIS.

- AliVParticle (AliVEvent) objects have been used whenever possible. This has the advantage that the same code can be used to apply selections on the kinematic properties on MC/ESD/AOD type track (event) objects

- For generator and acceptance level selections, the classes make use of the AliMCEvent/AliMCParticle interface (that also allows for an easy access to the Track Reference information)

- For classes handling a number of selections on different variables, the classes provide access to a bitmap keeping track of the cuts which were passed, or failed (to investigate losses, correlations in higher detail)

- The majority of these classes allow also to accumulate, while performing the selection, a set of diagnostic histograms of the variables used in the selections. These histograms, whose parameters (binning and range) are configurable by the user, can be used within the physisc analysis to perform correction "QA" (by comparing, for example, the distribution of the cut variables before and after the selection in real and simulated events).

Hereafter, a somewhat more detailed description of each of the selection classes can be found.

Base class for Selection Classes:

AliCFCutBase:

This class, which derives from AliAnalysisCuts, is the base class of all the CF selection classes.
It includes the following general methods (most of them having dummy implementation), that are then concretely coded in the derived classes and are related to the creation/filling and retrieval of the diagnostic histograms on the selection variables:

```
virtual Bool_t IsQAOn() const {return fIsQAOn;}; //QA flag getter
virtual void SetQAOn(Bool_t flagQA) {fIsQAOn=flagQA;}; //QA flag setter: request booking/filling of QA histos
virtual void FillHistogramsBeforeCuts(TObject* ) {;}; //Fill QA histos
virtual void FillHistogramsAfterCuts(TObject* ) {;}; //Fill QA histos
virtual void DefineHistograms(){;}; //Book QA histos
virtual void AddQAHistograms(TList*) const {;}; //Add QA histos to a TList
virtual void GetBitMap(TObject*, TBits*){;} //retrieve selection bitmap
virtual void SetEvtInfo(TObject *) {;}; //Pass pointer to event-level info (for selections which need access to some global info)
```

Other common methods used by all the CF Selection Classes are:

- Init()
- IsSelected(TObject *obj)

These methods are inherited from class AliAnalysisCuts.

Particle-Level Selections:

AliCFParticleGenCuts:

Derives from AliCFCutBase ; this cut class performs basic non-kinematic selections on generated particles (AliMCParticle) : PDG code, 3D- decay and production vertices, charge, primary/secondary etc.
Some of the member functions of more general interest are declared static for convenience.
Here the definition of a primary generated particle is given by the function AliStack::IsPhysicalPrimary(AliMCParticle*).

AliCFAcceptanceCuts:

Derives from AliCFCutBase ; this class performs selections on the number of track references (AliTrackReference) created by a generated particle (AliMCParticle) in the detectors ITS, TPC, TRD, TOF and MUON.
The current state of this class allows a basic (rough) calculation of acceptance corrections. However tools using a more complete information than the one available from the track references in order to implement more refined acceptance definitions are planned and still under discussion.

AliCFTrackKineCuts:

Derives from AliCFCutBase; it uses a pointer to AliVParticle and therefore can act on both generated and reconstructed tracks; this cut class performs selections on 10 different quantities (range in momentum space, charge, emission angle) and stores the decision on the individual cuts in a bitmap; it also provides QA histograms for cut quantities.

AliCFTrackQualityCuts:

Derives from AliCFCutBase; acts only on reconstructed tracks, uses a pointer to AliESDtrack; this cut class performs selections on 11 different quantities (number and chi2 of clusters, successful refit, covariance

matrix elements) and stores the decision on the individual cuts in a bitmap; it also provides QA histograms for cut quantities

AliCFTrackIsPrimaryCuts:

Derives from AliCFCutBase; acts only on reconstructed tracks, uses a pointer to AliESDtrack; this cut class performs selections on 3 different quantities (normalised 3d dca, successful calculation of dca, kink daughter) and stores the decision on the individual cuts in a bitmap; it also provides QA histograms for cut quantities and 2d displays of the (normalised) dca.

AliCFTrackCutPid

This class derives from AliCFCutBase and it is intended to select single tracks according their identification. The considered species are electrons, muons, kaons, pions and protons. The class acts specifically on AliESDtracks, by means of a single detector PID or of the combined PID.

In the most general case the user just needs to set the particle species to be selected, the identification mode (single/combined), the detector (or more detectors for the combined identification) and the a priori concentrations. The user can also set a momentum dependent a priori concentrations (*SetPriorFunctions*). The main method of the class is IsSelected: it loads the needed track information, asks for the identification and returns true if such an identity corresponds to the required specie. The identification (together with some further checks) is performed in the *GetID* method, which calls *Identify*. The *CombPID* method calculates the combined responses according to the selected detectors: here, the user has also the possibility to ask for the AND status between a set of PID detectors to estimate the combined responses.

The class provides some tools to have a more refined identification, as applying explicit cuts on the probability, asking for restrictions on the detector responses (*SetDetectorProbabilityRestriction*) and requiring a minimum allowed difference between the probability of the selected species and all the others (a cut on the minimum difference between the detector responses can also be required).

If requested, the class books and fills some QA histograms which consist of the chosen detector responses and (depending on the identification mode) the final probabilities for the five particle species.

Event-Level Selections:

AliCFEventGenCuts:

This class can be used to select Monte Carlo events on the base of conditions on:

- Generator-level Vertex Position
- Total Generated Particle Multiplicity
- Minimum Bias Process Type (Non Diffractive, Single Diffractive, Double Diffractive)

The class accepts as input argument for the IsSelected method a TObject, which is then cast into an AliMCEvent-type object. For what concerns the selection on the MB process type in Pythia pp events, the user has the following choices: Double Diffractive (defined by process id=94), Single Diffractive (defined by process id=92 || id=93) and Non-Diffractive (defined as being neither SD nor DD) Events. Static methods are also available, to check on the MB process type and to look at the process ID in pythia events:

```
static Int_t AliCFEventGenCuts:: IsMBProcType(AliMCEvent *ev,  PrType iproc);
static Bool_t AliCFEventGenCuts::ProcType(AliGenEventHeader *header);
```

are also available.

AliCFEventClassCuts:

This class allows the user to set requirements on the bits of the Trigger Mask and on the energy observed in different sections of the ZDC Calorimeters. It accepts as input argument for the IsSelected method a TObject, which is then cast into an AliVEvent-type object. At the moment, only the pp trigger configuration is implemented**.** The user can require that a set of trigger bits have fired, both in AND and OR configuration

(OR being the default). On top of the standard trigger descriptors in the pp Trigger configuration, the class allows also to check on the following Minimum Bias Trigger combinations:

- MB1: (ITS GFO || V0OR) && !BG
- MB2: (ITS GFO && V0OR) && !BG
- MB3: (ITS GFO && V0AND) && !BG
- MB4: (ITS GFO || V0AND) && !BG
- MB5: ITS GFO && !BG

A static checker AliCFEventClassCuts:: IsTriggered(AliVEvent *ev, TriggerType trigger=kMB1) is also implemented.
For what concerns the selection on the ZDC variables, the user can require the energy observed in the proton and/or neutron calorimeters on each side to be within a given range, and impose conditions on the EM energy observed on either side of the ZDC. Optionally, the user has the possibility to require the booking/filling of control histograms of the fired bits in the Trigger Mask, and of the energy observed in the individual sections of the ZDC calorimeter.

AliCFEventRecCuts:

This class can be used to select ESD events on the base of conditions on:

- Reconstructed Vertex 3-D Position and its uncertainty
- Number of ESD tracks

It accepts as input argument for the IsSelected method a TObject, which is then cast into an AliESDEvent-type object. The user has optionally the possibility to require the booking/filling of control histograms of the fired bits in the Trigger Mask, and of the energy observed in the individual sections of the ZDC calorimeter.

User Template Task and Macros

As an example of the usage of the CF classes described above, a template Analysis Task and configuration macro have been prepared:

AliCFSingleTrackTask:

This class derives from AliAnalysisTask ; it has therefore all the functionalities allowing to process a list of data files (AliESDs.root). This class is in addition provided with several directives to fill a container (AliCFContainer) as the events are processed for a single-particle analysis.
The Exec() function is divided in 2 steps :

1st step: - *loops on MC particles and fills the map at* generator *and* acceptance *levels*
2nd step : - *loops on reconstructed tracks and fills the map at* reconstructed *and* selected *levels*

Inputs:

TChain (the chain of events to be processed)

Outputs:

- *TH1I* (simple histogram that counts the number of events processed)
- *AliCFContainer* (container used to calculate the correction map afterwards)
- *TList* (list of QA histograms)

This class works under both local and AliEn sessions. Support to run on CAF is nearly finalized and will be provided very soon.

AliCFSingleTrackTask.C

This macro is an example of how-to-use the Correction Framework. Is consists of the following steps:

- connection to AliEn and creation of the data chain

- configuration of the container (number of variables, bins, number of selection steps, bin limits)
- creation of the selection cuts and initialization of the AliCFManager (see below)
- definition of the inputs and outputs of the task

The only input required is an XML file in which are specified the data tags.

The template Task and macro also make use of a dedicated class, **AliCFManager**, which serves as an interface to pass to the Analysis Task the N-dimensional Containers, Event-level and Particle-level selections that have been configured in the user macro. The class also includes methods to loop on the various cuts which were required at different selection steps, and collect the QA histograms and the selection bitmaps.

Finally, some additional information on the usage of the Container Classes (how to book/fill multidimensional Containers, derive from them an efficiency map and apply it to a container which stores the observed data) can be found in the macro **testCFContainers.C .**