The FLOW Analysis Package

a short writeup

January 26, 2014

Various authors, edited by Redmer Alexander Bertens (`rbertens@cern.ch`)

Contents

Chapter 1

Introduction

The intro to everything.

Chapter 2

A Quick Start

2.1 The flow package

The `ALICE flow package`[a] contains most known flow analysis methods. In this chapter we give a few examples how to setup an analysis for the most common cases. The chapters that follow provide more detailed information on the structure of the code and settings of the various flow methods. This write-up is however not a complete listing of the methods, for this the reader is referred to the header files.

2.2 On the fly - getting started on a Toy MC

To get started with the `flow package` we begin by generating a few simple Toy Monte Carlo events and performing a flow analysis on these simulated events. The steps which will be followed will be the same as when performing an analysis on data:

1. Prepare your `(Ali)ROOT` session by loaded the necessary libraries
2. Create the analysis method objects
3. Initialize the methods (which creates their histograms)
4. Define track cuts

---

[a] The `ALICE` flow package is part of `AliROOT`, the ALICE extension of the `ROOT` framework, which can be obtained from http://git.cern.ch/pub/AliRoot. The flow package itself is located in the folder `$ALICE_ROOT/PWG/FLOW/`, where `$ALICE_ROOT` refers to the source directory of `AliROOT`.

5. Create flow events, which is a container class holding all necessary information (e.g. tracks) for the flow analysis of an event (collision) and actually do the analysis
6. Finish the analysis, which will calculate the final v_n values
7. Write the results to an output file

In this Monte Carlo exercise, the flow event class will not receive data from a detector (e.g. an `NTuple`) , but instead generate toy events itself.

The macro `runFlowOnTheFlyExample.C`[b] is a basic example of how the flow package works. In this section we explain the main pieces of that macro.

1. To use the flow code the flow library needs to be loaded. In `AliROOT`:

```
gSystem->Load("libPWGflowBase");
```

   In `root` additional libraries need to be loaded:

```
gSystem->Load("libGeom");
gSystem->Load("libVMC");
gSystem->Load("libXMLIO");
gSystem->Load("libPhysics");
gSystem->Load("libPWGflowBase");
```

2. We need to instantiate the flow analysis methods which we want to use. In this example we will instantiate two methods: the first which calculates the flow versus the reaction plane of the Monte Carlo, which is our reference value (see section 5.1), and second the so called Q-cumulant method (see section 5.4).

```
AliFlowAnalysisWithMCEventPlane *mcep = new
    AliFlowAnalysisWithMCEventPlane();
AliFlowAnalysisWithQCumulants *qc = new
    AliFlowAnalysisWithQCumulants();
```

3. Each of the methods needs to initialize (e.g. to define the histograms):

```
mcep->Init();
qc->Init();
```

4. To define the particles we are going to use as Reference Particles (RP's, particles used for the **Q** vector) and the Particles Of Interest (POI's, the particles of which we calculate the differential flow) we have to define two track cut objects:

---

[b] In aliroot, this macro can be found at
`$ALICE_ROOT/PWGCF/FLOW/Documentation/examples/runFlowOnTheFlyExample`

```
AliFlowTrackSimpleCuts *cutsRP = new AliFlowTrackSimpleCuts();
AliFlowTrackSimpleCuts *cutsPOI = new AliFlowTrackSimpleCuts();
cutsPOI->SetPtMin(0.2);
cutsPOI->SetPtMax(2.0);
```

5. Now we are ready to start the analysis. For a quick start we make an event on the fly, tag the reference particles and particles of interest and pass it to the two flow methods.

   We do this in an event loop. First define the number of events that need to be created, their multiplicity, and a value v_2 value, which can either be supplied as a fixed number (no p_t dependence) of a function (to generate p_t differential flow[c]

```
  Int_t nEvents = 1000; // generate 1000 events
  Int_t mult = 2000;    // use track multiplicity of 2000
  Double_t v2 = .05;    // 5 pct integrated flow
  // or sample differential flow
  TF1* diffv2 = new TF1("diffv2", "((x<1.)*(0.1/1.)*x+(x>=1.)*0.1)", 0., 20.);
```

   Now we have all the ingredients to our first flow analysis

```
for(Int_t i=0; i<nEvts; i++) {
      // make an event with mult particles
      AliFlowEventSimple* event = new AliFlowEventSimple(mult,AliFlowEventSimple::kGenerate);
      // modify the tracks adding the flow value v2
      event->AddV2(diffv2);
      // select the particles for the reference flow
      event->TagRP(cutsRP);
      // select the particles for differential flow
      event->TagPOI(cutsPOI);
      // do flow analysis with various methods:
      mcep->Make(event);
      qc->Make(event);
}
```

6. To fill the histograms which contain the final results we have to call Finish for each method:

```
mcep->Finish();
qc->Finish();
```

7. This concludes the analysis and now we can write the results into a file. Two options for writing the input to a file are available:

---

[c] The on the fly event generator is not limited to the generation of the second harmonic v_2, but to get started, this is a nice example

- Create a new output file and write the output to this file

```
TFile *outputFile = new TFile("outputMCEPanalysis.root","RECREATE");
mcep->WriteHistograms();
TFile *outputFile = new TFile("outputQCanalysis.root","RECREATE");
qc->WriteHistograms();
```

  Please note that this will create a new output file, and overwrite any existing filse called `AnalysisResults.root`.

- To write the output of multiple analyses into subdirectories of one file, one can do the following:

```
TFile *outputFile = new TFile("AnalysisResults.root","RECREATE");
TDirectoryFile* dirQC = new TDiretoryFile("outputQCanalysis", "outputQCanalysis");
qc->WriteHistograms(dirQC);
TDirectoryFile* dirMCEP = new TDiretoryFile("outputMCEPanalysis", "outputMCEPanalysis");
mcep->WriteHistograms(dirMCEP);
```

  Note that `AnalysisResults.root` is the default name given to analyses in `AliROOT`. Many macros in `AliROOT` will expect a file `AnalyisResults.root` as input, so for most users it will be convenient to follow this convention.

  When done with running the analysis, do not forget to write the file to disk by calling

```
TFile::Close(); // write the buffered file to disk
```

2.3 What is in the output file ?

Now we have written the results into a file, but what is in there?

Although the output of different flow analysis techniques might differ slightly as a result of their different approaches at estimating v_2, the output files containers are always built in a similar way.

2.3.1 AliFlowCommonHists - Output objects

Objects of two types are stored in the output of the flow analysis[d]

---

[d] Make sure that `libPWGflowBase.so` is loaded in your `(Ali)ROOT` session, otherwise these objects will be unknown.

(a) `AliFlowCommonHist`, which is a class that contains common histograms for the flow analysis (e.g. QA histograms and histograms that contain the analysis flags which were used). Depending on the type of flow analysis that was used, this object contains histograms from the following list:

```
Bool_t     fBookOnlyBasic;         // book and fill only control histos needed for all methods
TH1F*      fHistMultRP;            // multiplicity for RP selection
TH1F*      fHistMultPOI;           // multiplicity for POI selection
TH2F*      fHistMultPOIvsRP;       // multiplicity for POI versus RP
TH1F*      fHistPtRP;              // pt distribution for RP selection
TH1F*      fHistPtPOI;             // pt distribution for POI selection
TH1F*      fHistPtSub0;            // pt distribution for subevent 0
TH1F*      fHistPtSub1;            // pt distribution for subevent 1
TH1F*      fHistPhiRP;             // phi distribution for RP selection
TH1F*      fHistPhiPOI;            // phi distribution for POI selection
TH1F*      fHistPhiSub0;           // phi distribution for subevent 0
TH1F*      fHistPhiSub1;           // phi distribution for subevent 1
TH1F*      fHistEtaRP;             // eta distribution for RP selection
TH1F*      fHistEtaPOI;            // eta distribution for POI selection
TH1F*      fHistEtaSub0;           // eta distribution for subevent 0
TH1F*      fHistEtaSub1;           // eta distribution for subevent 1
TH2F*      fHistPhiEtaRP;          // eta vs phi for RP selection
TH2F*      fHistPhiEtaPOI;         // eta vs phi for POI selection
TProfile*  fHistProMeanPtperBin;   // mean pt for each pt bin (for POI selection)
TH2F*      fHistWeightvsPhi;       // particle weight vs particle phi
TH1F*      fHistQ;                 // Qvector distribution
TH1F*      fHistAngleQ;            // distribution of angle of Q vector
TH1F*      fHistAngleQSub0;        // distribution of angle of subevent 0 Q vector
TH1F*      fHistAngleQSub1;        // distribution of angle of subevent 1 Q vector
```

```
TProfile* fHarmonic;               // harmonic
TProfile* fRefMultVsNoOfRPs;       // <reference multiplicity> versus # of RPs
TH1F*     fHistRefMult;            // reference multiplicity distribution
TH2F*     fHistMassPOI;            // mass distribution for POI selection
```

This information is from the header file of the AliFlowCommonHist object[e]

(b) `AliFlowCommonHistResults` is an object designed to hold the results of the flow analysis. The possible histograms stored in this object are

```
TH1D* fHistIntFlow; // reference flow
TH1D* fHistChi;     // resolution
// RP = Reference Particles:
TH1D* fHistIntFlowRP;      // integrated flow of RPs
TH1D* fHistDiffFlowPtRP;   // differential flow (Pt) of RPs
TH1D* fHistDiffFlowEtaRP; // differential flow (Eta) of RPs
// POI = Particles Of Interest:
TH1D* fHistIntFlowPOI;      // integrated flow of POIs
TH1D* fHistDiffFlowPtPOI;   // differential flow (Pt) of POIs
TH1D* fHistDiffFlowEtaPOI; // differential flow (Eta) of POIs
```

The titles of the histograms in the output object differ from the names of the pointers given in the two lists printed above, but the lists give an overview of what is available; the easiest way however of getting acquainted with where to find histograms in the output is browsing them in `ROOT's TBrowser` (see figure 2.2).

```
new TBrowser();
```

Analysis specific outputs will be discussed in later sections.

Comparing flow results

A convenient way of comparing the results of the different flow analysis strategies that have been used is invoking the macro `compareFlowResults.C`[f]. This macro will read the analysis output file `AnalysisResults.root`, extract the requested results from it and plot

---

[e] The headers of both output objects can be found in `$ALICE_ROOT/PWG/FLOW/Base/`.

[f] `$ALICE_ROOT/PWGCF/FLOW/macros/compareFlowResults.C`

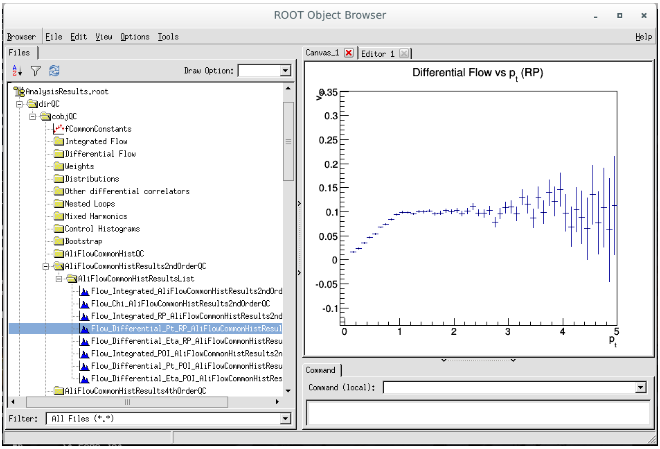


Figure 2.1: Example of output file opened in a TBrowser, results of differential v_2 analysis with second order Q-cumulant analysis are shown.

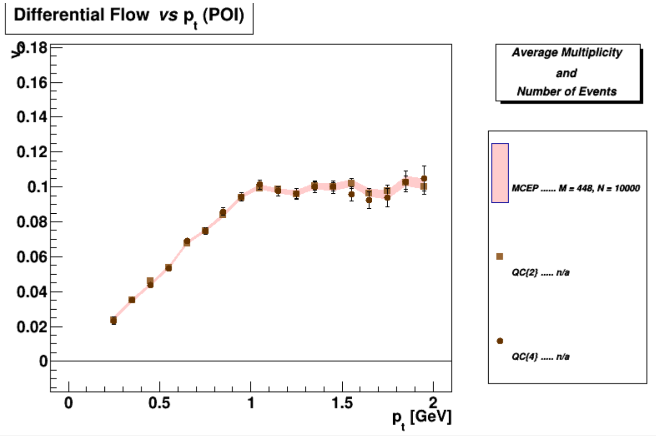


Figure 2.2: Example of inspecting the output file of the on the fly analysis with the `compareFlowResults.C` macro.

them. For a full overview of what can be done with the macro, the reader is referred to the macro itself and its ample documentation. To run the macro on the dataset that we have just generated, simply do

```
.L compareFlowResults.C
compareFlowResults(TSring(""))  // the empty suffix indicates on
    the fly events
```

2.3.2 AliFlowCommonConstants - The Common Constants class

All flow analysis use a common output container to store their histograms. To set the configuration for the histograms in these containers - e.g. the p_t ranges of histograms, the number of bins, etc, etc - all flow analysis methods initialize their output containers using variables from a global instance of the `AliFlowCommonConstants` class. This global object, which can be obtained via the a static function

```
static AliFlowCommonConstants* GetMaster();
```

can be tuned to the user's liking by requesting a pointer to it via the static access method, and using the available setter functions, e.g. the following

```
AliFlowCommonConstants* cc = AliFlowCommonConstants::GetMaster();
cc->SetNbinsPt(10);
cc->SetPtMin(0);
cc->SetPtMax(10);
```

will result in an analysis which is performed in 10 p_t bins of 1 GeV/c width. For a full overview of the available common constants the user is referred to the class header[g].

2.3.3 redoFinish.C

When analysis is run in parallel, resulting in large, merged files (e.g. when running on `GRID`) the output of the flow analysis tasks in `AnalysisResults.root` is typically wrong, as merging files via `ROOT's TFileMerger` will trivially sum up results in all histograms.

The `redoFinish.C`[h] macro re-evaluates all output that cannot trivially be merged. To use `redoFinish.C`, make sure your analysis output file is called `mergedAnalysisResults.root` and simply run the macro

```
.L redoFinish.C
redoFinish();
```

`redoFinish.C` will produce a new `AnalysisResults.root` file with the corrected results by calling the `::Finish()` function on all known output structures in the `mergedAnalysisResults.root` file. Additionally

---

[g] `$ALICE_ROOT/PWG/FLOW/Base/AliFlowCommonConstants.h`

[h] `$ALICE_ROOT/PWGCF/FLOW/macros/refoFinish.C`

`redoFinish.C` can be used to repeat the call to `::Finish()` with different settings, which might alter the outcome of the flow analysis (e.g. use a different strategy to correct for non-uniform acceptance). This will be explained in more detail in the following sections.

2.4 Getting started on Data

The macro `Documentation/examples/runFlowReaderExample.C` is an example how to setup a flow analysis if the events are already generated and for example are stored in ntuples.

2.5 A simple flow analysis in ALICE using Tasks

The macro `Documentation/examples/runFlowTaskExample.C` is an example how to setup a flow analysis using the full `ALICE` Analysis Framework.

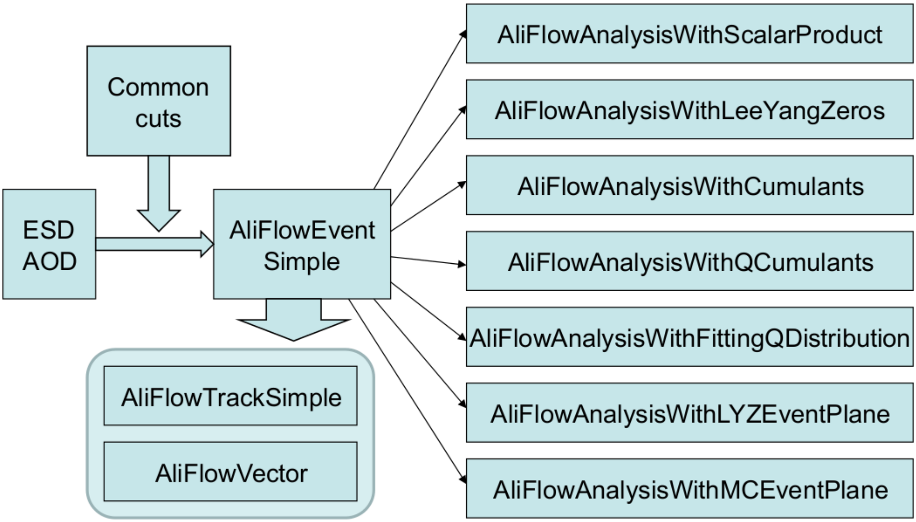
Chapter 3

The Flow Event

Here we describe the flowevent, flowtracks, general cuts and cuts for RPs POIs. OntheFly, AfterBurner. Filling with ESD, AOD, Ntuples, etc.

Chapter 4

The Program



Here we describe the program.

Chapter 5

Methods

The flow package aims at providing the user with most of the known flow analysis methods. Detailed theoretical overview of the methods can be found in the following papers, which are included in the folder `$ALICE_ROOT/PWGCF/FLOW/Documentation/otherdocs/`

- The Monte-Carlo Truth
- Scalar Product Method
  `EventPlaneMethod/FlowMethodsPV.pdf`
- Generating Function Cumulants
  `GFCumulants/Borghini_GFCumulants_PracticalGuide.pdf`
- Q-vector Cumulant method
  `QCumulants/QCpaperdraft.pdf`
- Lee-Yang Zero Method
  `LeeYangZeroes/Borghini_LYZ_PracticalGuide.pdf`
- Lee-Yang Zero Method
  `LeeYangZeroesEP/LYZ_RP.pdf`

The structure of this chapter is as follows: of each of the available methods a short description is given in the `theory` subsection (for more detailed information, see the papers listed above) followed by details which are specific to the implementation in the subsection `implementation`. Caveats, possible issues, etc, are listed in the `caveats` subsections.

5.1 The Monte-Carlo Truth

Here we describe the implementation of the monte-carlo truth.

5.2 Scalar Product Method

5.2.1 Theory

The scalar product method

The scalar product method - as well as the Q-cumulant method which will be described later - does not depend on explicit construction of an (sub)event plane, but estimates v_n directly from multi-particle correlations.

To do so, firstly all particles in an event are labeled either as *reference particles* (RP's) or *particles of interest* (POI's). The RP and POI selections are in turn divided into sub-events, which are again taken from different η ranges, in analogy to the approach taken for the event plane method. Each POI is correlated with a sub-event Q-vector from the RP selection, which allows for the calculation of v_n without any explicit reconstruction of an event plane[**?**].

The reason for the division into RP's and POI's is the fact that the two particle correlator of POI's,

$$v_n^{POI} = \sqrt{\left\langle e^{in(\phi_i^{POI} - \phi_j^{POI})} \right\rangle} \tag{5.2.1.1}$$

is generally not stable statistically. Introducing reference flow, 5.2.1.1 can be rewritten as

$$v_n^{POI} = \frac{\left\langle e^{in(\phi_i^{POI} - \phi_j^{RP})} \right\rangle}{\left\langle e^{in(\phi_i^{RP} - \phi_j^{RP})} \right\rangle} = \frac{v_n^{POI} v_n^{RP}}{\sqrt{v_n^{RP} v_n^{RP}}}. \tag{5.2.1.2}$$

By taking an abundant particle source as RP's - in the case of this study the RP selection comprises all charged particles - both correlators in 5.2.1.2 are statistically stable.

The scalar product method In the scalar product method, POI's u_k,

$$u_k = e^{in\phi_k}, \tag{5.2.1.3}$$

are correlated with Q_a^*, the complex-conjugate Q-vector built from RP's in a given sub-event a. First, the scalar product of u_k and Q_a^* is taken,

$$u_k \cdot \sum_{\substack{j=1,\\ j \neq k}}^{M_{RP,a}} u_j^* \tag{5.2.1.4}$$

where $M_{RP,a}$ denotes RP multiplicity for a given sub-event a and the inequality $j \neq k$ removes auto-correlations. From this, differential v_n of POI's (v'_n) and v_n of RP's (v^a_n) in sub-event a can be obtained in a straightforward way from the correlation of POI's and RP's:

$$\langle u \cdot Q^*_a \rangle = \frac{1}{M_{RP,a} - k} \sum_{i=k}^{M_{RP,a}} \left(u_k \sum_{\substack{j=1,\\ j \neq k}}^{M_{RP,a}} u^*_j \right) \tag{5.2.1.5}$$

where POI multiplicity is expressed in terms of $M_{RP,a}$; $M_{POI} = M_{RP,a} - k$. Since for any function $f(x)$ and constant a

$$\sum af(x) = a \sum f(x) \tag{5.2.1.6}$$

5.2.1.5 can be rewritten as

$$\begin{aligned} \langle u \cdot Q^*_a \rangle &= \frac{1}{M_{RP,a} - k} \sum_{i=k}^{M_{RP,a}} e^{in[\phi_k - \Psi_n]} \sum_{j=1}^{M_{RP,a}} e^{-in[\phi_j - \Psi_n]} \\ &= M_{RP,a} v'_n v^a_n \end{aligned} \tag{5.2.1.7}$$

where in the last step of 5.2.1.7 it has been used that

$$v_n = \frac{\sum_i^M e^{in[\phi_i - \Psi_n]}}{M}. \tag{5.2.1.8}$$

To obtain the estimate of v_n, one must still disentangle the reference flow contribution from the event averaged correlation given in 5.2.1.5. Proceeding in a fashion similar to that presented in equation 5.2.1.5, it can be shown that

$$\left\langle \frac{Q_a}{M_a} \cdot \frac{Q^*_b}{M_b} \right\rangle = \langle v^a_n v^b_n \rangle \tag{5.2.1.9}$$

where Q_a, Q_b are the Q-vectors of RP's in sub-event a, b. Under the assumption that

$$\langle v^2_n \rangle = \langle v_n \rangle^2, \tag{5.2.1.10}$$

- an assumption which will be spoiled in the case of flow fluctuations - and requiring that the v_n estimates in both sub-events are equal, one simply evaluates

$$v'_n = \frac{\left\langle \left\langle u \cdot \frac{Q^*_a}{M_a} \right\rangle \right\rangle}{\sqrt{\left\langle \frac{Q_a}{M_a} \cdot \frac{Q^*_b}{M_b} \right\rangle}} \tag{5.2.1.11}$$

to obtain v_n^a. For equal multiplicity sub-events $M_a = M_b$, 5.2.1.11 is simplified to

$$v_n' = \frac{\langle\langle u\cdot Q_a^*\rangle_t\rangle}{\sqrt{\langle Q_a\cdot Q_b^*\rangle}}. \tag{5.2.1.12}$$

v_n^b can be obtained by switching indices a and b in expressions 5.2.1.11 and 5.2.1.12, and should equal v_n^a. This principle can be generalized straightforwardly to allow for a selection of RP's which has been divided into three subevents.

$$\begin{aligned} v_n^a &= \frac{\left\langle\left\langle u\cdot \frac{Q_a^*}{M_a}\right\rangle\right\rangle}{\sqrt{\langle v_n'^a v_n'^b\rangle \langle v_n'^a v_n'^c\rangle \,/\, \langle v_n'^b v_n'^c\rangle}} \\ &= \frac{\left\langle\left\langle u\cdot \frac{Q_a^*}{M_a}\right\rangle\right\rangle}{\sqrt{\left\langle \frac{Q_a}{M_a}\cdot\frac{Q_b^*}{M_b}\right\rangle \left\langle \frac{Q_a}{M_a}\cdot\frac{Q_c^*}{M_c}\right\rangle \Big/ \left\langle \frac{Q_b}{M_b}\cdot\frac{Q_c^*}{M_c}\right\rangle}} \end{aligned} \tag{5.2.1.13}$$

where cyclic permutation of a, b, c (in analogy to the switching of indices in **??** gives the estimates of v_n^b and v_n^c.[insert some discussion here: is this result actually true, and some light on va, vb, (vc)]

5.2.2 Implementation

Extension to Event Plane method

As explained earlier, the event plane analysis results in this study are actually obtained by normalizing the Q-vectors in the scalar product by their length $|Q_n|$. Consider the following:

$$\frac{Q_n^*}{|Q_n^*|} = \frac{|Q_n^*|e^{-in\Psi_{Q_n}}}{|Q_n^*|} = e^{-in\Psi_{Q_n}}. \tag{5.2.2.1}$$

For a full event, the enumerator of 5.2.1.11 can be expressed as

$$\langle\langle u\cdot e^{-in\Psi_{Q_n}}\rangle\rangle = \langle\langle e^{in\phi_i}\cdot e^{-in\Psi_{Q_n}}\rangle\rangle = \left\langle\left\langle e^{in(\phi_i-\Psi_{Q_n})}\right\rangle\right\rangle = \langle\langle\cos(n[\phi_i - \Psi_{Q_n}])\rangle\rangle$$

which corresponds to the all-event average of **??**. As shown in the previous subsection this expression equals v_n^{obs}.

For normalized Q-vectors, the denominator of 5.2.1.11 reads (using 5.2.2.1):

$$\sqrt{\left\langle \frac{Q_a}{|Q_a|}\cdot\frac{Q_b^*}{|Q_b^*|}\right\rangle} = \sqrt{\left\langle e^{in[\Psi_{Q_{n_a}}-\Psi_{Q_{n_b}}}\right\rangle} = \sqrt{\left\langle \cos(n[\Psi_{Q_{n_a}} - \Psi_{Q_{n_b}}]\right\rangle)} \tag{5.2.2.2}$$

from which the event plane resolution can be calculated using **??** or **??**.

Caveats

5.3 Generating Function Cumulant Method

Here we describe the generating function cumulant method and how it is implemented.

5.4 Q-vector Cumulant Method

5.4.1 Theory

The Q-cumulant (QC) method[a] uses multi-particle correlations to estimate v_n estimates for RP's and POI's, but does not limit itself to two-particle correlations. Although higher-order Q-cumulant calculations are available, this section will discuss the method using two- and four-particle correlations.

Multi-particle correlations in the QC method are expressed in terms of cumulants, which are the the expectation values of correlation terms in joint probability density functions. Consider the following: if two observables f for particles x_i and x_j are correlated, the joint probability $f(x_i, x_j)$ is the sum of the factorization of the constituent probabilities and a covariance term:

$$f(x_i, x_j) = f(x_i)f(x_j) + f_c(x_i, x_j) \tag{5.4.1.1}$$

When taking as an observable azimuthal dependence,

$$x_i \equiv e^{in\phi_i}, \qquad x_j \equiv e^{in\phi_j} \tag{5.4.1.2}$$

the two-particle cumulant is expressed as the covariance of the expectation value:

$$E_C(e^{in[\phi_i-\phi_j]}) = E(e^{[in(\phi_i-\phi_j]}) - E(e^{in[\phi_i]})E(e^{in[-\phi_j]}). \tag{5.4.1.3}$$

Symmetry arguments (along the lines of those given in appendix **??**) dictate that the product of separate expectation values is equals zero, from which

---

[a] The overview given in this section is inspired by [**?**], for further reading the reader is referred there. A full derivation of results that are relevant in this study is given in appendix **??**.

a familiar expression for the two-particle correlation is obtained:

$$\begin{aligned} E_C(e^{in[\phi_i-\phi_j]}) &= E(e^{in[\phi_i]})E(e^{in[-\phi_j]}) \\ &= \left\langle e^{in[\phi_i]} \right\rangle \left\langle e^{in[-\phi_j]} \right\rangle \\ &= \left\langle e^{in[\phi_i-\phi_j]} \right\rangle \\ &= \langle 2 \rangle \,, \end{aligned} \tag{5.4.1.4}$$

the all-event average of which is denoted by

$$c_n\{2\} = \langle\langle 2 \rangle\rangle \tag{5.4.1.5}$$

where $c_n\{2\}$ is called the two-particle cumulant. For the four-particle case, one proceeds likewise:

$$\begin{aligned} E_c(e^{in[\phi_i+\phi_j-\phi_k-\phi_l]}) &= E(e^{in[\phi_i+\phi_j-\phi_k-\phi_l]}) \\ &- E(e^{in[\phi_i-\phi_k]})E(e^{in[\phi_j-\phi_l]}) \\ &- E(e^{in[\phi_i-\phi_l]})E(e^{in[\phi_j-\phi_k]}). \end{aligned} \tag{5.4.1.6}$$

The four-particle cumulant can be expressed in terms of two- and four-particle correlations as well,

$$c_n\{4\} = \langle\langle 4 \rangle\rangle - 2 \left\langle\langle 2 \rangle\right\rangle^2 . \tag{5.4.1.7}$$

From 5.4.1.5 and 5.4.1.7 it follows that v_n harmonics are related to cumulants following

$$\begin{aligned} v_n\{2\} &= \sqrt{c_n\{2\}} \\ v_n\{4\} &= \sqrt[4]{-c_n\{4\}}. \end{aligned} \tag{5.4.1.8}$$

where $v_n\{2\}$, $v_n\{4\}$ denote flow estimates obtained from two- and four-particle correlations.

In a fashion similar to that explained in the previous subsection, the Q-cumulant method uses reference flow to obtain a statistically stable estimate of the differential flow of POI's. Differential POI flow, for the two- and four-particle case, can be expressed as

$$\begin{aligned} d_n\{2\} &= \langle\langle 2' \rangle\rangle \\ d_n\{4\} &= \langle\langle 4' \rangle\rangle - 2\cdot \langle\langle 2' \rangle\rangle \langle\langle 2 \rangle\rangle \end{aligned} \tag{5.4.1.9}$$

where $d_n\{2\}, d_n\{4\}$ denotes the two-, four-particle differential flow and the $'$ is used as an indicator for differential (p_t dependent) results. Disentangling from this the reference flow contributions, one is left with the final expression for the estimate of differential v_n for POI's:

$$\begin{aligned} v'_n\{2\} &= \frac{d_n\{2\}}{\sqrt{c_n\{2\}}} \\ v'_n\{4\} &= -\frac{d_n\{4\}}{(-c_n\{2\})^{3/4}}. \end{aligned} \tag{5.4.1.10}$$

5.4.2 Implementation

Here we describe the Q-vector cumulant method and how it is implemented.

5.5 Lee-Yang Zero Method

Here we describe the Lee-Yang Zero method and how it is implemented.

5.6 Lee-Yang Zero Method

Here we describe the Lee-Yang Zero method and how it is implemented.

5.7 Fitting the Q-vector Distribution

Here we describe how the fitting of the Q-vector distribution is implemented.

Chapter 6

Summary

This sums it all up.

Chapter 7

Bibliography

[1] J. Y. Ollitrault, Phys. Rev. D **46** (1992) 229.

[2] P. Danielewicz, Nucl. Phys. A **661** (1999) 82.

[3] D. H. Rischke, Nucl. Phys. A **610** (1996) 88C.

[4] J. Y. Ollitrault, Nucl. Phys. A **638** (1998) 195.

[5] S. Voloshin and Y. Zhang, Z. Phys. C **70** (1996) 665.

[6] K. H. Ackermann *et al.* [STAR Collaboration], Phys. Rev. Lett. **86** (2001) 402

[7] C. Adler *et al.* [STAR Collaboration], Phys. Rev. Lett. **87** (2001) 182301

[8] T.D. Lee *et al.*, New Discoveries at RHIC: Case for the Strongly Interacting Quark-Gluon Plasma. Contributions from the RBRC Workshop held May 14-15, 2004. Nucl. Phys. A **750** (2005) 1-171

Appendix I

Here we put short pieces of code.

Index