BABAR Analysis Document #102, Version 7.00
February 7, 2005

The *BABAR* Vertexing

Vertexing and Composition Tools Group [1]

Abstract

This document describes the vertexing algorithms in *BABAR*, including the practical use and basic performances.

[1]Direct contributors to this document are: Massimo Carpinelli (editor) (*INFN-Pisa*), Riccardo Faccini (*Univ. di Roma La Sapienza*), Chih-hsian Cheng (*Stanford University*), **Fernando Martinez-Vidal (editor and contact person)** (*INFN-Pisa*), Stéphane Plaszczynski (*LAL Orsay*), Patrick Robbe (*LAPP, Annecy*), Abi Soffer (*Colorado State Univ.*), Jan Stark (*LPNHE, Univ. Paris 6&7*), Christos Touramanis (*University of Liverpool*)

Contents

1 Overview

This document describes the vertexing algorithms in *BABAR*, including their practical use and basic performances. A brief summary of the beam spot reconstruction technique, performances in the 2000 year and how to use it for physics analyses is also part of this document. The document is intended to provide the primary reference for vertexing algorithms in *BABAR*. Detailed performance studies, control samples and data/Monte Carlo comparisons using the 2000 year data are provided in a separated document [1].

Vertexing and kinematic fitting is an important tool for *BABAR*. It has to deal with the following two considerations. First, the *BABAR* detector has been designed to achieve high resolution tracking. Second, the asymmetric design of the PEP-II collider with a boost of $\beta\gamma \approx 0.56$ in the laboratory frame has been done with the purpose of obtaining a separation of $\langle\beta\gamma c\tau\rangle \approx 250\mu$m between the two B vertices (comparable to the experimental resolution, ~ 130 μm RMS), crucial component for studying time-dependent CP asymmetries. In this way vertexing is largely used in most of the *BABAR* analyses to improve four-momenta and position measurements, as well as to measure the time difference between decaying B hadrons in the $\Upsilon(4S) \rightarrow B\overline{B}$ decay.

If you want to have the most up-to-date version you should check-out the head of CVS:

```
% cvs co BAD/note102
% cd BAD
% cvs co pubboard
% cd note102
% ln -s ../pubboard/ .
% latex paper.tex
% ...
```

The description in this document refer to the *analysis-7* release, with the following vertexing related tags on top:

```
VtxFitter V00-08-48-02
VertexingTools V00-08-44-02
BetaTools V00-10-06-03
FastVtx V03-03-07
```

2 Vertexing algorithms

2.1 Introduction

The physics properties of decay are used to apply constraints which translate to better mass and position resolutions and larger signal-to-background ratios. For example, in the case of $B^0 \to J/\psi K^0_S$, the position measurement of the B^0 can be improved using the fact that the line-of-flight of the K^0_S intersects the J/ψ vertex and that it is aligned with the line that joins the K^0_S and the J/ψ vertices. The energy resolution of the B^0 can also be improved by applying a mass constraint to the J/ψ. Similarly, the momentum resolution of neutrals can be improved if a decay vertex can be identified to apply a mass constraint. Several constraints have been considered: common decay vertex, mass, energy, momentum, beam energy (with and without smearing), beam spot position and line-of-flight. Given the similar tracking resolutions of the *BABAR* detector in the transverse plane and longitudinal direction, calculations are in all cases performed in three dimensions. The design of vertexing algorithms took a significant profit from other previous HEP experiments [3].

Non linearities in the fits require the application of an iterative procedure. Simple fits involving only vertex constraints (except long-lived particles, i.e. V^0's) are, however, accurate enough with a single iteration. This has an important impact on the amount of time consumed in vertex fitting. The other fits involving kinematic constraints and V^0's require, in general, more than one iteration.

One of the fundamental principles in the design was to deal in a simple way with complex decay chains. Virtual *composite* particles and their error matrices are built from the original particles. The composite particle then replaces the daughters in subsequent fits and analysis. Once the composite particle has been built one can forget about the original particles that went into it. The vertexing software makes use of an advanced interface for building composite (virtual) particles based on constraints and particle type: each intermediate particle has a set of associated constraints and a vertex. To account for intermediate resonance, vertices can be shared by different virtual states. With this approach the daughter tracks are first fitted to a common vertex and the updated tracks are used to compute the 4-momenta of the tracks.

To the extent that the field is uniform and the material is thin, the trajectories of a charged particle can be approximated by a helix. Candidates used for vertexing are parameterized in the so-called *X-P representation* which gives the position and momenta of the candidate at distance of closest approach for tracks (with respect to the *BABAR* origin) and at decay point for composites. 3-momentum of neutrals is calculated assuming the *BABAR* origin, but when using them for vertexing it is possible to provide the point at which it has to be calculated (see sections below). The energy has been excluded from the internal representation to avoid problems derived from its large correlation with the 3-momentum components. The advantage of this representation is simplicity and better accuracy when implementing kinematic constraints, a part of the fact that we use directly for fitting physically meaningful quantities. The main drawbacks of using the 6-parameter representation instead of helices are: i) the *BABAR* tracks have to be converted into this representation at the point of closest approach, ii) the six parameters are not independent. Appendix A gives the details of the transformation of parameters and weight/covariance matrices from the helix to the X-P representation. Only the `FastVtx` algorithm makes use of the *helix representation* (5 parameters).

Building virtual particles and decay chains has an additional complication derived from the fact that we need flexibility for merging different kinds of particles, some of which have poor or no position information, as it is the case of resonances like $\rho \to \pi^+ \pi^0$ or neutrals. Manipulation of neutrals in vertex constrained fits is performed using the standard 6-parameter approach. The derivation of the covariance matrix is performed by assigning a large error to the vertex position of the neutral, and then correcting the number of degrees of freedom. Extensive studies showed that this approach is equivalent to use a 3-parameter representation. Resonances are resolved by attaching the daughters to the common vertex of the mother (only `GeoKin`, see below).

2.2 The `BtaAbsVertex` class

The information available for a candidate after fitting, is defined in the `BtaCandidate` structure, and can be accessed using the `BtaAbsVertex` interface, which class is available in the Beta package. The following list shows the most relevant information stored, along with the name of the corresponding accessor:

`double chiSquared()` χ^2 of the fit;

`int nDof()` number of degrees of freedom of the fit;

`VtxStatus status()` word indicating the status of the vertex. `VtxStatus` has the following possible values: `Success=0, NonConverged, BadInput, Failed, UnFitted`. When creating composite candidates, the default vertex status is `UnFitted`. `NonCoverged` means that the maximum number of iterations has been exceeded, and `Failed` that an error has been produced in the matrix manipulation, typically when inverting matrices. `BadInput` is there by historical reasons, but now it is obsolete and will never occur (except when one is using the `SingleTrackGeoKin` option of `VtxFitterOper` (section 2.8) and the beam spot constraint was not applied.

`VtxType type()` word indicating whether the candidate has valid position information. Particles such as π^0's and γ's do not have useful position information that can be used or updated in a fit. This information is used by `GeoKin` to correct for the 'ghost' degrees of freedom introduced by the *X terms*. `VtxType` can take one of the following values: `None=-1, Geometric, Kinematic`;

`HepPoint point()` fitted vertex;

`HepLorentzVector p4()` four-momentum at the fitted vertex;

`double mass()` mass at the fitted vertex;

`double p()` momentum magnitude at the fitted vertex;

`HepSymMatrix xxCov(), HepSymMatrix ppCov(), HepMatrix xpCov()` 3×3 covariance matrices;

`HepSymMatrix xxWeight()`, `HepSymMatrix ppWeight()`, `HepMatrix xpWeight()` 3×3 weight matrices;

2.3 Vertexers and fitters

We can distinguish two kinds of vertexing operators:

- *Fitters*, which deal with the fit of full decay chains with any kind of vertex and kinematic constraint.
- *Vertexers*, a simplified version of fitters which exploit the special feature of the vertex constraints (track's parameters can be related with the vertex coordinates independently of the other tracks) to gain in execution time.

In the subsections below we describe the recommended fitters and vertexers which should be used for *BABAR* analyses: `GeoKin` and `FastVtx` as fitters, `VtxLeastChiVertexer` as vertexer. `GeoKin` is the general and default operator for *BABAR*.

2.4 First guess

A common issue in vertexing is that an initial guess of the location of the vertex is required; usually the primary interaction point can be used, but the convergence radius is of the order of a few centimeters. This is good enough for decays of short-lived particles (B and D) but not for long-lived particles (V^0's). This problem is handled generally for all cases by solving analytically (with an iterative procedure) the point of closest approach of the two tracks or candidates (both with defined spatial information), using a second-order approximation for the track at each point. When more than two tracks appear, the point of closest approach of the closest pair of tracks is used. No cut on the minimal doca is applied. When the search for the poca point fails, the *BABAR* origin is assumed and the vertex is still attempted. The class `VtxGeomCalculator` in VtxFitter provides the access to this quantity. For neutrals, the *BABAR* origin is assumed by default for estimating its momentum, but it can be changed as explained in the following sections.

2.5 GeoKin algorithm

2.5.1 General formalism

The general `GeoKin` algorithm is based on the generalized least squares method using the well-known Lagrange Multiplier technique [2]. In this method it is assumed that the constraint equations can be linearized and summarized in three matrices, labeled in this document as $\tilde{A}$, $\tilde{B}$ and $\vec{c}$. Specific expressions for these matrices will be given in subsections below and section 4.2.

Let $\vec{\eta}$ represent the measurable quantities (n-vector). The actually measured quantities $\vec{y}$ (for instance the measure track parameters) deviate from $\vec{\eta}$ by errors $\vec{\delta}$. We assume that the errors δ_i, $i = 1, ..., n$, are normally distributed. There are r unknowns grouped in a vector $\vec{x}$. The $\vec{x}$ and $\vec{\eta}$ are related by m functions (constraints),

$$f_k(\vec{x}, \vec{\eta}) = f_k(\vec{x}, \vec{y} + \vec{\delta}) = 0, \quad k = 1, ..., m \tag{1}$$

Assuming that functions f_k can be linearized at $\vec{x} = \vec{x_0}$ and $\vec{\eta} = \vec{\eta_0}$, the χ^2 anszat can be written as [2]:

$$\chi^2 = \vec{\delta}^T \tilde{W}_y \vec{\delta} + 2\vec{\mu}^T \left(\tilde{A}\vec{\xi} + \tilde{B}\vec{\delta} + \vec{c} \right) \tag{2}$$

where $\vec{\mu}$ is the vector of Lagrange multipliers, and

$$\vec{\xi} = \vec{x} - \vec{x_0} \tag{3}$$

$$\vec{\delta} = \vec{\eta} - \vec{\eta_0} \tag{4}$$

$\tilde{W}_y$ is the weight matrix of parameters. Matrices $\tilde{A}$ and $\tilde{B}$ are, respectively, the derivatives (estimated at point $\vec{x_0}$, $\vec{\eta_0}$) of the constraints with respect to the unknowns and parameters:

$$a_{kl} = \left(\frac{\partial f_k}{\partial x_l} \right)_{\vec{x_0}, \vec{\eta_0}} , \qquad \tilde{A} = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1r} \\ a_{21} & a_{22} & ... & a_{2r} \\ ... & & & \\ a_{m1} & a_{m2} & ... & a_{mr} \end{pmatrix} \tag{5}$$

$$b_{kl} = \left(\frac{\partial f_k}{\partial \eta_l} \right)_{\vec{x_0}, \vec{\eta_0}} , \qquad \tilde{B} = \begin{pmatrix} b_{11} & b_{12} & ... & b_{1n} \\ b_{21} & b_{22} & ... & b_{2n} \\ ... & & & \\ b_{m1} & b_{m2} & ... & b_{mn} \end{pmatrix} \tag{6}$$

$\vec{c}$ is the vector of values of the constraints at point $\vec{x_0}$, $\vec{\eta_0}$,

$$c_k = f_k(\vec{x_0}, \vec{\eta_0}) , \qquad \vec{c} = \begin{pmatrix} c_1 \\ c_2 \\ ... \\ c_m \end{pmatrix} \tag{7}$$

As a first approximation, we take $\vec{\eta_0} = \vec{y}$. The expansion point requires also $\vec{x_0}$. Is here where the first guess (section 2.4) enters in.

Resolving (2) for unknowns and parameters [2], we get:

$$\vec{\xi} = -\left(\tilde{A}^T \tilde{G}_B \tilde{A} \right)^{-1} \tilde{A}^T \tilde{G}_B \vec{c} \tag{8}$$

$$\vec{\delta} = -\tilde{W}_y^{-1} \tilde{B}^T \tilde{G}_B \left(\vec{c} + \tilde{A}\vec{\xi} \right) \tag{9}$$

where

$$\tilde{G}_B = \left(\tilde{B} \tilde{W}_y^{-1} \tilde{B}^T \right)^{-1} \tag{10}$$

The covariance matrices for unknowns and parameters are:

$$\tilde{G}_x^{-1} = \tilde{G}_\xi^{-1} = \left(\tilde{A}^T \tilde{G}_B \tilde{A}\right)^{-1} \tag{11}$$

$$\tilde{G}_\eta^{-1} = \tilde{W}_y^{-1} - \tilde{W}_y^{-1} \tilde{B}^T \tilde{G}_B \tilde{B} \tilde{W}_y^{-1} + \tilde{W}_y^{-1} \tilde{B}^T \tilde{G}_B \tilde{A} \tilde{G}_\xi^{-1} \tilde{A}^T \tilde{G}_B \tilde{B} \tilde{W}_y^{-1} \tag{12}$$

and the correlation matrix between unknowns and parameters is:

$$\tilde{G}_{x\eta}^{-1} = -\tilde{W}_y^{-1} \tilde{B}^T \tilde{G}_B \tilde{A} \tilde{G}_\xi^{-1} \tag{13}$$

The minimum of (2) can be written as:

$$\chi^2_{min} = \vec{\delta}^T \left(\tilde{B}^T \tilde{G}_B \tilde{B}\right) \vec{\delta} \tag{14}$$

which follows a χ^2-distribution with $m - r$ degrees of freedom. Equation (14), projected over a "group" of parameters allow us to estimate the contribution of it to the total χ^2. By "group" of parameters we mean any set of parameters for which the covariance matrix $\tilde{W}_y^{-1}$ is block diagonal.

When (1) are already linear, equations (3), (4), (11) and (12) give the final solution. In general constraints are not linear, and then we need a better approximation, which can be obtained replacing $\vec{x}_0$, $\vec{\eta}_0 = \vec{y}$ by $\vec{x}$, $\vec{\eta}$ into (5), (6) and (7). This iteration process is repeated until a satisfactory solution is found. The default convergence criteria consist in requiring a change in χ^2, as given by equation (14), between two successive iterations less than 0.005, with a maximum of 6 iterations.

The equations above and iterative procedure described above are implemented in the general classes `VtxGeoKin` and `VtxGeoKinIterator`, both in the VtxFitter package. They constitute the general engines of the `GeoKin` fitter, and, as described latter on, will be used also for the vertex tag reconstruction. Some tricks are applied to speed-up matrix manipulation.

In the particular case that parameters are candidates (in *X-P representation*), the covariance matrix $\tilde{W}_y^{-1}$ will be block diagonal, and each block can be written as

$$\tilde{W}_{y,i}^{-1} = \begin{pmatrix} \tilde{W}_{xx,i}^{-1} & \tilde{W}_{xp,i}^{-1} \\ \tilde{W}_{xp,i}^{-1} & \tilde{W}_{pp,i}^{-1} \end{pmatrix} \tag{15}$$

where $\tilde{W}_{xx,i}^{-1}$, $\tilde{W}_{xp,i}^{-1}$ and $\tilde{W}_{pp,i}^{-1}$ are the 3×3 covariance matrices for position and momentum. It should be noted that with this approach the position-momentum correlations are accounted for, important for kinematic fitting [2] (less relevant for pure vertex fitting). Due to their small size the correlation can introduce some inestabilities, especially in complex decay trees with both vertex and kinematic constraints[3].

2.5.2 Constraints

This section contains a description of all the constraints available with the `GeoKin` fitter. We distinguish two different kinds of constraints: Lagrange and χ^2. Lagrange constraints

[2]E.g. the dramatic improvement in Δm resolution in D^* decays is possible thanks to these correlations.

[3]These inestabilities will cause error codes in `GeoKin`, generally due to the impossibility to perform matrix inversions.

are exact, χ^2 constraints account for the errors defining the parameters of the constraints. Technically, the two types of constraints differ on the fact that any χ^2 constraint requires the addition of new parameters, in the same number as parameters are used to define the constraint.

Vertex

For each input candidate i there are two constraint equations, corresponding to the bend and non-bend planes, respectively. The linerized equations are:

$$\frac{1}{p_{\perp,i}}\left[\Delta y_i p_{x,i} - \Delta x_i p_{y,i}\right] = 0 \tag{16}$$

$$\Delta z_i - \frac{p_{z,i}}{p_{\perp}^2}\left[\Delta x_i p_{x,i} + \Delta y_i p_{y,i}\right] = 0 \tag{17}$$

where $p_{\perp,i} = \sqrt{p_{x,i}^2 + p_{y,i}^2}$ is the transverse momentum of candidate i, $\Delta x_i = x - x_i$ $\Delta y_i = y - y_i$ and $\Delta z_i = z - z_i$. (x, y, z) is the unknown vertex position.

Starting from *analysis-8* release, the equations (16) and (17) are corrected by track curvature. The new equations read as:

$$\frac{1}{p_{\perp,i}}\left[\Delta y_i p_{x,i} - \Delta x_i p_{y,i}\right] - \frac{a_i}{2p_{\perp,i}}\left[(\Delta y_i)^2 + (\Delta x_i)^2\right] = 0 \tag{18}$$

$$\Delta z_i - \frac{p_{z,i}}{a_i}\sin^{-1}\left[\frac{a_i}{p_{\perp,i}^2}(\Delta x_i p_{x,i} + \Delta y_i p_{y,i})\right] = 0 \tag{19}$$

where $a_i = -0.00299792458 B q_i$ MeV/c, q_i being the charge of the track and B the magnetic field (in Tesla).

These equations, evaluated at each point, constitute the $2 \times n$ vector of constraints, where n is the number of tracks to vertex. The derivatives with respect to the parameters and unknowns allow us to construct the matrices $\tilde{A}$ and $\tilde{B}$:

$$\frac{\partial(18)}{\partial x_i} = \frac{p_{y,i} + a_i \Delta x_i}{p_{\perp,i}} \tag{20}$$

$$\frac{\partial(18)}{\partial y_i} = \frac{-p_{x,i} + a_i \Delta y_i}{p_{\perp,i}} \tag{21}$$

$$\frac{\partial(18)}{\partial z_i} = 0 \tag{22}$$

$$\frac{\partial(18)}{\partial p_{x,i}} = \frac{\Delta y_i}{p_{\perp,i}} - \left[\Delta y_i p_{x,i} - \Delta x_i p_{y,i}\right]\frac{p_{x,i}}{p_{\perp,i}^3} + \frac{a_i}{2}\left[(\Delta y_i)^2 + (\Delta x_i)^2\right]\frac{p_{x,i}}{p_{\perp,i}^3} \tag{23}$$

$$\frac{\partial(18)}{\partial p_{y,i}} = \frac{-\Delta x_i}{p_{\perp,i}} - \left[\Delta y_i p_{x,i} - \Delta x_i p_{y,i}\right]\frac{p_{y,i}}{p_{\perp,i}^3} + \frac{a_i}{2}\left[(\Delta y_i)^2 + (\Delta x_i)^2\right]\frac{p_{y,i}}{p_{\perp,i}^3} \tag{24}$$

$$\frac{\partial(18)}{\partial p_{z,i}} = 0 \tag{25}$$

$$\frac{\partial(19)}{\partial x_i} = p_{x,i}p_{z,i}\frac{1}{p_{\perp,i}^2\sqrt{1-\left(a_i[\Delta x_i p_{x,i}+\Delta y_i p_{y,i}]/p_{\perp,i}^2\right)^2}} \tag{26}$$

$$\frac{\partial(19)}{\partial y_i} = p_{y,i}p_{z,i}\frac{1}{p_{\perp,i}^2\sqrt{1-\left(a_i[\Delta x_i p_{x,i}+\Delta y_i p_{y,i}]/p_{\perp,i}^2\right)^2}} \tag{27}$$

$$\frac{\partial(19)}{\partial z_i} = -1 \tag{28}$$

$$\frac{\partial(19)}{\partial p_{x,i}} = -p_{z,i}\frac{1}{p_{\perp,i}^2\sqrt{1-\left(a_i[\Delta x_i p_{x,i}+\Delta y_i p_{y,i}]/p_{\perp,i}^2\right)^2}}\left[\Delta x_i - 2p_{x,i}\frac{\Delta x_i p_{x,i}+\Delta y_i p_{y,i}}{p_{\perp,i}^2}\right] \tag{29}$$

$$\frac{\partial(19)}{\partial p_{y,i}} = -p_{z,i}\frac{1}{p_{\perp,i}^2\sqrt{1-\left(a_i[\Delta x_i p_{x,i}+\Delta y_i p_{y,i}]/p_{\perp,i}^2\right)^2}}\left[\Delta x_i - 2p_{y,i}\frac{\Delta x_i p_{x,i}+\Delta y_i p_{y,i}}{p_{\perp,i}^2}\right] \tag{30}$$

$$\frac{\partial(19)}{\partial p_{z,i}} = -\frac{\sin^{-1}\left[a_i\left(\Delta x_i p_{x,i}+\Delta y_i p_{y,i}\right)/p_{\perp,i}^2\right]}{a_i} \tag{31}$$

$$\frac{\partial(18)}{\partial x} = -\frac{\partial(18)}{\partial x_i}\,, \qquad \frac{\partial(18)}{\partial y} = -\frac{\partial(18)}{\partial y_i}\,, \qquad \frac{\partial(18)}{\partial z} = -\frac{\partial(18)}{\partial z_i} \tag{32}$$

$$\frac{\partial(19)}{\partial x} = -\frac{\partial(19)}{\partial x_i}\,, \qquad \frac{\partial(19)}{\partial y} = -\frac{\partial(19)}{\partial y_i}\,, \qquad \frac{\partial(19)}{\partial z} = -\frac{\partial(19)}{\partial z_i} \tag{33}$$

These constraints are implemented in the `BtaGeoConstraint` class in `VtxBase`.

Pseudo-momentum

The calculation of the 6×6 covariance matrix requires special care because it includes the momentum contribution from the daughter tracks, the position from the vertex fit, the correlations from the constraints and the errors and correlations arising from the summing process. This can produce non-negligible numerical inaccuracies. The solution has been incorporate the 3-momentum of the virtual particle as additional unknowns (like the vertex position) with the corresponding kinematic constraints. These constraints are known as *Pseudo-momentum* constraints, and their trivial definition is:

$$\sum_i p_{x,i} - p_x = 0 \tag{34}$$

$$\sum_i p_{y,i} - p_y = 0 \tag{35}$$

$$\sum_i p_{z,i} - p_z = 0 \tag{36}$$

(p_x, p_y, p_z) is the unknown momentum of the mother at the vertex position. The energy E of the mother is calculated after fitting and update of the parameters of the daughters,

$$E = \sum_i E_i = \sum_i \sqrt{(\vec{p}_i^2 + m_i^2)} \tag{37}$$

where m_i is the mass of the candidate. Similarly, the error on E is calculated using the updated covariance matrices of the daughters.

The use of these pseudo-momentum constraints has been proven very efficient in improving error matrix accuracy and speed, especially in complex decay chains. These constraints are implemented in the `BtaMomentumConstraint` class in `VtxBase`.

<u>Momentum</u>

A Lagrange momentum constraint is also available. The constraint is applied in a given reference frame $(\vec{\beta})$ defined by the user. The equations are:

$$\left(\begin{array}{cccc} -\gamma\beta_x & 1+(\gamma-1)\beta_x^2/\beta^2 & (\gamma-1)\beta_x\beta_y/\beta^2 & (\gamma-1)\beta_x\beta_z/\beta^2 \\ -\gamma\beta_y & (\gamma-1)\beta_y\beta_z/\beta^2 & 1+(\gamma-1)\beta_x^2/\beta^2 & (\gamma-1)\beta_y\beta_z/\beta^2 \\ -\gamma\beta_z & (\gamma-1)\beta_z\beta_x/\beta^2 & (\gamma-1)\beta_z\beta_y/\beta^2 & 1+(\gamma-1)\beta_x^2/\beta^2 \end{array} \right) \left(\begin{array}{c} E \\ p_x \\ p_y \\ p_z \end{array} \right) - \left(\begin{array}{c} p_x^{fix} \\ p_y^{fix} \\ p_z^{fix} \end{array} \right) = 0 \tag{38}$$

These constraints are implemented in the `BtaMomentumConstraint` class in `VtxBase`.

<u>Invariant mass</u>

The constraint which forces the candidate to have an invariant mass M is

$$\left(E^2 - p_x^2 - p_y^2 - p_z^2\right)/M^2 - 1 = 0 \tag{39}$$

This constraint has been implemented in the `BtaMassConstraint` class in `VtxBase`.

<u>Beam-spot</u>

With this χ^2 constraint the vertex position in the transverse plane (x, y) is constrained to be compatible with the beam-spot position

$$(x - x_{BS})^2 + (y - y_{BS})^2 = 0 \tag{40}$$

As this is a χ^2 constraint, (x_{BS}, y_{BS}) are introduced as additional parameters in the fit, so their covariance matrix is used to account for the actual size of the beam.

Beam-spot single track

This constraint is equivalent to the previous one with the difference that it is applied to a single track (or candidate) instead of a vertex. It forces the track to intersect the beam-spot position in the transverse plane. The equation of the constraint is similar to the previous one:

$$(x_i - x_{BS})^2 + (y_i - y_{BS})^2 = 0 \tag{41}$$

Energy

Similar to the Lagrange momentum constraint, there is a Lagrange energy constraint which can be applied in a generic reference frame ($\vec{\beta}$). The equation is

$$\gamma E - \gamma\beta_x p_x - \gamma\beta_y p_y - \gamma\beta_z p_z - E^{fix} = 0 \tag{42}$$

This constraint has been implemented in the `BtaEnergyConstraint` class in `VtxBase`.

Beam energy

There is the possibility of applying a beam-energy constraint. It is in fact a particular case of the energy constraint described above, but for the specific reference frame of the $\Upsilon(4S)$, for which $\gamma = \frac{E_{e^+}+E_{e^-}}{m_{\Upsilon(4S)}c^2}$ and $\vec{\beta}\gamma = \frac{\vec{p}_{e^+}+\vec{p}_{e^-}}{m_{\Upsilon(4S)}c}$. Contrary to the previous case, here the constraint is of χ^2 type, so we are accounting for the spread of the beams. The equation defining the constraint is:

$$(E_{e^+} + E_{e^-})\, E - (\vec{p}_{e^+} + \vec{p}_{e^-})\, \vec{p} - m_{\Upsilon(4S)}/2 = 0 \tag{43}$$

where $m_{\Upsilon(4S)} = ((E_{e^+} + E_{e^-})^2 - (\vec{p}_{e^+} + \vec{p}_{e^-})^2)^{1/2}$.

This constraint is available in `BtaBeamEnergyConstraint` class in `VtxBase`.

Zero-lifetime constraint

There is an special constraint used for constraining the position of a zero-lifetime state (known as "resonances" in Beta) [4] to be the same as its mother. This constraint implies that the momentum direction of the non-resonant state has to be aligned with the line that joints its vertex and that of the resonance (its mother). The condition is specified by minimizing the distance between the positions of both particles:

$$(x - x_{res})^2 + (y - y_{res})^2 + (z - z_{res})^2 = 0 \tag{44}$$

where $(x_{res}, y_{res}, z_{res})$ is the vertex previously fitted of the daughter resonance. This constraint helps in improving the vertex resolution for modes like $B^0 \to J/\Psi K^0_S$ (the K^0_S can be used to improve the J/Ψ vertex in order to build the B^0 vertex).

Non-flying constraints for resonances for which only a vertex constraint has been applied are in fact resolved by `GeoKin` by attaching the daughters of the resonances

[4]The exact definition of "resonance" is any state with $c\tau < 1$ nm.

directly to its mother and vertexing all together (vertex constraint). In the case that a kinematic constraint has been also applied, for instance a mass constraint, equation (44) is used instead. The advantage of attaching all the daughters to the same vertex rather than the application of the above equation is twofold: first, it is better adapted to the design of Beta, for which resonances and their mothers share vertices. This means that they share the `BtaAbsVertex` information, in particular they have the same χ^2 and $ndof$. The consequence of not proceeding in this way would be that the χ^2 and $ndof$ of the mother of the resonance would not contain the internal degrees of freedom of the resonance (which would be lost), and therefore the meaning of such quantities would be not clear. Second, and more important, allows vertexing of resonances with poor or no position information (without this approach they would be lost). And there are lots of resonances like this. For instance, in the decay $B^0 \to D^{*-}\rho^+$, $\rho^+ \to \pi^+\pi^0$, both the D^{*-} and the ρ^+ are resonances. When the D^{*-} is reconstructed in the mode $D^-\pi^0$ it has no position information, and when the mode under consideration is $D^0\pi^-$ the information is rather poor (of course it can be largely improved with a beam spot constraint). The ρ^+ has no position information. Vertexing of the B^0 would not be possible using equation (44). If no mass constraint is applied to the D^{*-}, then the vertex of the B^0 is made by vertexing directly the D^0/D^+ and π^-/π^0 from the D^{*-} with the π^+ and π^0 from the ρ^+. In this way everything is then defined and vertexing is possible. We can find lots of other examples like this.

An example which can be vertexed using both approaches is the $B^0 \to J/\Psi K^0_S$. In this case, if a mass constraint is applied to the J/Ψ, then equation (44) is applied. If it is not, then leptons from the J/Ψ decay are vertexed directly with the K^0_S. Extensive testing was made to check the equivalence of results when vertexing in these two ways when the mass constraint is not applied.

The zero-lifetime constraint is applied by default, but it can be disabled using the `setFlyResonances()` modifier, as described in section 2.8.

Line-of-flight

The equations defining the constraint are

$$x - x_0 - t_x s = 0 \tag{45}$$

$$y - y_0 - t_y s = 0 \tag{46}$$

$$z - z_0 - t_z s = 0 \tag{47}$$

where (x_0, y_0, z_0) and (t_x, t_y, t_z) are the fixed position and direction defining the line of flight. The length s is the only parameter left free.

2.6 LeastChiVertexer algorithm

Riccardo

2.7 FastVtx algorithm

The purpose of `FastVtx` is to provide a user-friendly package with emphasis put on low CPU, therefore suitable to high combinatorics vertexing. The former requirement implies that the user can vertex any number of `BtaCandidate` (charged and neutrals) and use the resulting candidate in a further fit. Unlike `GeoKin`, `FastVtx` does not work at the level of the `BtaCandidate`, but at he level of the trajectory parametrized, near the vertex, by a helix (charged) or a slope (neutrals). This avoids hardcoding in the determination of the vertex the local magnetic field. However as soon as four-momenta are to be extracted from the fit (which is the case), using the magnetic field value cannot be avoided.

P. Billoir [7] [8] has explicited the χ^2 linearized near the vertex and provided the least square estimate of the vertex position and refitted trajectories, in terms of matrix algebra. He showed that the estimates can be obtained without inverting a large ($3n$+1,$3n$+1) matrix (n being the number of tracks), reducing the complexity of the algorithm to $o(n)$. Furthermore many of the matrices involved exhibit a large number of zeros: by performing explicitly only the computation of non-zero terms , one gains further on CPU. In `FastVtx` , a comparison between the explicit computations (`BtaFastVertexer`) and the same `CLHEP` matrix based algorithm (`FvtBilloirVertexer`) shows a factor of 2 improvement on CPU.

Finally note that `FastVtx` solves analytically exactly the best χ^2 estimate: there is no idea of convergence in this procedure: there is always one result (unless some track has some problems as having non positive definite error matrix).

2.7.1 Vertexers/Operators/Fitters

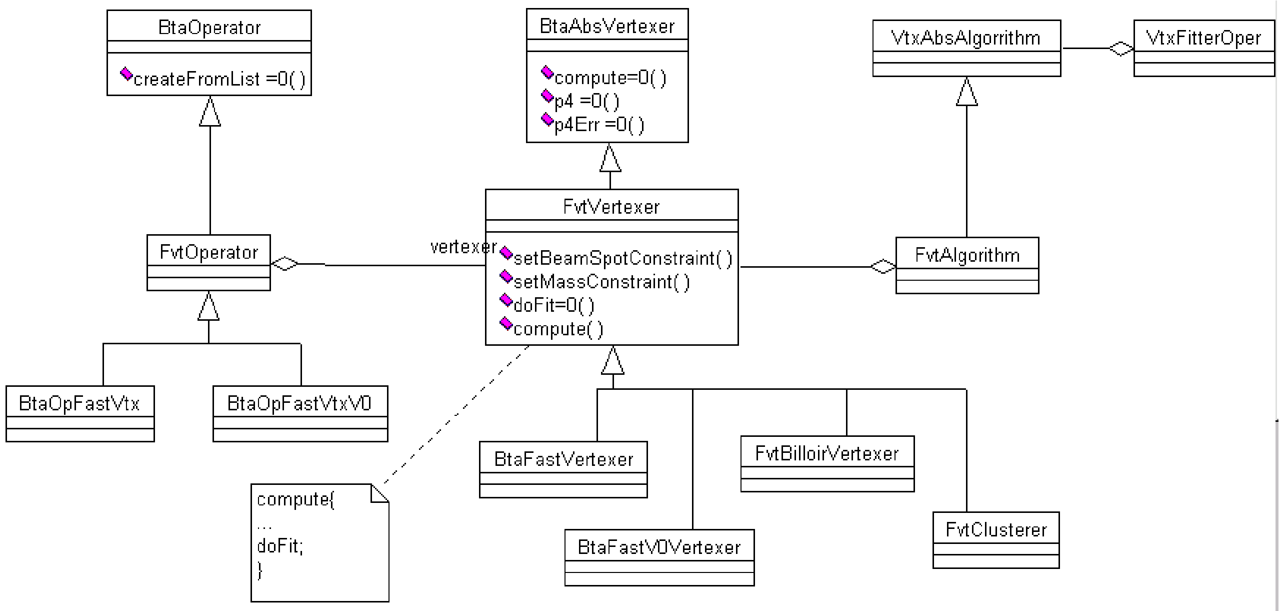


Figure 1: schematic view of FastVtx design

The main engine in `FastVtx` is the *vertexer*. Its goal is to provide a vertex and re-fitted quadrimomenta together with error matrices.All vertexers inherits from `FvtVertexer` (itself deriving from `BtaAbsVertexer` which allows to apply coherently all constraints (beam spot and kinematic) via a template method pattern to any vertexer.

Operators are devoted to providing from a set of `BtaCandidate` another valid `BtaCandidate` with proper parent-daughters relationship. Operators in `FastVtx` derive from `FvtOperator` (itself deriving from `BtaAbsOperator`) and make use of the strategy pattern to get vertex information through an abstract pointer to `FvtVertexer*`. Constraints can be applied to any operator , which forwards them to the corresponding vertexer.

The goal of *fitters* is to fit recursively a tree of `BtaCandidate`. The main class is the *VtxFitterOper* which uses (strategy pattern) a pointer to a `VtxAbsAlgorithm`. Since the interface of a `BtaAbsVertxer` only slightly differs from the one of a `VtxAbsAlgorithm`, the adapter pattern has been used to forward the requests to the underlying concrete `FvtVertexer` algorithm.

This design allow the various aspects of vertexing to be treated by the same method and presents a user friendly interface to the user who can decide whether to use an operator, a vertexer or a fitter depending on his/her analysis.

We now review the different cases of fitting the user may encounter.

2.7.2 Direct charged tacks (as $B^0 \to \pi^+\pi^-$)

- `Vertexer`: BtaFastVertexer
- `Operator`: BtaOpFastVtx
- `Fitter`: VtxFitterOper(`BtaCandidate`*,VtxFitterOper::FastVtx)

This is the simplest case to fit and all matrix computations can be found in [7]. In order to decrease CPU, BtaFastVertexer uses a wrapping to `PXFVTX` F77 routine, written by P. Billoir and which was widely used in DELPHI. In this routine the matrix elements are computed explicitly avoiding all the 0 terms.

In the process of fitting a common vertex and the re-fitted tracks at this vertex, some track-track and track-vertex correlations arise. These terms have generally been neglected in other experiments. In order to check that approximation another vertexer has been developed: `FvtBilloirVertexer` which incorporates all these terms. Due to the complexity of matrix products here one uses CLHEP matrix computations.

Many comparisons have been made between these 2 algorithms, and the conclusions is that in all cases correlations can be safely neglected. For example Fig. 2 shows the resolutions obtained without/with these correlations turned and any difference is barely visible.

Actually these correlations tend to introduce some instabilities (due to their extremely tiny size) and one may end up with non-positive definite matrices. Also the the CPU in for this class is about a factor 2 higher than the Fortran wrapping (about 1 ms per fit). One should however not conclude that Fortran is a factor two faster than C++, but rather that CLHEP based computations are not very fast. For these reasons it is not recommended to use this class.

2.7.3 Direct neutrals (as $D^0 \to K\pi\pi^0$)

- `Vertexer`: BtaFastVertexer

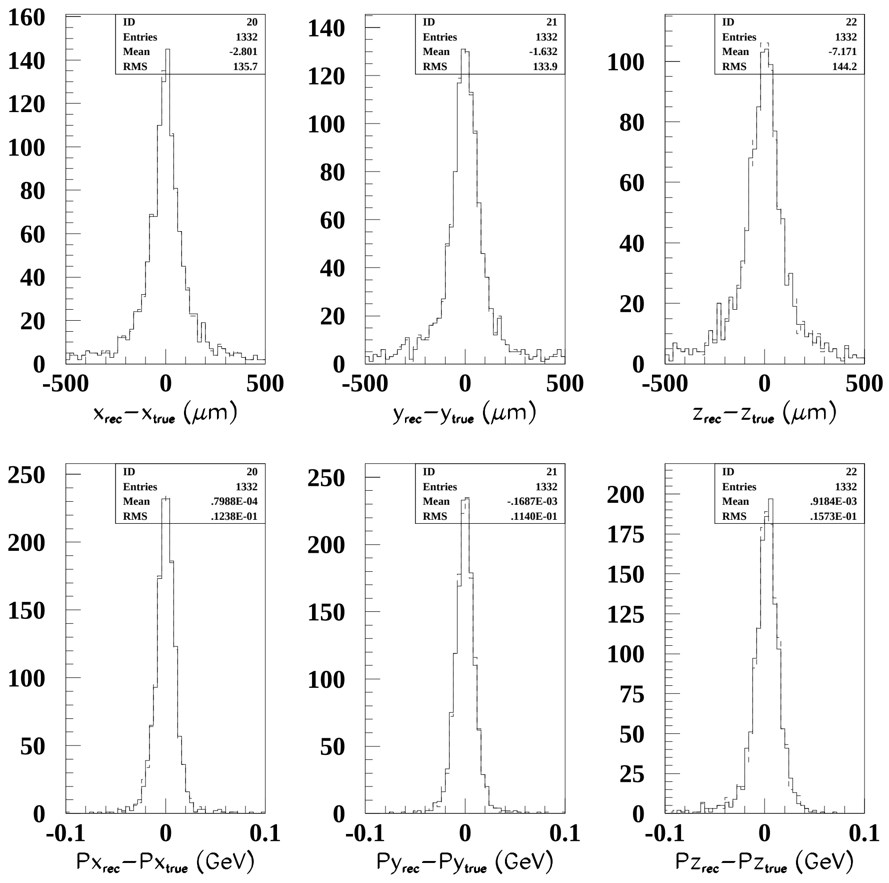


Figure 2: Resolutions obtained in the three directions on the B^0 vertex position (upper plots) and momentum (lower plots) in J/ψ K^0_S decays (where $J/\psi \rightarrow \mu^+\mu^-$ and $K^0_S \rightarrow \pi^+\pi^-$.). The full histogram shows the default `FastVtx` operator and the dashed one the version that includes tracks-tracks and tracks-vertex correlations.

- `Operator`: BtaOpFastVtx
- `Fitter`: VtxFitterOper(`BtaCandidate`*,VtxFitterOper::FastVtx)

In some fits the user wants to add some neutrals (generally γ's or 2 γ's resonances). These "tracks" differs from the charged , since they can just be defined *after* a reference point has been chosen(in order to draw the ray between the reference point and the EMC cluster), they are just affecting the kinematic part of the fit and cannot be used to fit better any position. The reference point in `FastVtx` has been chosen as the *vertex position obtained from charged tracks only.* Therefore `FastVtx` supports this kind of fits if there are enough tracks to define a geometrical vertex. For instance you can fit $D^0 \rightarrow K\pi\pi^0$ but not $B^0 \rightarrow \pi^0\pi^0$ (this latter requires some external output, as the beam spot position, which lies beyond the scope of a purely fitting package).

A kinematic mass constraint can be applied to any of the 2 γ's resonances (as pi^0, η...) through the "setTwoGammaMass" routine in operator/vertexer or usual `BtaCandidate` constraints in the case of fitters.

2.7.4 V^0s (as $K_S \to \pi^+\pi^-$)

- `Vertexer`: BtaFastV0Vertexer
- `Operator`: BtaOpFastVtxV0
- `Fitter`: VtxFitterOper(`BtaCandidate`*,VtxFitterOper::FastVtx)

The case of V^0s is special since one cannot assume anymore that (0,0,0) is a good starting point due to the possible large flight: the χ^2 linearization approximation is becoming improper when the (transverse) radius exceeds about 1.5 cm [8].

For $V0$'s one must therefore propagate the tracks (with their error matrices) up to a good starting point. In `FastVtx` this point is computed as the POCA of the first two tracks. In order to gain on CPU a numerical derivation is performed for the propagation of error matrices (which is significantly faster than using DifNumbers). All terms must be computed in order to get a flat χ probability.

One could use this fitting procedure in all cases and not let the user define. However it has been chosen to let the user decide whether he/she should use BtaOpFastVtx or BtaOpFastVtxV0 for the following reasons:
-it could be unsafe to systematically use the intersect of the first two tracks as a starting point. Think about $D^0 \to K3\pi$: if the first two tracks are very collinear, their POCA could be significantly displaced and the fit consequently be wrong.
-it would result in unnecessary CPU increase.

In the fitter implementation the right vertexer is chosen according to the type of the fitted node, so do not forget to fill it with the usual "setType" method; otherwise a warning message is issued and the *non*-V^0 vertexer is chosen.

2.7.5 γ conversions (ie. $\gamma \to e^+e^-$)

- `Operator`: BtaOpFastV0Add4

The case of gamma conversions is a special case of V^0 fitting. At the decay vertex the two electrons are extremely collinear and the χ^2 linearization breaks down. `FastVtx` supplies an operator which is not a genuine vertexer. The two tracks are propagated to their POCA and a simple 4-vector addition is performed at this point. The returned vertex has not any valid matrix error and its status is set to "unfitted".

Roughly corresponding to the χ^2 probability cut, the user provides in the constructor the minimum distance of approach of the two tracks (xy and z directions separately) required for the operator to return a valid `BtaCandidate`. If these cuts fail, the returned BtaAbsVertex* is simply NULL. If you don't know which value to use for these cuts, just use the defaults.

2.7.6 Composites(as $B^0 \to J/\psi K_S^0$)

- `Vertexer`: BtaFastVertexer or BtaFastV0Vertexer
- `Operator`: BtaOpFastVtxV0 or BtaOpFastVtxV0
- `Fitter`: VtxFitterOper(`BtaCandidate`*,VtxFitterOper::FastVtx)

The case of composite is transparent to the user and is equivalent to any of the previous cases. Note that composites are propagated to the same reference point (0,0,0) than other direct trajectories using DiFNumbers since in some case the distance of propagation from the vertex to the perigee is too small to be performed accurately by numerical derivations.

2.7.7 Exception handling

When a fit fails, for some reason quantified in TrkdelphiPar.hh (enum Status), `FastVtx` simply does not return any vertex: it does NOT make use of the mechanism of BtaAbsVertex::status (however it sets it to "success" when it succeeds for compatibility with other software). However a valid `BtaCandidate` is returned by the operators, which contains the proper daughter relationships but no associated "decayVtx()". This candidate is obtained from simple 4-vector additions at the right point.

2.7.8 Constraints

The Beams-spot and mass (sometimes called kinematic) constraints are available in `FastVtx` . The beam spot constraint is naturally taken into account in the Billoir method itself, while the mass constraint is obtained through an iterative Lagrangian multiplier method. The default precision of this constraint is set to 1 MeV. If you wish to change that (or increase the number of cycles to perform the computation) you have to provide some extra arguments to the "setMassConstraint' method.

Due to the design, these constraints can be applied to any operator,vertexer or fitter uniformally (see next part for example). However since `FastVtx` neglects the correlations, the kinematic constraint is applied *after* the fit (not during as in `GeoKin`). This means that the user may have to define a different geometric tree from the kinematic one.

The default beam spot (as obtained from a simple "setBeamSpotConstraint()") is obtained from the `eventInfo` object which is loaded from the "BtaLoadBeamSpot" module. For B-related analyses one must take into account the possible spread due to the B flight. This is done by default in the module. If you wish to disable that feature and use the genuine beam spot spread, add as an argument:
"setBeamSpotConstraint(false)".

2.7.9 Accessing refitted daughters informations

A more advanced topic is to access some information about the refitted daughters as the refitted daughter trajectories at the vertex or the individual track χ^2 contribution.

The individual χ^2 contribution is obtained through the method

```
double chi2Contribution(const BtaCandidate& bc)
```

of the vertexer, where bc represents the input daughter.

The refitted trajectories is obtained through:

```
const BtaCandidate* fittedCand(const BtaCandidate& bc)
```

which returns the associated refitted `BtaCandidate`.

Note Operators allow an access to the underlying vertexer, through the "vertexer()" method, on which you can therefore apply the previous requests.

For fitters, the usual procedure must be followed, ie. "fitAll()" must be explicitly invoked while daughters refitting is always performed in `FastVtx`.

As an example we show a possible use of `FastVtx` to obtain simply the refitted ΔM distribution of the $D^*D^0\pi$ decay, after applying a beam spot constraint which improves the soft pion refitting:

```
BtaOpFastVtx op;
op.setBeamSpotConstraint();

BtaCandidate dstar=op.create(*d0,*pi);

BtaCandidate* d0fit=op.vertexer()->fittedCand(*d0);
BtaCandidate* pifit=op.vertexer()->fittedCand(*pi);
double deltaM=(d0fit->p4()+pifit->p4()).mag()- d0fit->p4().mag()));
```

2.7.10 `FastVtx` utilities

`FastVtx` has a class `FvtUtil` which contains static services to simplify the user's life:

- `probMass`

  given a p4 with errors, one may want to test the compatibility of this vector with a given mass. The returned value is the χ^2 probability for the kinematic fit on which the user might ant to cut.

- `impactParm`

  computes the impact parameters of a `BtaCandidate` w.r.t to a given BtaAbsVertex. xy and z component are separated and the errors on these values are computed. The main advantage of this method (wrt to a more standard one) is again speed and simplicity.

- `timer()`

  is a service for CPU studies. It returns a pointer to a `FvtTimer`, which is a slightly enhanced version of the Rogue-Wave timer, which keeps also the number of calls to it and can therefore produce a mean time. This service returns a single instance of FvtTimer (via singleton pattern) in order to simplify the user instantiation in any part of the code. So be aware to not duplicate it in many parts of the code (a very unlikely situation). An example could be, for each event:

```
FvtUtil::timer()->start();
.....
FvtUtil::timer()->stop();
```

and get at the end of the job the mean CPU spend through:

```
FvtUtil::timer()->meanTime()
```

- `constrainMasses`

  a utility related to tree fitting. Given the head of a tree apply all subsequent mass constraints to intermediate nodes given their type. This avoids storing many pointers and doing many operations.

2.8 Common vertexing interface and examples

This section is an update of the *Vertexing User's Guide* [4] and from now replaces it.

2.8.1 Building trees. Vertexers

As it has been mentioned in section 2, the vertexing design is based on the concept of *composite* particle. Therefore, before any other operation, we have to build the tree we want to fit. For instance, for a tree like $D^{*+} \to D^0\pi^+$, $D^0 \to K^-\pi^+$, and given the following three pointers to `BtaCandidates`:

```
pi, pi_s, K
```

the syntax would be

```
#include ‘‘BetaCoreTools/BtaOpMakeTree.hh’’
...
BtaOpMakeTree comb;
BtaCandidate* D0 = comb.create(*pi,*K);
D0->setType(K->charge()<0.0 ? ‘‘D0’’ : ‘‘anti-D0’’);
BtaCandidate* Dstar = comb.create(*D0,*pi_s);
Dstar->setType(Dstar->charge()<0.0 ? ‘‘D*-’’ : ‘‘D*+’’);
...
delete D0;
delete Dstar;
```

At this point, `Dstar` is the head of the tree. Arguments between quotes in `setType()` have to be genuine entries in the PDT table. When `BtaOpMakeTree` has no arguments, it just adds four-momenta. Vertexers can be used as argument:

```
#include "VtxFitter/VtxLeastChiVertexer.hh"
...
VtxLeastChiVertexer vertexer;
BtaOpMakeTree comb(vertexer);
```

Other vertexers are:

```
#include "BetaCoreTools/BtaAdd4Vertexer.hh"
#include "FastVtx/BtaFastVertexer.hh"
#include "FastVtx/BtaFastV0Vertexer.hh"
BtaAdd4Vertexer vertexer;
BtaFastVertexer vertexer;
BtaFastV0Vertexer vertexer;
```

`BtaAdd4Vertexer` is the default vertexer for `BtaOpMakeTree`.

The tool `BetaCoreTools/BtaTreeNavigator` provides information about the tree:

```
#include ``BetaCoreTools/BtaTreeNavigator.hh''
...
BtaTreeNavigator navigator(*Dstar);
```

It allows to know the number of final state candidates (`nFinalCand()`), unstable candidates (`nUnstableCand()`), and the number of vertices (`nVertex()`). The tool implements also iterators on the final and intermediate state candidates as well as vertices of the tree:

```
#include <rw/tpslist.h>
#include "Beta/BtaCandidate.hh"
#include "Beta/BtaAbsVertex.hh"
...
RWTPtrSlistIterator<BtaCandidate> finalIter = navigator.finalCandIterator();
RWTPtrSlistIterator<BtaCandidate> unstableIter = navigator.unstableCandIterator();
RWTPtrSlistIterator<BtaAbsVertex> vertexIter = navigator.vertexIterator();
```

It contains also an utility function member,

```
navigator.isCloneOf(*DstarCopy,true)
```

or equivalently,

```
Dstar->isCloneOf(*DstarCopy,true)
```

which can be used to test if two trees are “clones”[5]. If the second argument is changed to `false`, then the PDT condition for final and intermediate states is removed. If this argument is not specified, then `true` is assumed.

2.8.2 Fitters: the `VtxFitterOper` operator

Once the tree has been built with the corresponding constraints (see next subsection), there are several *fitters* which can be used to resolve the tree, fitting it recursively. All of them are integrated in a common operator, called `VtxFitterOper` (available in the `VtxFitter` package). The syntax of this operator is:
This is the main and default *BABAR* vertexing operator.

The second parameter, `algo`, can take one of the following values:

[5]Two composite candidates are “clones” if the topology of the trees is the same and the final stable candidates are copies with the same mass hypothesis (same PDT entry) of the same reconstruced objets (tracks and clusters).

```
VtxFitterOper(  const BtaCandidate&, algType algo=GeoKin,
                const HepPoint& orig=HepPoint(0,0,0) )
```

`GeoKin` GeoKin fitter. *BABAR* default.

`FastVtx` FastVtx fitter.

`Plain` LeastChi fitter. DO NOT USE.

`SingleTrackGeoKin` Beam Spot single track fitter. Makes use of the general GeoKin fitter. This option has to be used when one is interested in constraining a single candidate to pass through a point within errors (defined via a beam spot constraint).

`Add4` Four-momenta addition in fitter version. Useful for debugging.

The third parameter of the `VtxFitterOper` operator is used by `GeoKin` to define an origin for vertexing with neutrals (see below). The `VtxFitterOper::fit()` member function has to be called to perform all the operations. Then, to get the fit version, the following accessors are available:

```
   BtaCandidate getFitted( const BtaCandidate& ) const
```
or
```
   const BtaCandidate* fittedCand( const BtaCandidate& ) const
```
In the previous example of the $D^{*+} \to D^0\pi+$, $D^0 \to K^-\pi^+$ decay,

```
  ...
  setGeoConstraint(*D0);
  setMassConstraint(*D0);
  setGeoConstraint(*Dstar);
  setBeamConstraint(*Dstar,eventInfo);
  VtxFitterOper fitter(*Dstar);
  fitter.fit();
  BtaCandidate* fittedDstar = new BtaCandidate(fitter.getFitted(*Dstar));
  BtaCandidate* fittedD0 = new BtaCandidate(fitter.getFitted(*D0));
  ...
  delete D0;
  delete Dstar;
  ...
  delete fittedDstar;
  delete fittedD0;
```

Another example which involves the `SingleTrackGeokin` option is the case of the D^0 lifetime where one is interested in refitting the D^0 using the beam spot information (which provides an improved beam spot position), and then improve the slow pion track from the D^{*+} decay. The sequence would be the following:

```
// first fit the D0 to the beam spot
setGeoConstraint(*D0);
setBeamConstraint(*D0,eventInfo);
VtxFitterOper fitter(*D0,VtxFitterOper::SingleTrackGeokin);
fitter.fit();
BtaCandidate refitD0 = fitted.getFitted(*D0);
if (refitD0.decayVtx()->status()==BtaAbsVertex::Success) {
  BtaFitParams paramD0 = refitD0.fitParams();
  BbrPointErr beamSpotD0 = BbrPointErr(paramD0.pos(),paramD0.posCov());
  // then fit the slow pion to the smaller beam spot from D0 fit
 setGeoConstraint(*piSlow);
 setBeamConstraint(*piSlow,beamSpotD0);
 VtxFitterOper fitter(*piSlow,VtxFitterOper::SingleTrackGeokin);
 fitter.fit();
 BtaCandidate refitPiSlow = fitter.getFitted(*piSlow);
 if (refitPiSlow.decayVtx()->status()==BtaAbsVertex::Success) {
   // add improved D0 and pi together (summ of four-momentum)
   BtaOpMakeTree comb;
   BtaCandidate* refitDstar = comb.create(refitD0,refitPiSlow);
   refitDstar->setType(refitDstar->charge()<0.0 ? ``D*-'' : ``D*+''));
 }
}
```

It should be noted that this, rather complex sequence, is basically equivalent to the previous one where the beam spot constraint is directly applied to the D^{*+} vertex, provided that access to the refitted daughters is inquired.

The individual χ^2 contribution of daughters is obtained through the following `VtxFitterOper` method:

```
double chi2Contribution( const BtaCandidate& ) const
```

2.8.3 Operation modes

The `GeoKin` fitter have two flavors for operation:

`Fast` Single step (only 1 iteration, i.e. no check for convergence is applied). This operation mode is intendeed for optimizing CPU, and it provides enough precision for fits involving "smooth" constraints. By "smooth" we mean constraints without highly non-linear constraints. This is the case of the vertex constraints (except V^0's). It is recomended not to use this flavor when kinematic constraints are involved, and NEVER use it if there is a mass constraint, since this constraint is far of being linear. The syntax is:

```
#include ``VtxBase/VtxAbsAlgorithm.hh''
...
fitter.setMode(VtxAbsAlgorithm::Fast);
```

`Standard` Default option. Performs iterations testing convergence criteria. The syntax is:

```
        #include ``VtxBase/VtxAbsAlgorithm.hh''
        ...
        fitter.setMode(VtxAbsAlgorithm::Standard);
```

By design, `FastVtx` has only `Fast` mode, except for V^0's.

2.8.4 Resonances: zero-lifetime constraint

As it has been already explained, `GeoKin` contains an special zero-lifetime constraint. The syntax to turn off this condition is

```
  fitter.setFlyResonances();
```

and to turn on (default configuration) is

```
  fitter.setNoFlyResonances();
```

This modifier has to be called before to perform the fit using `fit()` or `fitAll()`. This modifier has no effect on all the other fitters and vertexers.

As a consequence of the different treatment of resonances performed by `GeoKin` (when the zero-lifetime constraint is active) and the other fitters and vertexers, the meaning of χ^2 and $ndof$ is different. As it is explained in section 2.5, resonances in Beta share vertex information with their mothers. As the trees are fit recursively, the vertex information of the mother superseeds that of the daughter resonances. This is not a problem for the vertex position itself (since it will contain the one of the mother), but it is so for the χ^2 and $ndof$ values since they will not contain the internal degrees of freedom of the resonances below. By its design, only `GeoKin` accounts for this and return the correct values. Therefore, any comparison of `GeoKin` with other fitters/vertexers should take into account the different meaning of χ^2 and $ndof$, in case that resonances are involved.

2.8.5 Appling constraints. Kinematic fitting

The constraints are owned by the candidates so they have to be applied on the `BtaCandidate`s before inserting them into a tree. The constraints are applied via global functions (declared in `Beta/BtaConstraint.hh`). Table 1 summarizes the syntax of all the available constraints, as well as for which fitters they are available.

It should be stressed that the constraints must be applied BEFORE building the tree, otherwise they are applied to a "copy" of what is actually used in the fit.

The `GeoKin` fitter does not know by default about any constraint[6]. We can consider four different situations where different minimal constrains are required. They are:

3D Vertexing This is the genuine vertexing where a 3D geometric constraint is applied. It is identified by the following sequence(s):

[6]All the other fitters and vertexers apply always by default a vertex constraint.

Constraint	Syntax	Description	Availability
Vertex	`setGeoConstraint(BtaCandidate&)`	3D vertex constraint	`GeoKin`
Pseudo-momentum	`setMomentumConstraint(BtaCandidate&)`		`GeoKin`
Mass	`setMassConstraint(BtaCandidate&)`	mass constraint with PDT mass	All
	`setMassConstraint(BtaCandidate&,double)`	mass constraint with given mass (MeV/c^2)	All
Energy	`setEnergyConstraint(BtaCandidate&,double, Hep3Vector&boost=(0,0,0))`	Lagrange energy constraint	`GeoKin`
Beam Energy	`setEnergyConstraint(BtaCandidate&, BbrVectorErr&p3eMinus,BbrVectorErr&p3ePlus)`	beam energy with smearing	`GeoKin`
	`setEnergyConstraint(BtaCandidate&, const EventInfo*)`	beam energy from DB with smearing	`GeoKin`
3-momentum	`setMomentumConstraint(BtaCandidate&, Hep3Vector&,Hep3Vector&boost=(0,0,0))`	Lagrange 3-momentum constraint	`GeoKin`
Beam Spot	`setBeamConstraint(BtaCandidate&,) BbrPointErr&beamSpot)`	beam spot with smearing	All
	`setBeamConstraint(BtaCandidate&, const EventInfo*)`	beam spot from DB with smearing	All
Line-of-flight	`setLineOfFlightConstraint(BtaCandidate& Hep3Vector dir,HepPoint point=(0,0,0))`	given direction and origin (Lagrange)	`GeoKin`
	`setLineOfFlightConstraint(BtaCandidate& Hep3Vector dir,const EventInfo*)`	given direction, origin is PV (Lagrange)	`GeoKin`
	`setLineOfFlightConstraint(BtaCandidate& BtaCandidate* cdir)`	direction and origin from candidat (Lagrange)	`GeoKin`

Table 1: Syntax for the available constraints in `VtxFitterOper`.

- `setGeoConstraint(bc)`,

or equivalently,

- `setGeoXYConstraint(bc)`,
- `setGeoZConstraint(bc)`,
- `setMomentumConstraint(bc)`,

where `bc` is a `BtaCandidate`. On top of them, any other kinematic constraint can be applied.

Kinematic fitting When pure kinematic fitting has to be applied, the minimal configuration requires:

- `setMomentumConstraint(bc)`

The kinematic constraint can then be applied on top of this one.

Four-momenta addition This is an special case of the previous one where no additional kinematic constraints is applied on top of the `setMomentumConstraint(bc)`. This is required when the candidate has to be used for further fitting and one is interested only in four-momenta addition. This is the case when we are building composites with no position information, like $K^{*+} \to K^+\pi^0$, where we cannot apply a geometric constraint, `setGeoConstraint(*Kstar)`.

Typical diagnostics alerting you that the constraint configuration is not correct are:

```
VtxGeoKinAlgorithm fatal error: Missing setGeoConstraint(*bta) and
       setMomentumConstraint(*bta).
       Please read again the Vertexing User's Guide.
       Hint: In geometric and geometric+kinematic fits you must impose the
       setGeoConstraint(*bta)+any other geo/kin constraint. In pure kinematic
       fits you must impose the setMomentumConstraint(*bta)+any other kin constraint.
```

or

```
VtxGeoKinAlgorithm warning:  Missing setMomentumConstraint(*bta) in a geometric fit.
    Hint: In geometric fits is recomended to impose also the
    setMomentumConstraint(*bta). Proceed ahead anyway...
```

or

```
VtxGeoKinAlgorithm fatal error: XY and Z vertex constraints do not match.
```

or

```
VtxGeoKinAlgorithm warning: nDof negative. Your constraints could be wrong...
```

An example of pure kinematic fitting is the π^0 (unmerged) vertexing. Here, the momenta of the reconstructed π^0 can be significantly improved by applying a mass constraint to the π^0 and assuming a fixed origin for the γ's. The fit proceeds as follows:

- assume a fixed origin for the neutrals. Default is (0,0,0), but it can be guessed (beam spot, primary vertex, vertex position of related decay);
- the calorimeter measurements for the photons, $(E^{\gamma}_{clus}, x^{\gamma}_{clus}, y^{\gamma}_{clus}, z^{\gamma}_{clus})$, can be converted within errors into 3-momentum $(P^{\gamma}_{x}, P^{\gamma}_{y}, P^{\gamma}_{z})$, assuming the previously guessed origin. This operation is performed by the `AbsCalo` package;
- now we can fit the 3-momenta assuming: i) the mass of the π^0, ii) the clusters are γ's (i.e. null mass).

The fit can easily be configured (we assume the primary vertex as origin for the neutrals):

```
...
gamma1->setType(‘‘gamma’’);
gamma2->setType(‘‘gamma’’);
BtaOpMakeTree comb;
BtaCandidate* pi0Ptr = comb.create(*gamma1,*gamma1);
pi0Ptr->setType(‘‘pi0’’);
setMomentumConstraint(*pi0Ptr);
setMassConstraint(*pi0Ptr);
BbrPointErr primaryVertex = eventInfo->primaryVertex();
HepPoint orig(primaryVertex.x(),primaryVertex.y(),primaryVertex.z());
VtxFitterOper fitter(*pi0Ptr,VtxFitterOper:GeoKin,orig);
fitter.fit();
BtaCandidate pi0Fitted = fitter.getFitted(*pi0Ptr);
...
delete pi0Ptr;
```

The fit then adjusts the momenta $(P_x^\gamma, P_y^\gamma, P_z^\gamma)$ of both photons, in other words, we fit the energy and position of the clusters for the given origin and the π^0 mass constraint. A symmetric error matrix with large values is assigned to the position of the π^0. As it will be described in section 3, this technique is the basis of the $K_S^0 \to \pi^0\pi^0$ reconstruction.

2.8.6 Accessing refitted daughters

Everytime that a new instance of `VtxFitterOper` is made, a clone of the whole tree in input is created. This copy is the one used by the algorithms themselves. When fitting a tree using the `VtxFitterOper::fit()` function member, then only the head of the cloned tree is updated with the new parameters. All their daughters remain unchanged. `VtxFitterOper` offers also the possibility to update the parameters for all the daughters. This is interesting, for instance, to improve the track parameters of the soft pion in a $D^{*+} \to D^0\pi^+$ decay, after applying a beam spot constraint. The way to access this feature is to use `fitAll()` instead of `fit()`.

But, a word of caution. When fitting with `fitAll()`, the parameters and also their covariance matrices of the candidates are updated. In particular, the track/neutral candidates are reset. However, it must be noted that the inter-candidate correlations are not kept (altought they are always calculated by the fitters). If two or more of these candidates are refit in a subsequent fit the loss of these correlations will lead to incorrect results. This problem is automatically resolved with the "lock" of candidates, i.e. when using a candidate for further fitting (as it happens in the recursive fitting of trees) it will not be refit but their parameters (as obtained from the previous fit) will be used instead[7]. This option can, however, be disabled:

```
fittedDstar->invalidatePresentFit();
```

will "unlock" only the head of the tree (technically this just turn the vertex status to `UnFitted`, regardless its previous value). A similar modifier which performs the same operation but now recursively for all the nodes of the tree is:

```
fittedDstar->invalidateFit();
```

With the possibility of unlocking fitted candidates, the problem of inter-candidate correlations goes back: the combination of `fitAll()` and `invalidatePresentFit()/invalidateFit()` will refit the candidate using the updated parameters and covariance matrices, which will produce wrong results.

2.8.7 Verbosities

Some verbosities are available in `VtxFitterOper`. The information they provide are:

- if you are or not refitting daughters;
- option: 0=Add4,1=Plain,2=GeoKin,2=FastVtx,3=not used,4=SingleTrackGeoKin;

[7]BTW, this technique improves very significantly CPU since it avoids to refit candidates everytime they appear in different trees.

- number and type of constraints applied;
- do fly or do not fly resonances;
- operation mode: 0=Fast, 1=Standard.

Verbosities are enabled using the following command:

```
  fitter.setVerbose();
```

before to call the `fit()` or `fillAll()` function.

In addition, for the first 50 calls of the fit operation, `VtxFitterOper` prints out a message telling you if you are attempting to fit a tree containing lock composites:

```
  ...VtxFitterOper: vertex already fitted with status 0. Return the same vertex
```

2.8.8 Error codes

In the process of vertex fitting, complex matrix manipulations, especifically matrix inversions, are involved. Innacuracies in error matrices (in particular tracking correlation terms and neutrals) and largely inconsistent vertices can manifest through mathematical errors. A typical error message which alerts you that an attempted fit failed because an error has been produced when trying to invert a matrix is the following:

```
...::VtxGeoKinAlgorithm.cc(1142): VtxGeoKin engine call with error code = 1
```

Of course, the frecuency of this message depends of how tight is your preselection before vertexing, as well as the complexity of your decay modes and neutrals involved. Starting from *analysis-8* release, only for the first failed fits the above message will be printed out. In *analysis-7*, `VtxFitter V00-08-48-02` has the same feature as that included as default in *analysis-8*.

3 Dedicated vertexing algorithms

3.1 Vertexing of γ conversions

3.1.1 Description

Opposite charged tracks are combined to construct $\gamma \rightarrow e^+e^-$ candidates. In case of real conversions the two charged tracks are parallel at the conversion point. The principle of the algorithm is to calculate the distance between the two tracks. This algorithm was used by the ALEPH Collaboration, and it is also described in [12]. It first determines the distance between the projections of the two tracks in the $x-y$ plane and then computes the distance along the z axis.

Calculation in the $x-y$ plane

The two tracks are circles in the $x-y$ plane. The first step is to find the coordinates of the points A and B defined in fig. 3. They are the points where the two circles are tangent, *ie* where the tangent vectors are parallel.

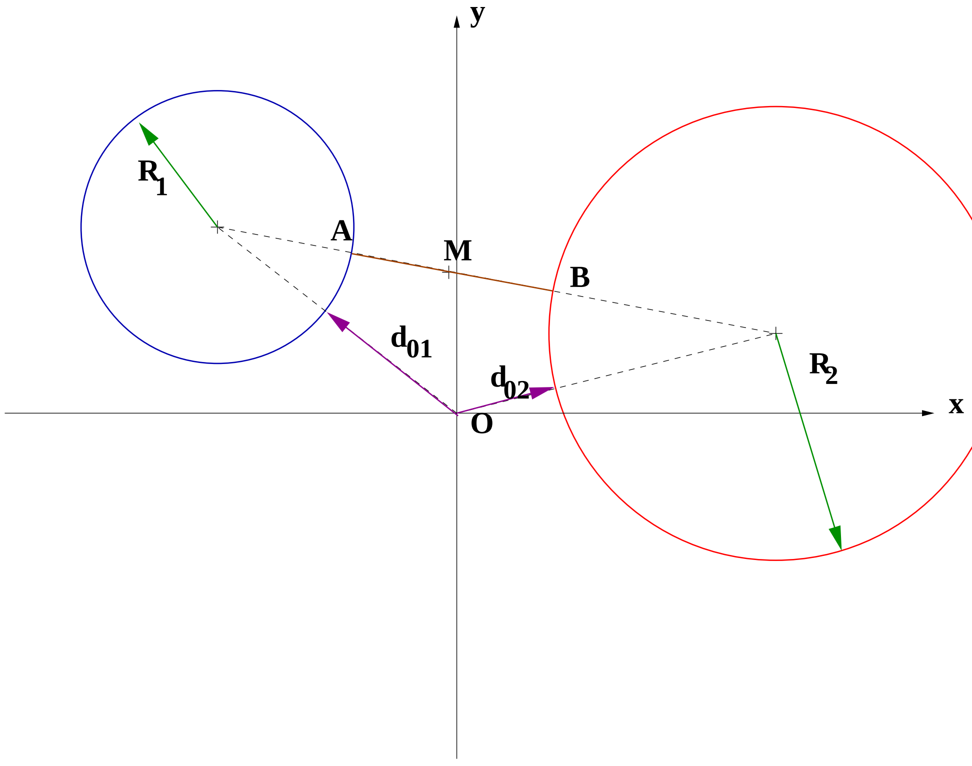


Figure 3: Principle of the conversion algorithm

From the parameters R, the radius of the track, d_0, the distance of closest approach of the track and ϕ_0, the angle with respect to the x-axis of the momentum 3-vector at the point of closest approach, one can calculate the position of the points A and B. All these parameters are available at the micro level.

Two quantities are computed :

- Δxy, the distance between the points A and B. This quantity is defined to be negative if the two tracks intersect, as represented in fig. 4.
- M, the middle of $[AB]$. This point is the Conversion point or the vertex given by the algorithm.

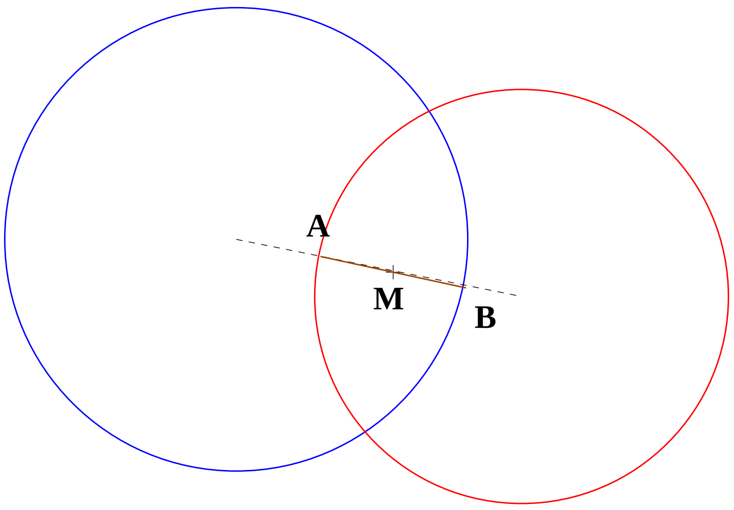


Figure 4: $\Delta xy < 0$

Here are the details. We know that the distance between the origin point and the circle is d_0 at the point of closest approach. At this point the tangent of the circle (the momentum of the particle) makes an angle ϕ_0 with the x-axis. Then if we call x_i $(i = 1, 2)$ and y_i $(i = 1, 2)$ the coordinates of the center of each track, we have :

$$\begin{cases} x_i &= -\,(R_i + d_{0i})\,\sin\phi_i \\ y_i &= (R_i + d_{0i})\,\cos\phi_i \end{cases} \tag{48}$$

Then the coordinates of the points A and B are :

$$\begin{cases} x_A &= x_1 + \frac{x_2 - x_1}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}\,|R_1| \\ y_A &= y_1 + \frac{y_2 - y_1}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}\,|R_1| \end{cases} \tag{49}$$

and :

$$\begin{cases} x_B &= x_2 - \frac{x_2 - x_1}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}\,|R_2| \\ y_B &= y_2 - \frac{y_2 - y_1}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}\,|R_2| \end{cases} \tag{50}$$

The position of the Conversion point M is given by :

$$\begin{cases} x_M &= \frac{x_A + x_B}{2} \\ y_M &= \frac{y_A + y_B}{2} \end{cases} \tag{51}$$

The value of Δxy is given by :

$$\Delta xy \;=\; \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \tag{52}$$

When the distance between the two centers is smaller than the sum of the two radiuses :

$$\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} < |R_1| + |R_2|$$

the two circles intersect and a negative value is assigned to Δxy.

Calculation in the z direction

Once the position of the points A and B has been determined, the elevation of these two points can be calculated knowing z_0 the z coordinate of the point of closest approach of the track and tan λ, the angle with respect to the $x - y$ plane of the momentum at the point of closest approach:

$$z_A \;=\; z_{0A} \;+\; s_A \,\tan\lambda_A \tag{53}$$

and

$$z_B \;=\; z_{0B} \;+\; s_B \,\tan\lambda_B \tag{54}$$

where s_A and s_B are the length on the circle between the point of doca and the points A and B respectively. The length s_A (similarly s_B) is guessed as

$$s_A \;=\; \alpha_A \, R \tag{55}$$

where α_A is the angle between $\vec{O_cP}$ and $\vec{O_cA}$ in the transverse plane where O_c is the center of the circle, P the point of closest approach to the origin. Actually P is not needed because it is on the line joining the origin O (figure 3) and O_c. If the particle moves clockwise, a negative sign is assigned to α_A. We require also that α_A be between $-\pi$ and $+\pi$ adding 2π to it when necessary.

After this step, the z the z coordinate of the point of conversion M is then :

$$z_M \;=\; \frac{z_A + z_B}{2} \tag{56}$$

and the value of Δz is :

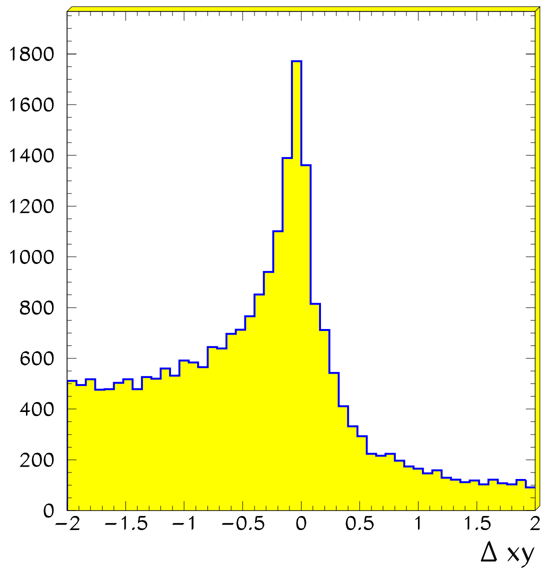
$$\Delta z \;=\; |z_A - z_B| \tag{57}$$

Final cuts

Once all parameters have been calculated, the following cuts are applied to the conversion candidate :

- $|\Delta xy|$ <5 mm,
- Δz <1 cm,
- $m(e^+ \, e^-)$ <30 MeV/c^2, where the invariant mass has been calculated at the conversion point.

The figures 5 and 6 are respectively showing the Δxy, Δz and invariant mass distributions obtained running on *BABAR* data.



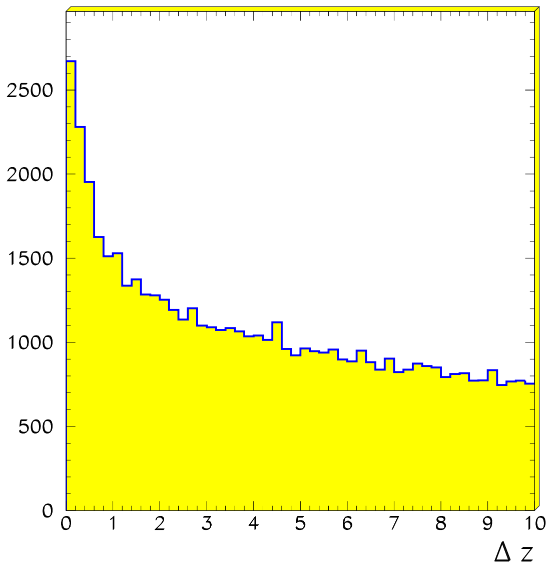


Figure 5: Δxy (left) and Δz (right) distributions (data)

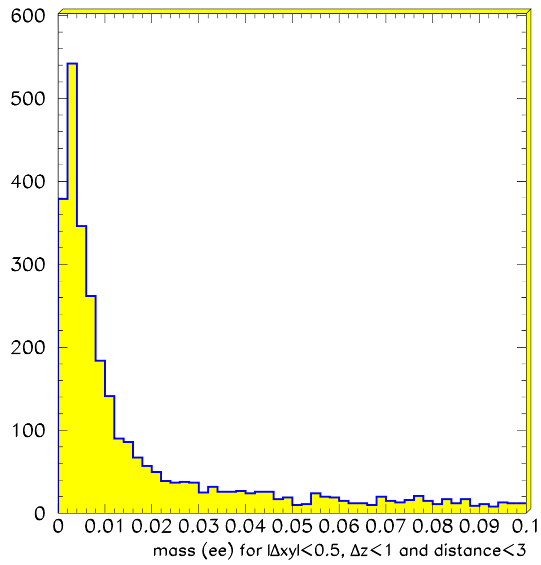


Figure 6: Invariant electron-positron mass distribution (data)

3.1.2 Interface and examples

The conversion algorithm has been implemented as a vertexing operator. The main class is called `VtxGammaConvOper` and can be found in the `VtxFitter` package. The interface to this class is the usual operator interface. The example below shows how to use it :

```
BtaCandidate *electron; // the e-
BtaCandidate *positron; // the e+

// Construction of the operator
VtxGammaConvOper operator;

// Construction of the conversion candidate, conv
BtaCandidate *conv = operator.create(*electron,*positron);
```

The output values of the algorithm (Δxy, Δz, the conversion point and the invariant mass) can be accessed via the resulting `BtaCandidate` using the `VtxGammaConv` vertex class available in the `VtxFitter` package :

```
// delta xy
double deltaxy = ((VtxGammaConv *)conv->decayVtx())->dxy();

// delta z
double deltaxy = ((VtxGammaConv *)conv->decayVtx())->dz();

// Conversion point
HepPoint M = conv->decayVtx()->point();

// mass
double mass = conv->mass();
```

There is no error matrix (yet) associated to the vertex.

3.2 $K^0_S \to \pi^0\pi^0$ vertexing

3.2.1 Description

Non-overlapping π^0 candidates are combined to construct $K^0_S \to \pi^0\pi^0$ candidates. In the initial step the *pi0AllLoose* list entries are combined. The photons are assumed to originate from the detector's origin. The wide π^0 mass window (100–155 MeV/c^2) allows to catch long flying K^0_S decays. Lower energy cuts of 200 MeV on the individual π^0 s and 800 MeV on the K^0_S candidate are currently used in the *KsToPi0Pi0Loose* selector. The mass of succesfull candidates is restricted in the 340-610 MeV/c^2 window. The energy cuts are wide enough to allow the best efficiency for charmonium B decays and can be lowered in the future for more general use. For this energy region (1–3 GeV) the cosines of the photon pair opening angles are restricted to positive values and the cosine of the two π^0 is restricted to the region above 0.6 .

Despite the cuts listed so far, at this stage the number of random combinatorics is too high for the time consuming vertex fit that follows. Instead of a straight K^0_S mass cut, a combined cut on the two π^0 and the K^0_S is used. This is motivated by the simple geometrical argument that as the decay point moves further away from the production point (thus approaching the EMC), the opening angles defined by the detector origin and the photon impact points in the EMC appear smaller than the real ones. The resulting linear relation of the K_S^0 squared invariant mass to the sum of the two π^0 ones can be rotated to form the one dimensional variable $0.17 m_{K_S^0}^2 - m_{\pi^0 1}^2 - m_{\pi^0 2}^2$. A cut in this variable achieves the best possible signal/background ratio and is shown in Figure 7. The 0.00–0.01 region is accepted.

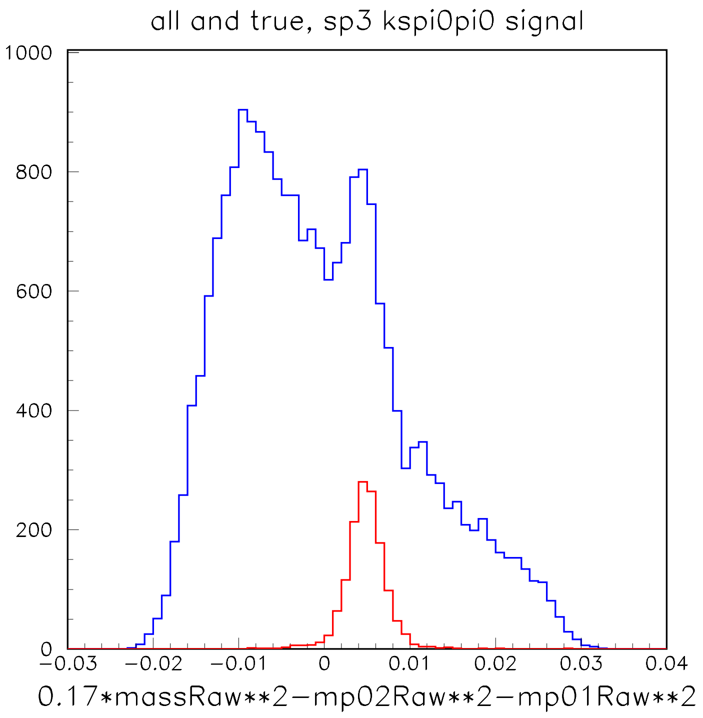


Figure 7: The correlated mass variable for true and random π^0 combinations (MC)

These candidates are the input of the *KsToPi0Pi0_Default* selector, which performs the final vertexing :

The K_S^0 is assumed to fly along the line defined by its momentum and the origin. At regular intervals along this line (2cm) both π^0 s are fitted to their mass, adjusting the energy and position of the cluster (in fact, the free parameters of the fit are the three momentum components). This procedure starts at negative distance ("behind" the origin) and moves forward. The point where the product of the probabilities from the two π^0 mass constraint fits is maximum is chosen as the K_S^0 decay vertex. We demand that this point lie in a region between -10 and $+40$ cm from the primary vertex and that the K_S^0 mass at that point is in the range 446–540 MeV/c^2. The succesfull candidates are in the *KsToPi0Pi0Default* list.

The method has been evaluated using $B^0 \to J/\psi K_S^0$ Monte Carlo. The space resolution can be seen in Figure 8. This shows the distance in 3 dimensions between the true and reconstructed K_S^0 vertex. It is fitted with two gaussians, one constrained at 0 and one with all parameters free. The resolutions are 1.5cm and 4.1cm with the narrow component containing 65% of the total. The wide gaussian is displaced by 2.2cm from the real vertex. Events in the narrow Gaussian have a better energy resolution. This is shown in Figure 8, where events with a vertex resolution of less than 6cm and more than 6cm are shown separately. The dominance of the later events in the tails of the ΔE distribution is clear.

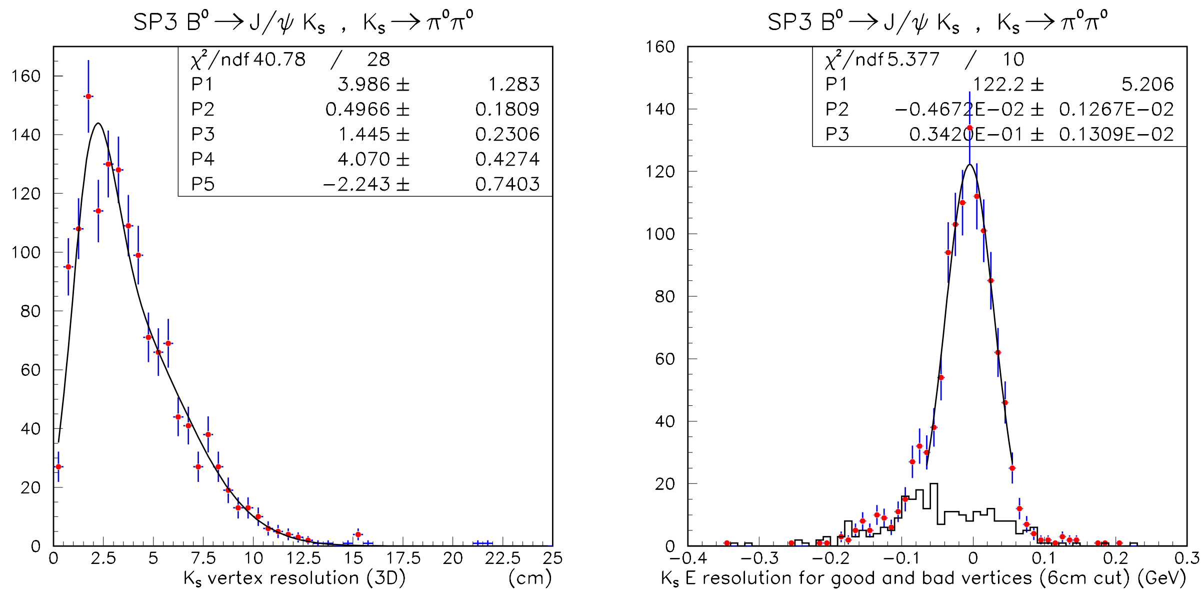


Figure 8: The space and energy resolution of the K_S^0 vertexer (MC)

The inclusive K_S^0 mass peak from real data, exhibiting a resolution better than 10 MeV/c^2, is shown in Figure 9.

3.2.2 Interface and examples

The algorithm to find the best vertex position has been implemented in the `VtxKsToPi0Pi0WalkFit` class, living in the `VtxFitter` package. The main part of the interface is the following:

```
// ctor
VtxKsToPi0Pi0WalkFit(const BtaCandidate *bc, const HepPoint origin);

// Does the fitting
void doFit();
void doFit(int, float, int);

// Accessors
BtaCandidate* getFittedKs();
float getBestprob();
float getBestx();
float getBesty();
float getBestz();
bool getRefitted();
```

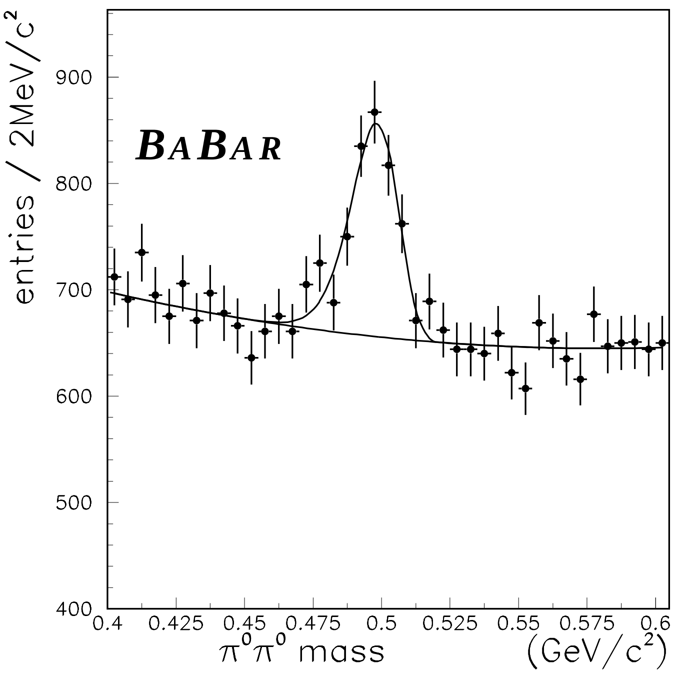


Figure 9: π^0 π^0 mass spectrum and K_S^0 signal from hadronic events

The main function performing the fitting, `doFit(int, float, int)` has three parameters: the first gives the maximum number of steps, and the second and third the maximum distance (positive and negative, respectively) with respect to the origin. The negative distance (thrid parameter) is in fact the starting point of the algorithm. The default configuration is defined in the `doFit()` function: 20,40 cm and -2 cm respectively. The accessors return, respectively: the fitted candidate, the probability and position at best point, and the status whether the fit was able to find a solution.

Below is a real example (taken from the corresponding *CompositeSelector*) of how to fit a $K_S^0 \to \pi^0\pi^0$ candidate. We assume that the the initial (raw) K_S^0, `rawks` pointer, has been built using mass constrained π^0's:

```
#include ‘‘VtxFitter/VtxKsToPi0Pi0WalkFit.hh’’
#include ‘‘VtxFitter/VtxFitterOper.hh’’
...
// access the primary vertex
BbrPointErr primaryVertex = eventInfo->primaryVertex();
HepPoint prodVtxPoint(primaryVertex.x(),primaryVertex.y(),primaryVertex.z());

// make Ks with fitted pi0s at given position
VtxKsToPi0Pi0WalkFit first(*rawks,prodVtxPoint);
```

```
  first.doFit(1,0,0);
  BtaCandidate* newks = first.getFittedKs();

  // find best decay point
  VtxKsToPi0Pi0WalkFit walker(newks,(HepPoint)prodVtx);
  walker.doFit();
  double bestp=walker.getBestprob();
  double bestxv=walker.getBestx();
  double bestyv=walker.getBesty();
  double bestzv=walker.getBestz();
  delete newks;
  newks = walker.getFittedKs();

  // refit Ks at best position
  BtaCandidate* finalks(0);
  if (walker.getRefitted())
    BtaCandidate* fnewks(0);
    HepPoint orig(bestxv,bestyv,bestzv);
    BtaCandidate *atemp = new BtaCandidate(*newks);
    atemp->invalidateFit();
    setMomentumConstraint(*atemp);
    setMassConstraint(*atemp);
    VtxFitterOper fitter(*atemp,VtxFitterOper::GeoKin,orig);
    fitter.fitAll();      // fit Kshort to its mass
    finalks = new BtaCandidate(fitter.getFitted(*atemp));
    delete atemp;
    const BtaAbsVertex* theDecayVtx = finalks->decayVtx();
    // do whatever with theDecayVtx
    if( theDecayVtx!=0 )     {
      ...
    }
    ...
  }
  delete newks;
  ...
```

VtxKsToPi0Pi0WalkFit algorithm makes internally extensive use of VtxFitterOper with the three arguments in order to refit π^0's to each attempted origin.

3.3 $B^0 \to D^{*-}\ell\nu$ vertexing

3.3.1 Description

The goal and motivations for this specific algorithm is to measure the D^0 lifetime in $B^0 \to D^{*-}\ell\nu$ events in both, the $R\phi$ and z projections, which allows, from the vertexing point of view, a simultaneous check of the $R\phi$ and z resolutions and scales. This algorithm was originally proposed in [13].

The experimental fact that the refitting of the soft pion improves dramatically the Δm resolution is a consequence that the determination of the D^0 decay length, the D^0 momentum and the soft pion momentum are all correlated. To account properly for these correlations one needs to fit both the D^0 and D^* vertices simultaneously. And this is not possible with the general purpose fitters. As a consequence, the Δm (and its error) itself can also be easily computed in this way.

The vertexing algorithm is similar in spirit to `VtxTagBtaSelFit` (section 4.2). The list of parameters, unknowns and constraints are the following:

Parameters:

- charged D^0 decay daughters;
- slow pion
- lepton

Unknowns:

- D^0 decay vertex ($\vec{v}_{D^0}$);
- D^* decay vertex ($\vec{v}_{D^*}$);
- D^0 lifetime (τ_{D^0}). It can be split into $R\phi$ and z lifetime.

Constraints:

- XY vertex constraint at $\vec{v}_{D^0}$ for D^0 daughters;
- Z vertex constraint at $\vec{v}_{D^0}$ for D^0 daughters;
- XY vertex constraint at $\vec{v}_{D^*}$ for D^0, π_s, ℓ;
- Z vertex constraint at $\vec{v}_{D^*}$ for D^0, π_s, ℓ;
- $\vec{v}_{D^0} - \vec{v}_{D^*} = c\tau_{D^0}/m_{D^0} \sum_i^{D^0\ daughters} \vec{p}_i$

The output of the algorithm is:

- improved versions of the inputs;
- the values of the unknowns;
- all correlations.

3.3.2 Interface and examples

The algorithm is available in the `VtxB0_DstarlnuAlgorithm` class of the `VtxFitter` package. The specific interface is the following:

```
#include ``VtxFitter/VtxB0_DstarlnuAlgorithm.hh''
...
// ctor
VtxB0_DstarlnuAlgorithm(const BtaCandidate& Dstar, const BtaCandidate& lepton);

// accessors
bool isOK() const;
void chisq(double& chisq, unsigned& ndof);
void Dstar(double& deltaM, double& sigmaDeltaM,Hep3Vector& p3) const;
void D(double& mass, double& sigmaMass, Hep3Vector& p3,
       double& life, double& lifeError) const;
void piSoft(Hep3Vector& p3) const;
void lepton(Hep3Vector& p3) const;
```

4 Δz algorithms

4.1 Introduction and definition of Δz

In measurements of the B lifetime, mixing rate and time-dependent CP-violating asymmetries, we fully or partially reconstruct only one B and then determine the distance, Δz, between the two B decays. The decay length of each B is not observable because of the absence of charged stable particles emerging from the $\Upsilon(4S)$ decay point and so the decay length difference Δz has to be reconstructed. One B meson is kinematically fully reconstructed and the decay vertex position can be measured using all the particles of the decay chain. The other B vertex (tagging B vertex) is reconstructed with the remaining charged tracks in the event. To retain high efficiency this has to be done using inclusive techniques. The tagging B vertex reconstruction is difficult because secondary tracks from short and long lived particles will bias the vertex position. Algorithms try to deal with these problems and minimize their impact using slightly different approaches (see below). Reconstructed V^0 candidates and veto from gamma conversions are used in an attempt to minimize tails and outliers in the resolution. By convention, Δz is defined as the difference between the reconstructed B ($B_{\ CP}$) and the recoiling B ($B_{\ TAG}$):

$$\boxed{\Delta z{=}z_{CP} - z_{TAG}}$$

The general strategy to find the tagging B vertex, already outlined in some earlier studies [21, 22], is the following:

- reconstruct the signal B ($B_{\ CP}$);
- start with a list of all charged tracks in the event, except those from the $B_{\ CP}$ decay;
- remove charged tracks from reconstructed V^0's (K^0_S and Λ), and add the corresponding composite track. Useful V^0's will be those which decay close to the IP and have still SVT information. The remaining ones will be removed to reduce the bias in the vertex position and outliers in the resolution.
- ideally, we should reconstruct D and D_s mesons, and for each successful candidate remove daughter tracks from the list and add the composite track. In practice the fraction of D mesons we can recover is very small, so the vertex tag reconstruction benefits very little (or nothing) by using exclusive D mesons. This step of the strategy has never been implemented.
- fit the tagging side candidates to a common vertex. The way in which this is performed depends on the algorithm. This part is described in detail in section 4.2 and 4.3;
- if the fit is bad (i.e. $\chi^2 > \chi^2_{cut}$), remove the candidate with the largest contribution to χ^2 and then refit (previous step). The details of the stopping criteria are also different for each algorithm, as it is described in 4.2 and 4.3.

There are two different available algorithms, `VtxTagBtaSelFit` and `FvtClusterer` which will be described below. The default one is `VtxTagBtaSelFit`.

4.2 VtxTagBtaSelFit (GeoKin) algorithm

The `VtxTagBtaSelFit` algorithm tries to make use of all the position and kinematic constraints available in the $\Upsilon(4S) \to B\overline{B}$ decay. The $B\overline{B}$ pair creation point is reconstructed by intersecting the reconstructed signal B_{CP} with the beam spot elipsoid: the line of flight of the B_{TAG} is given by the (reverse) momentum vector of the reconstructed B_{CP} and its vertex, and the interaction point is estimated from the intersection of this line with the beam spot position in y. In practice, the information from the beam spot can be reduced to just the y coordinate (figure 10).

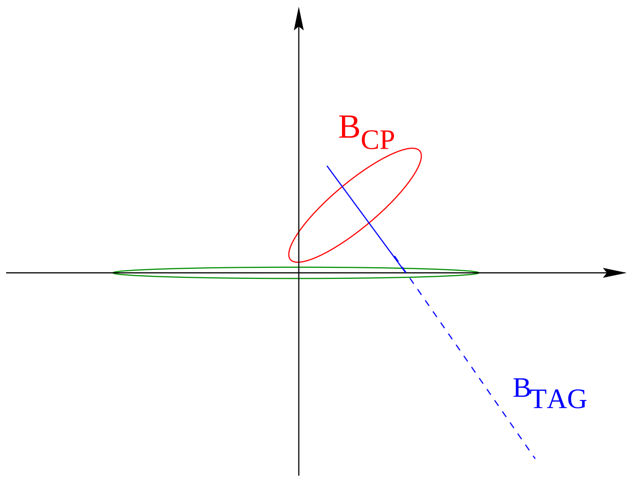


Figure 10: Geometry of $\Upsilon(4S) \to B\overline{B}$ decay in transverse plane. The line of flight of the B_{TAG} is given by the (reverse) momentum vector of the reconstructed B_{CP} and its vertex. The interaction point is the intersection of this line with the beam spot position in y.

Due to the very small size of the beam-spot in y, the decay path $L_z^{CP} = z_{CP} - z_{\Upsilon(4S)}$ of the B_{CP} can be extracted insuring that the direction of the line connecting the $\Upsilon(4S)$ and the vertex of the B_{CP} is given by the fitted momentum of the reconstructed B_{CP}. In the limit where $\sigma_y(\Upsilon(4S)) \approx 0$,

$$L_z^{CP} \approx \frac{y_{CP} - y_{\Upsilon}}{p_{y,CP}/p_{z,CP}} \tag{58}$$

The dominant factor in the precision on L_z^{CP} is the accuracy on y_{CP}: the error on L_z^{CP} is, however, about one order of magnitude larger than the error on y_{CP}. Since typically $\sigma(y_{CP}) \approx 45\ \mu$m, the final precision on L_z^{CP} is very poor.

From momentum conservation, the momentum of the tag B meson, B_{TAG}, is known:

$$\vec{p}_{TAG} = \vec{p}_{\Upsilon(4S)} - \vec{p}_{CP} \tag{59}$$

From this momentum and the $B\overline{B}$ pair creation point we can form a "pseudo-track" which can be used to fit a common vertex as any other tag side candidate. From the knowledge of

$\vec{p}_{TAG}$ from equation (59) and the $B_{\ TAG}$ vertex position from the vertex fit, one can get an estimate of the decay path for the tag side, L_z^{TAG}. In the $\sigma_y(\Upsilon(4S)) \approx 0$ limit,

$$L_z^{TAG} \approx \frac{y_{TAG} - y_{\Upsilon}}{p_{y,TAG}/p_{z,TAG}} \tag{60}$$

Unfortunately, as for the case of the $B_{\ CP}$ case, the precision on L_z^{TAG} is extremelly poor.

The method above assumes that the momentum of the $B_{\ CP}$ is known, which implies full reconstruction. Therefore, in the case of partial B reconstruction the constraint from the $B_{\ TAG}$ pseudo-track cannot be used, and only a beam-spot constraint to the tag side candidates can be applied. It should be noted that, contrary to the previous case, the beam-spot used in this configuration is applied to the B decay point (B production point in the previous case), so the beam spot size must be modified properly to take into account the small transverse flight of the $B_{\ TAG}$ due to the B lifetime and the small $\Upsilon(4S) \to B\overline{B}$ energy release (`beamSpotBFlight()` event accessor, see section 6.5).

4.2.1 Parameters, unknowns and constraints

The $B_{\ TAG}$ vertex fit strategy described above is rather simple to implement using the `GeoKin` general engines described in section 2.5, so a simultaneous fit of all physical quantities can be performed. The problem basically consists in the definition of the list of parameters, unknowns and constraints.

Parameters:

- position and momentum of tagging side candidates: $(x, y, z, p_x, p_y, p_z)_i$, $i = 1, ..., n$, where n is the total number of recoiling candidates (with defined position information);
- position and momentum of B_{CP} candidate: $(x_{CP}, y_{CP}, z_{CP}, p_{x,CP}, p_{y,CP}, p_{z,CP})$;
- position of interaction point in transverse plane: $(x_{\Upsilon}, y_{\Upsilon})$;
- three-momentum of the beams: $(p_{x,e^-}, p_{y,e^-}, p_{z,e^-})$, $(p_{x,e^+}, p_{y,e^+}, p_{z,e^+})$.

Unknows:

- position and momentum of the tagging B: $(x_{TAG}, y_{TAG}, z_{TAG}, p_{x,TAG}, p_{y,TAG}, p_{z,TAG})$;
- decay lengths for CP and TAG B's: L_z^{CP}, L_z^{TAG}.

Constraints:

$$\text{XY}\ Vertex\ constraint \quad \times\ n\ candidates \tag{61}$$

$$\text{Z}\ Vertex\ constraint \quad \times\ n\ candidates \tag{62}$$

$$p_{x,CP} + p_{x,TAG} - p_{x,e^-} - p_{x,e^+} = 0 \tag{63}$$

$$p_{y,CP} + p_{y,TAG} - p_{y,e^-} - p_{y,e^+} = 0 \tag{64}$$

$$p_{z,CP} + p_{z,TAG} - p_{z,e^-} - p_{z,e^+} = 0 \tag{65}$$

$$x_{CP} - x_{\Upsilon} - \frac{p_{x,CP}}{p_{z,CP}} L_z^{CP} = 0 \tag{66}$$

$$y_{CP} - y_{\Upsilon} - \frac{p_{y,CP}}{p_{z,CP}} L_z^{CP} = 0 \tag{67}$$

$$x_{TAG} - x_{\Upsilon} - \frac{p_{x,TAG}}{p_{z,TAG}} L_z^{TAG} = 0 \tag{68}$$

$$y_{TAG} - y_{\Upsilon} - \frac{p_{y,TAG}}{p_{z,TAG}} L_z^{TAG} = 0 \tag{69}$$

$$z_{TAG} - z_{CP} + L_z^{CP} - L_z^{TAG} = 0 \tag{70}$$

Equations (61) and (62) are the standard vertex constraints as described in section 2.5, equations (16) and (17).

In the case of partial reconstruction, the "pseudo-track" constraint cannot be applied. This is equivalent to remove constraints (63), (64), (65), (66), (67), (68), (69) and (70). The position and momentum of B_{CP} candidate and the three-momenta of the beams are anymore needed, as well as L_z^{CP} and L_z^{TAG}. In this case, the beam-spot constraint can be applied to the tag side candidates in the standard way as it was described in section 2.5, equation (40).

Let us remark that when using the beam-spot as described in section 6 the tiny effect of the xz tilt is automatically accounted for. Given that the beam-spot constraint is basically effective only in the y component, its effect is completely negligible.

Owing to the pinched geometry, L_z^{CP} and L_z^{TAG} have a positive correlation close to 1, therefore when we compute the difference Δz=$L_z^{CP} - L_z^{TAG}$ the effect of the correlation is basically removed, and this observable becomes much more precise than the separate decay lengths.

The beam energies have spreads of about 5.5 MeV (electrons) and 2.5 MeV (positrons). The actual $\Upsilon(4S)$ momentum distribution is a convolution of the Gaussian beam overlap distribution with the Breit-Wigner probability for $\Upsilon(4S)$ production and the phase space for producing $B\overline{B}$ pairs. Generator level studies [17] show a Gaussian $\Upsilon(4S)$ momentum distribution with a width of about 6 MeV. The effect of the momentum spread is two-fold. Firstly, at Δz reconstruction level (as the algorithm make use of the total momentum constraint), secondly when estimating Δt, as described in section 4.5. The impact of the spread when reconstructing Δz has been evaluated to be completely negligible since errors are dominated by reconstruction[8]. Therefore, in the default configuration, $\vec{p}_{y,e^-}$ and $\vec{p}_{e^+}$ are not considered as parameters but fixed quantities.

The simultaneous vertex fit involving all the constraints (61)-(70) makes the understanding of the algorithm rather difficult. Reference [1] documents the systematic studies performed which help to understand the details. Roughly speaking, here is the mechanism. When the $B_{\ CP}$ direction is perperdicular to the direction in which the beam size is small (y), $\phi \sim 0$, then the direct z constraints provided by the reconstructed $B_{\ CP}$ do not help. But in this case the beam spot constraint applies directly to the B decay point, providing the whole resolution of the beam spot in y ($\sim$ 15 μm) (this is equivalent to apply a beam spot constraint alone). In this case, the improvement in z resolution comes basically from exploiting the $y-z$ and to a less extend the $x-z$ correlations of the tag side tracks. When ϕ is maximal, then $B_{\ CP}$ constraints apply providing direct constraint on z, compensating

[8]χ^2 contributions from this term are about ten order of magnitude smaller than the rest.

the loss of beam spot constraint effectiveness as consequence of the $B_{\ CP}$ transverse flight. As the change of effective resolution of the beam spot from $\phi = 0$ to $\phi = \pi/2$ is not dramatic (from $\sim$ 10 to $\sim$ 30 μm) compared with the tracking resolution, the improvement in resolution (with respect to no constraints) is mostly provided by the beam spot alone. A part of the rather marginal gain in resolution provided by the $B_{\ CP}$ constraints, their purpose is basically to provide a more robust approach to select the tag side tracks, reducing charm contamination and fraction of outliers. Thus Δz resolution and scales are finally rather insentivite to resolution and scales in $B_{\ CP}$, and only $y-z$, $x-z$ correlations from the tagging side tracks and the reconstructed vertex, and the beam spot determine the improvement in error on the reconstructed Δz, compared to the case when no constraints are applied. This mechanism allows to improve Δz track selection and resolution, without spoiling the very basic assumption in all the analyses that the resolution function is independent of the specific $B_{\ CP}$ mode. Moreover, from the constraint equations (67) and (69), and to a less extend (66) and (68), it can be seen that L_z^{CP} and L_z^{TAG} are sensitive to the displacement of the CP and TAG vertices with respect to the beam spot position. As it has been already said above, its resolution is dominated by that of the CP and TAG vertices). However, when computing Δz as the difference between L_z^{CP} and L_z^{TAG}, it turn out that finally the main variable, Δz, is rather insensitive to the displacement of the vertices with respect to the beam spot, and the sensitivity is mainly to the relative displacement of the TAG and CP vertices.

Given a set of tag side candidates and the $B_{\ CP}$, the $B_{\ TAG}$ vertex fit has been implemented in the class `VtxTagAlgorithm` in the `VtxFitter` package. The convergence criteria in the minimization process requires a change in χ^2, as given by equation (14), between two successive iterations less than 0.001, with a maximum of 10 iterations.

4.2.2 Convergence criteria

The iterative procedure for track selection removes the candidate with the largest contribution to the total χ^2. The procedure is iterated until there are no tracks contributing more than 6 units to the χ^2 or only n tracks remain, where n (which can be 0) is a parameter to be tuned. The use of the "pseudo-track" allows us recover events with only an opposite track, making the algorithm highly efficient.

The best value of parameter n has been determined studying 36K $B^- \to D^0(K^-\pi^+)\pi^-$ events. In this study we used the V^0's as described in section 4.4, and then we compared $n = 0$ and $n = 2$. It should be noted that even in the case $n = 2$, events with a single track (candidates) can appear. The motivation for using $n > 1$ is because we know from previous studies that a large fraction of the outlier contribution to the Δz resolution function is due to events for which the Δz was reconstructed using a single candidate. The requirement of an additional track in the vertex (when there are more than one in the tagging side) introduces, in principle, a lever arm which can help in rejecting events with poor information. Additional quality cuts based on the χ^2 probability will be then needed to reject events with poor compatibility with single vertex. Figure 11 shows the residuals, pulls and number of candidates used to make the vertex for three configurations: $n = 2$ with a cut on the global $p(\chi^2)$ of the vertex at 0.1%, $n = 2$ without $p(\chi^2)$ cut, and $n = 0$ with no cut on $p(\chi^2)$. From these plots it is concluded that by requiring $n = 2$ we increase the fraction of outliers, but they are killed and reduced by requiring an overall χ^2 probability of the vertex fit higher than 0.1%. In such a configuration there is a gain in outlier reduction with respect to the case $n = 0$. From these plots it can also be seen that only the number of events with 0,1 and 2 candidates are different among the different configurations. $n = 2$ basically moves events from 1 to 2 candidates, and the overall χ^2 cut only affects events with 2 tracks.

When $n = 0$, the fraction of events having a single track is $\sim 5.5\%$. Therefore, if we throw away these events then we have a substantial drop in efficiency. When $n = 2$, this fraction is only $\sim 1.7\%$. The drop in efficiency caused by the $prob(\chi^2)$ cut is $\sim 3\%$. As there is no overlap between the single track events and those events at $prob(\chi^2) < 0.1\%$, the total drop in efficiency requiring both conditions is $\sim 4.7\%$. However, it has been verified that there is no gain by rejecting single track events, once $n = 2$ and $prob(\chi^2) > 0.1\%$ have been applied. In conclusion, in such a configuration, there is a net outlier reduction with a small drop in the efficiency. The issue here is whether the $prob(\chi^2)$ cut introduces differences in efficiencies among different B modes, as well as whether enhances data/Monte Carlo differences. This is investigated in reference [1]. Another problem which can arise is the fact that good events for vertexing can also be rejected. For instance, if there is a hard prompt lepton and kaons from a secondary decay, it can happen that the two final tracks used to make the vertex are the lepton and a kaon. If the D meson did fly far, more likely the χ^2 probability of the vertex will be very small, and the vertex will be rejected by the χ^2 cut, when in fact, a good vertex could be defined using only the lepton and the pseudo-track, as the $n = 0$ configuration would do. Examples of events falling into this category in the data have been found [24]. In order to minimize these effects and keep as high as possible the reconstruction efficiency, $n = 0$ was adopted as default configuration. In such a configuration it has been verified that applying a cut on $prob(\chi^2)$ at 0.1% has basically no effect (the cut is somehow implicit in the track removal strategy), beyond the small drop in efficiency.

The iterative procedure for track removal is available in the `VtxTagBtaSelFit` class in the `VertexingTools` package. The class makes use of `VtxTagAlgorithm`, living in `VtxFitter`. Both classes have a common interface.

4.3 FvtClusterer (FastVtx) algorithm

4.3.1 Description

The `FvtClusterer` algorithm is a `FastVtx` vertexer (see section 2.7.1) whose role is to "clusterize" a set of (charged)tracks to the best common vertex, based on purely geometrical criteria. It can therefore be used to determine the tag vertex of an event (once the reconstructed B tracks are subtracted) or fit rapidly the common vertex of an event ("hadronic" beam spot, see sections 5 and 6, and reference [10]).

In order to remove V^0's daughters and badly reconstructed tracks, a common vertex is formed with all charged tracks, and unless it satisfies a *global* $p(\chi^2, ndof)$ vertex cut, the track with the higher χ^2 contribution is rejected. The procedure is repeated iteratively until the cut is satisfied or until the minimum value for the χ^2 vertex allowed is exceeded.

For the Btag vertex, the differences with `VtxBtaSelFit` are therefore of two sorts:

- the stopping condition is based on the global χ^2 of the vertex.
- no cinematic information is used (however a beam spot constraint can be applied)

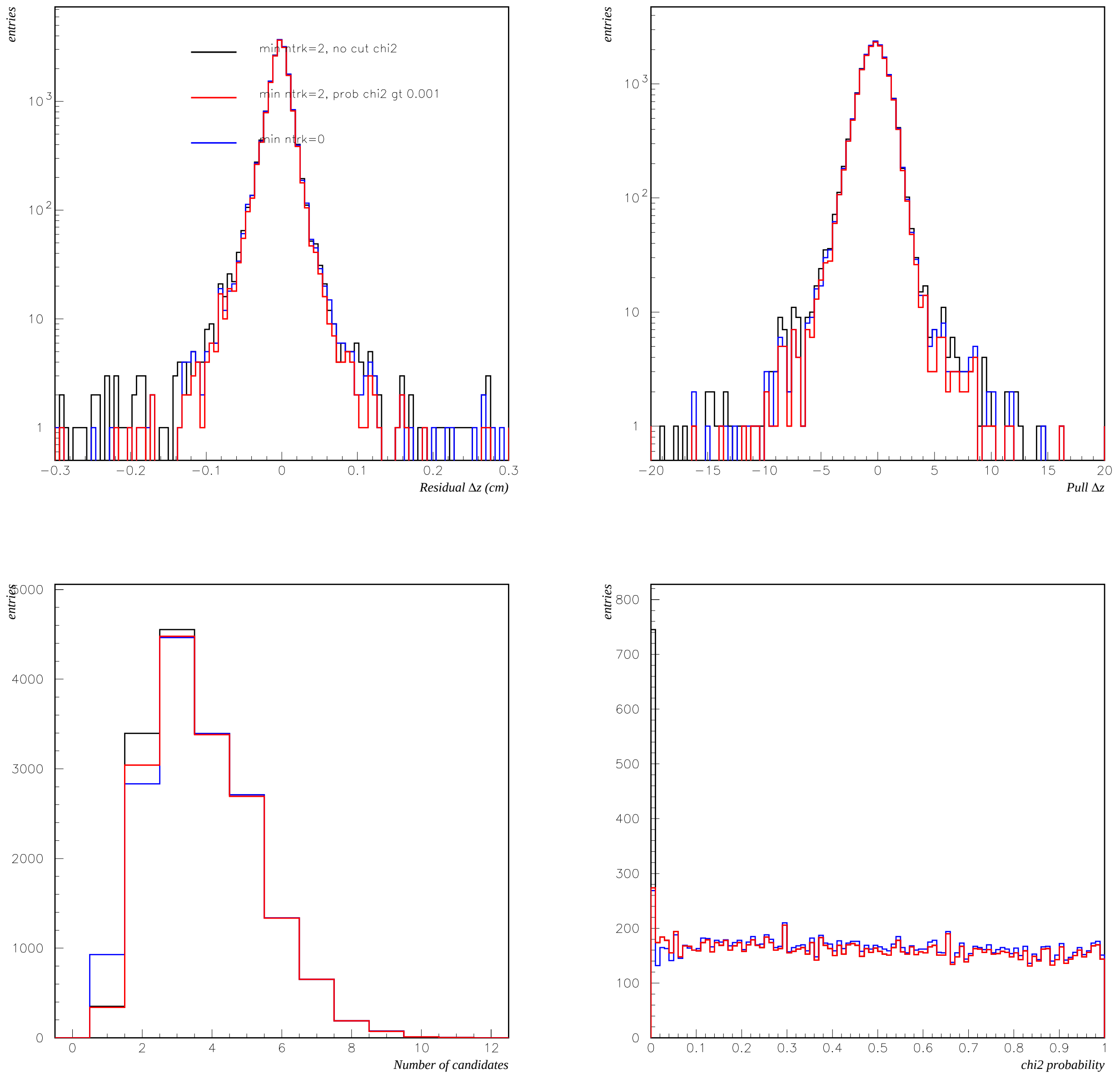


Figure 11: Comparison of Δz residuals (top-left) and pulls (top-right), number of tracks (bottom-left) used to make the vertex and χ^2 probability (bottom-right) for $n = 2$ with a cut on the global $p(\chi^2)$ of the vertex at 0.1%, $n = 2$ without $p(\chi^2)$ cut, and $n = 0$ with no cut on $p(\chi^2)$.

4.3.2 How to

The class use is equivalent to that of any `FastVtx` vertexer (section 2.7.1). The constructor defines the upper limit of χ^2 probability allowed and it is suggested to use the default value. Recall that `FvtClusterer` provides an un-owned pointer , so that it is the client responsibility to delete it. An example of use could be:

```
FvtClusterer cluster; \\uses default value of 0.001 for pchi2 cut
cluster.setBeamSpotConstraint(); \\use default BSC

BtaAbsvertex* tagB=cluster.compute(iter);

...//retrieve information from BtaAbsvertex*

delete tagB;
```

a few remarks:

- the iterator on charged tracks (iter) must be cleaned from B_{CP} tracks.
- concerning the beam-spot constraint details see 2.7.8
- `FvtClusterer` does not compute immediately the δz variable: it is up to the user to determine the B_{CP} vertex in order to compute it.

After the fit, the user can access the individual χ^2 contributions and the refitted tracks trajectories in the way described in 2.7.9. Another important piece of information is to know whether a given track was rejected during the iterative procedure. This is performed through the "status" method of the vertexer in the following way:

```
FvtCandidate::Status status=cluster(*trkptr);
```

The status information is of 3 types:

- `locked`: the track is used in the fit
- `unlocked`: the track was rejected
- `nailed`: the track is always kept in the fit (see next part)

4.3.3 Nailing tracks

The user has the possibility to fix ("nail") a given track, so that it is never rejected in the iterative procedure. This can be used for instance to enhance tagging/vertexing correlation: in order to reduce the charm lifetime bias one can nail the lepton so that the vertex converges onto it.

The way to proceed is to explicitly create a list of "FvtCandidate" (which is build from a `BtaCandidate`plus relevant `FastVtx` information) through:

```
HepAList<FvtCandidate> fvtList;
BtaCandidate* trkptr(0);

while(trkptr=iterCh()){

//if not lepton:
fvtList.append(new FvtCandidate(*trkptr,FvtCandidate::locked));

//if lepton:
fvtList.append(new FvtCandidate(*trkptr,FvtCandidate::nailed));
}
```

and then:

```
HepAListIterator<FvtCandidate> itfvt(fvtList);\\create iterator

FvtClusterer cluster; \\uses default value of 0.001 for pchi2 cut
cluster.setBeamSpotConstraint(); \\use default BSC
BtaAbsvertex* tagB=cluster.doFit(itfvt);

...//retrieve information from BtaAbsvertex*

delete tagB;
```

`FvtClusterer` predefines an algorithm that increases tagging-vertexing correlation maximally: the leptons are nailed, and the kaon "unlocked"(ie. not used). It has a more user-friendly interface through:

```
BtaAbsVertex* tagVtx(HepAList<BtaCandidate>& tags,
       HepAList<BtaCandidate>& chtrks);
```

where the user just provides the list of tags (leptons and kaons) and the list of charged tracks (the two may overlap).

Studies [11] show that this procedure does indeed minimize the bias for right lepton tagging, but (as expected) increases it for wrong leptons: one therefore increases the vertex asymmetry between right and wrong tags which is not a desired feature. For kaons the difference is negligible. Therefore the relative gain is not obvious and it has been decided to *not* exploit these correlations.

The common interface (section 4.6) makes the use of the algorithm in a much more userfriendly, apart of many other advantages as described there. It is therefore strongly recomended to use it instead of the particular implementation of each algorithm.

4.4 V^0's and γ conversions

In an attempt to reduce bias and outlier contribution to the resolution function, V^0's and γ conversions are used for vertexing. Special care have to be put in the selection of the V^0 candidates since fake candidates can potentially result with the opposite effect. The main effect from V^0's and γ conversions is expected from those decaying far away of the interaction region.

A dedicated sequence has been implemented in `VertexingTools`, called `VtxTagSequence`, which provides the three lists inputs for vertexing: charged tracks (*GoodVtxTagTracks*), V^0's (*V0sVtxTag*) and conversions (*gammaConversionVtxTag*). Currently the *GoodVtxTagTracks* list is identical to *GoodTracksVeryLoose*. The selection cuts used for K_s^0's, Λ's and γ conversions is described below. For the optimization of selection criteria, about 200K B^0 events from the "TagMix cocktail" have been used.

The selection cuts for K_s^0 (Λ) are the following:

- the probability of the vertex fit (without mass constraint) is higher than 0.1%;
- the decay length with respect to the primary vertex in the transverse plane is higher than 2 (5) mm;
- the aperture angle of the daughters greater than 200 mrad;
- mass of the reconstructed candidate within 7 (4) MeV of the nominal mass.

Figure 12 shows the mass distribution of the selected candidates. The dotted histograms represent the background as predicted from the Monte Carlo truth matching. The purity within the mass window is estimated to be about 65%. It should be noted that these plots are obtained by considering also the tracks used for reconstructing the B meson. This tracks will contribute mainly to the combinatorial background since the number of real K_S^0 in the B side is very small, and in the case of the Λ's is zero. As a result the purity of the final candidates finally used for vertexing is slightly higher: it has been evaluated to be higher than 70% for both K_S^0 and Λ's (in the latter case there are large statistical uncertainties due to the very small number of candidates found).

γ conversions are used for veto of tracks to be used for vertexing. The selection cuts are the following:

- the xy (z) distance between the two tracks is required to be less than 5 mm (1 cm);
- the three-dimensional distance between the tracks <3 cm;
- the invariant mass of the pair of tracks < 10 MeV/c^2.

Figure 13 shows the mass distribution of the selected candidates. As above, the dotted histograms represent the background as predicted from the Monte Carlo truth matching. The conversion contents of the sample is estimated to be about 25%.

The effect of using V^0 and γ conversions on the Δz outliers has been evaluated running over 36K $B^- \to D^0(K^-\pi^+)\pi^-$ events. Figure 14 compares the Δz residuals and pulls with and without V^0's and conversions in the vertex tag reconstruction. There is an indication that when using them there is a net reduction of the number of events at large residual and to a less extend at large pull. In the previous figure one can also see that there is an almost perfect overlapping between the case that all the reconstructed V^0 and conversions are used, and when only the truth candidates (from Monte Carlo matching) are used. This is an indication that with the present selection fake candidates are not introducing additional outliers. In all cases the vertex tag reconstruction efficiency is the same. The mean number of K_S^0 and Λ used to make a vertex tag is 0.09 and about 0.001, respectively.

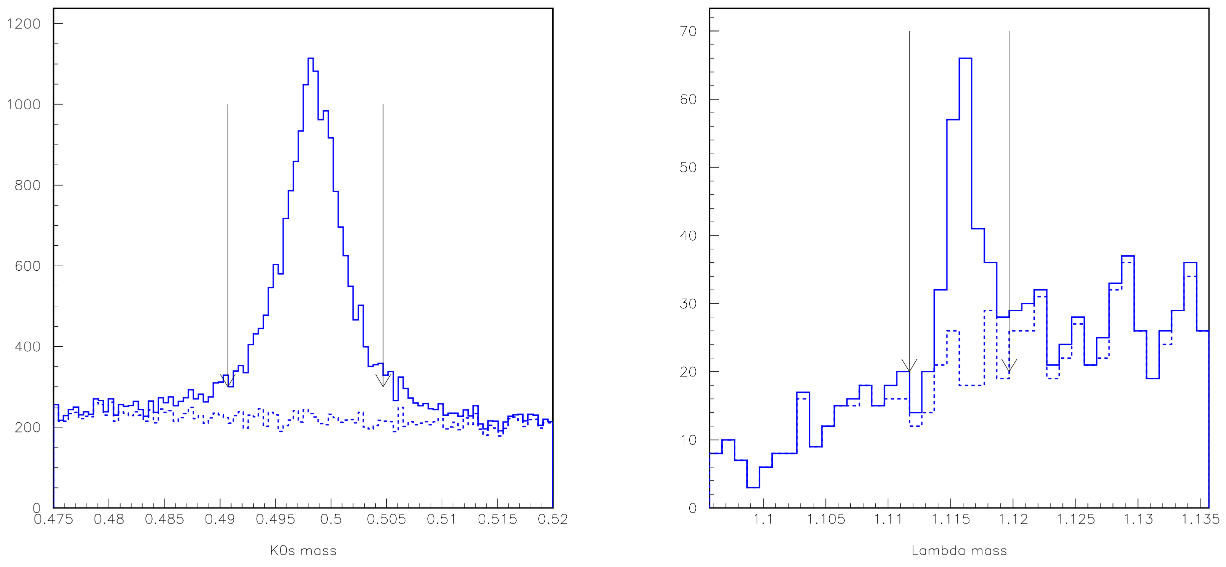


Figure 12: Mass distribution of the selected K_S^0 and Λ candidates. Arrows indicate the mass cut applied. These plots are obtained by using also the tracks used for the CP side. Therefore, the combinatorial background of the candidates used for vertexing will be significantly lower.

4.5 Δz to Δt transformation

A time-dependent analysis must estimate Δt for each event,

$$\Delta t = t_{CP} - t_{TAG} = M_B \left[\frac{z_{CP}}{p_{z,CP}} - \frac{z_{TAG}}{p_{z,TAG}} \right] \tag{71}$$

t_{CP} and t_{TAG} are, however, defined in different frames, therefore the transformation from Δz to Δt is not straightforward. The standard approach in *BABAR* consist in assuming that both B's have the same average boost, $p_{z,CP} = p_{z,TAG} \approx \langle p_z \rangle$, and take it to be the same as for the $\Upsilon(4S)$ frame. This is equivalent to assume that the momentum of the B mesons in the $\Upsilon(4S)$ decay frame is negligible ($\langle p_B^{cms} \rangle = \sqrt{E_{e^+}E_{e^-} - M_B^2} \approx 340$ MeV/c). With these assumptions,

$$\Delta t = \Delta z / \gamma \beta_z c \tag{72}$$

where γ is the boost factor of the $\Upsilon(4S)$ in laboratory frame and β_z its velocity projected on the *BABAR* z axis. In *BABAR*, $\beta_z\gamma \approx 0.56$. Using β_z instead of β we account for the rotation in the (x, z) plane of the beam axis with respect to the *BABAR* z axis (tilt angle about 20 mrad), reducing the full boost by about 1.2 MeV (0.02%). As a result, there is a small boost along the x axis (about 188 MeV) which generates azimuthal dependencies. This has, however, no direct effect when estimating Δz. The Δz provided by the algorithms described above is in the *BABAR* z axis. This approximation, for which the B momentum does not need to be measured, is commonly called *boost approximation*. Compared with the experimental resolution on Δz, the effects on Δt produced by this approximation are small

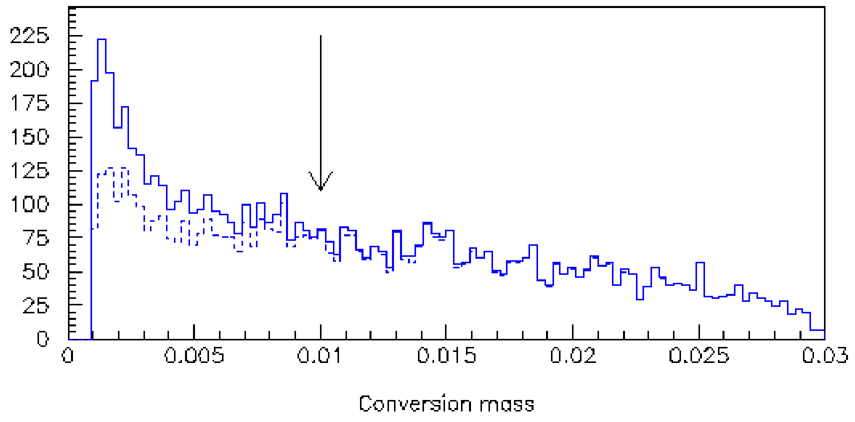


Figure 13: Mass distribution of the γ conversion candidates. The arrow indicates the mass cut applied. As before, this plot is obtained by using also the tracks used for the CP side.

(see below). However, this approach is a potential source of bias and systematic error in Δt [14, 17]. In particular, reference [14] shows analytically that for B lifetime measurements the bias introduced by this approach about 0.4%, which is non negligible for a precision measurement as we seek in *BABAR*. This result is consistent with toy Monte Carlo studies.

The discussion below shows how we can correct partially and even completely to account for the different boosts of the two B's in the event, in the case that we fully reconstruct one of them.

The distance Δz can be written in terms of t_{CP} and t_{TAG} [15] as:

$$\Delta z = \beta_z \gamma \gamma_{CP}^{cms} c(t_{CP} - t_{TAG}) + \gamma \beta_{CP}^{cms} \gamma_{CP}^{cms} \cos\theta_{CP}^{cms} c(t_{CP} + t_{TAG}) \tag{73}$$

where β_z and γ are the same quantities as defined above. γ_{CP}^{cms}, β_{CP}^{cms} and $\cos\theta_{CP}^{cms}$ are, respectively, the boost factor, velocity and angle with respect to the beam direction of the fully reconstructed B in the $\Upsilon(4S)$ frame. The above expression uses the fact that both B's are anticorrelated in the center-of-mass frame.

As (in principle) we don't know $t_{CP} + t_{TAG}$, so we have to neglect it or best put in some average value. Assuming an experiment with no polar angle bias and neglecting the event-by-event variation of $t_{CP} + t_{TAG}$ due to changes in Δt, $\langle \cos\theta_{CP}^{cms} \rangle = 0$, i.e. on average the second term of equation (73) vanishes. If we also neglect the energy release of the $\Upsilon(4S) \to B\overline{B}$ decay, then we have $\gamma_{CP}^{cms} = 1$. These two assumptions lead with the *boost approximation* described above and given by equation (72). This approach can be corrected, on average, to account for the small momentum of the B's,

$$\Delta t = \Delta z / \gamma \beta_z c \gamma_{CP}^{cms} \tag{74}$$

where $\gamma_{CP}^{cms} \approx 1.002$. Equation (74) will be known thereafter as *improved boost approximation.*

The effect of neglecting the per-event variation of $t_{CP} + t_{TAG}$ is an extra contribution to the resolution function [16]. The size of this term can easily be estimated:

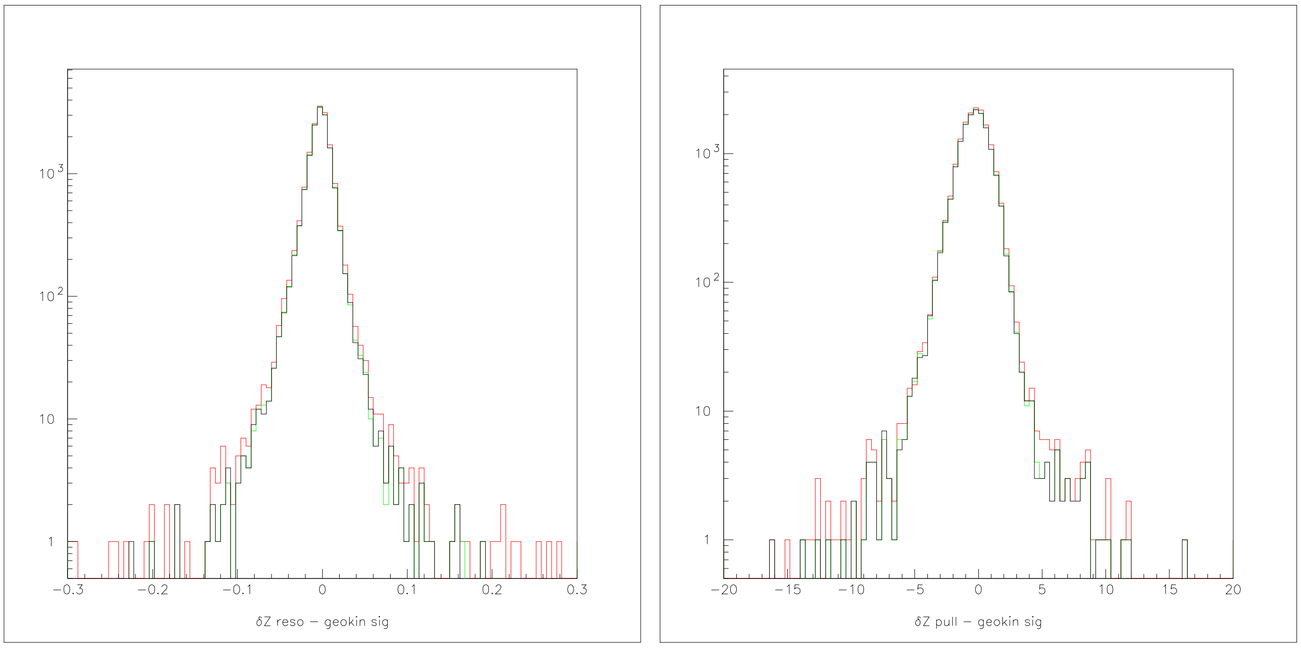


Figure 14: Comparison of Δz residuals (left) and pulls (right) with (black histogram) and without V^0 (red histogram) reconstruction. Green histogram corresponds to the case when only MC truth V^0's are used.

$$(\gamma\beta_{CP}^{cms}\gamma_{CP}^{cms})^2 \langle\cos^2\theta_{CP}^{cms}\rangle\langle(ct_{CP}+ct_{TAG})^2\rangle \approx 0.074^2\langle(ct_{CP}+ct_{TAG})^2\rangle\langle\cos^2\theta_{CP}^{cms}\rangle\ (\mu\text{m}^2) \quad (75)$$

since $\beta_{CP}^{cms} \approx 0.064$ and $\gamma \approx 1.15$. The angular distribution of $\Upsilon(4S) \to B\overline{B}$ is given by $(1-\cos^2\theta)d(\cos\theta)$, which gives $\langle\cos^2\theta\rangle = 1/5$. Similarly, $\sqrt{<(t_{CP}+t_{TAG})^2>} = \sqrt{6}\tau_B$. The RMS contribution is then 35-40 μm, to be compared to the total resolution of about 110 μm. Therefore, in terms of resolution the effect of neglecting the second term of equation (73) is small (about 5% in quadrature). If the resolution is extracted from data themselves from B lifetime and mixing fits, this effect is included automatically in any time-dependent analysis.

When one of the B's in the event is fully reconstructed we know with good precision $\cos\theta_{CP}^{cms}$. The problem then is to get an estimate of $t_{CP}+t_{TAG}$. We could take $\langle t_{CP}+t_{TAG}\rangle = 2\tau_B$. This approach, however, does not account for the variation of $\langle t_{CP}+t_{TAG}\rangle$ with Δt. This effect can be taken into account by averaging over the Δt range,

$$\langle t_{CP}+t_{TAG}\rangle\,|_{\Delta t} = \tau_B + \mid\Delta t\mid \quad (76)$$

Thus Δt can be extracted combining equations (73) and (76),

$$\Delta z = \beta_z\gamma\gamma_{CP}^{cms}c\Delta t + \gamma\frac{p_{z,CP}^{cms}}{M_B}c(\tau_B + \mid\Delta t\mid) \quad (77)$$

following a two steps method: firstly, we resolve (77) for Δt assuming that the sign of Δt is the same as that of Δz; secondly, we check the consistency of signs, and in case that the signs are opposite the equation is resolved again assuming now opposite signs for Δz and Δt. This approach will be known thereafter as *average τ_B approximation* and it corrects for any possible polar angle bias of the experiment, as well as it minimizes the additional contribution to the RMS from the per-event variation of $t_{CP} + t_{TAG}$, as given by equation (75). Similarly to the boost approximation, when estimating the statistical error on Δt we only consider the contribution coming from Δz, which is largely dominating the total error.

The `VtxTagBtaSelFit` algorithm brings a second way to correct by the different boost of the B mesons in center-of-mass frame. As described in section 4.2, this algorithm performs a simultaneous fit maximizing the available information in the $\Upsilon(4S) \to B\overline{B}$ decay. In the fit, the position of the vertex tag, as well as the decay lengths of the CP and TAG sides (with respect to the $\Upsilon(4S)$ decay point, L_z^{CP} and L_z^{TAG} respectively) are free parameters. The beam-spot constraints applied in the transverse plane, especially in the y component, are able to put constraints on the z component, therefore some information is obtained by fitting L_z^{CP} and L_z^{TAG}. As said before, these parameters are basically +100% correlated. In fact, Δz is calculated using directly those parameters and their covariance matrix. The knowledge of L_z^{CP} and L_z^{TAG} provide a way to estimate directly the second term of equation (73),

$$(\Delta z)_{correction} = \gamma\beta_{CP}^{cms}\gamma_{CP}^{cms}\cos\theta_{CP}^{cms}c(t_{CP}+t_{TAG}) = \gamma\gamma_{CP}^{cms}p_{z,CP}^{cms}\left[\frac{\mid L_z^{CP}\mid}{\mid p_{z,CP}\mid} + \frac{\mid L_z^{TAG}\mid}{\mid p_{z,TAG}\mid}\right] \quad (78)$$

where $p_{z,CP}$ and $p_{z,TAG}$ are the z components of the momenta of the CP and TAG B mesons, both reconstructed. Therefore, equation (73) reads here

$$\Delta z = \beta_z\gamma\gamma_{CP}^{cms}c\Delta t + \gamma\gamma_{CP}^{cms}p_{z,CP}^{cms}\left[\frac{\mid L_z^{CP}\mid}{\mid p_{z,CP}\mid} + \frac{\mid L_z^{TAG}\mid}{\mid p_{z,TAG}\mid}\right] \quad (79)$$

from which Δt can be extracted. This approach will be known thereafter as *exact Δt calculation*. The errors on L_z^{CP} and L_z^{TAG} are large and fully correlated, but they rescale according with $p_{z,CP}^{cms}$. Therefore, as it will be shown later, it turns out that the error contribution to Δt coming from this term is similar to that due to the pure Δz term. The estimation of the total error on Δt accounts only for the contributions from Δz, L_z^{CP} and L_z^{TAG}. All the other contributions can safely be neglected.

As already mentioned in section 4.2, the energy spread of the beams affects the Δt estimation (via the boost parameters β and γ). Provided that a $\cos\theta^{cms}$ correction is used when estimating Δt, in the generator level study of reference [17] it is shown that this is a minor source of troubles. On the other hand, the accuracy on $\beta_z\gamma$ using 2-prong events [18] has been evaluated to be better than 0.3% [19, 20].

The calculation of Δt from Δz for all the four approaches has been implemented in the class `BtaDeltaTConverter` in `BetaTools`. The boost quantities (β_z and γ) used by this class are those computed in rolling calibration by using 2-prong events [18].

The expected effect of the *average τ_B approximation* and the *exact calculation* methods is to reduce the Δt bias (due to the non-uniform acceptance of the *BABAR* detector). In the case of the *average τ_B approximation* an improvement in the RMS of the resolution (about 4% according with the estimation above) is also expected. The four different approaches for estimating Δt have been tested using 36K $B^- \to D^0(K^-\pi^+)\pi^-$ events. Figure 15 shows the

Δt residuals, errors and pulls for the four different Δt calculations: (a) boost approximation, (b) improved boost approximation, (c) average τ_B approximation and (d) exact calculation. Table 2 shows the mean values and widths after fitting the residuals and pulls of figure 15 to two Gaussians. In order to account for outliers, a third Gaussian has been included with width fixed to 8 ps for residuals and 8 for pull. The fraction of this third Gaussian is left free. From the figure and table it can be seen that there is an almost perfect overlap between (a) and (b), as expected. The only change which should appear is a small (0.2%) overall shift in the distributions, which is consistent with the numbers shown in the table (although the statistics is too low to be sensitive to this effect). The RMS for the residual and pull distributions for (c) is improved with respect to (a) by about 5%, consistent (within errors) with the estimation before. The bias reduction is difficult to be estimated with the present statistics, but it is at the level of 1-2%. As expected, the RMS for (d) is, compared to (a), slightly worse, but the difference is not dramatic ($\sim$ 5%). This is due to the additional error introduced by the correction term in the exact calculation method. There is also some indication of bias reduction, at the same level as in (c).

	f_{core}	μ_1	σ_1	μ_2	σ_2	RMS	μ
Δt Residual (ps)							
boost approximation	0.681 ± 0.023	-0.124 ± 0.007	0.571 ± 0.012	-0.329 ± 0.019	1.28 ± 0.04	0.80	-0.189
improved boost approx.	0.689 ± 0.022	-0.125 ± 0.007	0.573 ± 0.012	-0.327 ± 0.017	1.30 ± 0.04	0.80	-0.188
average τ_B approximation	0.660 ± 0.022	-0.117 ± 0.007	0.524 ± 0.011	-0.329 ± 0.016	1.23 ± 0.04	0.76	-0.188
exact calculation	0.623 ± 0.022	-0.115 ± 0.007	0.548 ± 0.012	-0.306 ± 0.021	1.31 ± 0.03	0.84	-0.187
Δt Pull							
boost approximation	0.60 ± 0.07	-0.191 ± 0.023	1.00 ± 0.04	-0.48 ± 0.05	1.58 ± 0.06	1.23	-0.31
improved boost approx	0.60 ± 0.08	-0.187 ± 0.023	1.00 ± 0.04	-0.49 ± 0.05	1.58 ± 0.06	1.23	-0.31
average τ_B approximation	0.72 ± 0.05	-0.202 ± 0.020	0.99 ± 0.03	-0.59 ± 0.07	1.63 ± 0.08	1.17	-0.31
exact calculation	0.66 ± 0.06	-0.157 ± 0.021	0.95 ± 0.03	-0.50 ± 0.03	1.49 ± 0.05	1.13	-0.27

Table 2: Comparison of Δt resolution function parameters for residuals and pull after fitting to two Gaussians, for the four different Δt approximations. In order to account for outliers, a third Gaussian has been included with width fixed to 8 ps for residuals and 8 for pull. The fraction of this third Gaussian is left free (values always $\sim$ 0.015 for residuals and $\sim$ 0.012 for pulls).

The previous comparison of the different Δz to Δt conversions has been redone with more recent Monte Carlo samples (based on 8.8.0 series) -the previous 36K $B^- \to D^0(K^-\pi^+)\pi^-$ events were produced with releases 8.6.3a to 8.6.5a-. This new check also includes some recent vertexing bug fixes (not related to the Δt conversion) not yet available at the time of the previous tests. We have used now $B^- \to J/\psi K^-$ events. Figure 16 shows the Δt residuals and pulls for both approximations and their two Gaussian fits with outlier component, and table 3 summarizes the results of these fits. The net gain in RMS for both, residual and pull distributions, is cleary visible, and it is at the level of 4%, consistent with the results above. And there is also a clear reduction of bias, $\sim$ 4% in residual and $\sim$ 2% in pull. In the results shown in table 2 there was not sizeable gain in bias.

It should be noted that the differences between the average τ_B and the boost approximations are, on average, small. However, it is not so much on an event-by-event basis. To ilustrate this effect, figure 17 shows the spread of the Δt and $\sigma_{\Delta t}$ differences between the average τ_B and boost approximations for charmonium Monte Carlo events. From the fit to a

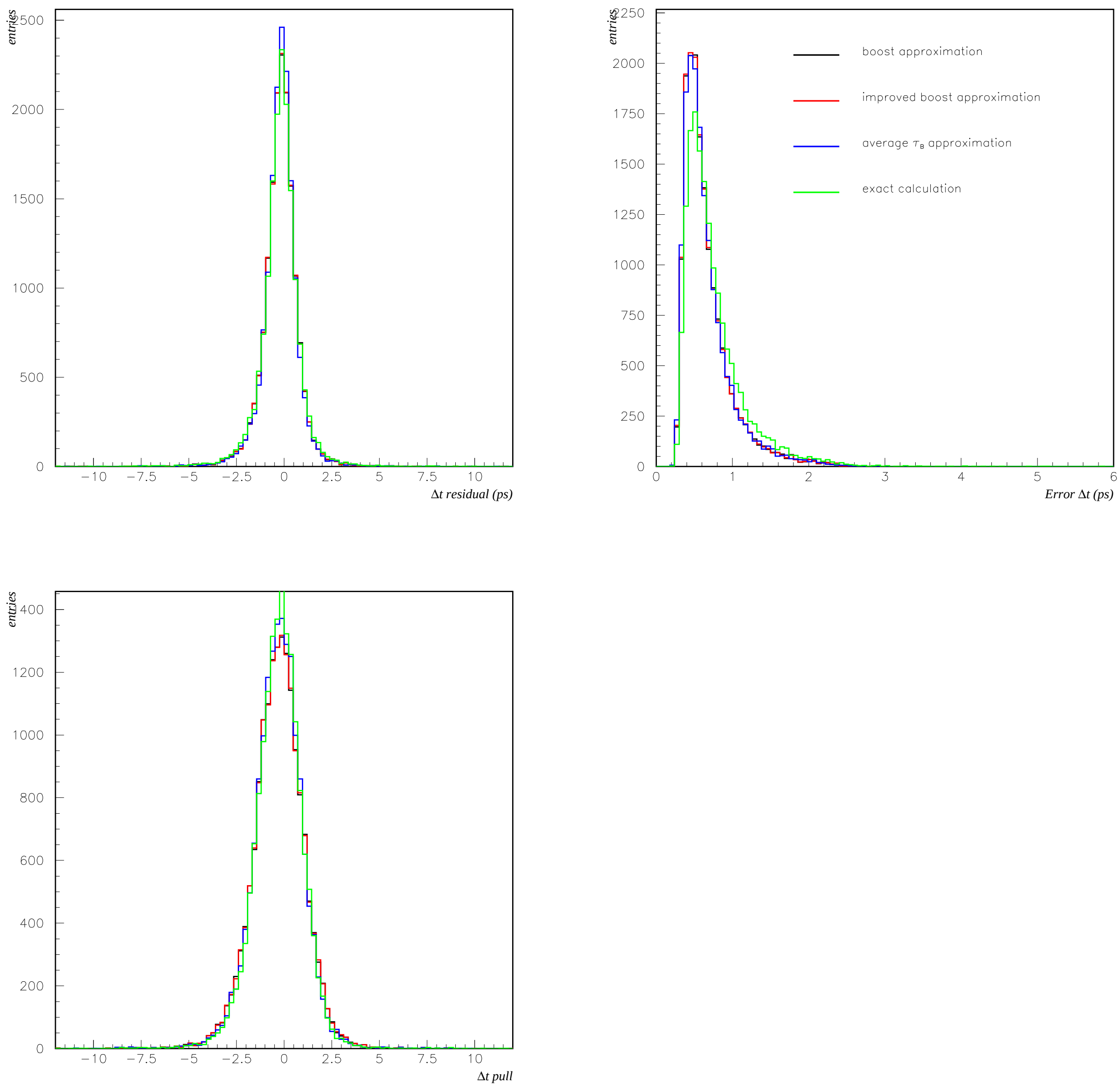


Figure 15: Comparison of Δt residuals (top,left), errors (top,right) and pull (bottom) for the three different Δt calculations: *boost approximation*, *improved boost approximation*, *average τ_B approximation* and *exact calculation*.

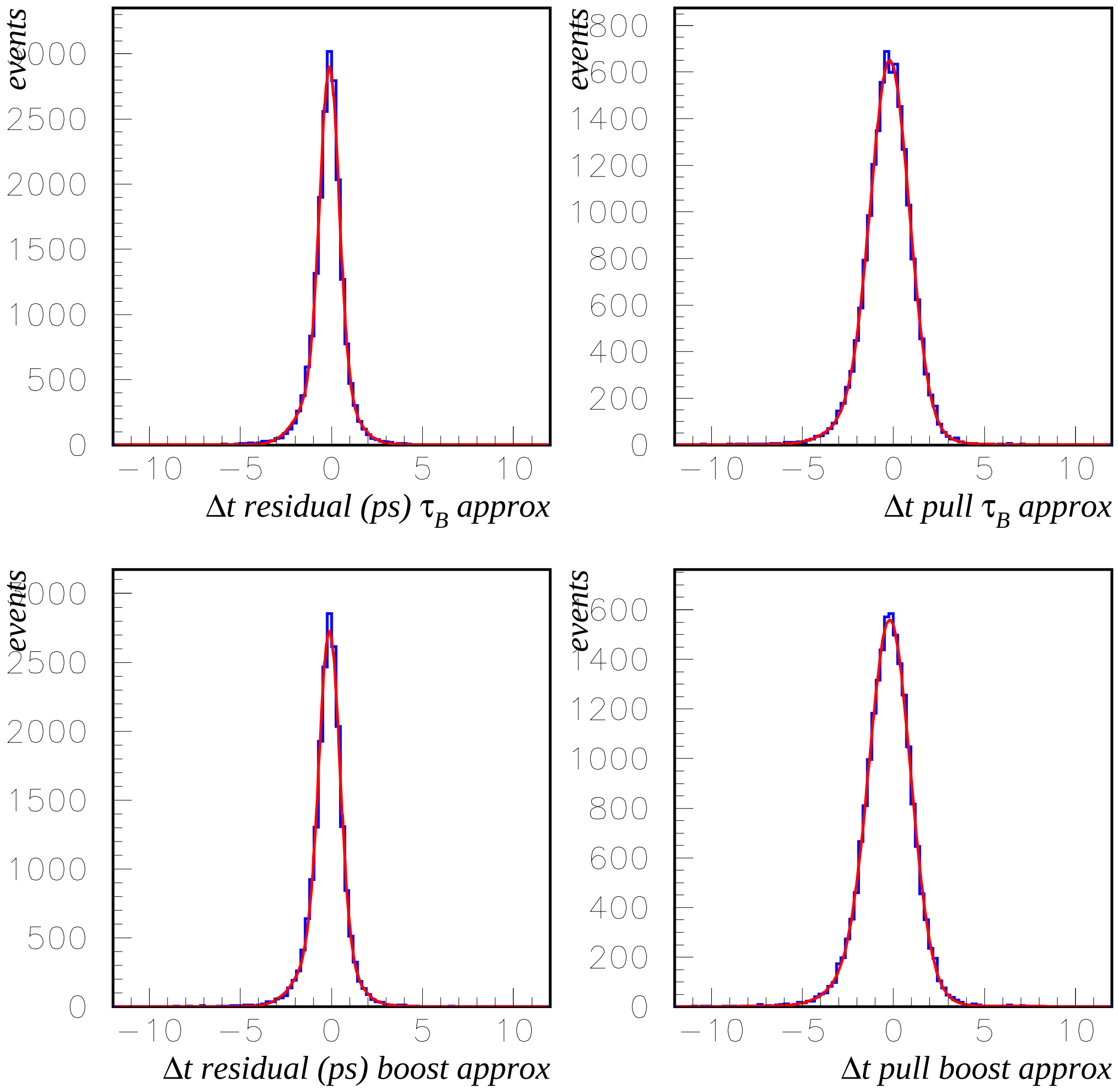


Figure 16: Δt residuals (left) and pulls (right) for the average τ_B (top) and boost (bottom) approximations. Superimposed are the two Gaussian fits with a third Gaussian describing the outlier component.

	f_{core}	μ_1	σ_1	μ_2	σ_2	RMS	μ
Δt Residual (ps)							
average τ_B approximation	0.648 ± 0.023	-0.101 ± 0.007	0.514 ± 0.011	-0.286 ± 0.020	1.21 ± 0.04	0.831	-0.166
boost approximation	0.664 ± 0.024	-0.104 ± 0.007	0.557 ± 0.012	-0.310 ± 0.022	1.26 ± 0.04	0.860	-0.173
Δt Pull							
average τ_B approximation	0.82 ± 0.04	-0.178 ± 0.015	1.046 ± 0.019	-0.77 ± 0.08	1.81 ± 0.09	1.226	-0.289
boost approximation	0.81 ± 0.04	-0.178 ± 0.017	1.111 ± 0.022	-0.81 ± 0.11	1.84 ± 0.09	1.279	-0.296

Table 3: Comparison of Δt resolution function parameters for residuals and pull after fitting to two Gaussians, for the average τ_B and boost Δt approximations. In order to account for outliers, a third Gaussian has been included with width fixed to 8 ps for residuals and 8 for pull. The fraction of this third Gaussian is left free (values always ~ 0.013 for residuals and ~ 0.011 for pulls).

single Gaussian, we obtain a bias of 0.004 ps with RMS 0.19 ps. The RMS of the per-event errors is determined similarly to be 0.04 ps.

4.5.1 An additional note

In the subsection we discuss some subtleties of the use of Eq. 73 and Eq. 77 for converting Δz to Δt.

To derive Eq. 73, we first assume the boost direction of $\Upsilon(4S)$ is along the z-axis. The B decays with a polar angle $\theta^*_{\rm CP}$ with respect to the boost direction. The decay distance along the boost direction can be derived:

$$\begin{aligned} p_z &= \gamma p^*_z + \gamma\beta E^* \\ c\gamma_B\beta_B &= p_z/m_B = \frac{\gamma p^*_z + \gamma\beta E^*}{m_B} \end{aligned}$$

and

$$\begin{aligned} z_{\rm CP} &= c\gamma_{\rm CP}\beta_{\rm CP}t_{\rm CP} = c(\gamma\gamma^*_{\rm CP}\beta^*_{\rm CP}\cos\theta^*_{\rm CP} + \gamma\beta\gamma^*_{\rm CP})t_{\rm CP} \\ z_{\rm TAG} &= c\gamma_{\rm TAG}\beta_{\rm TAG}t_{\rm TAG} = c(-\gamma\gamma^*_{\rm CP}\beta^*_{\rm CP}\cos\theta^*_{\rm CP} + \gamma\beta\gamma^*_{\rm CP})t_{\rm TAG}\,, \end{aligned}$$

where the asterisk represents the center-of-mass system, and we have use the fact that at the center-of-mass system, $\beta^*_{\rm CP} = \beta^*_{\rm TAG}$. Therefore, the distance along the boost axis is

$$\Delta L = c\gamma\beta\gamma^*_{\rm CP}(t_{\rm CP} - t_{\rm TAG}) + c\gamma\gamma^*_{\rm CP}\beta^*_{\rm CP}\cos\theta^*_{\rm CP}(t_{\rm CP} + t_{\rm TAG})\,. \tag{80}$$

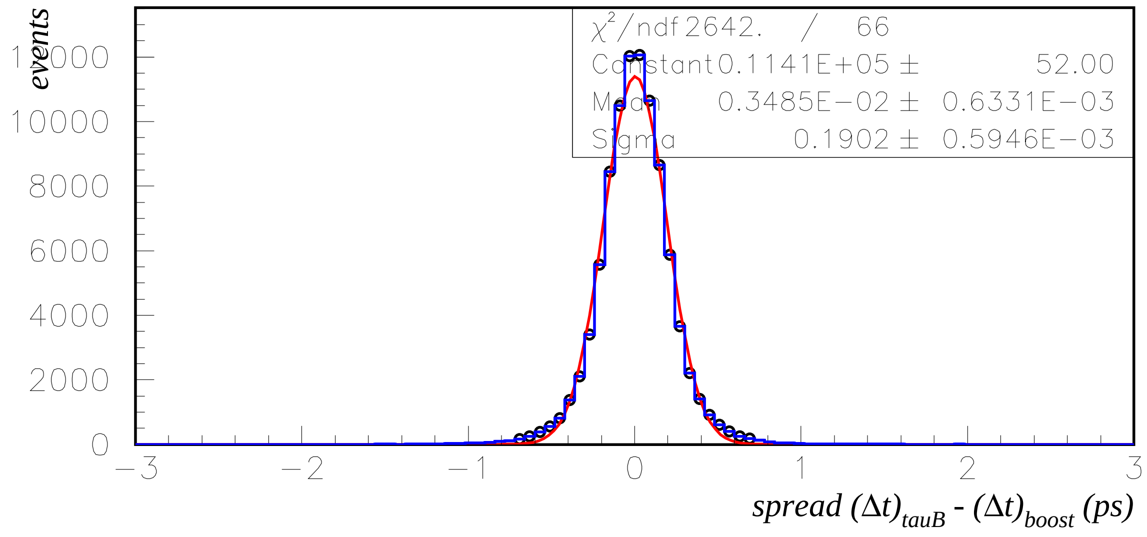
Since we have no way to know $t_{\rm CP} + t_{\rm TAG}$, we integrate out $t_{\rm CP} + t_{\rm TAG}$ and get the expectation value

$$\langle t_{\rm CP} + t_{\rm TAG}\rangle\,|_{\Delta t} = \tau_B + |\Delta t|\,. \tag{81}$$

Combining Eq. 80 and 81, and naively project the distance to the real z-axis (considering the boost angle $\theta_{\rm boost} \simeq 20\,{\rm mrad}$), we get

$$\Delta z = c\gamma\beta_z\gamma^*_{\rm CP}\Delta t + c\gamma\gamma^*_{\rm CP}\beta^*_{\rm CP}\cos\theta^*_{\rm CP}(\tau_B + |\Delta t|)\,\cos\theta_{\rm boost}\,. \tag{82}$$

Comparing Eq. 82 with Eq. 77, we found that the latter does not have the factor $\cos\theta_{\rm boost}$ in its second term. Moreover, the projection in Eq.82 is WRONG. Because Eq.80 is already a



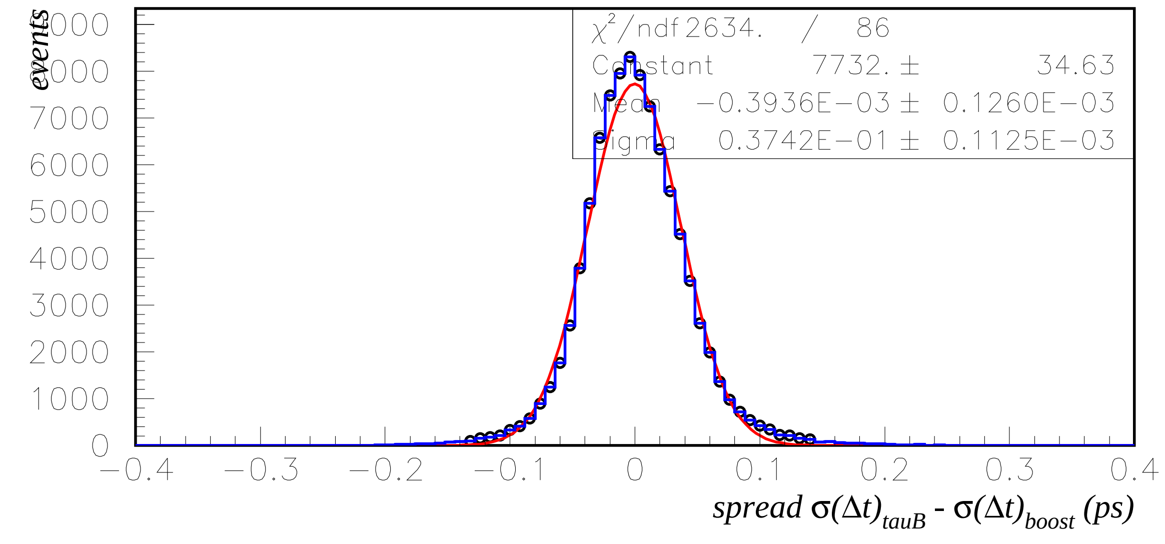


Figure 17: Spread of the Δt and $\sigma(\Delta t)$ differences between the average τ_B and boost approximations for charmonium Monte Carlo.

projection, we cannot project it to another axis by multiplying $\cos\theta$. The actual projection is much more complicated and involves ϕ angle of B decay. Obviously the better choice is to measure ΔL along the boost axis. Fortunately this effect is much smaller than the approximation we take using Eq. 81. See Fig. 18.

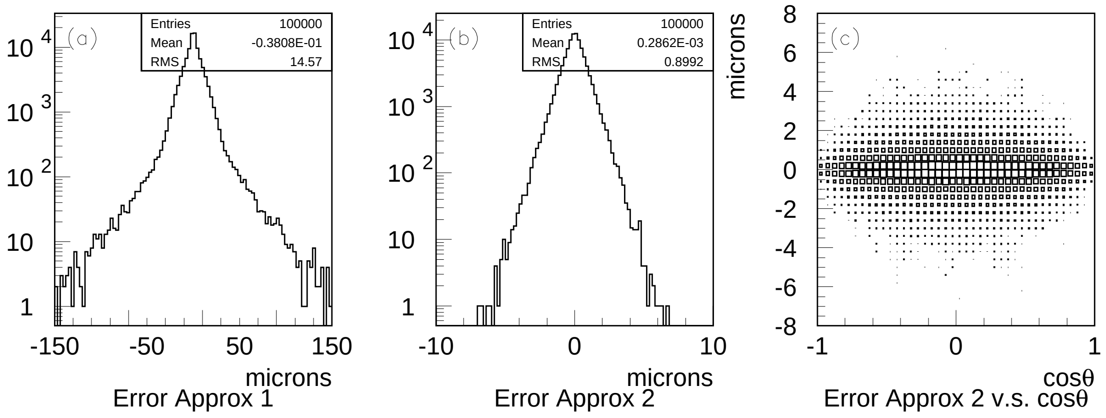


Figure 18: (a) Errors on Δz using true Δt to calculate Δz with Eq. 82. (b) Errors on Δz using true t_{CP} and t_{TAG} with Eq. 80 multiplied by $\cos\theta_{\mathrm{boost}}$. (c) Scatter plot of Δz errors (for (b)) versus $\cos\theta^*_{\mathrm{CP}}$.

To address the issue that whether Δt is biased using Eq. 72 and Eq. 77, we run 1 million events using the event generator. We comapre the difference between two Δt's and $\cos\theta^*_{\mathrm{CP}}$ distributions for all events and those survive “acceptance cut”, i.e., at least one B's final daughters (charged and neutral) are all within polar angle $17° - 150°$ in the lab frame. 759440 events survive the cut.

The results are shown in Fig.19. The mean and rms of $\delta\Delta t$ for all events are 0.000201 ps and 0.198876 ps, and for selected events are 0.000109 ps and 0.198788 ps. Both means are consistent with zero. The statistics for $\cos\theta^*_{\mathrm{CP}}$ are $mean = -0.000565$, $rms = 0.447005$ for all events and $mean = -0.000377$ and $rms = 0.446766$. Again, they are consistent with no bias. I fit the ratios of these two sets of histograms to a linear function. The slopes are consistent with one.

The conclusions are: (1) there are no biases between Δt using Eq. 72 and Eq. 77; (2) there is no evidence that the acceptance can bias Δt; (3) Eq. 77 is not exactly right, but it does not change the conclusion.

4.6 Common interface

The access to the vertex tag algorithms can be done, either using their own interface, either using an special class, `VtxTagMaker`, available in `VertexingTools`. The definition of a common interface would be a more suitable solution but the current differences in design, implementation and input/output for both algorithms, makes it somewhat difficult. It is

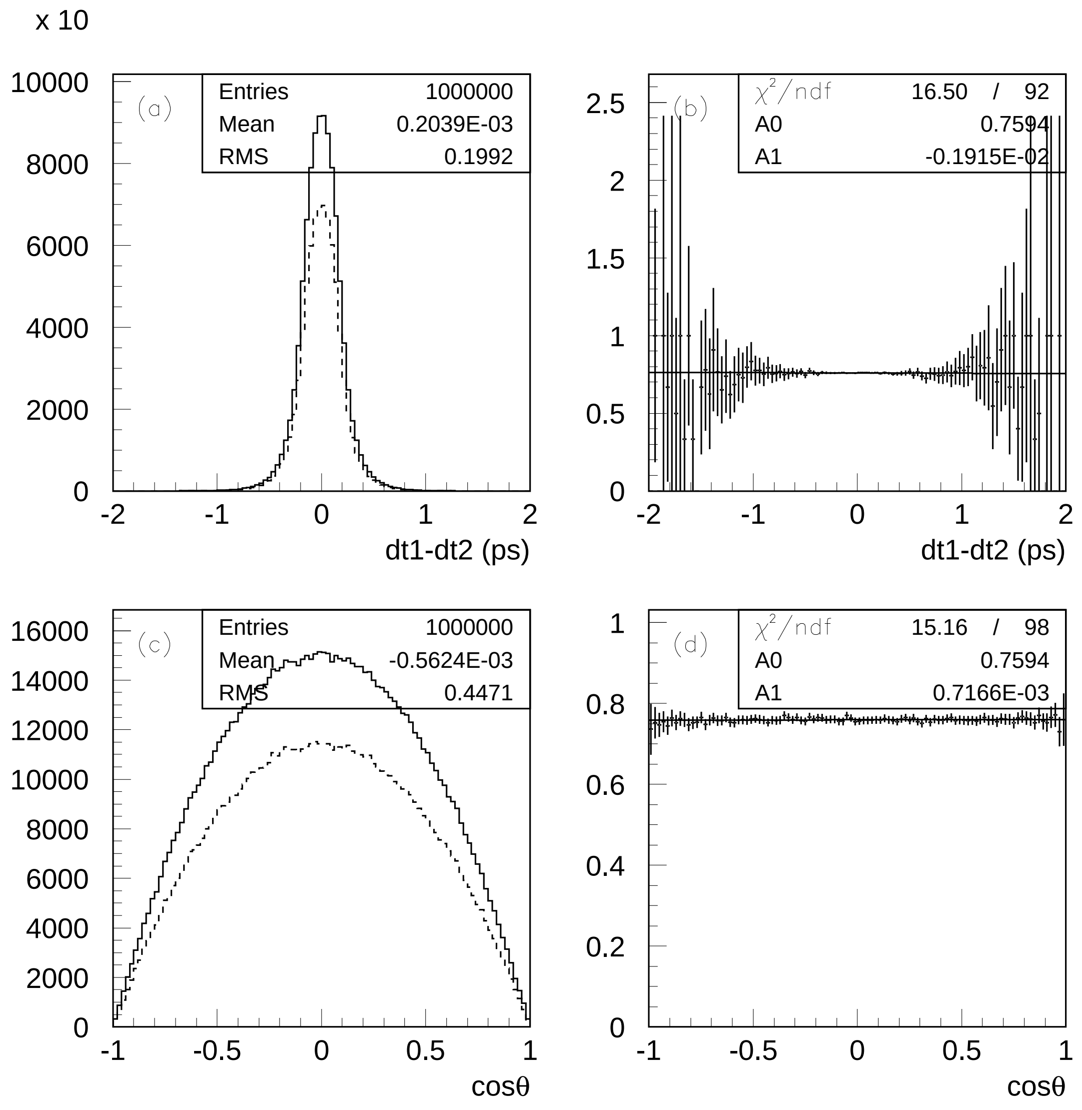


Figure 19: (a) Δt Difference between using Eq. 72 and Eq. 77. Solid histogram is for all events. Dashed one is for events that pass the acceptance cut. (b) Ratio of the two histograms in (a) fitted to a linear function. (c) $\cos\theta^*_{\rm CP}$ distribution for all events and passed events. (d) Ratio of the two histograms in (c). Histogram statistics are a little different from those mentioned in text because of the binning effect.

recommended to use this class rather than the specific interfaces [6, 9]. The advantages of using it are the following:

- the class manages properly the input for tag vertexing, providing a common input for all algorithms;
- one can switch easily from one to another algorithm;
- defines the default configuration of the algorithms according to our best understanding of them;
- it defines common modifiers and accessors for all the algorithms;
- gives access to all the possible Δt approaches;
- gives access to Δz and Δt blinding.

The list of candidates for vertexing (charged tracks and $V0$'s) as well as the list of candidates to veto are provided at constructor level:

```
VtxTagMaker(const BtaCandidate* theB,
            HepAList<BtaCandidate>* listOfTracks,
            HepAList<BtaCandidate>* listOfV0s,
            HepAList<BtaCandidate>* listOfVetos,
            bool doUseV0s=true, bool doUseVeto=true,
            HepString blindString="");
```

The build of the final list for vertexing (direct input for the algorithms) is the following:

- if *listOfV0s* is provided and *doUseV0s* is true, then the list of V^0's is made exclusive (overlapping V^0's are removed), using as criteria the χ^2 probability of the vertex fit;
- candidates appearing in the *listOfTracks* and not overlapping with any of the above V^0 candidates are then added to the list. In this way, V^0's will be used for vertexing instead of their daughters;
- when the list *listOfVetos* is provided and *doUseVeto* is true, candidates in the resulting list which overlap with any of the candidates in *listOfVetos* are then removed;
- finally, candidates which have a common track with the fully reconstructed B (*theB*), are excluded from the list.

Default configuration of the algorithms is performed at constructor level, and it is the following:

- the default algorithm is `VtxTagBtaSelFit`;
- `VtxTagBtaSelFit` defaults:

  - make use of the beam constraints (“pseudo-track” and beam-spot constraints), without beam energy smearing. With this constraint the beam spot used is the one provided by `beamSpot()`;
  - the stopping criteria requires that there are no tracks contributing more than 6 to the total χ^2 or than only 2 tracks remain;
  - a “good” Δz (`goodDeltaZ`, see below) is considered if $\mid \Delta z \mid < 0.3$ cm, $\sigma_{\Delta z} < 0.04$ cm and the global χ^2 of the fit is $< -0.1\%$ (i.e. no χ^2 cut);

- `FvtClusterer` defaults:

  - make use of the beam spot constraint. The beam spot used is the one provided by `beamSpotBFlight()`;
  - the χ^2 probability cut for track removal is 0.001;
  - a “good” Δz (`goodDeltaZ`, see below) is considered if $\mid \Delta z \mid < 0.3$ cm and $\sigma_{\Delta z} <$ 0.04 cm;

The list of modifiers and accessors provided by the interface, with defaults for the Δt accessors, are listed in tables 4 and 5, respectively. Names are self-explanatory.

It should be stressed the fact that the default configuration is the one which has been found optimal when the reconstructed side has been fully reconstructed. In the case of partial reconstruction these defaults have to be changed. The required changes in order do not screw-up results are the following:

- disable the beam constraints (“pseudo-track”) via `setBeamConstraints(false)` modifier. The beam spot constraint will be still enabled after the above command;

- don’t use the average τ_B approximation since it requires the momentum of the reconstructed B. Use the standard boost approximation `deltaT(VtxTagMaker::NoCorrection)` or the improved boost approximation `deltaT(VtxTagMaker::BoostCorrection)` instead.

All the other items of the configuration are common to the case of fully reconstructed events.

Modifiers	Default	Other values/comments
	Common	
`void setAlgorithm(algType)`	`BtaSelFit`	`BtaSelFit ,Clusterer`
`void setBeamSpotConstraint(bool)`	true	
`void setDeltaZCut(double)`	0.3 cm	Only used for `goodDeltaZ` definition
`void setDeltaZErrCut(double)`	0.04 cm	Only used for `goodDeltaZ` definition
	`VtxTagBtaSelFit`	
`void setBeamConstraints(bool)`	true	
`void setBeamEnergySmearing(bool)`	false	
`void chi2JumpCut(double)`	6.	
`void setMinTrk(int)`	0	
`void chi2ProbGlobalCut(double)`	-0.001	Only used for `goodDeltaZ` definition
	`FvtClusterer`	
`void chi2ProbJumpCut(double)`	0.001	

Table 4: List of modifiers available with the `VtxTagMaker` interface.

Accessors	Remarks
Common	
`HepAList<BtaCandidate>* usedList()`	List of candidates used in the vertex
`HepAList<BtaCandidate>* usedListRefitted()`	List of refitted candidates used in the vertex
`HepAList<BtaCandidate>* initialList()`	Input list of candidates used for vertexing
`BtaAbsVertex* vertex()`	vertex tag
`BtaCandidate* getInitialCP()`	initial CP candidate
`BtaCandidate* getRefittedCP()`	refitted CP candidate
`HepMatrix getCPTagVertexCov()`	CP-Tag vertex correlation matrix
`HepMatrix getTagVertexBSCov()`	Tag vertex-beam spot correlation matrix
`HepMatrix getCPVertexBSCov()`	CP vertex-beam spot correlation matrix
`bool defined()`	is the vertex defined?
`BtaAbsVertex::VtxStatus status()`	status of the vertex
`int nDof()`	*ndof* of the fit
`double chiSquared()`	χ^2 of the fit
`double prob()`	χ^2 probability of the fit
`HepPoint position()`	vertex position
`HepSymMatrix positionCov()`	covariance matrix of vertex tag position
`BbrDoubleErr deltaZ()`	Δz and it covariance
`double deltaZBlind(double)`	blind Δz
`bool goodDeltaZ()`	is a "good" Δz?
`BbrDoubleErr deltaT(deltaTtype dTtype=TauBCorrection)`	Δt and its covariance
	`dTtype` can be: `NoCorrection,BoostCorrection`
	`TauBCorrection,ExactCorrection`
`double deltaTBlind(double,deltaTtype dTtype=TauBCorrection)`	blind Δt
`int nK0s()`	number of K^0_S in vertex tag
`int nLambda()`	number of Λ in vertex tag
`int nSingleTrack()`	number of single tracks in vertex tag
`int nComp()`	number of composite objects in vertex tag
`int nUsedTrk()`	total number of single tracks in vertex tag
`HepLorentzVector p4CP()`	four-momentum of the B_{CP} candidate
`double getChi2Contribution(BtaCandidate* c)`	χ^2 contribution of a given candidate
`VtxTagBtaSelFit`	
`double LzCP()`	L_z^{CP}
`double LzTAG()`	L_z^{TAG}
`HepSymMatrix LzCov()`	Covariance matrix of L_z^{CP} and L_z^{TAG}
`double pHighest()`	momentum of most energetic object in the vertex
`HepLorentzVector p4TAG()`	four-momentum of B_{TAG} candidate
`double pTAG()`	momentum of B_{TAG} candidate
`double phiTAG()`	azimuthal angle of B_{TAG} candidate
`double thetaTAG()`	polar angle of B_{TAG} candidate
`double pstar()`	momentum of B_{TAG} in center-of-mass frame

Table 5: List of accessors available with the `VtxTagMaker` interface.

`BTaggingTools` ntuples [5] make use of this new class to access the vertex tag information. The default configuration is the same as defined in `VtxTagMaker` with only one exception: the definition of "good" Δz only considers the cut on Δz, needed for blinding. Al the other cuts are not applied. V^0's and vetoes are used with the configuration defined in the following section. The tcl piece with the current (*analysis-7* release) vertex tag configuration is given below (extracted from `BTaggingTools/BTaggingMicroSequence.tcl`):

```
# Configuration of tag vertex reconstruction
doVtxTag                set f
doFvtClusterer          set f
vtxTrackList            set GoodVtxTagTracks
vtxV0List               set V0sVtxTag
vtxVetoList             set gammaConversionVtxTag
useVtxV0                set t
```

```
useVtxVeto               set t
useBeamSpotConstraint    set t
useDeltaTCorrection      set t
useDeltaTTauBCorrection  set t
# Configuration of VtxTagBtaSelFit
cutVtxChi                set 6.
useDeltaTExactCorrection set f
cutVtxChiProbGlobal      set 0.
cutVtxNtrk               set 0
doVtxTrackRemoval        set true
useBeamConstraints       set true
useBeamEnergySmearing    set false
# Configuration of FvtClusterer
cutFvtChiProb            set 0.001
```

4.7 Comparison among algorithms

Let us first recall the difference between the two algorithms:

1. dropping/stopping criterion: individual track χ^2 (`VtxBtaSelFit`) or global vertex χ^2 (`FvtClusterer`)
2. kinematics information used in `VtxBtaSelFit` not in `FvtClusterer`

Both algorithms use a beam spot constraint. The second point is only possible in the case of fully exclusive B reconstruction, so one has to distinguish performances depending on the channel under study.

4.7.1 Fully exclusive B reconstruction

This is the case where `VtxBtaSelFit` uses more information than `FvtClusterer` and should therefore be more precise.

Efficiency and χ^2 The χ^2 probability distributions are different, as shown in figure 20 (left). For `FvtClusterer` it is significantly more peaked at 0. This is natural since one does not fit *geometrically* the genuine B vertex position (due to the charm bias). The fact that the same distribution for the other algorithm is flatter can be an artifact of the threshold on the single track χ^2 contribution. Figure 20 (right) shows the efficiency of both algorithms as a function of the track event multiplicity (`ChargedTracks`). At low multiplicity `VtxTagBtaSelFit` tends to be slightly more efficient, but differences decrease at high multiplicity. The mean efficiency measured in B^0 charmonium signal events is 0.9589 ± 0.0011 and 0.9383 ± 0.0013, respectively. And for $\mathrm{B_0} \rightarrow \mathrm{J\Psi K_S}$ events, 0.9598 ± 0.0018 and 0.9454 ± 0.0021.

Resolution and pulls Figure 21 compares the Δz residuals and pulls for both algorithms, using charmonium events. A similar comparison is shown in figure 22. In this case both distributions are fitted to a double gaussian and the outliers fraction (defined as the relative number of events lying above $3.5 \times \sigma_2$) is also shown. An extra quality cut is

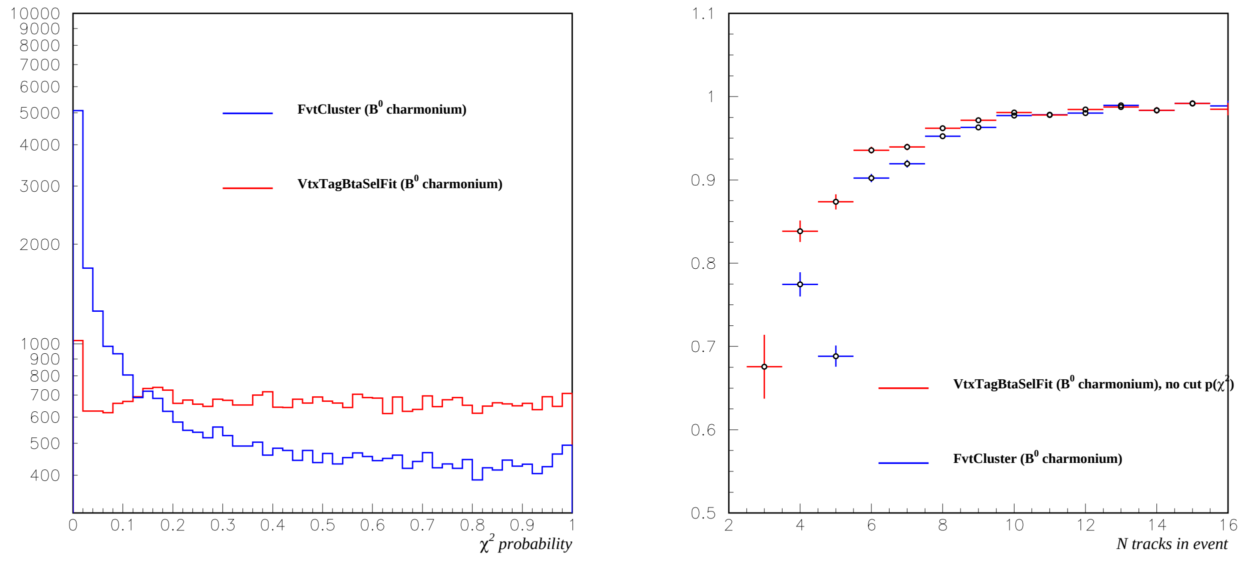


Figure 20: (Left) χ^2 probability distributions and (Right) efficiency as a function of the track event multiplicity for `VtxTagBtaSelFit` and `FvtClusterer` for the B^0 reco signal events.

also used: $|\Delta z| < 3$ mm and $\sigma_{\Delta z} < 400\mu$m. It appears indeed that `VtxBtaSelFit` has a better resolution and lower tails. And it is also clearly less biased.

Correlations between algorithms Figure 23 (left) shows the correlation between the Δz residuals from both algorithms. Figure 23 (right) shows the difference between the Δz measured with each algorithm. The distribution can be fitted to two Gaussians, with $\sim 82\%$ for the core component. The RMS is estimated to be 40 μm, with no bias.

4.7.2 Semi-exclusive B reconstruction

This is the case where all the B meson final state particles are not explicitly reconstructed; however all the other charged tracks in the event come really from the other B, as in $B^0 \to D^*\ell\bar{\nu}$. `VtxBtaSelFit` has therefore to turn off the kinematics information part and both algorithms should give similar results.

Figure 24 shows the Δz residuals and pulls for both algorithms. Both distributions are fitted to a double gaussian and the outliers fraction (defined as the relative number of events lying above $3.5 \times \sigma_2$) is also shown. An extra quality cut is also used: $|\Delta z| < 3$ mm and $\sigma_{\Delta z} < 400\mu$m.

Both algorithms give indeed similar results. It is interesting to compare `VtxBtaSelFit` results between Fig. 22 and 24 which allows to asses the importance of the kinematic constraints

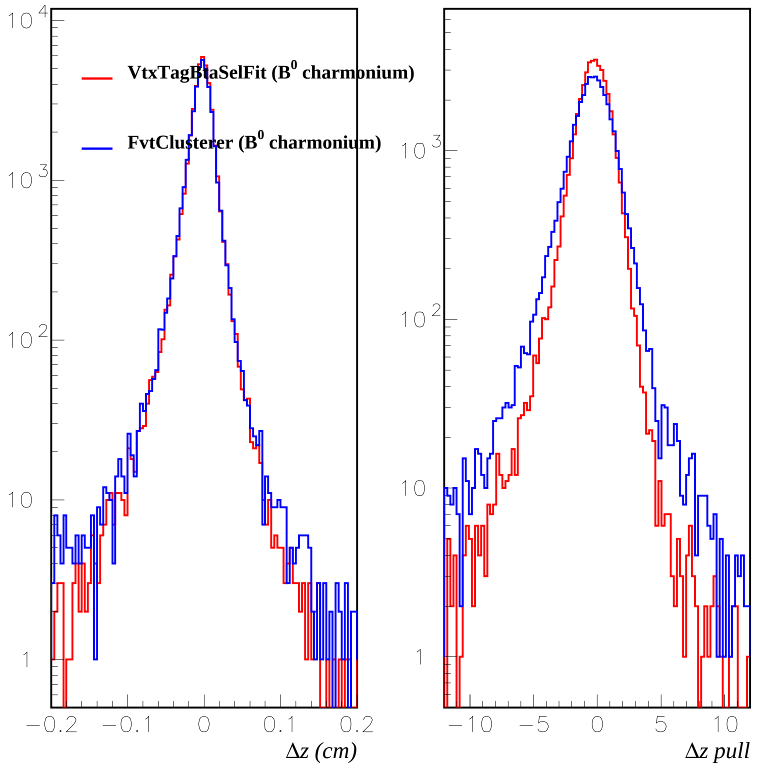


Figure 21: Δz residuals and pulls for `VtxTagBtaSelFit` and `FvtClusterer`, from Bz charmonium signal events.

4.7.3 Partial reconstruction

This is the case where all B meson final states particles are not explicitly reconstructed; however all the other charged tracks in the event do not come really from the other B. As before, `VtxBtaSelFit` has therefore to turn off the kinematic information part and both algorithms should give similar results. As an additional problem , one has to remove in some way all the tracks which are not candidate to come from the B tag vertex. This can be an additional source of biases, and it has to be evaluated for each particular analysis.

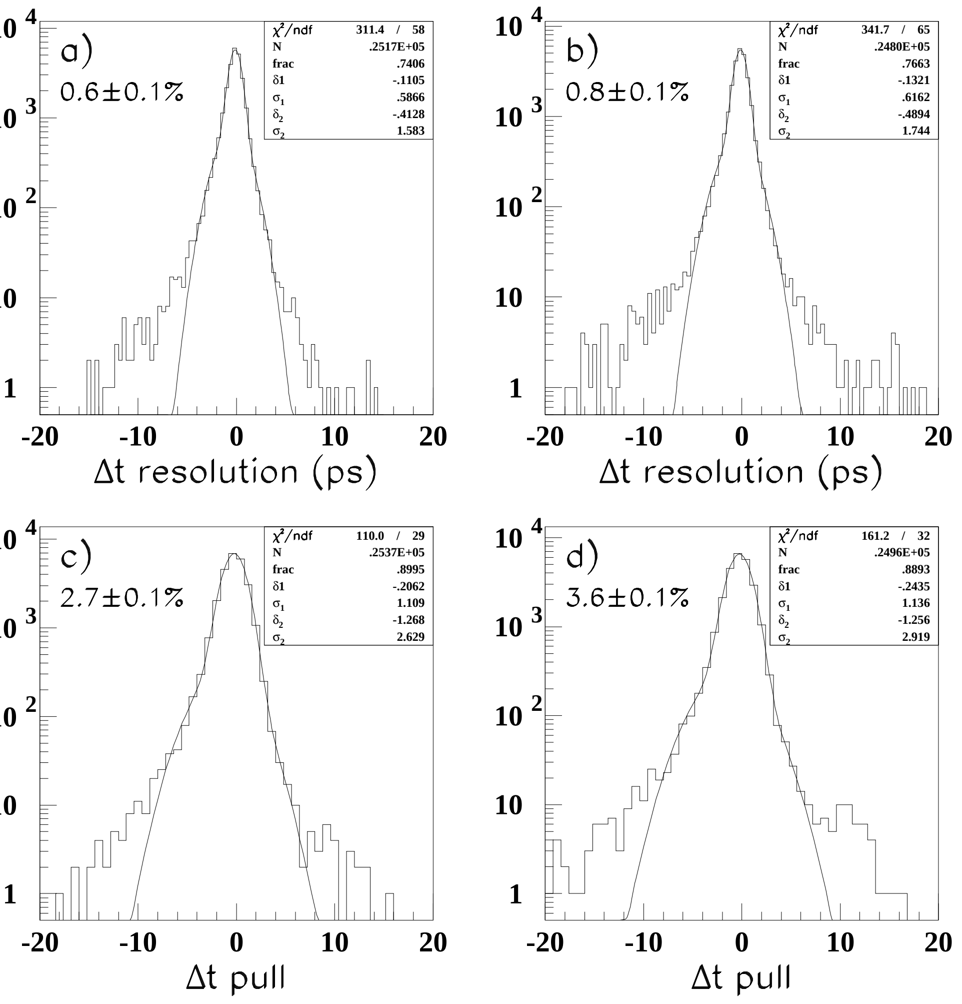


Figure 22: Δz residuals and pulls for `VtxTagBtaSelFit` and `FvtClusterer`, on fully reconstructed B^0 signal events($B^0 \rightarrow J/\psi K^0_s$). The box shows the parameters to a double gaussian fit. The fraction of outliers is also shown: a)-c) `VtxBtaSelFit` , b)-d) `FvtClusterer`

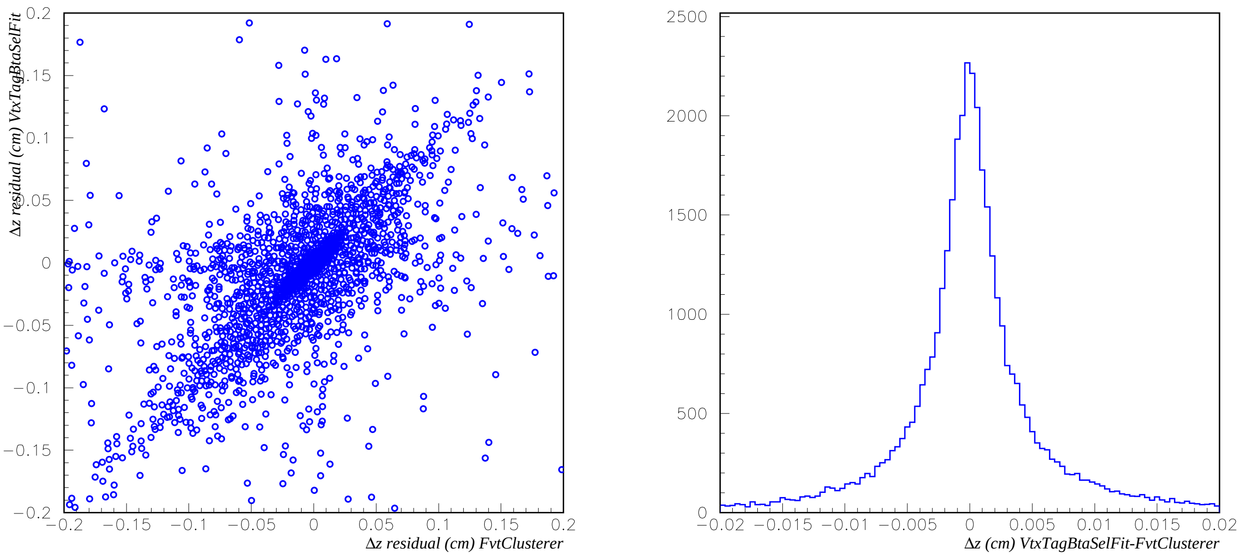


Figure 23: Comparison among `VtxTagBtaSelFit` and `FvtClusterer`: (left) correlation between the Δz residuals for both algorithms; (right) difference between the Δz measured with each algorithm.

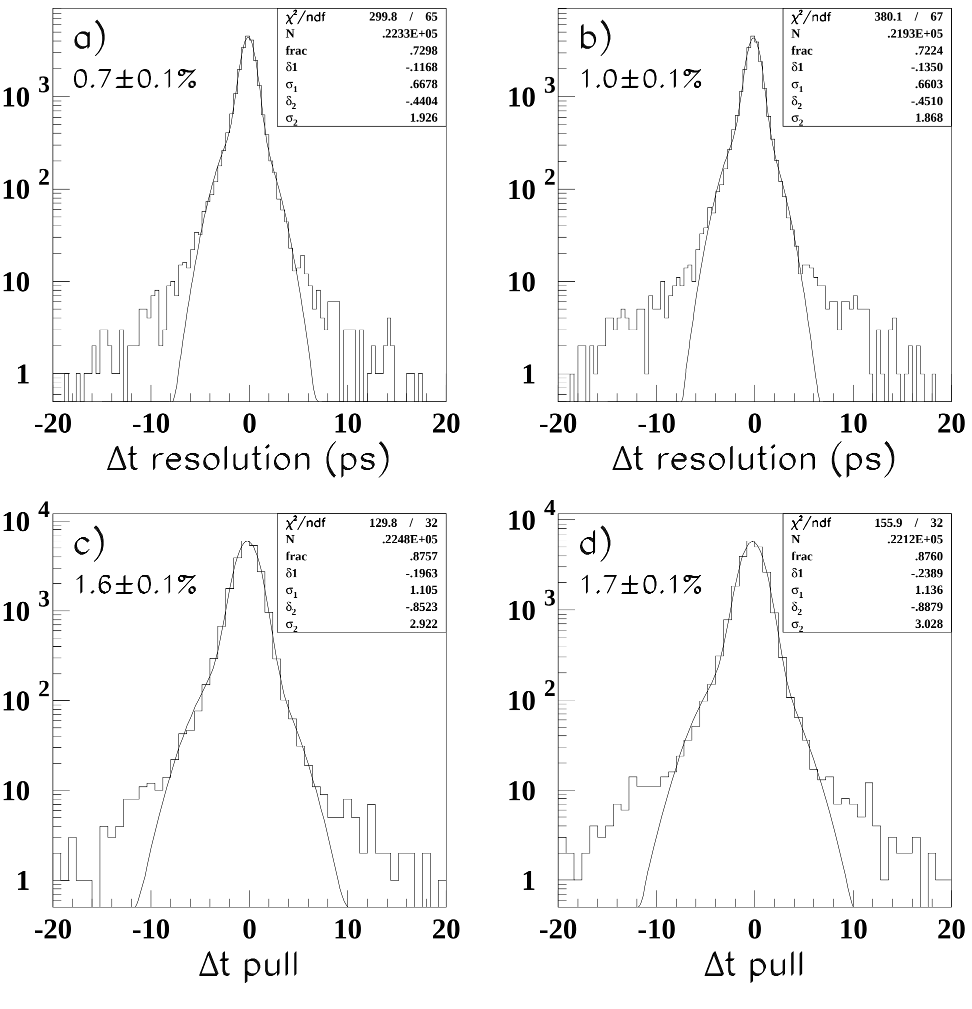


Figure 24: Δz residuals and pulls for `VtxTagBtaSelFit` and `FvtClusterer`, on semi-exclusive B^0 signal events $(B^0 \rightarrow D^* \ell \bar{\nu})$. The box shows the parameters to a double gaussian fit. The fraction of outliers is also shown: a)-c) `VtxBtaSelFit` , b)-d) `FvtClusterer`

5 Primary vertex reconstruction

For many physics analyses and reconstruction tasks, it is necessary to have a more accurate estimate of the collision point in x and z. For instance, reconstruction of γ and π^0 candidates uses, in many cases, the primary vertex as the point of origin in reconstructing vectors.

The primary vertex is estimated on an event-by-event basis from a vertex fit (for both two-prong and hadronic events) which uses charged tracks with an impact parameter (with respect to the beam spot position measured in the previous run with the two-prong DOCA-ϕ fit) less than 1 mm in the transverse plane. The window is set to ± 3 mm in z. Tracks are required to have also a minimum transverse momentum of 100 MeV/c, and a maximum total momentum of 10 MeV/c^2, with 20 DCH hits[9]. Tracks with high χ^2 contribution to the vertex fit are removed until an overall χ^2 probability greater than 1% is obtained.

First and second moments of vertex coordinates are accumulated. The mean position is,

$$\langle x^i \rangle = \frac{\sum_{k=1}^{N} x_k^i}{N}, \tag{83}$$

where N is the number of vertices, and the covariance matrix of vertices distribution is

$$M^{ij} = \frac{\sum_{k=1}^{N} x_k^i x_k^j}{N} - \langle x^i \rangle \langle x^j \rangle. \tag{84}$$

The errors on x^i are estimated by $\sqrt{M^{ii}/N}$, and those on σ^i are $\sigma^i/\sqrt{2N}$ where σ's are along the beam axes. The error on the tilt angles on i-j plane is $\frac{\sigma_i \sigma_j}{(\sigma_i^2 \sigma_j^2)\sqrt{N}}$ in a small angle approximation. The nine errors squared can form an error matrix with diagonal terms only.

The primary vertex is computed in production by the `VtxProdCreateSequence` sequence (VertexingTools package), module `VtxEvent`. It makes use of the `FvtClusterer` algorithm. Before release 8.6.0, the vertex was reconstructed using the `PrimVtxFinder` module.

For events where the e^+e^- collision produces light quarks, the reconstructed vertex is a good estimate of the true primary vertex. For $B\overline{B}$ events, where both B mesons travel along the z axis in lab frame, the z position of the primary vertex will be shifted in the positive z direction with respect the e^+e^- collision point, giving an average B decay position (midway point between the two decays). The resolution is about 70 μm in x, y and z for hadronic events.

Few Monte Carlo plots of resolution...

The primary vertex information can be accessed from `eventInfo->primaryVtx()`.

[9]This corresponds to the definition of the `GoodTracksTight` list.

6 Beam spot reconstruction and monitoring

The determination of the position and size of the beam-spot (luminous region) is an important component of many *BABAR* analyses: D^* reconstruction, D lifetime and Δz reconstruction are standard examples. Details about the physics motivation and reconstruction methods can be found in reference [10]. In this document we provide a brief summary of how it is reconstructed as well as we provide some monitoring quality plots. We put special emphasis in how the beam spot information is used in reconstruction and how it has to be used for analysis.

In order to accommodate the movements of the PEP-II beam with respect to the *BABAR* detector, the position of the collision point at PEP-II is determined from track-based methods. The apparent size from two-prong events is about $200\,\mu$m in x (after rotation), $35\,\mu$m in y and 0.8 cm in z. The apparent size in the y direction is totally dominated by the track resolution. A better estimate of σ_y can be obtained from the knowledge of the luminosity, the beam currents and the size in x, providing a value of about $4\,\mu$m, varying within 10% on a time scale of hours.

6.1 Beam spot determination

Details about how the beam spot is determined can be found in reference [10].

To determine the position, size and rotation of the beamspot, we select well-reconstructed tracks from two-prong events and fit the distance of closest approach with respect to the z-axis to a sinusoidal function $-x \sin\phi + y \cos\phi$. The z information can be obtained from simple geometry with the fact that these two tracks are back-to-back in transverse plane and are boosted approximately along the z-axis. The beamspot size can be determined from residuals. The parameter space is then extended to 9 parameters (3 means, 3 sigmas, and 3 rotations). The fitting procedure is linearized so that we can accumulate track information for a given period of time (typically a run) and approximately maximize the likelihood function with one matrix operation. Typically this procedure converges in a few runs after a major changes in beamspot position. We also accumulate the pirmary vertices of two-prong and multi-hadron events for cross check of beamspot parameters.

The beamspot parameters are stored in the conditions database on a run-by-run basis with rolling callibration. Since we fit parameters with one linear operation, the initial values of the parameters have to be close to the final estimation. Otherwise the linear approximation would not hold. In rolling calibration, the initial condition is taken from the results of the previous run. During normal data taking, the rolling calibration can keep up the slow beamspot movement (typically less than 10 μm/hour) mainly due to the diurnal motion of support tube with respect to the whole BaBar detector.

6.2 Quality of the beam spot determination

The beam spot parameters across the whole year 2000 data are shown in Fig. 25-Fig. 28. The values are taken from the top layer of the conditions database with sampling time being 5 minutes. The band at the bottom of each plot (labeled C, D and E) indicates the ranges that differnt SVT local alignment constants are used. From these plots there is no clear correlation between beam spot parameters and local alignment sets. Figure 29 shows the differences in the x and y beam spot positions for SVT local alignment sets E and D. A shift of $\sim 9\ \mu$ and m$\sim 17\ \mu$ m for x and y, respectively, is observed.

The major discontinuities are corresponding to machine accesses. A couple of "flat" regions (the parameters remain constant) across several days are during normal data taking. This indicates some problem with conditions database.

The errors on the mean are typically of the order of 1 μm on x and y. The histograms for error and size on y-axis is shown in Fig 30. Although the error on the mean y position is typically around 1 μm, the beam position can move significantly compared with 1 μm during a run due to machine operation and diurnal motion. We have not had a reliable way to examine this issue. Nevertheless, by looking the fluctuation between adjacent runs, we can get some idea of the order of magnitude of the smearing within a run. Fig. 31 shows the changes on x and y mean position between adjacent runs. The RMS for y is about 10 μm, and about 20 μm for x.

The xz and yz tilt angles are, respectively, $\sim$ 19.6 and 1 mrad.

6.3 Database storage and access

The 10 beamspot parameters determined by rolling calibration are stored in the BaBar conditions database along with their 10×10 error matrix. Parameters obtained using events collected during a given time period are stored in a time validity interval that spans the corresonding event collection period. The unique time-stamp of each event enables users of this information to obtain the beamspot parameters the correct time interval. By default, the beamspot parameters provided by PepBeams and EventInfo are those obtained with the 2-prong sample.

6.4 Beam spot size from luminosity measurement

The PEP-II luminosity is given by

$$\mathcal{L} = \frac{I_{HER}\, I_{LER}}{8\pi e^2 f_c\, \sigma_x \sigma_y}, \tag{85}$$

where I_{HER} and I_{LER} are the individual beam currents, e is the electron charge, f_c is the collision frequency, and σ_x and σ_y are the x and y Gaussian widths of the luminous region. Note that σ_x and σ_y are smaller than the single beam sizes by a factor of $sqrt2$. $sigma_y$ can be extracted from this expression, given a measurement of all the other quantities. Fig. 32 shows $sigma_x$ and σ_y as a function of time during owl shift on September 1, 1999. σ_y is seen to be around $4 - 5$ μm, decreasing as the bunch currents decrease during the course of each run. Since σ_y is much smaller than the BaBar vertexing resolution, it is not monitored online.

Variation of σ_y within a run...

6.5 Beam spot information for analysis in data and Monte Carlo

At the analysis level the beam spot is obtained from the condition database via `eventInfo->beamSpot()`

```
...
HepAList<EventInfo>* eventInfoList;
getTmpAList (anEvent, eventInfoList, IfdStrKey(``Default'') );
EventInfo* eventInfo = eventInfoList->first();
BbrPointErr beamSpot(eventInfo->beamSpot());
```

If you are running on data, the values returned here are those shown in the previous subsections. We have seen there that the apparent beam spot size in y is determined to be of the order of 35 μm, while the true σ_y is estimated to be between 4 and 5 μm. In Monte Carlo, the beam spot in the condition database is taken from a typical run from the range 5765-8000. The position, withs and xz tilt are:

- position: (0.1,0.33,-0.9) cm;
- widths: (125,4.2,8500) μm;
- xz tilt: -18.8 mrad.

The 4.2 μm for σ_y is an estimate from luminosity measurement, smaller than the tracking resolution (rolling callibration). It should be noted that in the Monte Carlo the beam spot position is generated at a fixed position (no smearing), although the B vertex positions are well smeared.

The question now is which is the *right* value of σ_y to be used for physics analysis. It is obvious that it will depend on our analysis, especifically whether we are analysing continuum or B events, and in the case of Δz reconstruction which algorithm and configuration are we using.

In order to deal with the different possibilities, there is a set of tcl parameters available in the `BtaLoadBeamSpot` object which allows to adapt the beam spot to the analysis needs. The default configuration is intendeed to be valid for most (if not all) the *BABAR* analyses:

sigmaY if positive, superseeds the width in y from the condition database. In order to account for the smaller real size compared to the apparent one, the movement within a run (following the previous results) and the small yz tilt angle, the default value has been set-up at 10 μm;

errorYPos if positive, an smearing on the y position is applied in the Monte Carlo. This is done in an attempt to account for errors on the y mean position. The default value is 5 μm;

errorXPos same as before but in x. Default is -1 (no smearing);

errorZPos same as before but in z. Default is -1 (no smearing);

sigmaYBFlight if positive, defines the width in y for the beam spot accounting for the flight of the B as well as the intrinsic beam spot spread. This new beam spot is available via `eventInfo->beamSpotBFlight()`. The default value is 30 μm, quadratic sum of the B flight component ($\sim$ 25 μm) and the intrinsic beam spot spread (same as before, 10 μm). Parameters `errorYPos`, `errorXPos` and `sigmaYBFlight` also apply here.

offsetY defines an offset in the y position of the beam spot. To be used for systematic studies. Default is 0;

offsetX same as before but in x;

offsetZ same as before but in z;

A particular application of these two beam spots, `beamSpot()` and `beamSpotBFlight()`, is the vertex tag reconstruction. `FvtClusterer` and `VtxTagBtaSelFit` in partial reconstruction have to apply the beam spot constraint directly to the tagging B, therefore the beam spot to be used has to account for the B flight. Meanwhile, `VtxTagBtaSelFit` in full reconstruction applies the constraint directly to the $\Upsilon(4S)$ decay point (in fact, it makes use of the B momentum to translate it to the tagging B vertex), therefore the standard beam spot can be used in this case. This operation is, however, transparent for the user.

The estimation of 25 μm RMS as effect of the B lifetime is similar to that used in section 4.5 to estimate the impact in terms of RMS of the assumptions made in the boost approximation. The transverse flight of the B meson can be written as $\Delta y = \beta_{CP}^{cms} \sin\theta_{CP}^{cms} ct_{CP}$. Taking into account the $\Upsilon(4S) \to B\overline{B}$ angular distribution and assuming no polar angle bias, $\langle \sin\theta_{CP}^{cms} \rangle = 0$, so $\langle \Delta y \rangle = 0$, as expected. The B lifetime component can then be extrated from the RMS, $\langle \sin^2\theta_{CP}^{cms} \rangle = 4/5$, therefore

$$\sqrt{\langle \Delta^2 y \rangle} \approx \beta_{CP}^{cms} \langle \sin^2\theta_{CP}^{cms} \rangle^{1/2} \langle ct_{CP} \rangle \approx 0.057 c\tau_B \approx 25\mu m \tag{86}$$

Generator level charmonium events have been used to check this estimation. Figure 33 shows the distance between the B decay vertex and the $\Upsilon(4S)$ production point. The RMS of the distribution is about 25 μm. However, it is clear that the distribution is highly non-Gaussian. If we attempt a single Gaussian fit (in order to estimate the RMS) to that distribution, χ^2 and likelihood approaches provides significantly different results, $\sim$ 15 μm for the former, $\sim$ 25 μm for the latter. As in the beam spot constraint we use a Gaussian approach for the meaning of the width induced by the B flight, one could rise the question of which is the most appropiate estimate, 15 or 25 μm. Using $B^0 \to D^{*-}\ell\nu$ it has been verified that there is no sizeable difference when using either of both values (see for instance [23]). Therefore, as default value we have adopted finally 25 μm (30 μm when accounting for the intrinsic width).

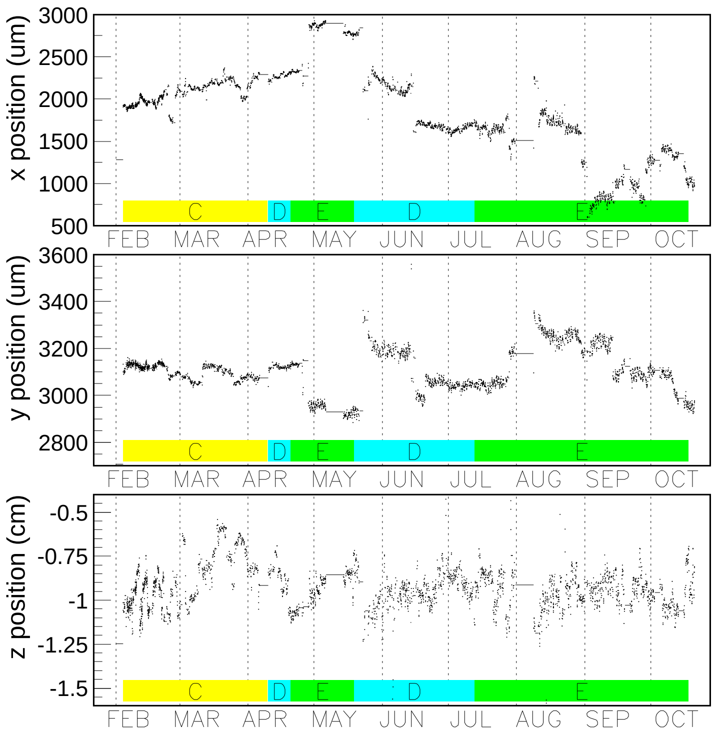


Figure 25: Mean positions on x, y and z axes, respectively.

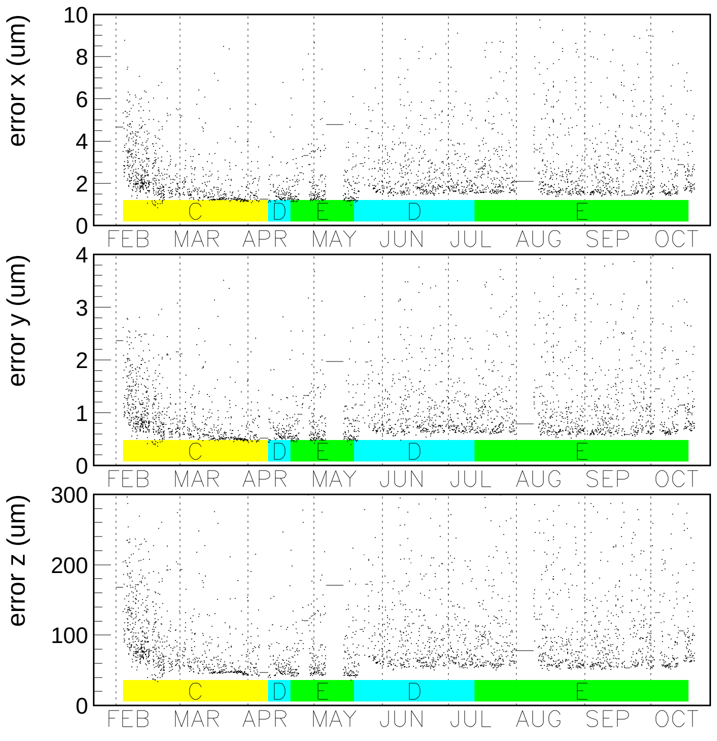


Figure 26: Errors on mean positions.

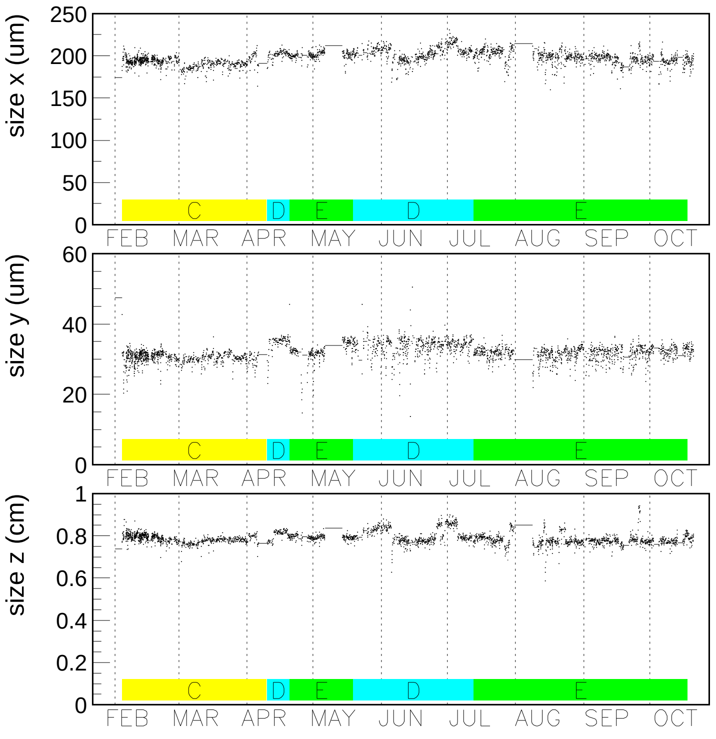


Figure 27: Apparent sizes of the projections on three axes.

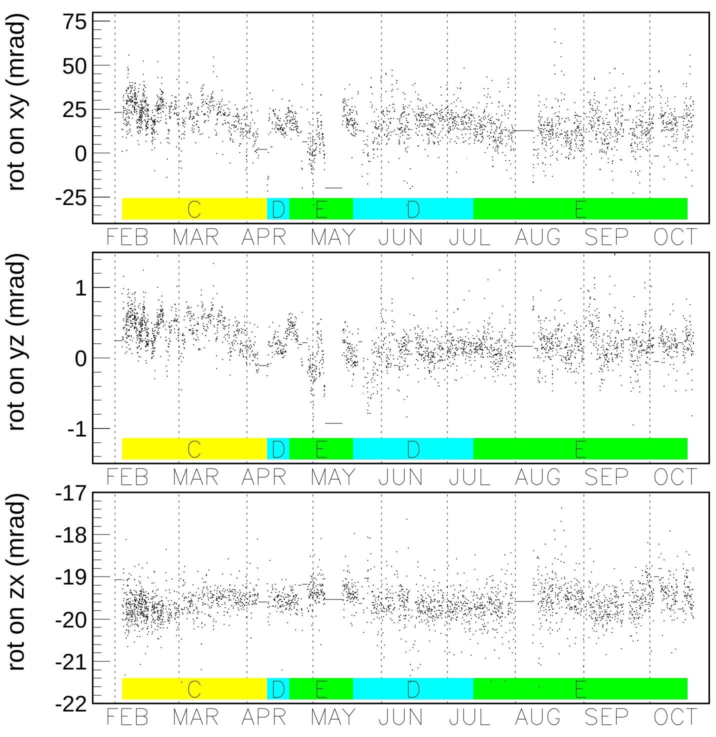


Figure 28: Major axes tilt angles projected on three planes.

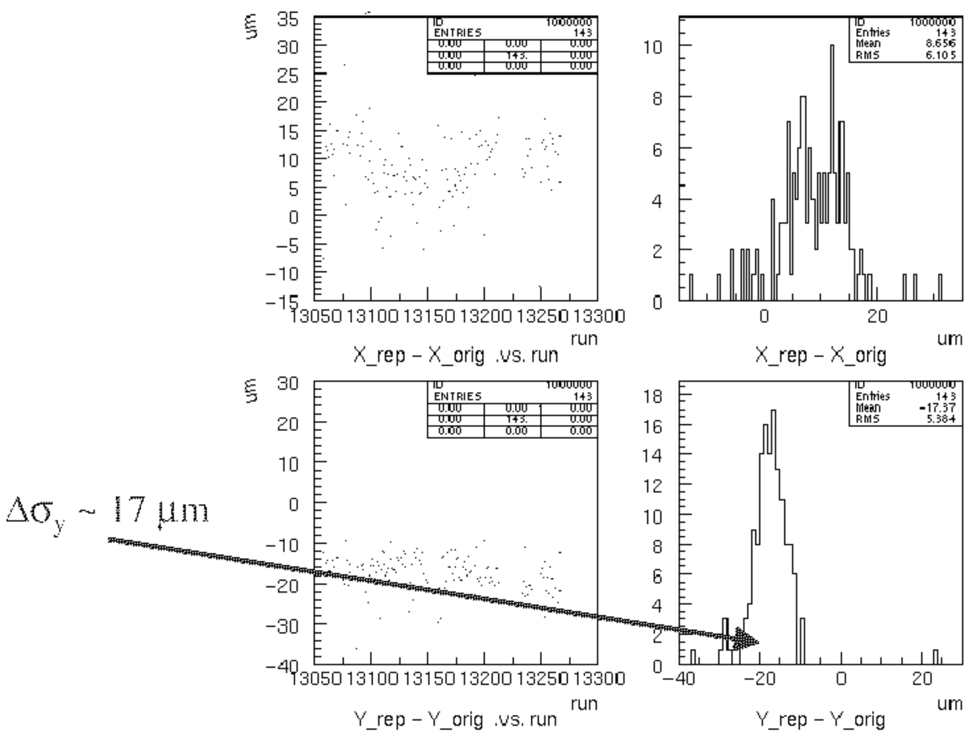


Figure 29: Dependence of x and y beam spot positions on SVT Local Alignment (sets E - D).

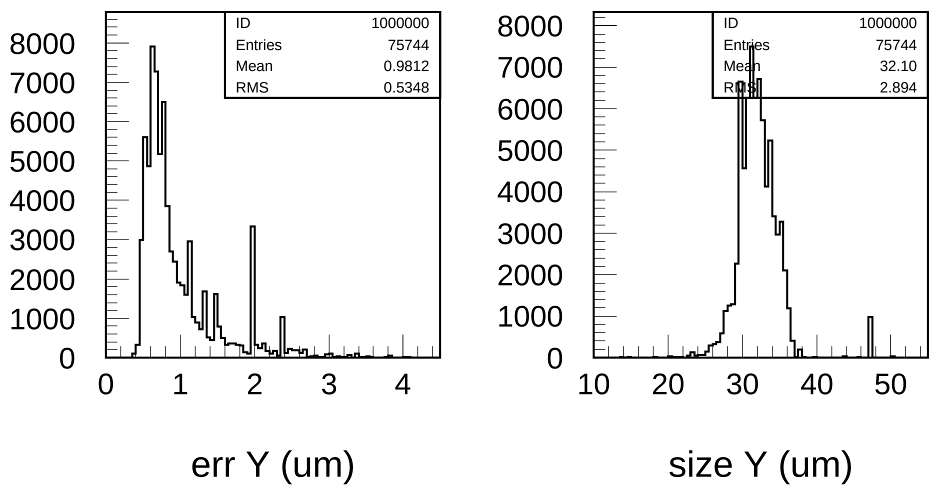
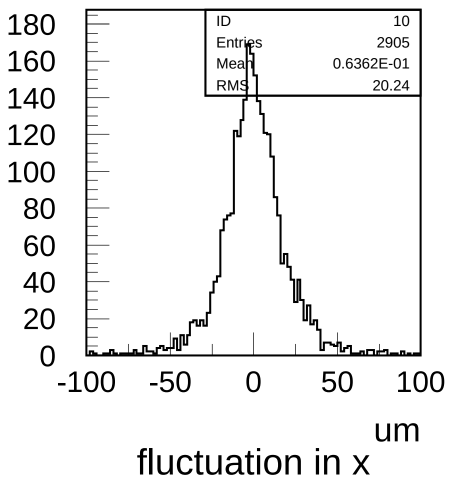


Figure 30: Error on the mean y position and apparent size.



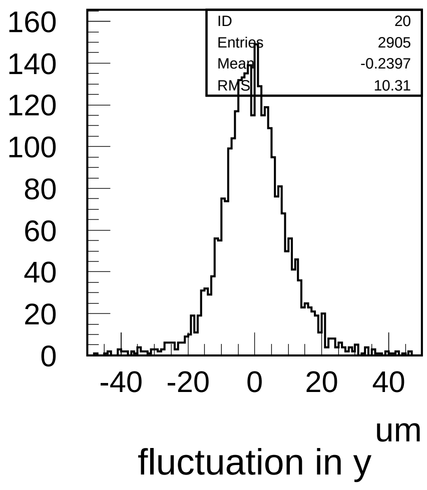


Figure 31: Changes of mean beam position between adjacent runs.

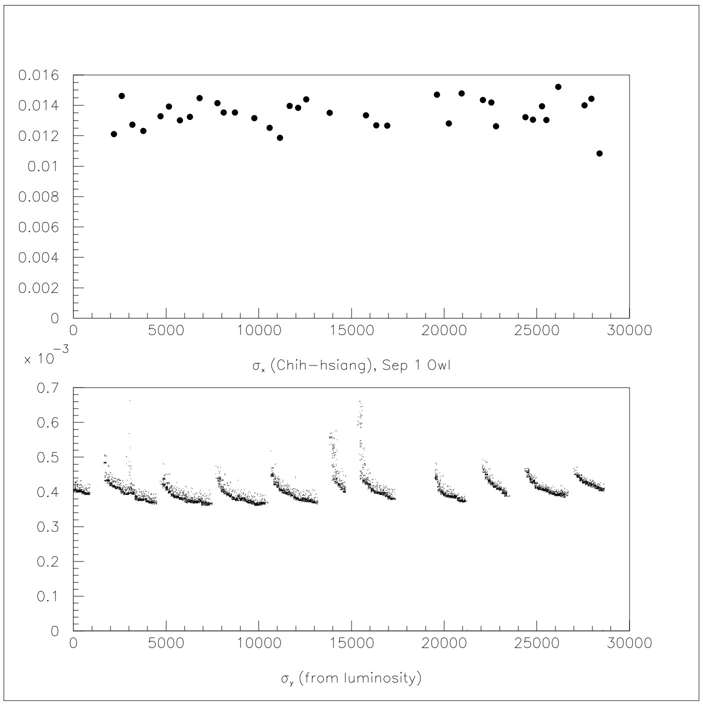
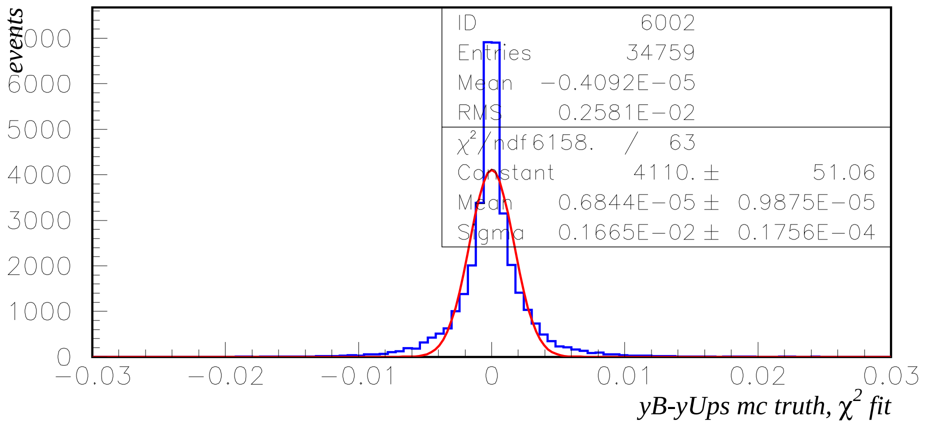


Figure 32: Top: The measured horizontal beamspot size, σ_x, as a function of time during owl shift, September 1, 1999. Bottom: The vertical beam size, σ_y, calculated from σ_x, the luminosity, and the beam currents.



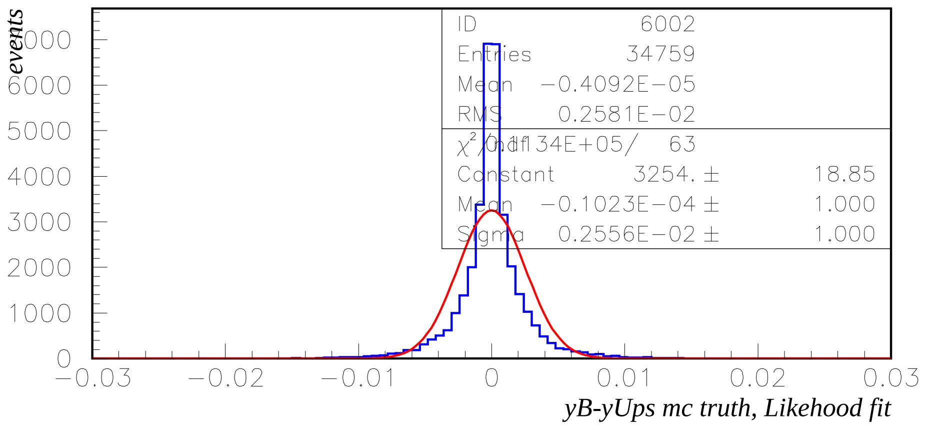


Figure 33: y distance between the B decay vertex and the $\Upsilon(4S)$ production point at generator level. Fits are to a single Gaussian, using χ^2 (top) and likelihood (bottom) fits. .

A Transformation from track helix to X-P representations

The transformation from the *X-P representation* to the helix parameters is:

$$d_0 = \frac{-xp_x + yp_y}{p_\perp} \tag{87}$$

$$\phi_0 = \tan^{-1}\frac{p_y}{p_x} \tag{88}$$

$$\omega = \frac{1}{p_\perp} \tag{89}$$

$$z_0 = z - \frac{p_z(xp_x + yp_y)}{p_\perp^2} \tag{90}$$

$$\lambda = \tan\theta_{dip} = \frac{p_z}{p_\perp} \tag{91}$$

where $p_\perp = \sqrt{p_x^2 + p_y^2}$ is the transverse momentum.

Using the `difNumber`'s class, the derivatives needed are also obtained.

B Doca calculations

References

[1] *BABAR* Analysis Document # 130, Performances and control samples of the *BABAR* Vertexing.

[2] Statistical and Computational Methods in Data Analysis, Siegmund Brandt, North Holland Publishing.

[3] Data Analysis and Kinematic Fitting With the KWFIT Library, Paul Avery, CSN 98-355. 1998

[4] R. Faccini, F. Martinez-Vidal *Vertexing/Kinematic Fitting User's Guide*, http://www.slac.stanford.edu/BFROOT/www/Physics/Tools/Vertex/VtxGuide/index.html

[5] *BABAR* Analysis Document # 119, B Tagging in *BABAR*: Status for the Spring 2001 Conferences.

[6] F. Martinez-Vidal *VtxTagBtaSelFit: a vertex tag reconstruction tool. User's Guide*, http://www.slac.stanford.edu/BFROOT/www/Physics/Tools/Vertex/VtxGuide/VtxTagBtaSelFit.html

[7] P Billoir, R Fruhwith and M Regler , Nucl. Instr. Meth. A **2**42 (1985) 115-131

[8] P Billoir, S. Qian , Nucl. Instr. Meth. A **3**11 (1992) 139-150

[9] S. Plaszczynski, *FastVtx How-To*, http://www.lal.in2p3.fr/recherche/babar/Analyse/FastVtx/HowTo/index.html

[10] *BABAR* Analysis Document # 13, Beam spot determination and use in *BABAR*.

[11] S. Plaszczynski, Tagging/vertexing correleations at $\sin 2\beta$ workshop, 3rd Nov 2000.

[12] B. Dunwoodie et al, *BABAR* Analysis Document # 106, Study of material interactions with gamma conversions and protons.

[13] G. Raven, *$D^{*-} - D^0$ vertexing in $B^0 \to D^{*-}\ell\nu$ events*, talk at $\sin 2\beta$ WorkShop, 3rd November, 2000, http://www.slac.stanford.edu/BFROOT/www/Physics/CP/beta/Meetings/03Nov00/S2bWorkshop00.html

[14] S. Meztler, *BABAR* Analysis Document # 65, Measurement of B Lifetimes at *BABAR*.

[15] Bill Dunwoodie, private communication.

[16] Art Synder, Δz vs Δt, http://babar-hn.slac.stanford.edu:5090/HyperNews/get/recoTracking/386/4.html

[17] D. Kirkby, Generator Level Studies for $B\ \bar{B}$ Mixing, http://www.slac.stanford.edu/~davidk/BBMix/GenStudy/

[18] *BABAR* Analysis Document # 14, Measuring the PEP-II Boost.

[19] Presentations at the Forum meeting on May 16th (200), http://www.slac.stanford.edu/BFROOT/www/Physics/Forum/forum/phonemeetings/forum_16may00/doc.html

[20] A. Soffer, Beam parameters in data and Monte Carlo, talk at $\sin 2\beta$ WorkShop, 3rd November, 2000,
http://www.slac.stanford.edu/BFROOT/www/Physics/CP/beta/Meetings/03Nov00/S2bWorkshop00.html

[21] Ch. De la Vaissiere et al., Lifetimes with full B reconstruction, BaBar Note # 436 (1998).

[22] R. Muller-Pfefferkorn, R. Waldi, BaBar Note # 373 (1997).

[23] http://babar-hn.slac.stanford.edu:5090/HyperNews/get/VertexTools/173/1/1/1.html

[24] An example is the event 160029/c6888dd3:H reported by Art Snyder, http://www.slac.stanford.edu/ snyder/badVertex-160029-c6888dd3.html