The Fit Algorithm in muo_trackreco

Frédéric Déliot

21 novembre 2000

Résumé

The goal of the fit algorithm implemented in the *muo_trackreco* package is to identify and determine the momentum of muon tracks from couples of segments (two segments respectively in A and B-C layers) in the muon system using a fit. This algorithm takes into account: magnetic field, energy loss and multiple scattering in the iron toroid. The three main classes implemented for the algorithm are *MuoTrackAlgFit*, *Estimation* and *SegmentFit*.

1 MuoTrackAlgFit:

The *MuoTrackAlgFit* class is the basic class of the fit algorithm. It inherits from *MuoTrackAlg* and implements its virtual method *findTracks*. An instance of *MuoTrackAlgFit* is constructed in *MuoTrackReco* if the parameter $"TrackAlgorithm"$ string is set to $"Fit"$ in *MuoTrackReco.rcp* (see section 4).

There are three steps in this algorithm. First the segments are grouped into pair. Then different cuts are applied on the segment pairs and an estimation of the track momentum is computed. Finally the good pairs are fitted to find muon tracks.

1.1 Group Segment:

void groupSegments(const MuoSegmentVector& segments,SegmentPairVector& pairVector);

This method produces a segment pair vector from a segment vector. First an index is given to each segment of the input segment vector by creating a list of *MuoTrackSegment* objects. Then the *MuoTrackSegment* objects of this list are combined into pairs: the first element of the pair is a A layer segment and the second one a BC layers segment. A segment is associated with a BC segment if both segments are in the same region (barrel or endcaps) and in the same but adjacent octants. For instance, A segment in octant 0 with $Arctan(\frac{y_A}{x_A}) \leq \frac{\pi}{8}$ is associated with BC segment in octant 0 or 1. If $Arctan(\frac{y_A}{x_A}) > \frac{\pi}{8}$, this A segment is coupled with BC segment in octant 0 or 7.

1.2 Cuts on Segment Pair, First Estimation of Momentum:

We make five successive cuts to verify that the two segments of the pair are compatible.

First we project the A segment on the deviation plane (either (z,x) or (z,y) according to the octant) using the *projectInDev* method (see section 1.2.1). In this plane, the angle between the z axis and the projected segment direction is named deviation angle. Then we use the projected deviation angle error in *coneInterceptVertex* (see section 1.2.2) to define a cone around the A segment direction and see if it intercepts the vertex (only when the A segment has more that one hit).

Secondly we calculate the phi angles related to the segment positions:

$$\phi_A = atan2(-ya, -xa) + \pi \text{ and } \phi_B = atan2(-yb, -xb) + \pi. \tag{1}$$

We verify that $|\phi_A - \phi_B| < conePhi$ where $conePhi$ is a value set in the RCP. Then we look if the A segment position is a cone around B segment direction using $AinConeB$ (see section 1.2.3).

At this step, a first estimation of the track momentum is computed. This calculation is implemented in the *Estimation* class (see section 2).

The fourth cut consists of verifying that the deviation angle difference between A and BC segments is compatible with the estimated bending angle computed in the *Estimation* class (see section 2.1).

Next we control that the estimated momentum at A layer in the deviation plane is greater than $ptAmin$ fixed in the RCP file .

Finally if the A segment has only one hit, we calculate its deviation angle using the estimated bending angle and verify that this direction in compatible with the vertex position using *coneInterceptVertex.*

1.2.1 projectInDev method:

void projectInDev(int octant, SpacePoint point, Direction dir, SpacePoint& ptproj, double& angleDev, double& sigmaAngleDev);

This method projects a given point (x, y, z) and a given spherical direction (θ, ϕ) in the deviation plane.

The inputs are the octant, a position, a direction and errors on direction σ_θ and σ_ϕ. The ouputs are the projected point, the deviation angle $angle_{Dev}$ and the error on this angle $\sigma_{angleDev}$.

The deviation angle is definied as:

$$angle_{Dev} = atan2(sin(theta) * sin(phi), cos(theta)) \tag{2}$$
$$angle_{Dev} = atan2(sin(theta) * cos(phi), cos(theta)) \tag{3}$$

(2) stands for octants 1,2,5,6 and (3) for octants 0,7,3 or 4. The deviation angle error is computed as:

$$\frac{\sigma^2_{angleDev}}{cos^4(angle_{Dev})} = \sigma^2_\theta \frac{sin^2\phi}{cos^4\theta} + \sigma^2_\phi tan^2\theta cos^2\phi \tag{4}$$
$$\frac{\sigma^2_{angleDev}}{cos^4(angle_{Dev})} = \sigma^2_\theta \frac{cos^2\phi}{cos^4\theta} + \sigma^2_\phi tan^2\theta sin^2\phi \tag{5}$$

(4) stands for octants 1,2,5,6 and (5) for octants 0,7,3,4.

1.2.2 coneInterceptVertex method:

bool coneInterceptVertex(const SpacePoint pointProj, double angleDev, double errorAngleDev);

This method runs in the deviation plane. The *coneInterceptVertex* method looks if a cone with the A segment position as origin intercepts a beam axis region compatible with the vertex extension.

The cone angle $angle_{cone}$ is computed as $angle_{cone} = 3 * \sigma_{thetaA} + \sigma_{MS}$ where σ_{thetaA} is the A segment direction error in the deviation plane and σ_{MS} an estimation of the direction error due to multiple scattering in the calorimeter (in the deviation plane). σ_{MS} is an RCP parameter. The vertex position is assumed to be along the beam axis (z axis) between $-maxVertex$ and $+maxVertex$. $maxVertex$ is set by RCP.

In the deviation plane, the cone is defined by two vectors $\overrightarrow{cone1}$ and $\overrightarrow{cone2}$. The vector $\overrightarrow{cone1}$ makes an angle $-angle_{cone}$ with respect to the A segment direction and $\overrightarrow{cone2}$ an angle $+angle_{cone}$. The point $V1(0, 0, maxVertex)$ is in the cone if the components of $\overrightarrow{VA1} \times \overrightarrow{cone1}$ and $\overrightarrow{VA1} \times \overrightarrow{cone2}$ orthogonal to the plane have different signs. The same calculation is made with the point $V2(0, 0, -maxVertex)$.

1.2.3 AinConeB method:

bool AinConeB(int octant, const SpacePoint pointProjA, const SpacePoint pointB, const Direction dirB);

This method runs in the deviation plane. It looks if the A segment position is contained in a cone with B segment position as origin and B segment direction as axis.

The projected B position and direction are obtained with the *projectInDev* method. The angle of the cone is given by $coneTheta$ set in RCP. We define the cone vector limits $\overrightarrow{cone1}$ which makes an angle $-coneTheta$ with the projected B segment direction and $\overrightarrow{cone2}$ which makes an angle $+coneTheta$. The $\overrightarrow{AB}$ vector joins the projected A segment position with the projected B segment position. To see if $\overrightarrow{AB}$ is between $\overrightarrow{cone1}$ and $\overrightarrow{cone2}$, we compute the same cross products as in *coneInterceptVertex*. Hence, if the components of $\overrightarrow{AB} \times \overrightarrow{cone1}$ and $\overrightarrow{AB} \times \overrightarrow{cone2}$ orthogonal to the plane have different signs, A is in the cone defined by B.

1.3 Fit and Output: MuoTrack:

If the pair passes all the cuts, it is fitted by the *SegmentFit* class (see section 3). The track position and momentum at A and BC layers are stored in a vector of *MuoTrack* objects. A *MuoTrack* object constructed by the fit algorithm has the following attributs (the Combinatorial algorithm uses a different *MuoTrack* constructor):

- a *MuoSegment* constructed from A segment position and direction but with all the hit indices from A and BC segments.
- a 2 components vector of indices refering to the A and BC segments used to find the track (see section 1.1).
- the track position at layer A result from the fit (*SpacePoint*)

- the track momentum at layer A result from the fit (*SpaceVector*)
- the track position at layer B result from the fit (*SpacePoint*)
- the track momentum at layer B result from the fit (*SpaceVector*)
- the particule charge result from the fit
- the track position at layer A computed in the first estimation (*SpacePoint*)
- the track momentum at layer A computed in the first estimation (*SpaceVector*)
- the track position at layer B computed in the first estimation (*SpacePoint*)
- the track momentum at layer B computed in the first estimation (*SpaceVector*)
- the particule charge computed in the first estimation
- the χ^2 of the fit
- a 21-component vector filled with the (6×6) symmetric error matrix values result from the fit
- the multiple scattering angle result from the fit at the first fixed multiple scattering plane in the toroid
- the multiple scattering angle result from the fit at the second fixed multiple scattering plane in the toroid
- the fit iteration number

2 Estimation:

The aim of the *Estimation* class is to compute a first quick estimation of the track momentum from geometrical calculation. This estimation is used as input for the fit. The calculation is performed in local coordinate in the deviation plane. For octants 0,7,3 or 4, the deviation plane is the (z, x) plane in global coordinates. For octants 1,2,5 or 5, it is the (z, y) plane. By a frame rotation depending of octants, the deviation plane is set to (x, y), x orthogonal the wire plane and y in the wire plane. The calculation is implemented in the *firstEstimate* method.

2.1 firstEstimate method:

void firstEstimate(const Toroid localtor,const TrackParameters & localsegA, const TrackParameters& localsegB);*

This method computes the momentum estimation using segmentA and segment BC position and direction in local coordinates transformed into $TrackParameters$ objects. It uses a simple toroid geometry set in the *Toroid* class.

It fills two $TrackParameters$ outputs: $\_trackA$ and $\_trackB$, attributs of the class. This method is called by the *computeEstimation* method of *SegmentFit* class (see section).

The deviation angles θ_A or θ_B are angles from x axis to projected segment direction in the plane (x, y). If A segment has only one hit, this angle is set to: $atan2(-ya, -xa) + \pi$. First one finds point A and point B, straight line extrapolation of segment A and segment B to the toroid planes (see figure 1). If *begin_toro* and *end_toro* are the toroid planes x

coordinates, the vector $\overrightarrow{AB}$ components are:

$$BAx = begin\_toro - end\_toro \quad (6)$$
$$BAy = y_A + (begin\_toro - x_A) * tan\theta_A - y_B - (end\_toro - x_B) * tan\theta_B \quad (7)$$

where (x_A, y_A) and (x_B, y_B) are the coordinates of segment A and segment B center of gravity positions.

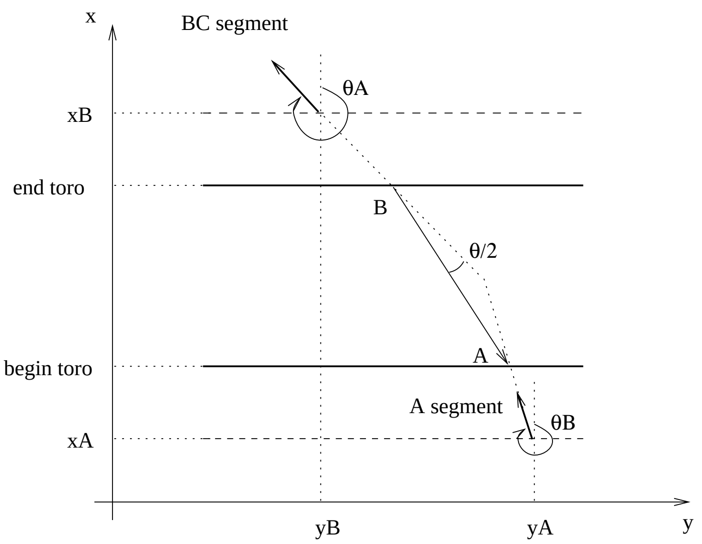

FIG. 1 – *first momentum estimation by calculating angle θ*

BA, norm of $\overrightarrow{BA}$, provides an estimation of the particule path in the toroid. We define $\overrightarrow{segB}(cos(\theta_B), sin(\theta_B))$, and the angle θ between the projected B segment direction and $\overrightarrow{BA}$:

$$sin\frac{\theta}{2} = \frac{1}{BA}\overrightarrow{segB} \times \overrightarrow{BA} \quad (8)$$
$$= \frac{1}{BA}(cos\theta_B BA_y - sin\theta_B BA_x) \quad (9)$$
$$\theta = 2\pi - 2Arcsin(sin\frac{\theta}{2})\ [2\pi] \quad (10)$$

The sign of θ gives the particule charge. θ provides also the possibility to compute θ_A from θ_B:

$$\theta_A = \theta_B + \theta\ [\pi] \quad (11)$$

We assume a constant energy loss in the toroid set to $\frac{dE}{dx} = 0.016GeV/cm$ and a constant magnetic field $B_f = 1.8T$. Then a momentum estimation in the plane is:

$$P = \frac{0.3 \times B_f \times BA}{|tan\theta|} \quad (12)$$

with BA in meter, B_f in Tesla and P in GeV. At layer A and BC, a momentum estimation in the deviation plane is:

$$P_A = P - \frac{dE}{dx}\frac{BA}{2}$$
$$P_B = P + \frac{dE}{dx}\frac{BA}{2}$$

So the A layer momentum components are:

$$\begin{aligned} P_{Ax} &= P_A \cos\theta_A \\ P_{Ay} &= P_A \sin\theta_A \end{aligned}$$

We have the same kind of equation for the B layer momentum components with P_B and θ_B. The momentum component along the wire (z axis in the local frame) is computed by using the ϕ global angle of the segment.

The error on the estimate angle θ derives from (9) and (7):

$$\begin{aligned} d(\sin\frac{\theta}{2}) &= \frac{1}{AB}(-\sin\theta_B BA_y d\theta_B + \cos\theta_B dBA_y - \cos\theta_B BA_x d\theta_B) & (13) \\ dBA_y &= dy_A - dy_B + (begin\_toro - x_A)\frac{d\theta_A}{\cos^2\theta_A} - (end\_toro - x_B)\frac{d\theta_B}{\cos^2\theta_B} & (14) \\ & +tan\theta_B dx_B - tan\theta_A dx_A & (15) \end{aligned}$$

Then:

$$\sigma_\theta^2 = \left(\frac{2\cos\theta_B}{BA\cos\frac{\theta}{2}}\right)^2 \left(\sigma_{\theta_B}^2(tan\theta_B BA_y + BAx + \frac{end\_toro - x_B}{\cos^2\theta_B}) + \sigma_{\theta_A}^2\frac{begin\_toro - x_A}{\cos^2\theta_A} + \sigma_{y_A}^2 + \sigma_{y_B}^2\right) \quad (16)$$

The fourth cut in *MuoTrackAlgFit* (see section 1.2) relies on (10) and on σ_θ calculation. Indeed it compares $tan\theta$ and $tan(\theta_A - \theta_B)$ and requires that

$$|tan\theta - tan(\theta_A - \theta_B)| < sigmaThmax * \sigma_{tan\theta}$$

where $sigmaThmax$ is fixed by RCP.

3 SegmentFit:

The *SegmentFit* class provides two main methods for computing the first estimation and the fit algorithm.

3.1 momentum estimation:

void computeEstimation();

This method first makes a coordinates change from global to local frame via the *GlobalToLocal* method and redefines the spherical ϕ of the global A and BC segments (*phiRedefinition* method):

$$\phi = atan2(-yA, -xA) + \pi.$$

It also redefines the segment positions along the wire:

$$\begin{aligned} x_B &= x_A + (y_B - y_A)/tan(\phi) \text{ for octants 1,2,5 and 6} \\ y_B &= y_A + (x_B - x_A) * tan(\phi) \text{ for octants 0,7,3 and 4} \end{aligned}$$

Then one calls *firstEstimate* to perform the momentum estimation (see section 2.1).

3.2 Fit:

void fit(const Estimation& firstEstimation, TrackParameters& trackAOut, TrackParameters& trackBOut, std::vector<double> & errorOut);

This method creates an object of *NonLinearFitter* class and calls its *doFit* method with the first estimation as input for the fit. It computes the track postions and directions at A and BC layers and the error matrix at BC layer. At the end one transforms the (5×5) error matrix to a (6×6) error matrix in (x, y, z, px, py, pz) global variables (see section 3.2.2).

3.2.1 the NonLinearFit class:

This class performs a non linear fit to find the best momentum compatible with segment A and segment BC positions and directions in local frame.

The track is propagated step by step from the center of gravity of BC segment to the one of A segment using circular helices in space in the *BackTrack* class. At each step, an energy loss correction is performed using a function giving $\frac{dE}{dx}$ as a function of momentum. *BackTrack* can either using a constant magnetic field in the toroid or a real mapping. This is controled by the RCP parameter: "$WhichField$".

This propagation allows to perform a 2D fit in the deviation plane ((x, y) in local coordinates) using matrix formalism with 5 parameters:

- the B layer track position along y-axis at fixed x_B (the segment B center of gravity position along x-axis)
- $c = \frac{q}{P}$ where P is the track momentum at fixed x_B in the deviation plane and q the particule sign.
- the angle θ in the deviation plane between x-axis and the momentum direction $(0 \leq \theta \leq 2\pi)$
- two multiple scattering angles θ_{MS1} and θ_{MS2} at two fixed planes. If e is the toroid width in the deviation plane, the multiple scattering planes are located at $\frac{e}{\sqrt{12}}$ from the toroid middle plane.

Positions and angle errors at A and BC layers used in the fit come from segment errors. Errors on multiple scattering angles are calculated with:

$$\delta\theta = \frac{0.014}{P_{MS}} \sqrt{\frac{x_{MS}}{1.76}}$$

where P_{MS} is the momentum at the multiple scattering plane and x_{MS} is the muon path from the enter of the toroid to the multiple scattering plane.

The outputs are the track angle and position at both A and BC layers (two *TrackParameters*) and the error matrix in the variables $(y, c, \theta, \theta_{MS1}, \theta_{MS2})$.

3.2.2 error matrix in global propagation coordinates:

The (5×5) error matrix computed in the fit is in the variables: $(y, c, \theta, \theta_{MS1}, \theta_{MS2})$ at layer B. The *errorMatrix3to6* method of *SegmentFit* performs the transformation to a (6×6) error matrix in the global variables: (x, y, z, px, py, px) appropriated to error matrix propagation.

One first reduces the error matrix from $(y, c, \theta, \theta_{MS1}, \theta_{MS2})$ to (y, c, θ) by calling the *removeMS* method of the *NonLinearFit* class. The input (5×5) error matrix M_5 is formed with the (3×3) M_3 matrix in the variables (y, c, θ) and with the (2×2) M_2 matrix in $(\theta_{MS1}, \theta_{MS2})$:

$$M_5 = \begin{pmatrix} M_3 & M_{32} \\ {}^t M_{32} & M_2 \end{pmatrix}$$

The reduced error matrix M_3' in (y, c, θ) is:

$$M_3' = M_3 - M_{32}(M_2)^{-1}\, {}^t M_{32}$$

We complete M_3' to obtain a (6×6) error matrix in $(x, y, z, c, \theta, \phi)$ where x, y, z are the position in global coordinates and ϕ the B segment spherical angle.

For octants 1,2,5 and 6, σ_x is given by the measurement precision along the wire, $\sigma_y = 0$ because the position in y is fixed at the segment center of gravity. The formula $cotan\phi = \frac{x_B - x_A}{y_B - y_A}$ allows us to compute the correlation between error on x and error ϕ. If not coming directly from B segment, we can also calculate σ_ϕ.

$$\delta(cotan\phi) = \frac{\delta x_B - \delta x_A}{(y_B - y_A)} \tag{17}$$

$$\sigma_\phi^2 = \frac{sin^4\phi}{(y_B - y_A)^2}(\sigma_{x_A}^2 + \sigma_{x_B}^2) \tag{18}$$

and

$$< \delta x_B \delta\phi >= -\frac{sin^2\phi}{y_B - y_A}\sigma_{x_B}^2 \tag{19}$$

For octants 0,7,3 and 4, $\sigma_x = 0$, σ_y comes from the measurement precision along the wire and $tan\phi = \frac{y_B - y_A}{x_B - x_A}$ gives:

$$\sigma_\phi^2 = \frac{cos^4\phi}{(x_B - x_A)^2}(\sigma_{y_A}^2 + \sigma_{y_B}^2) \tag{20}$$

$$< \delta y_B \delta\phi > = \frac{cos^2\phi}{x_B - x_A}\sigma_{y_B}^2 \tag{21}$$

The *changeVariable* method of the *SegmentFit* class performs the coordinates change from $(x, y, z, c, \theta, \phi)$ to (x, y, z, px, py, px). If we are in an octant with horizontal wires, we can write, with ϵ refering to the sign of $c = \frac{q}{P}$:

$$\begin{aligned}
x &= x \\
y &= y \\
z &= z \\
P_x &= \frac{1}{|c|} sin\theta.cotan\phi = \frac{\epsilon}{c} sin\theta.cotan\phi \\
P_y &= \frac{sin\theta}{|c|} = \frac{\epsilon sin\theta}{c} \\
P_z &= \frac{cos\theta}{|c|} = \frac{\epsilon cos\theta}{c}
\end{aligned}$$

The jacobian matrix J for this coordinates change is ($P_t = \sqrt{P_x^2 + P_y^2}$):

$$J = \frac{\partial(x,y,z,P_x,P_y,P_z)}{\partial(x,y,z,c,\theta,\phi)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-P_x}{c} & P_x.\frac{P_z}{P_y} & \frac{-P_t^2}{P_y} \\ 0 & 0 & 0 & \frac{-P_y}{c} & P_z & 0 \\ 0 & 0 & 0 & \frac{-P_z}{c} & -P_y & 0 \end{pmatrix}$$

In the case of an octant with vertical wires:

$$\begin{aligned} P_x &= \frac{1}{|c|}sin\theta \\ P_y &= \frac{sin\theta}{|c|}.tan\phi \\ P_z &= \frac{cos\theta}{|c|} \end{aligned}$$

And so the jacobian matrix is:

$$J = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-P_x}{c} & P_z & 0 \\ 0 & 0 & 0 & \frac{-P_y}{c} & P_y.\frac{P_z}{P_x} & \frac{P_t^2}{P_x} \\ 0 & 0 & 0 & \frac{-P_z}{c} & -P_x & 0 \end{pmatrix}$$

The final error matrix E in the new coordinates (x,y,z,px,py,px) is obtained from the old one M_3' in (x,y,z,c,θ,ϕ) by:

$$E = J.M_3'.{}^tJ$$

4 The rcp file MuoTrackReco.rcp:

string PackageName = "MuoTrackReco"
// choose algorithm: either Combinatorial or Fit:
string TrackAlgorithm = "Fit"
// choose if you want to perform the fit or to make an estimation:
int performFit = 1
float MaxDriftRes = 100.
float MaxAxialRes = 100.
float MaxFitChi2 = 500000.
// choose magnetic field: either constant or real
string WhichField = "constant"
// choose if you want to fit with MS or not: either 0 or 1
int FitMS = 1
int DebugLevel = 0
// select cuts for fitting only good segment pairs:

float coneMSA = 0.1
float maxVertex = 30.
float coneTheta = 0.6
float conePhi = 0.2
int sigmaThetamax = 5
float ptAmin = 1

- *TrackAlgorithm* provides to choose if you want to use the combinatorial or the fit algorithm.
- *performFit* provides to choose if you want to really perform the fit or only to run the first estimation.
- *MaxDriftRes, MaxAxialRes* are not used in the fit algorithm.
- *MaxFitChi2* sets the cut on χ^2 result from the fit. If the χ^2 is greater than this value, there is no *MuoTrack* object created.
- *WhichField* provides to choose if you want to fit with a constant toroid magnetic field or with the real mapping for toroid magnetic field.
- *FitMS* provides to choose if you want to include two multiple scattering angles in the fit. If the set value is 0, there are only 3 variables instead of 5 in the fit.
- *DebugLevel* sets the level of debug print.
- *coneMSA* sets the estimated multiple scattering angle error in the calorimeter used in *coneInterceptVertex* (see section 1.2.2).
- *maxVertex* sets the vertex extension used in *coneInterceptVertex* (see section 1.2.2).
- *coneTheta* sets the cone angle in *AinConeB* (see section 1.2.3).
- *conePhi* sets the maximum ϕ angles difference between A and BC segments (see section 1.2).
- *sigmaThetamax* sets the maximum tangent difference used in *firstEstimate* to match the estimated angle with the segment angle difference (see section 2.1).
- *ptAmin* sets the cut on the estimated momentum in the deviation plane (see section 1.2).