DOI: 10.17181my9jk-hv376

A vertex fitting package

Franco Bedeschi[1*]

[1*]INFN - Sezione di Pisa, Largo B. Pontecorvo 3, Pisa, 56127, Italy.

Corresponding author(s). E-mail(s): franco.bedeschi@pi.infn.it;

1 Introduction

Vertex fitting code is commonly found within the analysis packages of several HEP experiments, unfortunately it usually deeply packaged inside their software infrastructure, making it cumbersome to use in the context of external applications such, for instance, in the study of the performance of the proposed FCCee detectors. In this note a totally independent package is described. The only dependencies being the ROOT libraries, making it easy to use in a wide range of applications.

The code currently works with track trajectories that are either helices or straight lines, as generated by charged or neutral particles in a constant magnetic field, but is expandable in principle to other type of trajectories or parameterizations.

This vertexing package, in addition to fitting a vertex given a list of tracks, supports several features: addition and subtraction of tracks, momenta and their errors of the tracks at the vertex point, external vertex constraints, mass constraints, and the capability to perform fits of chain decays.

This note is organized as follows: in section 2 the basic formulas for the fit are derived, section 3 describes how to use the software and two examples are given in section 4. Details on the fit formulas derivation are shown in appendix **??**. Several formulas needed for trajectories in a uniform magnetic field, track parameters and related derivatives are contained in appendix B and C.

2 Basic formulas

Two main approaches are used for vertex fitting: the simplest finds the location of the point closest to all tracks without changing the track parameters; the second is more complex and forces all tracks to go through the same point by changing the track parameters. We look now at the first method, while the second will be developed in the next subsection.

2.1 Vertex fit without track parameter steering

Given a track trajectory $\vec{x}(s;\vec{\alpha})$, where $\vec{\alpha}$ are the track parameters and the phase, s, defines the position on the trajectory, the covariance matrix S of a point on the track at fixed s is given by

$$S = W^{-1} = \frac{\partial \vec{x}}{\partial \vec{\alpha}} C \left(\frac{\partial \vec{x}}{\partial \vec{\alpha}}\right)^t = ACA^t$$

where C is the covariance matrix of the track parameters $\vec{\alpha}$. The χ^2 to minimize is the following:

$$\chi^2 = \sum_{i=1}^{N} (\vec{x}(s_i;\vec{\alpha}_i) - \vec{x}_V)^t W_i (\vec{x}(s_i;\vec{\alpha}_i) - \vec{x}_V) \tag{1}$$

where $\vec{x}_V$ is the vertex to be determined. The unknown parameters in this fit, besides the vertex position, $\vec{x}_V$, are the phases, s_i. We now set to 0 the derivatives of the χ^2, after using a first order approximation for $\vec{x}(s;\vec{\alpha}) \simeq \vec{x}^0 + \frac{\partial \vec{x}}{\partial s}\delta s = \vec{x}^0 + \vec{a}\delta s$:

$$\frac{1}{2}\frac{\partial \chi^2}{\partial s_k} = \vec{a}_k^t W_k (\vec{x}_k^0 + \vec{a}_k\,\delta s_k - \vec{x}_V) = \vec{a}_k^t W_k (\vec{x}_k^0 - \vec{x}_V) + \vec{a}_k^t W_k \vec{a}_k\,\delta s_k = 0 \tag{2}$$

which can be solved for δs_k:

$$\delta s_k = \frac{\vec{a}_k^t W_k (\vec{x}_V - \vec{x}_k^0)}{\vec{a}_k^t W_k \vec{a}_k} = \frac{\vec{a}_k^t}{a_k} W_k (\vec{x}_V - \vec{x}_k^0) \tag{3}$$

then:

$$\frac{1}{2}\frac{\partial \chi^2}{\partial \vec{x}_V} = -\sum_{i=1}^{N} W_i (\vec{x}_i^0 + \vec{a}_i\,\delta s_i - \vec{x}_V) = -\sum_{i=1}^{N} W_i (\vec{x}_i^0 + \frac{\vec{a}_i \vec{a}_k^t}{a_k} W_i (\vec{x}_V - \vec{x}_i^0) - \vec{x}_V) \tag{4}$$

that can be solved for $\vec{x}_V$:

$$\vec{x}_V = (\sum_{i=1}^{N} D_i)^{-1} (\sum_{i=1}^{N} D_i \vec{x}_i^0) = D^{-1} (\sum_{i=1}^{N} D_i \vec{x}_i^0) \tag{5}$$

where:

$$D_i = W_i - W_i \frac{\vec{a}_i \vec{a}_i^t}{a_i} W_i \tag{6}$$

The error matrix on $\vec{x}_V$ is obtained by error propagation on the $\vec{x}_i^0$:

$$Cov(\vec{x}_V) = D^{-1} (\sum_{i=1}^{N} D_i W_i^{-1} D_i) D^{-1} \tag{7}$$

The procedure can be iterated by using $\vec{x}_V$ as obtained in eq. 5 to update the s_i (from eq. 3) and then the $\vec{x}_i^0$.

2.2 Vertex fit with parameter steering

The method described in section 2.1 is very fast and provides a reliable vertex position and error matrix, however it is not suitable to allow further functionality, such as mass constraints or applications in chain decays. We then describe another method where N input tracks are forced to pass through a common vertex by varying their track parameters. Let

$$\vec{x}_i(s_i,\vec{\alpha}_i) \simeq \vec{x}_i(s_i',\vec{\alpha}_i') + \frac{\partial \vec{x}_i}{\partial \vec{\alpha}_i}(\vec{\alpha}_i - \vec{\alpha}_i') + \frac{\partial \vec{x}_i}{\partial\, s_i}(s_i - s_i') = \vec{x}_i^{\,0} + A_i\delta\vec{\alpha}_i + \vec{a}_i\delta\, s_i$$

be all track equations and their first order expansion in the track parameters, $\vec{\alpha}_i$ and the variables describing the position in each trajectory, s_i. The symbol $'$ is used to indicate the expansion point. Let also C_i be the covariance matrices of the original track parameters $\vec{\alpha}_i^{\,0}$ The vertex fit amounts to minimizing the following χ^2:

$$\chi^2 = \sum_{i=1}^{N}(\vec{\alpha}_i - \vec{\alpha}_i^{\,0})\,{}^tC_i^{-1}(\vec{\alpha}_i - \vec{\alpha}_i^{\,0}) + 2(\vec{x}(s_i,\vec{\alpha}_i) - \vec{x}_V)^t\vec{\lambda}_i\} \tag{8}$$

where $\vec{\alpha}_i$ and s_i are the modified track parameters and the phases at the vertex, $\vec{x}_V$ is the vertex to be determined and $\vec{\lambda}_i$ are Lagrange multipliers. The fit varies the track parameters and the phases until all tracks go through a common vertex $\vec{x}_V$. The solution is found to be:

$$\vec{x}_V = \left(\sum_{i=1}^{N} D_i\right)^{-1}\left(\sum_{i=1}^{N} D_i(\vec{x}_i^{\,0} + A_i\,\delta\vec{\alpha}_i^{\,0})\right) = D^{-1}\left(\sum_{i=1}^{N} D_i(\vec{x}_i^{\,0} + A_i\,\delta\vec{\alpha}_i^{\,0})\right) \tag{9}$$

where $\delta\vec{\alpha}_i^{\,0} = \vec{\alpha}_i^{\,0} - \vec{\alpha}_i$ is the difference between the input and final track parameters. The corresponding error matrix on $\vec{x}_V$ is obtained by taking the average of the fluctuations of $\vec{x}_V$ due to those of the track parameters, $\vec{\alpha}_i^{\,0}$:

$$\begin{aligned} C_V = Cov(\vec{x}_V) \;&=\; D^{-1}\left(\textstyle\sum_{ij} D_iA_i < \delta\vec{\alpha}_i^{\,0}\delta\vec{\alpha}_j^{\,0t} > A_i^tD_j\right)D^{-1} \\ &=\; D^{-1}\left(\textstyle\sum_i D_iW_i^{-1}D_i\right)D^{-1} \end{aligned} \tag{10}$$

We note that the vertex position and its errors are almost the same as obtained with the simpler fit with no parameter steering, however here we have the updated parameters satisfying the vertex constraint. This is very useful to expand the functionality of the vertex fit as seen in later sections, when dealing with complex constraints or chain decays.

2.3 χ^2 and updated track parameter errors

Since the constraints are all zero after solving the minimization problem, the χ^2 is given only by the sum of the variation of the track parameters:

$$\chi^2 = \sum_{i=1}^{N} (\vec{\alpha}_i - \vec{\alpha}_i^{\,0})^{\,t} C_i^{-1} (\vec{\alpha}_i - \vec{\alpha}_i^{\,0}) = \sum_{i=1}^{N} \vec{\lambda}_i^{\,t} (A_i C_i A_i^t) \vec{\lambda}_i = \sum_{i=1}^{N} \vec{\lambda}_i^{\,t} W_i^{-1} \vec{\lambda}_i \tag{11}$$

Comparing with eq. 1 the Lagrange multipliers correspond to the vector distances of the tracks from the vertex. Given the structure of the χ^2 it is easy to separate the contribution of each track. The $\vec{\lambda}_i$ can be obtained directly from eq. A1:

$$\vec{\lambda}_i = D_i(\vec{x}_i^{\,0} + A_i\,\delta\vec{\alpha}_i^{\,0} - \vec{x}_V) = D_i \sum_{k=1}^{N} (I\delta_{ik} - D^{-1}D_k)(\vec{x}_k^{\,0} + A_i\,\delta\vec{\alpha}_i^{\,0}) \tag{12}$$

This leads to a closed form for the relation between the updated track parameters, $\vec{\alpha}_i$, and those in input, $\vec{\alpha}_i^{\,0}$:

$$\begin{aligned} \vec{\alpha}_i &= \vec{\alpha}_i^{\,0} - C_i\,A_i^t \vec{\lambda}_i \\ &= \vec{\alpha}_i^{\,0} - C_i\,A_i^t D_i \sum_{k=1}^{N} (I\delta_{ik} - D^{-1}D_k)(\vec{x}_k^{\,0} + A_k\,\delta\vec{\alpha}_k^{\,0}) \end{aligned} \tag{13}$$

so:

$$\begin{aligned} d\vec{\alpha}_i &= d\vec{\alpha}_i^{\,0} - C_i\,A_i^t D_i \sum_{k=1}^{N} (I\delta_{ik} - D^{-1}D_k) A_k\,d\vec{\alpha}_k^{\,0} \\ &= \sum_{k=1}^{N} [I_5\delta_{ik} - C_i\,A_i^t D_i (I_3\delta_{ik} - D^{-1}D_k) A_k]\,d\vec{\alpha}_k^{\,0} \qquad (14) \\ &= \sum_{k=1}^{N} M_k^i\,d\vec{\alpha}_k^{\,0} \qquad (15) \end{aligned}$$

where the subscript under the I indicates the dimensionality of the unit matrix. So

$$M_k^i = \frac{\partial\vec{\alpha}_i}{\partial\vec{\alpha}_k^{\,0}} \tag{16}$$

Finally the errors on the updated track parameters are given by:

$$< \delta\vec{\alpha}_i \delta\vec{\alpha}_j^{\,t} > = \sum_{k=1}^{N} M_k^i < \delta\vec{\alpha}_i^{\,0} \delta\vec{\alpha}_j^{\,0\,t} > M_k^{j\,t} = \sum_{k=1}^{N} M_k^i C_k (M_k^j)^{\,t} \tag{17}$$

2.4 Momenta at vertex and their errors

The track momentum at the vertex can be calculated from the updated track parameters and the phase: $\vec{p}_i = \vec{p}_i(\vec{\alpha}_i, s_i)$ therefore:

$$\frac{\partial \vec{p}_i}{\partial \vec{\alpha}_k^0} = \frac{\partial \vec{p}_i}{\partial \vec{\alpha}_i} \frac{\partial \vec{\alpha}_i}{\partial \vec{\alpha}_k^0} + \frac{\partial \vec{p}_i}{\partial s_i} \frac{\partial s_i}{\partial \vec{\alpha}_k^0}$$

where the phase derivatives are given by:

$$\frac{\partial s_i}{d\vec{\alpha}_k^0} = \vec{S_i^k}^t = \frac{\vec{a}_i^t W_i}{a_i} (D^{-1} D_k - I\delta_{ik}) A_k \tag{18}$$

Using a similar strategy as that used to calculate $< \delta\vec{\alpha}_i \delta\vec{\alpha}_j^t >$, we get all the $< \delta\vec{p}_i \delta\vec{p}_j^t >$ terms:

$$C_{ij} =< \delta\vec{p}_i \delta\vec{p}_j^t >= \sum_{k=1}^{N} \frac{\partial \vec{p}_i}{\partial \vec{\alpha}_k^0} C_k \left(\frac{\partial \vec{p}_i}{\partial \vec{\alpha}_k^0} \right)^t \quad \text{i, j = 1, N} \tag{19}$$

This can be used to get the covariance matrix of the total vertex momentum $\vec{p} = \sum_i \vec{p}_i$:

$$< \delta\vec{p} \delta\vec{p}^t >= \sum_{i,j}^{N} C_{ij} \tag{20}$$

The correlation between the momenta and the vertex position is given by:

$$C_{i,0} =< \delta\vec{p}_i \delta\vec{x}^t >= \sum_{k=1}^{N} \frac{\partial \vec{p}_i}{\partial \vec{\alpha}_k^0} C_k \left(X_k \right)^t \tag{21}$$

where $X_k = D^{-1} A_k$.

2.5 Covariance matrix of vertex track parameters

This code has the feature of using each found vertex as an additional track (charged or neutral) that can be used for vertexing. This requires to calculate the vertex track parameters, that is trivial given the position of the vertex and its total momentum, $\vec{\alpha}_V(\vec{x}, \vec{p})$, and their covariance matrix, C_V.

It is useful to define a global vector:

$$\vec{q} = \begin{pmatrix} \vec{x} \\ \vec{p}_1 \\ \cdots \\ \vec{p}_N \end{pmatrix} \tag{22}$$

and its covariance matrix:

$$Q = Cov(\vec{q}) = \begin{pmatrix} C_{00} & C_{01} & \cdots & C_{0N} \\ C_{10} & C_{11} & \cdots & C_{1N} \\ \cdots & \cdots & \cdots & \cdots \\ C_{N0} & C_{N1} & \cdots & C_{NN} \end{pmatrix} \tag{23}$$

Then, after noting that $\partial\vec{\alpha}_V/\partial\vec{p}_i = \partial\vec{\alpha}_V/\partial\vec{p}$:

$$C_V = \frac{\partial\vec{\alpha}_V}{\partial\vec{q}} Q \left(\frac{\partial\vec{\alpha}_V}{\partial\vec{q}}\right)^t \tag{24}$$

2.6 External constraint

Adding an external constraint to the χ^2 in the form $(\vec{x}_V - \vec{y})^t V^{-1}(\vec{x}_V - \vec{y})$, where $\vec{y}$ is an independent measurement of the vertex and V its covariance matrix, is relatively straightforward. Indeed all derivatives are the same except for those with respect to $\vec{x}_V$ that become:

$$\frac{1}{2}\frac{\partial\chi^2}{\partial\,\vec{x}_V} = -\sum_{i=1}^{N}\vec{\lambda}_i + V^{-1}(\vec{x}_V - \vec{y}) = \sum_{i=1}^{N} D_i(\vec{x}_V - \vec{x}_i^{\,0} - A_i\,\delta\vec{\alpha}_i^{\,0}) + V^{-1}(\vec{x}_V - \vec{y}) = 0$$

solving this the final vertex $\vec{x}_V$ is:

$$\vec{x}_V = \left(\sum_{i=1}^{N} D_i + V^{-1}\right)^{-1}\left(V^{-1}\vec{y} + \sum_{i=1}^{N} D_i(\vec{x}_i^{\,0} + A_i\,\delta\vec{\alpha}_i^{\,0})\right) = D_V^{-1}\left(V^{-1}\vec{y} + \sum_{i=1}^{N} D_i(\vec{x}_i^{\,0} + A_i\,\delta\vec{\alpha}_i^{\,0})\right) \tag{25}$$

Its covariance matrix can be easily obtained by error propagation:

$$C_V = Cov(\vec{x}_V) = D_V^{-1}\left(V^{-1} + \sum_{i=1}^{N} D_i W_i^{-1} D_i\right) D_V^{-1} \tag{26}$$

The effect of this type of contraint on all above formulas is just a modification of the matrix D:

$$D = \sum_i D_i \rightarrow D = \sum_i D_i + V^{-1} \tag{27}$$

2.7 Generic constraint

Both vertex and moments can be improved applying a constraint, for instance an invariant mass constraint on a subset of particles used for vertexing. We solve here the case of a generic set of contraints using the notations defined in the previous sections.

Let $\vec{f}(\vec{q})$ be the constraints, then the χ^2 is given by:

$$\chi^2 = (\vec{q} - \vec{q}_0)^t Q^{-1}(\vec{q} - \vec{q}_0) + 2\vec{f}^{\,t}(\vec{q})\vec{\lambda} \tag{28}$$

We linearize the constraint around a point $\vec{q}'$ to allow the solution by iteration,

$$\vec{f}(\vec{q}) \simeq \vec{f}(\vec{q}') + \frac{\partial \vec{f}}{\partial \vec{q}}(\vec{q} - \vec{q}') = \vec{f_0} + B\delta\vec{q} \tag{29}$$

we also define $\delta\vec{q_0} = \vec{q_0} - \vec{q}'$. The linearized problem is solved by setting to zero the derivatives of the χ^2 with respect to the vectors $\vec{p}$ and $\vec{\lambda}$:

$$\frac{1}{2}\frac{\partial \chi^2}{\partial \vec{q}} = Q^{-1}(\delta\vec{q} - \delta q_0) + B^t\lambda = 0 \quad \rightarrow \quad \delta\vec{q} = \delta q_0 - QB^t\lambda \tag{30}$$

Replacing $\delta\vec{q}$ in the constraint we obtain λ:

$$\vec{f_0} + B\delta\vec{q_0} - BQB^t\lambda = 0 \quad \rightarrow \quad \lambda = (BQB^t)^{-1}(\vec{f_0} + B\delta\vec{q_0}) \tag{31}$$

and finally :

$$\delta\vec{q} = \delta\vec{q_0} - QB^t(BQB^t)^{-1}(\vec{f_0} + B\delta\vec{q_0}) = (I - QB^t(BQB^t)^{-1}B)\delta q_0 - QB^t(BQB^t)^{-1}\vec{f_0} \tag{32}$$

The covariance of the updated parameters is obtained averaging over the fluctuations of the $\vec{q_0}$:

$$Cov(\vec{q}) = (I - QB^t(BQB^t)^{-1}B)Q(I - QB^t(BQB^t)^{-1}B)^t = Q - QB^t(BQB^t)^{-1}BQ \tag{33}$$

3 Software implementation

The vertex fitting code is implemented with three classes: `VertexFit`, that performs the vertex fit with some options, `VertexMore`, that provides additional features such as update the fit with mass constraints, calculation of track momenta at the vertex and their errors and calculation of the vertex track parameters to allow use in chain decays, while some usuful general formulas are kept in `TrkUtil`.These three classes are currently part of the `DELPHES` package available at `https://github.com/delphes/delphes`. These three classes are self-contained and the only other dependencies are with the standard `ROOT` package.

It is worth noting that, while `VertexFit` can be used with any units or magnetic field, since its formulas depend only on the track parameters, this is not the case for `VertexMore` since explicit track parameter to momenta conversions are needed. Meters, GeV and B-field of 2 Tesla are the default. An option to switch to millimeters is provided in the constructor, but at present the field needs to be changed by hand in the code.

Given an array of pointers to track parameter sets, `tPar`, and their covariance matrices, `tCov`, the vertex fit is obtained by the following simple code sequence:

```
const Int_t N = something;
TVectorD* tPar[N]; TMatrixDSym* tCov[N];
.... assign tPar[i] and tCov[i] ....
eg: tPar[i] = new TVectorD(par); tCov[i] = new TMatrixDSym(cov);
.....................................
VertexFit* Vfit = new VertexFit(N, tPar, tCov);
```

This default constructor assumes that all input tracks are charged. If this is not the case, one needs to tell the fit which tracks are charged and which are not by adding a `Bool_t` array that is `kTRUE` for charged tracks and `kFALSE` for neutrals as follows:

```
Bool_t Charge[N];
... assign Charge[i] values ...
VertexFit* Vfit = new VertexFit(N, tPar, tCov, Charge);
```

The fit results are provided by the following methods of `VertexFit`:

```
TVectorD XvFit = Vfit->GetVtx();     // Vertex position
TMatrixDSym XvCov = Vfit->GetVtxCov();// Vertex position covariance
Int_t Ntr = Vfit->GetNtr();          // Number of tracks in fit
Double_t Chi2 = Vfit->GetVtxChi2(); // Vertex χ^2
TVectorD Chi2List = Vfit->GetVtxChi2List();// χ^2 contribution of each track
TVectorD NewPar = Vfit->GetNewPar(i);// Updated i^th track parameters
TMatrixDSym NewCov = Vfit->GetNewCov(i):// Updated i^th track covariance
```

We note that the instantiation of `VertexFit` performs only some basic initializations, while the actual fitting is triggered only when any of these methods is invoked: `GetVtx(), GetVtxCov(), GetVtxChi2() or GetVtxChi2List()`.

The input tracks can also be added or removed incrementally as can be useful in pattern recognition combinatorics with the following methods:

```
Vfit->AddTrk(par, cov): // Adds one charged track to the fit track list
Vfit->AddTrk(par, cov, kFALSE): // Adds one neutral track to the fit track list
Vfit->RemoveTrk(i); // Removes the i^th track from the fit track list
```

Finally the `VertexFit` class supports also including an external vertex constraint, for instance an independent knowledge of the primary vertex position and width. Given the mean position, `Xpvc`, and the covariance matrix, `CovXpvc`, of this external constraint, one can include it in the fit with the call:

```
Vfit->AddVtxConstraint(Xpvc, CovXpvc);
```

N.B. when an external constraint is in place, vertex fits are allowed also with a single track in input.

Additional information after fitting can be obtained with the class `VertexMore`. Its constructor has in input the pointer to the vertex fit, `VertexFit* Vfit` and an optional `Bool_t` parameter that requests the use of millimiters if `kTRUE`, as shown in the following:

```
VertexMore* VM = new VertexMore(Vfit);
or
Bool_t opt = kTRUE;
VertexMore* VM = new VertexMore(Vfit, opt);
```

In the following the most relevant methods are described:

```
TVector3 GetMomentum(Int_t i);      // Momentum of the i^th track in the vertex
TVector3 GetMomentumC(Int_t i);     // Covariance of the above
TVector3 GetTotalP();               // Total momentum of vertex
TMatrixDSym GetTotalPcov();         // Covariance of the above
TVectorD GetXv();                   // Vertex position
TMatrixDSym GetXvCov();             // Covariance of the above
TVectorD GetVpar();                 // Vertex track parameters
TMatrixDSym GetVcov();              // Covariance of the above
```

Mass constraints on a subset of the tracks in the vertex fit are initialized by calls like:

```
VM->AddMassConstraint(Double_t Mass, Int_t Ntr, Double_t* masses,
Int_t* list);
```

where `Mass` is the constraining mass value, `Ntr` is the number of tracks involved, `masses` is an array with the masses of the tracks involved and `list` is an array with the list of track numbers involved. This function can be called more than once if there is more than one disjoined set of tracks to constrain. The fit is then activated with

`VM->MassConstrFit();`
that also updates all momenta and vertex positions and their covariance matrices, that can be accessed with the methods described above.

4 Examples

Some examples on the use of the code are provided with the standard `DELPHES` distribution available in `GitHub`. The most instructive ones are discussed in the following sections. It should be noted that all these examples assume input files generted with `Pythia8` and simulated with `DELPHES`. It is important to ensure consistency with the generated files when beam constraints are used in the examples.

4.1 Primary vertex determination

The code for this example is located in `examples/ExamplePVtxFind.C` and is executed from `ROOT` by :
`root>.X examples/ExamplePVtxFind.C("InputFile.root", 1000);`, where the second parameter is the number of events to process. A description of the algorithm and how the vertexing code is used is presented in the following.

The input is the full set of `NtrG` tracks in the event, that can be mapped to the track parameters, `pr[i]` and their covariance matrix, `cv[i]`. It is also assumed that there is an independent knowledge of the primary vertex with mean value `xpvc` and covariance `covpvc`, that wil be used as an external constraint.

The algorithm first loops over all tracks and selects those sufficiently close to the known primary vertex:

```
// Skim tracks
Int_t nSkim = 0;
Int_t* nSkimmed = new Int_t[NtrG];
TVectorD** PrSk = new TVectorD * [1];
TMatrixDSym** CvSk = new TMatrixDSym * [1];
Double_t MaxChi2 = 9.;
for (Int_t n = 0; n < NtrG; n++) {
     PrSk[0] = new TVectorD(*pr[n]);
     CvSk[0] = new TMatrixDSym(*cv[n]);
     // Vertex fit one track at a time
     VertexFit* Vskim = new VertexFit(1,PrSk, CvSk);
     // with external constraint
     Vskim->AddVtxConstraint(xpvc, covpvc);
     Double_t Chi2One = Vskim->GetVtxChi2();
     // Select depending on Chi2
     if (Chi2One < MaxChi2) {
       nSkimmed[nSkim] = n;
       nSkim++;}
}
```

Then setup the primary vertex candidate fit with all selected tracks.

```
// Load all skimmed tracks
TVectorD** PrFit = new TVectorD * [nSkim];
TMatrixDSym** CvFit = new TMatrixDSym * [nSkim];
for (Int_t n = 0; n < nSkim; n++) {
     PrFit[n] = new TVectorD(*pr[nSkimmed[n]]);
     CvFit[n] = new TMatrixDSym(*cv[nSkimmed[n]]);}
// Setup vertex fit
VertexFit* Vtx = new VertexFit(nSkim, PrFit, CvFit);
// add Constraint
Vtx->AddVtxConstraint(xpvc, covpvc);
```

and remove iteratively the tracks that contribute most to the χ^2 if above a given threshold.

```
//
// Remove tracks with large chi2
Double_t MaxChi2Fit = 8.0;
Int_t Nfound = nSkim;
const Int_t MaxFound = 100; Double_t Chi2LL[MaxFound];
Bool_t Done = kFALSE;
while (!Done) {
   // Find largest Chi2 contribution
   TVectorD Chi2List = Vtx->GetVtxChi2List(); // Contributions to Chi2
   Chi2L = Chi2List.GetMatrixArray();
   Int_t iMax = TMath::LocMax(Nfound, Chi2L);
   Double_t Chi2Mx = Chi2L[iMax]; // Largest Chi2 contribution
   if (Chi2Mx > MaxChi2Fit && Nfound > 1) {
       // Remove bad track
       Vtx->RemoveTrk(iMax);
       Nfound--;}
   else {Done = kTRUE;}
}
```

After this selection `Vtx` is the final primary vertex.

4.2 $B_s \to D_s\pi$

The code for this example is located in `examples/VtxBs2DsPi.C` and is executed from `ROOT` by :
`root>.L goBs.C+`
`root>goBs();`
`root>.X examples/VtxBs2DsPi.C("InputFile.root", 1000);`, where the second parameter is the number of events to process. The goal of this example is to first fit the decay $D_s^{\pm} \to K^+K^-\pi^{\pm}$ and then use the D_s track to fit the decay vertex

$B_s \to D_s^- \pi^+$ or charge coniugate. A mass constraint is applied to the D_s vertex fit to improve resolution. A description of the algorithm and how the vertexing code is used is shown.

```
// Find Ds Vertex with mass constraint
// Load Ds tracks
TVectorD* tDsPar[nDsT];
TMatrixDSym* tDsCov[nDsT];
for(Int_t k=0; k<nDsT; k++){
   TVectorD par(5); TMatrixDSym cov(5);
   TrkToVector(tDs[k], par, cov);
   tDsPar[k] = new TVectorD(par);
   tDsCov[k] = new TMatrixDSym(cov);
}
// Fit Ds vertex
VertexFit* vDs = new VertexFit(nDsT, tDsPar, tDsCov);
Double_t DsChi2 = vDs->GetVtxChi2(); // Ds fit Chi2
// More fitting
Bool_t Units = kTRUE; // Set to mm
VertexMore* VMDs = new VertexMore(vDs,Units);
// Mass constraint if requested
Bool_t MCst = kTRUE; // Mass constraint flag
if(MCst){
   Double_t DsMass = pBsDs->Mass; // Ds mass
   Double_t DsMasses[nDsT]; Int_t DsList[nDsT];
   for(Int_t k=0; k<nDsT; k++){
       DsMasses[k] = pDs[k]->Mass;
       DsList[k] = k;
   }
   VMDs->AddMassConstraint(DsMass, nDsT, DsMasses, DsList);
   VMDs->MassConstrFit();
}
TVectorD rDv = VMDs->GetXv(); // Ds vertex
```

Then use the D_s track and the remaining π to fit the B_s vertex.

```
// Find Bs vertex
// Load Bs tracks
TVectorD* tBsPar[nBsT];
TMatrixDSym* tBsCov[nBsT];
TVectorD par(5); TMatrixDSym cov(5);
TrkToVector(tBs[0], par, cov);
tBsPar[0] = new TVectorD(par); // Bs pion
tBsCov[0] = new TMatrixDSym(cov);
tBsPar[1] = new TVectorD(VMDs->GetVpar()); // Ds from previous fit
```

```
tBsCov[1] = new TMatrixDSym(VMDs->GetVcov());
//
// Fit Bs vertex
VertexFit* vBs = new VertexFit(nBsT, tBsPar, tBsCov); // Bs vertex
Double_t BsChi2 = vBs->GetVtxChi2();
```

One could also use `VertexMore` for the B_s vertex to extract more information or even apply an additional mass constraint on the B_s if useful.

4.3 $B_0 \to K_S^0 K_S^0$

The code for this example is located in `examples/VtxB02KsKs.C` and is executed from `ROOT` by :
`root>.L goKsKs.C+`
`root>goKsKs();`
`root>.X examples/VtxB02KsKs.C("InputFile.root", 1000);`, where the second parameter is the number of events to process. The goal of this example is to first fit the decays $K_s^0 \to \pi^+\pi^-$ and then use the K_s^0 tracks to fit the decay vertex $B_0 \to K_s^0\, K_s^0$ or charge coniugate. A mass constraint is applied to the K_s^0 vertices fits to improve resolution. A description of the algorithm and how the vertexing code is used is shown.

```
// Find first Ks vertex
// Load 1st Ks tracks
TVectorD* tKs1Par[nKsT];
TMatrixDSym* tKs1Cov[nKsT];
for(Int_t k=0; k<nKsT; k++){
   TVectorD par(5); TMatrixDSym cov(5);
   TrkToVector(tKs1[k], par, cov);
   tKs1Par[k] = new TVectorD(par);
   tKs1Cov[k] = new TMatrixDSym(cov);
}
// Fit 1st Ks vertex
VertexFit* vKs1 = new VertexFit(nKsT, tKs1Par, tKs1Cov);
Double_t Ks1Chi2 = vKs1->GetVtxChi2(); // Ks fit Chi2
// More fitting
Bool_t Units = kTRUE; // Set to mm
VertexMore* VMKs1 = new VertexMore(vKs1,Units);
// Mass constraint if requested
Bool_t MCst = kTRUE; // Mass constraint flag
if(MCst){
   Double_t KsMass = pB0[0]->Mass; // Ks mass
   Double_t KsMasses[nKsT]; Int_t KsList[nKsT];
   for(Int_t k=0; k<nKsT; k++){
       KsMasses[k] = pKs1[k]->Mass;
       KsList[k] = k;
   }
```

```
   VMKs1->AddMassConstraint(KsMass, nKsT, KsMasses, KsList);
   VMKs1->MassConstrFit();
}
```

A similar code is used to fit the second K_s^0 in the event. The B_0 vertex is then fit using the two neutral tracks:

```
// Find B0 vertex
// Load B0 tracks
TVectorD* tB0Par[nB0T];
TMatrixDSym* tB0Cov[nB0T];
tB0Par[0] = new TVectorD(VMKs1->GetVpar()); // 1st Ks previous fit
tB0Cov[0] = new TMatrixDSym(VMKs1->GetVcov());
tB0Par[1] = new TVectorD(VMKs2->GetVpar()); // 2nd Ks previous fit
tB0Cov[1] = new TMatrixDSym(VMKs2->GetVcov());
//
// Fit B0 vertex
Bool_t Charged[nB0T] = kFALSE, kFALSE; // Tag neutral tracks
VertexFit* vB0 = new VertexFit(nB0T, tB0Par, tB0Cov, Charged);
Double_t B0Chi2 = vB0->GetVtxChi2();
// More fit information
VertexMore* VMB0 = new VertexMore(vB0,Units);
TVectorD rvmB0 = VMB0->GetXv();
```

As in the previous case one could further improve the resolution by mass constraining the B_0 if useful.

Appendix A Steered parameter fit derivation

Differentiating the χ^2 shown in eq. 8 with respect to the track parameters and using the expansion of the track positions at the points $(s'_i, \vec{\alpha}'_i)$ we obtain:

$$\frac{1}{2}\frac{\partial \chi^2}{\partial \vec{\alpha}_i} = C_i^{-1}(\vec{\alpha}_i - \vec{\alpha}_i^{\,0}) + A_i^t \vec{\lambda}_i = C_i^{-1}(\delta\vec{\alpha}_i - \delta\vec{\alpha}_i^{\,0}) + A_i^t \vec{\lambda}_i = 0$$

where $\delta\vec{\alpha}_i = \vec{\alpha}_i - \vec{\alpha}'_i$ and $\delta\vec{\alpha}_i^{\,0} = \vec{\alpha}_i^{\,0} - \vec{\alpha}'_i$. The following derived relations will be used later:

$$\vec{\alpha}_i = \vec{\alpha}_i^{\,0} - C_i A_i^t \vec{\lambda}_i$$

and after mltiplication by $A_i\, C_i$:

$$A_i\, \delta\vec{\alpha}_i = A_i\, \delta\vec{\alpha}_i^{\,0} - (A_i C_i A_i^t)\vec{\lambda} = A_i\, \delta\vec{\alpha}_i^{\,0} - W_i^{-1}\vec{\lambda}$$

Differentiating with respect to the Lagrange multipliers returns the constraints:

$$0 = \vec{x}_i - \vec{x}_V = \vec{x}_i^{\,0} - \vec{x}_V + A_i\delta\vec{\alpha}_i + \vec{a}_i\delta\, s_i = \vec{x}_i^{\,0} + A_i\, \delta\vec{\alpha}_i^{\,0} - \vec{x}_V - W_i^{-1}\vec{\lambda}_i + \vec{a}_i\delta\, s_i$$

so:

$$\vec{\lambda}_i = W_i\{\vec{x}_i^{\,0} + A_i\, \delta\vec{\alpha}_i^{\,0} - \vec{x}_V + \vec{a}_i\delta s_i\}$$

Differentiating by s_i we can eliminate the dependence on δs_i in the above:

$$\frac{1}{2}\frac{\partial \chi^2}{\partial\, s_i} = \vec{a}_i^t\vec{\lambda}_i = 0$$

so replacing $\vec{\lambda}_i$ from the above:

$$\vec{a}_i^t\vec{\lambda}_i = \vec{a}_i^t W_i\{\vec{x}_i^{\,0} + A_i\, \delta\vec{\alpha}_i^{\,0} - \vec{x}_V + \vec{a}_i\delta\, s_i\} = 0$$

leading to:

$$\delta s_i = \vec{a}_i^t W_i \frac{\vec{x}_V - \vec{x}_i^{\,0} - A_i\, \delta\vec{\alpha}_i^{\,0}}{a_i}$$

where we have set $a_i = \vec{a}_i^t W_i \vec{a}_i$. Finally:

$$\begin{aligned}
\vec{\lambda}_i &= W_i\{\vec{x}_i^{\,0} + A_i\, \delta\vec{\alpha}_i^{\,0} - \vec{x}_V + \frac{\vec{a}_i\vec{a}_i^t}{a_i} W_i(\vec{x}_V - \vec{x}_i^{\,0} - A_i\, \delta\vec{\alpha}_i^{\,0})\} \\
&= \left(W_i - W_i\frac{\vec{a}_i\vec{a}_i^t}{a_i}W_i\right)(\vec{x}_i^{\,0} + A_i\, \delta\vec{\alpha}_i^{\,0} - \vec{x}_V) \\
&= D_i(\vec{x}_i^{\,0} + A_i\, \delta\vec{\alpha}_i^{\,0} - \vec{x}_V) \qquad \text{(A1)}
\end{aligned}$$

We now differentiate by $\vec{x}_V$ obtaining:

$$\frac{1}{2}\frac{\partial \chi^2}{\partial\, \vec{x}_V} = -\sum_{i=1}^{N}\vec{\lambda}_i = \sum_{i=1}^{N} D_i(\vec{x}_V - \vec{x}_i^{\,0} - A_i\, \delta\vec{\alpha}_i^{\,0}) = 0$$

that can be solved for $\vec{x}_V$ giving the solution:

$$\vec{x}_V = \left(\sum_{i=1}^{N} D_i\right)^{-1} \left(\sum_{i=1}^{N} D_i(\vec{x}_i^{\,0} + A_i\,\delta\vec{\alpha}_i^{\,0})\right) = D^{-1}\left(\sum_{i=1}^{N} D_i(\vec{x}_i^{\,0} + A_i\,\delta\vec{\alpha}_i^{\,0})\right) \quad \text{(A2)}$$

The corresponding error matrix on $\vec{x}_V$ is obtained by taking the average of the fluctuations of $\vec{x}_V$ due to those of the track parameters, $\vec{\alpha}_i^{\,0}$:

$$\begin{aligned} C_V = Cov(\vec{x}_V) &= D^{-1}\left(\textstyle\sum_{ij} D_i A_i < \delta\vec{\alpha}_i^{\,0}\delta\vec{\alpha}_j^{\,0t} > A_i^t D_j\right) D^{-1} \\ &= D^{-1}\left(\textstyle\sum_i D_i W_i^{-1} D_i\right) D^{-1} \end{aligned} \quad \text{(A3)}$$

The fitting procedure starts setting $\vec{\alpha}' = \vec{\alpha}^{\,0}$ and s' to some smart guess of its value. After each iteration $\vec{\alpha}'$ is set to the latest value of $\vec{\alpha}$ obtained, given by:

$$\vec{\alpha}_i = \vec{\alpha}_i^{\,0} - C_i A_i^t \vec{\lambda}_i$$

and s_i' to the latest value of s_i obtained given by:

$$s_i = s_i' + \vec{a}_i^t W_i \frac{\vec{x}_V - \vec{x}_i^{\,0} - A_i\,\delta\vec{\alpha}_i^{\,0}}{a_i}$$

Appendix B Charged track formulas

B.1 Trajectory formulas

In the point of minimum approach to the z-axis the 3D position and momentum are given by:

$$\begin{cases} x = x_0 + [sin(s+\varphi_0) - sin(\varphi_0)]/(2C) \\ y = y_0 - [cos(s+\varphi_0) - cos(\varphi_0)]/(2C) \\ z = z_0 + \lambda\, s/(2C) \end{cases} \quad \text{(B4)}$$

and

$$\begin{cases} p_x = p_t\, cos(s+\varphi_0) \\ p_y = p_t\, sin(s+\varphi_0) \\ p_z = p_t \lambda \end{cases} \quad \text{(B5)}$$

where

$$\begin{cases} x_0 = -D\, sin\,\varphi_0 \\ y_0 = D\, cos\,\varphi_0 \\ z_0 = z_0 \end{cases} \quad \begin{cases} a = -0.2998BQ\ (T/m/GeV) \\ \rho = 2C = a/p_t \end{cases} \quad \text{(B6)}$$

and s is the angular displacement from the point of minimum approach to the z axis (pma), D is the signed transverse impact parameter, φ_0 the track direction at the pma, z_0 the z coordinate at the pma, and $\lambda = p_z/p_t$ or the cotangent of the polar angle. The phase, s, at point $\vec{x}$ on the trajectory is given by:

$$s = sin^{-1}\{2C(x\, cos\,\varphi_0 + y\, sin\,\varphi_0)\} \quad \text{(B7)}$$

In the following we show the derivatives of the track trajectory, $\vec{x}(\vec{\alpha}, s)$, with respect to the track parameters, $\vec{\alpha} = (D,\, \varphi_0,\, C,\, z_0,\, \lambda)$, and the phase, s. They are shown in the following:

$$\vec{a} = \frac{\partial \vec{x}}{\partial s} = \frac{1}{2C} \begin{pmatrix} cos(s+\varphi_0) \\ sin(s+\varphi_0) \\ \lambda \end{pmatrix};\;\; a = \vec{a}^{\,t}\,\vec{a} = \frac{1+\lambda^2}{4C^2} \tag{B8}$$

$$A = \frac{\partial \vec{x}}{\partial \vec{\alpha}} = \begin{pmatrix} -sin\,\varphi_0 & -D\,cos\,\varphi_0 + \frac{[cos(s+\varphi_0)-cos\,\varphi_0]}{2C} & -\frac{[sin(s+\varphi_0)-sin\,\varphi_0]}{2C^2} & 0 & 0 \\ cos\,\varphi_0 & -D\,sin\,\varphi_0 + \frac{[sin(s+\varphi_0)-sin\,\varphi_0]}{2C} & \frac{[cos(s+\varphi_0)-cos\,\varphi_0]}{2C^2} & 0 & 0 \\ 0 & 0 & -\lambda\, s/(2C^2) & 1 & s/(2C) \end{pmatrix} \tag{B9}$$

Similarly for the track momentum:

$$\frac{\partial \vec{p}}{\partial \vec{\alpha}} = \begin{pmatrix} 0 & -p_y & -p_x/C & 0 & 0 \\ 0 & p_x & -p_y/C & 0 & 0 \\ 0 & 0 & -p_z/C & 0 & p_t \end{pmatrix} \tag{B10}$$

and

$$\frac{\partial \vec{p}}{\partial s} = (-p_y,\, p_x,\, 0) \tag{B11}$$

B.2 Track parameters from $\vec{x}$, $\vec{p}$

We provide the basic formulas to obtain the track parameters in cylindrical geometry given a point on the track, $\vec{x}$, and the momentum, $\vec{p}$, in that point. Two of the parameters, λ and C, depend only on the momentum:

$$\lambda = p_z/p_\perp;\;\; C = a/(2p_\perp) \tag{B12}$$

The transverse impact parameter, D, is given by:

$$D = \frac{1}{a}(T - p_\perp);\; T = \sqrt{p_\perp^2 - 2a(xp_y - yp_x) + a^2(x^2+y^2)} \tag{B13}$$

The angle φ_0 is given by:

$$cos\,\varphi_0 = \frac{p_x + ay}{T};\; sin\,\varphi_0 = \frac{p_y - ax}{T} \tag{B14}$$

or

$$tan\,\varphi_0 = \frac{p_y - ax}{p_x + ay} \tag{B15}$$

Finally z_0:

$$z_0 = z - \lambda s/2C \tag{B16}$$

The derivatives of the track parameters relative to both $\vec{x}$ and $\vec{p}$ are reported in the following.

$$\frac{\partial\lambda}{\partial\vec{x}} = (0,\,0,\,0);\;\frac{\partial\lambda}{\partial\vec{p}} = \left(-\frac{p_x p_z}{p_\perp^3}, -\frac{p_y p_z}{p_\perp^3}, \frac{1}{p_\perp}\right) \tag{B17}$$

$$\frac{\partial C}{\partial\vec{x}} = (0,\,0,\,0);\;\frac{\partial C}{\partial\vec{p}} = \frac{a}{2}\left(-\frac{p_x}{p_\perp^3}, -\frac{p_y}{p_\perp^3}, 0\right) \tag{B18}$$

$$\frac{\partial T}{\partial\vec{x}} = \left(-\frac{a}{T}(p_y - ax),\,\frac{a}{T}(p_x + ay),\,0\right);\;\frac{\partial T}{\partial\vec{p}} = \left(\frac{p_x + ay}{T},\,\frac{p_y - ax}{T},\,0\right) \tag{B19}$$

$$\frac{\partial D}{\partial\vec{x}} = \frac{1}{a}\frac{\partial T}{\partial\vec{x}};\;\frac{\partial D}{\partial\vec{p}} = \left(\frac{1}{a}\left(\frac{\partial T}{\partial p_x} - \frac{p_x}{p_\perp}\right),\,\frac{1}{a}\left(\frac{\partial T}{\partial p_y} - \frac{p_y}{p_\perp}\right),\,0\right) \tag{B20}$$

$$\frac{\partial\varphi_0}{\partial\vec{x}} = -\frac{a\,cos^2\varphi_0}{p_x + ay}\;(1,\,tan\,\varphi_0,\,0) \tag{B21}$$

$$\frac{\partial\varphi_0}{\partial\vec{p}} = \frac{cos^2\varphi_0}{p_x + ay}\;(-tan\,\varphi_0,\,1,\,0) \tag{B22}$$

The derivative of z_0 is calculated after noting that an alternate expression for the phase, s, is given by:

$$s = tg^{-1}\frac{p_y}{p_x} - \varphi_0 \tag{B23}$$

then:

$$\frac{\partial z_0}{\partial\vec{x}} = \left(\frac{\lambda}{2C}\frac{\partial\varphi_0}{\partial x},\,\frac{\lambda}{2C}\frac{\partial\varphi_0}{\partial y},\,1\right) = \left(\frac{p_z}{a}\frac{\partial\varphi_0}{\partial x},\,\frac{p_z}{a}\frac{\partial\varphi_0}{\partial y},\,1\right) \tag{B24}$$

$$\frac{\partial z_0}{\partial\vec{p}} = \left(\frac{p_z}{a}\left(\frac{p_y}{p_t^2} + \frac{\partial\varphi_0}{\partial p_x}\right),\,\frac{p_z}{a}\left(-\frac{p_x}{p_t^2} + \frac{\partial\varphi_0}{\partial p_y}\right),\,-s/a\right) \tag{B25}$$

Appendix C Neutral track formulas

The track trajectory equation for neutral tracks is a straight line. For consistency with the previous parameterization we find the trajectory equation by setting $C = 0$ in the charged track equations obtaining:

$$\begin{cases} x &= x_0 + s\,cos\varphi_0 \\ y &= y_0 + s\,sin\varphi_0 \\ z &= z_0 + s\,\lambda \end{cases} \quad \text{where} \quad \begin{cases} x_0 &= -D\,sin\varphi_0 \\ y_0 &= D\,cos\varphi_0 \\ s &= \sqrt{R^2 - D^2} \end{cases} \tag{C26}$$

we note that in this case the parameter s is not angle, but the distance from the pma, and is given by:

$$s = x\,cos\,\varphi_0 + y\,sin\,\varphi_0 \tag{C27}$$

Alternatively, as a function of the radius, R, the neutral track equation can be writtten as:

$$\begin{cases} \varphi(R) &= \varphi_0 + sin^{-1}(D/R) \\ z(R) &= z_0 + \lambda\sqrt{R^2 - D^2} \end{cases} \tag{C28}$$

There are only 4 parameters to define the trajectory, however we include also a 5^{th} parameter, p_t, to keep track of the neutral track momentum. The final set of

parameters describing a neutral is threrefore $\vec{\alpha} = (D,\, \varphi_0,\, p_t,\, z_0,\, \lambda)$. The derivatives of the trajectory with respect to the track parameters is:

$$\frac{\partial \vec{x}}{\partial \vec{\alpha}} = \begin{pmatrix} -sin\varphi_0 & -D\,cos\varphi_0 - s\,sin\varphi_0 & 0 & 0 & 0 \\ cos\varphi_0 & -D\,sin\varphi_0 + s\,cos\varphi_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s \end{pmatrix} = \begin{pmatrix} -sin\varphi_0 & -y & 0 & 0 & 0 \\ cos\varphi_0 & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s \end{pmatrix} \tag{C29}$$

Derivatives of the trajectory with respect to s are given by:

$$\frac{\partial \vec{x}}{\partial\, s} = (cos\varphi_0,\, sin\varphi_0,\, \lambda) \tag{C30}$$

Neutral track momenta are constant and given by:

$$\vec{p} = (p_t\, cos\,\varphi_0,\, p_t\, sin\,\varphi_0,\, p_t\lambda) \tag{C31}$$

with trivial derivatives with respect to track parameters:

$$\frac{\partial \vec{p}}{\partial \vec{\alpha}} = \begin{pmatrix} 0 & -p_y & cos\varphi_0 & 0 & 0 \\ 0 & p_x & sin\varphi_0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & p_t \end{pmatrix} \tag{C32}$$

C.1 Neutral track parameters from $\vec{x}$ and $\vec{p}$

The track direction is easily obtained from the momentum:

$$\begin{cases} cos\varphi_0 = p_x/p_t \\ sin\varphi_0 = p_y/p_t \quad \text{where } p_t = \sqrt{p_x^2 + p_y^2} \\ \lambda = p_z/p_t \end{cases} \tag{C33}$$

The parameters D and z_0 can be obtained by eliminating s in the trajectory equation:

$$D = y\,cos\varphi_0 - x\,sin\varphi_0,\ s = y\,sin\varphi_0 + x\,cos\varphi_0,\ z_0 = z - \lambda\, s \tag{C34}$$

Derivatives of the phase, s, with respect to $\vec{x}$:

$$\frac{\partial s}{\partial \vec{x}} = (cos\varphi_0,\, sin\varphi_0,\, 0) \tag{C35}$$

and with respect to the momentum, $\vec{p}$:

$$\frac{\partial s}{\partial \vec{p}} = \left(\frac{-D\,sin\varphi_0}{p_t},\, \frac{D\,cos\varphi_0}{p_t},\, 0 \right) \tag{C36}$$

Derivatives of the track parameters with respect to the starting point $\vec{x}$ and the momentum $\vec{p}$ are given by:

$$\frac{\partial \vec{\alpha}}{\partial \vec{x}} = \begin{pmatrix} -sin\varphi_0 & 0 & 0 & -\lambda\, cos\varphi_0 & 0 \\ cos\varphi_0 & 0 & 0 & -\lambda\, sin\varphi_0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{C37}$$

and

$$\frac{\partial \vec{\alpha}}{\partial \vec{p}} = \begin{pmatrix} \frac{s\, sin\varphi_0}{p_t} & -\frac{sin\varphi_0}{p_t} & cos\varphi_0 & \frac{\lambda}{p_t}(s \cdot cos\varphi_0 + D\, sin\varphi_0) & -\frac{\lambda}{p_t}\, cos\varphi_0 \\ \frac{-s\, cos\varphi_0}{p_t} & \frac{cos\varphi_0}{p_t} & sin\varphi_0 & \frac{\lambda}{p_t}(s \cdot sin\varphi_0 - D\, cos\varphi_0) & -\frac{\lambda}{p_t}\, sin\varphi_0 \\ 0 & 0 & 0 & -s/p_t & 1/p_t \end{pmatrix} \tag{C38}$$