

# A Proposed C++ Toolkit for Kalman Filtering in Larsoft

H. Greenlee

March 8, 2012

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>C++ Classes Overview</b>	<b>2</b>
<b>3</b>	<b>Linear Algebra</b>	<b>3</b>
<b>4</b>	<b>Surfaces</b>	<b>4</b>
4.1	Surface Base Class . . . . .	4
4.2	Unbounded Plane . . . . .	5
4.3	Bounded Plane . . . . .	6
4.3.1	Polygon-Bounded Plane . . . . .	6
4.3.2	Rectangle-Bounded Plane . . . . .	6
<b>5</b>	<b>Track State</b>	<b>7</b>
5.1	Track State Without Errors . . . . .	7
5.2	Track State With Errors . . . . .	8
5.3	Track State with Fit . . . . .	8
5.4	Track State With Measurements . . . . .	8
<b>6</b>	<b>Measurements</b>	<b>9</b>
6.1	Measurement Base Class . . . . .	9
6.2	Concrete Measurement Classes . . . . .	10
6.2.1	Wire-Time Measurement . . . . .	11
6.2.2	Space Point Measurement . . . . .	11
<b>7</b>	<b>Measurement Containers</b>	<b>11</b>
<b>8</b>	<b>Propagators</b>	<b>11</b>
<b>9</b>	<b>Interactors</b>	<b>12</b>
<b>10</b>	<b>Persistent Tracks</b>	<b>13</b>

<b>A Kalman Filter Algorithm</b>	<b>13</b>
<b>B Propagation</b>	<b>15</b>
B.1 Coordinate Transformation . . . . .	16
B.2 Chaining Propagation . . . . .	16
<b>C Global Fits and Kalman Smoothing</b>	<b>16</b>

## 1 Introduction

The subject of this document is a proposed set of c++ classes for implementing the Kalman Filter track reconstruction algorithm [1, 2, 3, 4, 5] in larsoft.

The Kalman Filter algorithm has been widely used in high energy physics experiments since the 1980s. The Kalman Filter is useful both as an efficient method of estimating track parameters, and as a road-following type pattern recognition tool. The proposals contained in this document are informed by the Kalman Filter implementation used by the D0 experiment, called TRF [6], as well as by the GENFIT framework [7], of which a copy exists in larsoft, without being a copy of either one. A brief description of the Kalman Filter algorithm is given in Appendices A–C.

## 2 C++ Classes Overview

There are various types of detector-specific classes that are needed or useful for implementing a Kalman Filter algorithm, including the following.

- Linear algebra
- Surfaces.
- Track state (i.e. information about a track on a single surface).
- Measurements.
- Measurement containers.
- Propagators.
- Interactors.
- Persistent tracks.

In the following sections, we will consider various aspects of the above types of classes, including use cases, interfaces, attributes, methods, and persistence.

### 3 Linear Algebra

General linear algebra classes will be used for track state and covariance matrix, measurements, and all of the related vectors and matrices involved in the Kalman Filter algorithm. Almost any decent linear algebra package would be good enough, presumably. Probably the most important design requirement that might not automatically be satisfied, is that the linear algebra package should be optimized for handling large numbers of small (up to  $5 \times 5$ ) matrices by avoiding dynamic memory allocations for such objects.

Two linear algebra packages that are readily available are ROOT and CLHEP. Of these ROOT seems the better choice for the following reasons.

- ROOT is actively maintained (not sure about CLHEP).
- Using ROOT does not introduce any new external package dependencies.
- ROOT vectors and matrices can be included in persistent classes without any issues.
- ROOT linear algebra has the important optimization of preallocating memory for small matrices (up to  $5 \times 5$ ) (I think CLHEP does something similar, but I'm not sure).

The basic ROOT linear algebra classes are `TVectorT<double>` for vectors, `TMatrixT<double>` for general matrices, and `TMatrixTSym<double>` for symmetric matrices.

One thing that concerns me about ROOT linear algebra classes is that they waste memory. Each ROOT vector preallocates 5 doubles, and each ROOT matrix preallocates 25 doubles, regardless of size. Also, ROOT symmetric matrices are stored in a square (rather than triangular) format, the triangular format being reserved for i/o only. The designers of ROOT linear algebra classes were obviously unconcerned about memory usage.

An ideal linear algebra package (for our purpose) would have the following properties.

- It would preallocate exactly the right amount of memory for each object. This implies that the dimensions of newly created linear algebra objects would have to be known at compile time.
- It would provide a dimension-independent interface so that classes could manipulate existing linear algebra objects without knowing the size.

I mention the above requirements in case someone reading this document knows of such a linear algebra package.

Note that the first of the above bullets (that sizes of newly created objects are known at compile time) is nontrivial, and needs to be designed in. The design of classes proposed here does follow this rule (that is why measurement classes have ownership of the Kalman gain matrix).

## 4 Surfaces

Surface classes must be able to be used in the following ways.

- As a destination for track propagation.
- Each kind of surface must have a defined set of track parameters.
- Any kind of surface that can hold a measurement must have a defined set of measurement coordinates.

Regarding the first bullet, some possible destinations for track propagations are:

- Propagate to plane (e.g. plane defined by wire and drift velocity).
- Propagate to point (e.g. reconstructed vertex or space point).

Classes described in this section are summarized in Fig. 1.

### 4.1 Surface Base Class

The surface base class should not have any data, and should have the following methods.

- Pure virtual method to convert track parameters (passed as linear algebra vector) to space point (without error).
- Pure virtual method to convert track parameters and direction flag to momentum vector (without error).
- Pure virtual methods for testing equality and near equality within some tolerance. The near equality method will be used to determine whether a propagation is necessary to reach a specified destination surface.

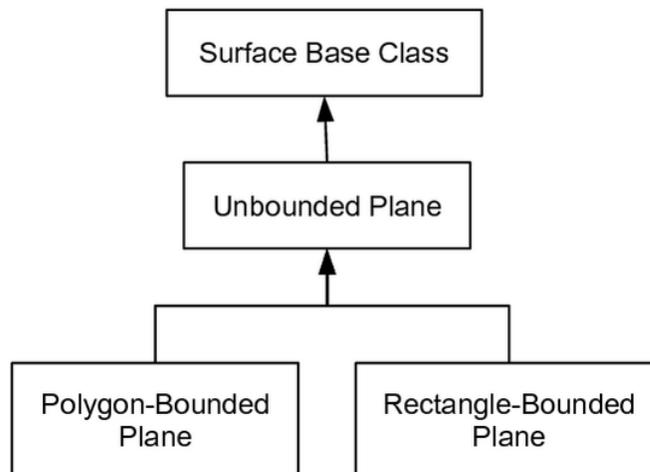


Figure 1: Inheritance diagram for surface classes.

- Pure virtual method for testing whether surfaces are parallel, within some tolerance (assuming that makes sense for a particular kind of surface).
- Pure virtual method for returning perpendicular separation between two parallel surfaces.
- Virtual method(s) to test whether track is in-bounds (with or without error). The base class will supply a default implementation that can be overridden by derived classes that implement a concept of boundedness.

It might seem odd that we don't want errors for the space point and momentum vector methods. The reason for this is that we will stipulate that any time we need to do calculations involving errors, we will use track objects. These methods are rather intended for things like event displays where precise errors are not needed, and adding errors will complicate things too much.

## 4.2 Unbounded Plane

In larsoft we will initially (and maybe only ever) be concerned with planar surfaces. The proposals given here generally follow the GENFIT implementation of a general plane (class GFDetPlane). GENFIT does one thing wrong by not having an abstract surface base class.

Note that plane surfaces are good enough to implement the propagation-to-point use case (also propagation-to-line), simply by making a destination surface that passes through the destination point (or line) with some convenient orientation. Therefore, there is no need to specify separate surface classes for these use cases.

All of the use cases described at the start of this section can be achieved by defining a general Cartesian coordinate transformation from the global  $(x, y, z)$  coordinate system to a local  $(u, v, w)$  coordinate system (not to be confused with the three wire readout views). Such a general transformation should include both a translation and a rotation, requiring at least six floating point parameters. The surface itself is defined as the plane  $w = 0$  in the local coordinate system. The  $u$  and  $v$  coordinates are used for measurement coordinates and track parameters.

The attributes of the plane surface class are:

- Displacement vector specified in global coordinated  $(x, y, z)$ .
- Rotation matrix. This could be specified in as few as three parameters using Euler angles, or redundantly using as many as nine parameters to store the entire rotation matrix. GENFIT stores the rotation matrix by storing unit vectors corresponding to the local  $u$  and  $v$  orthonormal basis vectors (six parameters).

The track parameters on the plane surface are as follows.

1.  $u$ .
2.  $v$ .

3.  $du/dw$ .
4.  $dv/dw$ .
5.  $1/p$  ( $q/p$  for magnetized detector).
6. Boolean direction parameter specifying whether  $dw/ds$  is positive or negative.

In the case of a measurement surface defined by a wire and the drift velocity, we take the  $u$ -axis parallel to the wire direction, and the  $v$ -axis parallel to the drift velocity.

### 4.3 Bounded Plane

Kalman Filter packages generally include classes for bounded surfaces. Although liquid argon TPCs lack internal structure, it is probably still useful to have bounded plane surfaces to implement the concept of TPC or fiducial volume boundaries in the local coordinate system of the plane.

Bounded plane surface classes should inherit from the unbounded plane class, add their own attributes for specifying the boundary, and override the bounds checking methods.

#### 4.3.1 Polygon-Bounded Plane

For a general plane, the bounding polygon corresponding to the TPC or fiducial volume can have anywhere from three to six sides. Such general polygon-bounded plane surfaces may be needed in the magnetic case where the drift velocity is not parallel to the  $x$ -axis. In this case, the extra attributes needed to specify the boundary are:

- A collection of points in the  $(u, v)$  plane, representing the vertices of the bounding polygon.

#### 4.3.2 Rectangle-Bounded Plane

In the non-magnetic case, it is probably enough to have bounded planes where the bounding polygon is a rectangle aligned with the  $u$ - and  $v$ -axes. In this case, the boundary attributes are:

- Minimum and maximum  $u$ .
- Minimum and maximum  $v$ .

## 5 Track State

Track state classes encapsulate information about a track on a single surface. The track state class does not need to be polymorphic in the same sense that one would expect to have different implementations of the same interface. Rather, I envision a cascading set of derived classes where each derived class adds attributes and methods. The class hierarchy described in this section is summarized in Fig. 2.

### 5.1 Track State Without Errors

This is the most basic track state classes. It has two attributes.

- Surface (pointer to Surface base class).
- Track state vector.
- Track direction (boolean). Specifies whether momentum vector is pointing in forward or reverse direction compared to surface.

This class includes methods for calling the underlying virtual methods of its surface.

- Return position vector (without errors).
- Return momentum vector (without errors).
- Return in-bounds status (without errors).

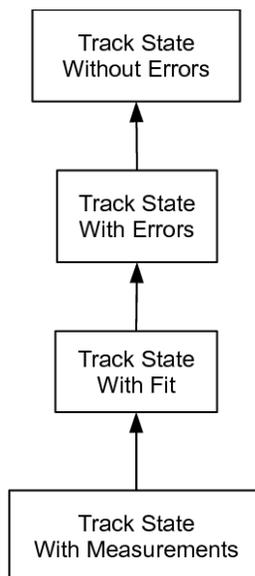


Figure 2: Inheritance diagram for track state classes.

## 5.2 Track State With Errors

This class inherits from the Track State Without Errors class and adds the following attribute.

- Track parameter covariance matrix.

And it adds the following method.

- Return in-bounds status (with errors).

## 5.3 Track State with Fit

This class inherits from Track State With Errors class and adds additional information obtained from the Kalman fit.

- Cumulative fit chisquare.
- Fit status (i.e. whether fit is optimal or not).
- Propagation distance.

## 5.4 Track State With Measurements

This class inherits from the Track State With Fit class and adds information about which measurements contributed to this track.

- A collection of measurements (pointers to measurement base class).

One of the measurements is designated as the latest or current measurement (i.e. the last measurement added to the collection).

The basic Kalman Filter algorithm is carried out with this kind of track using the following steps.

1. Start with a track containing a set of successfully fit measurements.
2. Identify a candidate measurement for addition to the track.
3. Make a trial track by copying Track State With Errors part of the track.
4. Propagate the trial track to the candidate measurement surface.
5. Call the prediction method of the candidate measurement.
6. Calculate the incremental chisquare.
7. If the incremental chisquare fails the incremental chisquare cut, discard this measurement and look for a new candidate measurement (go to step 2).
8. If the predicted incremental chisquare passes the incremental chisquare cut, then do the following:

- (a) Add the candidate measurement to the collection.
- (b) Add the incremental chisquare to the cumulative chisquare.
- (c) Update the track state vector and covariance matrix using the Kalman filtering formula (Eq. 7 and one of Eqs. 9–11).

To carry out these kinds of steps, there will need to be the following kinds of methods.

- Add a measurement to the collection.
- Update the track parameter state vector and covariance matrix using the Kalman updating formula.

## 6 Measurements

Measurement classes encapsulate data and methods related to measurements, predictions, and residuals (all specified in measurement coordinates). Classes described in this section are summarized in Fig. 3.

### 6.1 Measurement Base Class

The measurement base class should have the following attributes:

- Measurement surface.
- Art pointer to associated `RecoBase` object (art associations could possibly be used instead).

The measurement surface attribute should be held via pointer to the surface base class.

The measurement base class should have the following methods.

- Pure virtual accessors for:
  - Measurement vector and covariance matrix.
  - Prediction vector and covariance matrix.

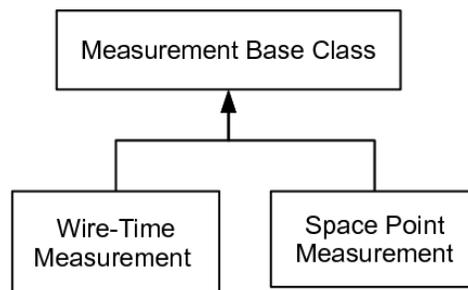


Figure 3: Inheritance diagram for measurement classes.

- Residual vector and covariance matrix.
- Inverses of residual and measurement covariance matrix.
- H-matrix.
- Kalman gain matrix.
- Predicted incremental chisquare (using Eq. 12).

All of the above methods (except the chisquare) should return references to linear algebra classes owned by the measurement object.

- Pure virtual method for calculating a prediction. This method should accept a track state as argument and fill attributes corresponding to prediction vector and covariance matrix, residual vector and covariance matrix, H-matrix, and Kalman gain matrix. The prediction method should test whether the track and the measurement surface are the same (within tolerance), and it should be prepared to deal with the case where the surfaces are not the same (either by propagating the track, returning an error, throwing an exception, etc.).

## 6.2 Concrete Measurement Classes

Concrete measurement classes derive from the measurement base class. Concrete measurements should own attributes corresponding to the virtual accessor methods described in the previous section, namely,

- Measurement vector and covariance matrix.
- Prediction vector and covariance matrix.
- Residual vector and covariance matrix.
- Inverses of residual and measurement covariance matrix.
- H-matrix.
- Kalman gain matrix.

This is a pretty fat set of attributes (probably amounting to a couple of kilobytes per measurement if ROOT linear algebra classes are used) for something that may conceptually be a single floating point number and error. The idea is that all possible measurements will be constructed at the beginning of the event and never copied (so the number of measurements does not become astronomical, hopefully).

Concrete measurement classes should override all of the virtual methods of the measurement base class including the all-important prediction method. Calling the prediction method should trigger updating many attributes of the measurement object.

### 6.2.1 Wire-Time Measurement

The wire-time measurement is a one-dimensional measurement based on `RecoBase/Hit`. Namely, it consists of a drift time (the measurement coordinate) measured on a particular wire. The measurement plane is defined by the wire and the drift velocity.

### 6.2.2 Space Point Measurement

The space point measurement is a two-dimensional measurement consisting of the coordinates of a point on the measurement surface. The corresponding `RecoBase` object is `RecoBase/SpacePoint`. One feature of this type of measurement is that there is no single uniquely defined measurement plane. In principle, any plane passing through the space point will do. There are a couple of ways this class could be implemented.

- The measurement surface could be chosen once and for all at the beginning of the event and never changed (e.g. a  $z$ -plane).
- The prediction method could update the measurement surface in some way, for example, by setting it to the plane perpendicular to the track. In this case, the prediction method would have to update the measurement vector and covariance matrix also.
- A third possibility is to define pointlike surface (destination of propagate-to-point) and use that as the measurement surface. We said in Sec. 4 that this wouldn't be necessary, but we mention it as a possibility.

## 7 Measurement Containers

A measurement container has ownership of all of the measurements of a particular kind in the event and makes them available to other objects in the form of abstract pointers to the measurement base class (measurements are never copied once the measurement container is filled). Conceptually, a measurement container is nothing more than a collection of pointers. The reason that I mention measurement containers as a separate category is because there is plenty of room for inventing clever containers with the goal of efficiently finding candidate measurements that are “near” the endpoint of the currently growing track (to help with step two of the track finding algorithm described in Sec. 5.4). I do not have any particular clever ideas to put forward in this proposal, though.

## 8 Propagators

Propagators accept a track state (tracks parameters and error matrix on specified surface), and calculate a new track state on a destination surface. The propagation

distance to reach the destination will also be calculated. Refer to Appendix B for a general discussion of propagation.

We will definitely want to have an abstract propagator base class that has basically a single pure virtual propagation method. Then we will want one or more concrete propagator classes that derive from the propagator base class and implement the propagation method (Fig. 4).

There are many possible use cases for propagators. The various use cases may be implemented by different classes, or the same class with different configuration parameters. Some possible use cases are.

- Propagate without error.
- Propagate with error, but without noise
- Propagate with error and noise.
- Propagate with or without magnetic field (uniform field or map).
- Propagate with or without average energy loss.
- Propagate with or without energy loss fluctuations.
- Propagate with or without multiple scattering.
- Propagate with or without short distance approximations.

## 9 Interactors

Interactors accept a track state (track parameters on specified surface), and calculate the noise matrix (see Appendix B) resulting from propagation over a specified distance through a specified material. In the case of larsoft, the material will usually be liquid argon, but could potentially be other kinds of deal material like walls of containment vessels. Interactors would be called from a propagator when propagation with noise was needed.

We should have an abstract interactor base class with a pure virtual interaction method. Various concrete derived classes will implement the interaction method (see Fig. 5). Here are some use cases, which probably will want different derived classes.

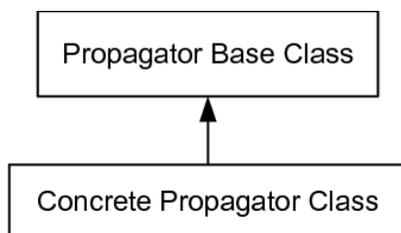


Figure 4: Inheritance diagram for propagator classes.

- Gaussian thin or thick multiple scatterer.
- Gaussian energy loss fluctuator.

As previously noted, the interaction associated with average energy loss is not noise, and would have to be handled within the propagator proper.

## 10 Persistent Tracks

I do not think we will want to save most of the classes described above as data products directly. In particular, I do not think we will ever save measurement objects. Rather, we will associate persistent track objects with the corresponding `RecoBase` objects (`Hit` or `SpacePoint`). Therefore, a persistent track object would have the following kinds of attributes.

- Track state information (`Track State With Fit` class) on one or more surfaces, including at least the endpoints.
- Art pointers or associations to `RecoBase` objects that are used as measurements.

## A Kalman Filter Algorithm

Let  $\mathbf{x}$  stand for the five-dimensional state vector of a track on a given surface  $S$ . The state vector can be regarded as being a function of the surface,

$$\mathbf{x} = \mathbf{x}(S). \tag{1}$$

If we know the state vector on some surface  $S$ , then it is possible to calculate the state vector on any other surface  $S'$  via propagation. The basic Kalman Filter algorithm factorizes almost completely with propagation algorithms. The latter are detector-specific and are often quite complicated, whereas the Kalman Filter algorithm per se is simple and detector-independent. In this appendix, we will limit ourselves to describing the Kalman Filter algorithm on a single surface. In Appendix B we examine some general aspects of track propagation.

We denote by the symbol  $\mathbf{C}$  the covariance matrix of the estimated state vector  $\mathbf{x}$  with respect to the true state vector  $\hat{\mathbf{x}}$ .

$$\mathbf{C} = \langle (\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T \rangle. \tag{2}$$

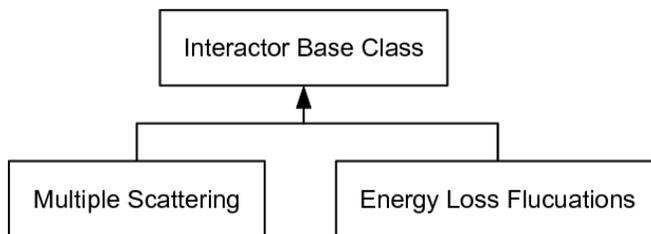


Figure 5: Inheritance diagram for interactor classes.

Let  $\mathbf{m}$  be a measurement vector on a surface  $S$ . The dimensionality of the measurement vector can vary depending on the type of measurement. We assume that there is a function  $\mathbf{h}(\mathbf{x})$  that maps track parameters to predicted measurement coordinates for any measurement surface.

$$\mathbf{m}_{\text{pred}} = \mathbf{h}(\mathbf{x}). \quad (3)$$

We furthermore assume that the prediction function can be linearized locally around some state vector  $\mathbf{x}_0$ .

$$\mathbf{h}(\mathbf{x}) = \mathbf{h}(\mathbf{x}_0) + \mathbf{H}(\mathbf{x} - \mathbf{x}_0), \quad (4)$$

where  $\mathbf{H}$  is the matrix of partial derivatives of the components of the prediction with respect to the track parameters.

$$H_{ij} = \frac{\partial h_i(\mathbf{x})}{\partial x_j}. \quad (5)$$

Define the residual vector  $\mathbf{r}$  as the difference between the measurement and the prediction on the same surface.

$$\mathbf{r} = \mathbf{m} - \mathbf{h}(\mathbf{x}). \quad (6)$$

Finally, define various covariance matrices in measurement coordinates:  $\mathbf{V}$  the intrinsic measurement error,  $\mathbf{T}$  the prediction error, and  $\mathbf{R} = \mathbf{T} + \mathbf{V}$  the residual error.

The so-called filtered (i.e. updated) track parameters  $\mathbf{x}'$  are obtained by simultaneously minimizing the chisquare of the filtered track parameters with respect to the original estimated track parameters  $\mathbf{x}$ , and the chisquare of the filtered prediction vector  $\mathbf{h}(\mathbf{x}')$  with respect to the measurement vector  $\mathbf{m}$ . The filtered track parameters are given by

$$\mathbf{x}' = \mathbf{x} + \mathbf{K}\mathbf{r}, \quad (7)$$

where

$$\mathbf{K} = \mathbf{C}\mathbf{H}^T\mathbf{R}^{-1} \quad (8)$$

is called the Kalman gain matrix.

The filtered track parameter covariance matrix is

$$\mathbf{C}' = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{C}, \quad (9)$$

$$= (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{C}(\mathbf{I} - \mathbf{K}\mathbf{H})^T + \mathbf{K}\mathbf{V}\mathbf{K}^T. \quad (10)$$

Alternatively, the filtered covariance matrix can be expressed in terms of inverse covariance matrices as

$$\mathbf{C}'^{-1} = \mathbf{C}^{-1} + \mathbf{H}^T\mathbf{V}^{-1}\mathbf{H}. \quad (11)$$

In view of Eq. 11, it can be an interesting design choice to store the inverse of the track covariance matrix rather than the track covariance matrix as part of track state information. Doing this solves the tricky problem of setting the track errors to be

initially “infinite.” On the other hand, it (slightly) complicates the calculation of propagation noise.

The incremental chisquare is

$$\chi^2 = \mathbf{r}^T \mathbf{R}^{-1} \mathbf{r}. \quad (12)$$

Equation 12 is invariant whether calculated in terms of the original or filtered residual. Therefore, it is easy to implement an incremental chisquare cut to select candidate measurements for inclusion in the track. Furthermore, hit selection can take place before updating the track parameters and error matrix.

## B Propagation

The process of evolving the track parameter state vector  $\mathbf{x}$  and covariance matrix  $\mathbf{C}$  from one surface  $S$  to another surface  $S'$  is called propagation. In general, the propagation function is complicated and nonlinear. However, we assume that the propagation function can be linearized locally around some reference trajectory  $(\mathbf{x}_0, S)$  to  $(\mathbf{x}'_0, S')$ .

$$\mathbf{x}' = \mathbf{x}'_0 + \mathbf{F}(\mathbf{x} - \mathbf{x}_0). \quad (13)$$

$\mathbf{F}$  is the forward propagation matrix, which is the matrix of partial derivatives of the propagated track parameters  $\mathbf{x}'$  with respect to the original track parameters  $\mathbf{x}$ .

$$F_{ij} = \frac{\partial x'_i}{\partial x_j}. \quad (14)$$

For the reverse propagation (from  $S'$  to  $S$ ) we write

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{B}(\mathbf{x}' - \mathbf{x}'_0), \quad (15)$$

where  $\mathbf{B}$  is the backward propagation matrix,

$$B_{ij} = \frac{\partial x_i}{\partial x'_j}, \quad (16)$$

and obviously  $\mathbf{B} = \mathbf{F}^{-1}$ .

The track parameter covariance matrix is modified by propagation as follows, for forward and backward propagation.

$$\mathbf{C}' = \mathbf{F}\mathbf{C}\mathbf{F}^T + \mathbf{Q}_F. \quad (17)$$

$$\mathbf{C} = \mathbf{B}\mathbf{C}'\mathbf{B}^T + \mathbf{Q}_B. \quad (18)$$

The  $\mathbf{Q}$  terms represent the irreversible noise contribution due to stochastic processes, such as multiple scattering and energy loss fluctuations (the effect of average energy loss is not irreversible, and is included in the propagation matrix and reference trajectory). In the absence of propagation noise, the propagation is fully reversible, with respect to both track parameters and error matrix. If there is propagation noise, the

propagation is not reversible. However, the forward and backward noise are related as

$$\mathbf{Q}_B = \mathbf{B}\mathbf{Q}_F\mathbf{B}^T, \quad (19)$$

$$\mathbf{Q}_F = \mathbf{F}\mathbf{Q}_B\mathbf{F}^T. \quad (20)$$

Note that the noise contribution to the error matrix is positive regardless of the direction of propagation.

## B.1 Coordinate Transformation

A special kind of propagation occurs when the track is already located on the destination surface, even though the destination surface may not be the same as the starting surface. In other words, one is simply transforming the track state from one surface to a different surface. A coordinate transformation of this type is perfectly well described using the formalism described in this appendix. In particular, the calculation of the propagation matrix and the transformation of the error matrix are according to Eqs 14 and 17, with zero noise.

## B.2 Chaining Propagation

One frequently wants to divide a complex propagation into several shorter or simpler steps. Assume that we have  $n$  forward propagations, each with its own propagation matrix  $\mathbf{F}_k$  ( $k = 1, \dots, n$ ), and propagation noise matrix  $\mathbf{Q}_{Fk}$ . The total propagation matrix  $\mathbf{F}$  is

$$\mathbf{F} = \prod_{k=1}^n \mathbf{F}_k, \quad (21)$$

To find the total noise, calculate the cumulative noise matrix  $\mathbf{Q}_{Fk}$  after  $k$  steps using

$$\mathbf{Q}_{Fk} = \mathbf{F}_k \mathbf{Q}_{Fk-1} \mathbf{F}_k^T + \mathbf{Q}_{Fk}, \quad (22)$$

and the total noise matrix is  $\mathbf{Q}_F = \mathbf{Q}_{Fn}$ .

Note that neither the total propagation matrix  $\mathbf{F}$  nor the total noise matrix  $\mathbf{Q}_F$  depend on the initial error matrix  $\mathbf{C}$ . Therefore, splitting a propagation into steps does not muddy the distinction between initial error and noise components in the final track parameter error matrix  $\mathbf{C}'$ .

## C Global Fits and Kalman Smoothing

In a typical track-fitting scenario, one has  $n$  measurements  $\mathbf{m}_k$  ( $k = 1, \dots, n$ ), on surfaces  $S_k$ . Starting from an initial estimated track with “infinite errors,” one propagates to each measurement surface, starting with  $S_1$  and ending with  $S_n$ . The fitted track parameters are optimal only at the final surface  $S_n$ , since that is the only surface that makes use of all available information. One can also do the Kalman fit in the reverse direction to obtain optimal track parameters at surface  $S_1$ .

The accuracy of the fit can sometimes be improved by iterating the Kalman fit, in alternating directions. Iterating will improve the accuracy of the linearized prediction function (Eq. 4), by providing better estimates of the initial track parameters.

It may or may not be good enough to have optimal estimates of the track parameters at either or both endpoints. Sometimes it is desirable to have optimal estimates of track parameters at interior surfaces. In such cases, optimal estimates can be obtained by combining the predicted track parameters at surface  $S_k$  from the forward fit ( $\mathbf{x}_{Fk}$ ) with the filtered track parameters from the backward fit ( $\mathbf{x}'_{Bk}$ ), or alternatively, forward-filtered and backward-predicted track parameters can be combined. In this way, one can obtain estimates of track parameters at interior surfaces that make use of all available information, and are therefore optimal. This type of algorithm is called Kalman smoothing.

## References

- [1] R.E. Kalman, J. Bas. Eng. **82D**, 35 (1960); R.E. Kalman and R.S. Bucy, J. Bas. Eng. **83D**, 95 (1961);
- [2] P. Billoir, Nucl. Instr. and Meth. **A225**, 352 (1984).
- [3] R. Frühwirth (DELPHI exp.), Nucl. Instr. and Meth. **A262**, 444 (1987).
- [4] P. Billoir and S. Qian (ZEUS exp.), Nucl. Instr. and Meth. **A294**, 219 (1990).
- [5] E.J. Wolin and L.L. Ho, Nucl. Instr. and Meth. **A329**, 493 (1993).
- [6] From my personal experience. There is no public document, however, the D0 code repository is browsable via url <http://cdcvs.fnal.gov/cgi-bin/public-cvs/cvsweb-public.cgi/?cvsroot=d0cvs>, specifically, packages beginning with “trf.”
- [7] C. Höppner, S. Neubert, B. Ketzer, S. Paul, Nucl.Instrum.Meth. A620 (2010) 518-525 (arXiv:0911.1008 [hep-ex]).