Linear Fit within Missing-Hit Roads in FTKSim

A. Annovi, F. Crescioli, M. Dell'Orso, P. Giannetti, G. Punzi, G. Volpi

November 28, 2017

1 Revision history

- Nov 28th, 2017: fixed formulas 18

2 Fitting missing-hit tracks

The linear fit, as proposed for the FTK track fitter (TF) [1], evaluates each track parameter p by using the equation:

$$p = \vec{c} \cdot \vec{x} + q \tag{1}$$

where $\vec{c}$ and q are fit constants (FCs) for the given sector [2] to which the $\vec{x}$ components, namely the hit *coordinates*, belong. The number N of coordinates is the number of strip detector layers plus twice the number of pixel detector layers used in the fit. In parallel with the track parameters, the TF also evaluates a fit-quality estimator χ^2:

$$\chi^2 = \sum_{i=1}^{N_f} \left[\left(\sum_{j=1}^{N} S_{ij}\, x_j \right) + h_i \right]^2 \tag{2}$$

where the sum indices are explicit, S is a rectangular matrix of size $N_f \times N = (N - N_p) \times N$, $\vec{h}$ is a constant vector of size N_f, N_p is the number of track parameters, and N_f is the number of track constraints (χ^2 degrees of freedom). S and $\vec{h}$ are additional FCs for the given sector.

For each sector a total of $N_p \times (N+1) + N_f \times (N+1) = N \times (N+1)$ pre-calculated FCs result from (1) and (2). In the current configuration $N = 14$, $N_p = 5$, and $N_f = 9$ [3]. These numbers, multiplied by about 300,000 sectors, mean that we need a set of at least 6.3×10^7 FCs to be stored somewhere.

It is important to remark that the dimensions of the arrays in (1) and (2) are fixed. This entails that only roads with hits on every layer can be processed this way with the given FC set. Thus, any single-module inefficiency would result in inefficiency of the tracking algorithm.

A practical solution to this problem is to admit also missing-hit roads, by progressively reducing the number of used coordinates to $N-1$, $N-2$, etc... However, each case must be processed separately, since *the FCs depend on the number of coordinates* *as well as on the identity of the missing layers.* In this approach the FC storage quickly becomes an issue. For instance, in order to process all roads having any single missing hit, the number of additional 6.3×10^7-manifold FC sets would be N, i.e. one set for each possibly missing coordinate [1].

3 Reclaiming N-hit FCs

In order to circumvent the disruptive increase of memory demand, FTKSim currently reclaims the N-hit FCs for use with N'-hit roads, where $N' < N$ is the number of actually measured coordinates. The procedure, which exploits the strong coordinate correlations arising from the membership to the same real trajectory, will be referred to as *guessing hits*. Once the missing coordinates are guessed, the incomplete $\vec{x}$ vector can be filled with the guessed values up to dimension N, then the standard N-hit FCs can be used in order to calculate the track parameters with (1).

A good guess for the missing hits is to position them exactly where they would minimize the χ^2 defined in (2), leaving of course unchanged the measured points. This, more or less (resolution losses under investigation), corresponds to positioning the missing hits at the intersections between the N'-hit fit track and the inefficient layers.

Let $M = N - N'$ be the number of missing hits, $\mathcal{I} = \{1, 2, \dots, N\}$ the set of coordinate indices, $\hat{\mathcal{I}}$ the set of missing-coordinate indices, and $\check{\mathcal{I}}$ the set of measured-coordinate indices. $\mathcal{I} = \hat{\mathcal{I}} \cup \check{\mathcal{I}}$, the size of $\hat{\mathcal{I}}$ is M and the size of $\check{\mathcal{I}}$ is N'. In order to minimize the χ^2 of (2), we must solve the M-equation system:

$$\frac{\partial \chi^2}{\partial x_{\hat{\jmath}}} = \sum_{i=1}^{N_f} S_{i\hat{\jmath}} \left(\sum_{k \in \mathcal{I}} S_{ik} x_k + h_i \right) = 0 \qquad \forall \hat{\jmath} \in \hat{\mathcal{I}} \tag{3}$$

where $\hat{\jmath}$ is the index of missing coordinates.

Now, let us split the $\vec{x}$ components in two groups: one with index $\hat{k}$ representing the missing values, the other with index $\check{k}$ representing the measured coordinates. Isolating the missing values on the left side, system (3) becomes:

$$\sum_{i=1}^{N_f} \sum_{\hat{k} \in \hat{\mathcal{I}}} S_{i\hat{\jmath}} S_{i\hat{k}} x_{\hat{k}} = - \sum_{i=1}^{N_f} S_{i\hat{\jmath}} h_i - \sum_{i=1}^{N_f} \sum_{\check{k} \in \check{\mathcal{I}}} S_{i\hat{\jmath}} S_{i\check{k}} x_{\check{k}} \qquad \forall \hat{\jmath} \in \hat{\mathcal{I}} \tag{4}$$

where the right side can be computed from the measured values.

By (4), the problem is translated into the linear system:

$$C\, \hat{\vec{x}} = \vec{t} \tag{5}$$

where $\hat{\vec{x}} \equiv (\hat{x}_1, \dots \hat{x}_M) = \left(x_{\hat{k}_1}, \dots, x_{\hat{k}_M} \right)$ is the projection of $\vec{x}$ onto the subspace of the missing coordinates, C is a square symmetrical matrix of dimensions $M \times M$, and $\vec{t}$ is a

[1] Actually, this example is true for N independent measurements only. The case of the pixel detectors, with $x - y$ correlated measurements, unfolds with slightly different mathematical details

constant vector term of dimension M. The components of C and $\vec{t}$ are evaluated from (4):

$$C_{jn} = \sum_{i=1}^{N_f} S_{i\hat{k}_j} S_{i\hat{k}_n} \qquad j, n \in \{1, \ldots, M\} \tag{6a}$$

$$t_j = - \sum_{i=1}^{N_f} S_{i\hat{k}_j} h_i - \sum_{i=1}^{N_f} S_{i\hat{k}_j} \sum_{\check{k} \in \check{\mathcal{I}}} S_{i\check{k}} x_{\check{k}} \qquad j \in \{1, \ldots, M\} \tag{6b}$$

Finally, the missing coordinates are obtained by:

$$\hat{\vec{x}} = C^{-1} \vec{t} \tag{7}$$

4 Estimating the number of arithmetical operations

In order to figure out how the procedure could be implemented in a FPGA chip, it can be useful to estimate the number of products $N_\times$ and the number of sums N_+ at different stages of the algorithm.

Evaluation of the N_p parameters in (1) requires:

$$N_\times = N_p N \tag{8a}$$

$$N_+ = N_p N \tag{8b}$$

Evaluation of the χ^2 in (2) requires:

$$N_\times = N_f (N + 1) \tag{9a}$$

$$N_+ = N_f N + (N_f - 1) = N_f (N + 1) - 1 \tag{9b}$$

Evaluation of the $M(M + 1)/2$ independent elements of the symmetrical C matrix in (6a) requires:

$$N_\times = \frac{M(M+1)}{2} N_f \tag{10a}$$

$$N_+ = \frac{M(M+1)}{2} (N_f - 1) \tag{10b}$$

These elements do not depend on the measured hits $\vec{x}$, thus the C^{-1} elements could be pre-calculated and stored once for all in memory. Taking into account all possible combinations of M missing coordinates, the total number of constants to be stored *per sector* is:

$$N_C = \binom{N}{M} \frac{M(M+1)}{2} \tag{11}$$

Evaluation of the M elements of $\vec{t}$ in (6b) requires:

$$N_\times = M(N_f + N_f(N' + 1)) = M N_f (N' + 2) \tag{12a}$$

$$N_+ = M((N_f - 1) + (N_f N' - 1)) = M(N_f (N' + 1) - 1) \tag{12b}$$

The values of the first sum in (6b) do not depend on the measured hits, but only on the missing-coordinate index $\hat{\jmath}$, and could be pre-calculated and stored once for all in memory. Taking into account all possible missing coordinates, the total number of constants to be stored *per sector* is:

$$N_t = N \tag{13}$$

Correspondingly, the number of arithmetical operations would reduce to:

$$N_\times = M N_f (N' + 1) \tag{14a}$$
$$N_+ = M N_f N' \tag{14b}$$

The most intensive calculations in equations (6) involve the product of two S matrix. This product does not depend on the measured hits. It can be pre-calculated as:

$$SS_{jn} = \sum_{i=1}^{N_f} S_{ij} S_{in} \qquad j, n \in \{1, \ldots, N\} \tag{15}$$

For elements that satisfy $j, n \in \{1, \ldots, M\}$ this is exactly the C_{jn} matrix of equation (6a). Most importantly, equation (6b) reduces to:

$$t_j = -\sum_{i=1}^{N_f} S_{i\hat{k}_j} h_i - \sum_{\check{k} \in \check{\mathcal{I}}} SS_{\hat{k}_j \check{k}} x_{\check{k}} \qquad j \in \{1, \ldots, M\} \tag{16}$$

The number of constants to be stored *per sector* corresponds to the elements of the symmetrical SS matrix of equation (15) :

$$N_{SS} = N(N-1)/2 \tag{17}$$

Correspondingly, the number of arithmetical operations would reduce to:

$$N_\times = M(N' + N_f) \tag{18a}$$
$$N_+ = M(N' + N_f - 1) \tag{18b}$$

We still have the option to pre-calculate the first term of (16). This would reduce both $N_\times$ and N_+ to MN'.

In order to calculate the missing hits of equation (7) we can use Cramer's rule [4] that requires the calculation of $M+1$ determinants of MxM matrixes. Thus the number of operations needed is:

$$N_/ = 1 \tag{19}$$
$$N_\times = (M+1)(M-1)M! + M \tag{20}$$
$$N_+ = (M+1)(M! - 1) \tag{21}$$

where 1 division is needed to take the reciprocal of the determinant of the C_{jn} matrix of equation (6a). It should be possible to further reduce the number of calculations, considering that several multiplications needed for the $M + 1$ determinant calculations are duplicated. Further optimization could take advantage of optimized algorithms for determinant calculation that exist in literature.

Finally, table 1 summarizes the number of arithmetical operations, as well as the number of constant values to be stored for each sector, for different choices of pre-calculations in the current configuration ($N = 14, N_p = 5$) and for the several cases of missing coordinates ($M = 1, 2, \ldots$).

The additional constants to be stored for the "Pre-Calc SS" case is 105. It represent a moderate increase with respect to the 210 FC constants. We observe that using the "Pre-Calc SS" option the number of calculations up to $M = 3$ becomes manageable. It is lower than those needed to calculate a χ^2.

			operations			stored constants		
		Task	$N_/$	$N_\times$	N_+	N_C	N_t	FC
	14-coord. fit	Parameters	0	70	70	-	-	75
		χ^2	0	135	208	-	-	135
$M=1$	No Pre-Calc	C	0	9	8	0	0	-
		T	0	135	125	0	0	-
	Pre-Calc	C	0	0	0	14	-	-
		T	0	126	117	-	14	-
	Pre-Calc SS	Cramer	1	1	0	105	-	-
		T	0	26	26	-	-	-
$M=2$	No Pre-Calc	C	0	27	24	0	0	-
		T	0	252	253	0	0	-
	Pre-Calc	C	0	0	0	273	-	-
		T	0	234	216	-	14	-
	Pre-Calc SS	Cramer	1	8	3	105	-	-
		T	0	48	48	-	-	-
$M=3$	No Pre-Calc	C	0	54	48	0	0	-
		T	0	351	321	0	0	-
	Pre-Calc	C	0	0	0	2184	-	-
		T	0	324	297	-	14	-
	Pre-Calc SS	Cramer	1	51	20	105	-	-
		T	0	66	66	-	-	-
$M=4$	No Pre-Calc	C	0	90	80	0	0	-
		T	0	432	392	0	0	-
	Pre-Calc	C	0	0	0	10010	-	-
		T	0	396	360	-	14	-
	Pre-Calc SS	Cramer	1	364	115	105	-	-
		T	0	80	80	-	-	-

Table 1: Number of arithmetical operations and number of stored constants for several computational tasks in different conditions of missing coordinates and pre-calculated constants. In the "No Pre-Calc" case, the numbers $N_\times$ and N_+ refer to the calculations of elements of C matrix and do not include the operations to invert it. In the "Pre-Calc" case, N_C is the number of stored elements of all possible C^{-1} matrices (11). In the "Pre-Calc SS" case, N_C is the number of stored elements of the SS matrix of equation (15), which are the same for all M values. The numbers $N_/$, $N_\times$ and N_+ refer to the calculations needed to solve the equations (5) using Cramer's rule.

5 Recycling calculations within a road

The Fast Tracker processor will use the linear fit to fit candidate tracks within a road found by the Associative Memory [1]. In this framework, for each road all possible combinations of hits will be fitted. For a road involving N layers that have n_j hits in the j-th layer with $j \in \{1, \ldots, N\}$, the number of combinations of hits to fit will be:

$$N_{fit} = \prod_{j=1}^{N_l} n_j \tag{22}$$

where N_l is the number of layers also defined as $N_l = N - N_{pixels}$. It is clear that, when multiples fits within a road are needed, the scalar products of equations (1) and (2) will repeat multiple times the product of the same hits with the same matrix elements. Two strategies are proposed to reduce the number of calculations needed to perform all the fits. In the following, we refer only the calculation of the χ^2 because the track parameters will be evaluated only if the found χ^2 is below threshold. The first strategy calculates the product of the S_{ij} matrix, see eq. (2), with every hit at the beginning of the road process. Thus reducing the χ^2 calculation for each fit to a certain number of additions and only N_f multiplications needed to square each track constraint. This strategy, needs the theoretical minimum number of multiplications. While it looks very promising, some further thought is needed to understand how to keep under control the time needed for the additions[2].

The second strategy takes advantage of the fact that different combinations share a fraction of the hits. For this reason, pieces of the scalar products in equation (2) calculated for one combination can be reused for the next combination(s). With this approash, the first step of the road fitting algorithm will sort the layers of the road to be fit by number of hits. The layers with less hits will become the first layers, while the most populated layers will become the last. In the follow we assume that layers are sorted. The combinations of hits will be fit sequentially. The first combination will be identified by

$$k_j = 1 \qquad \forall j \in \{1 \ldots N_l\} \tag{23}$$

where k_j is the index referring to the k-th hit in the j-th sorted layer. For the first combination the algorithm will calculate the full χ^2 of equation (2). The results of all partial sums will be stored in a memory. We need N_f memories (one for each constraint), with each memory being N locations deep. The $N_f \times N$ partial sums, which are summed up to the n-th coordinate, are defined by:

$$c_{in} = \left(\sum_{j=1}^{n} S_{ij}\, x_j \right) + h_i \tag{24}$$

[2]Here I'm assuming that there is not much saving by reducing only the number of multiplications, if the hardware is able to perform additions and multiplications at a similar rate as it is true for DSP blocks in FPGAs.

At this point the algorithm proceeds to loop over all combinations of hits in order to calculate the χ^2 for all of them. The loop will start incrementing the k_j index for $j = N_l$, when ever $k_j > n_j$ it will overflows. Thus is will sets $k_j = 1$ and it will increment k_{j-1} until all combinations will be scanned. For a given combination, lets call m the incremented and non overflowed layer. For this combination the χ^2 can be calculated as follows:

$$\chi^2 = \sum_{i=1}^{N_f} \left[\left(\sum_{j=m}^{N} S_{ij}\, x_j \right) + h_i + c_{i(m-1)} \right]^2 \tag{25}$$

Evaluation of the χ^2 in (27) requires:

$$N_\times = N_f(N - m + 2) \tag{26a}$$

$$N_+ = N_f(N - m + 2) + (N_f - 1) = N_f(N - m + 3) - 1 \tag{26b}$$

When m is close to N (e.g. $m = N$ or $m = N - 1$) the number of needed operation is much lower than those of equations (9).

What is the average of $N - m$ during the loop over all hit combinations? Neglecting the fact that pixel layers correspond to two coordinates, it can be calculated as

$$< N - m >= \frac{\sum_{m=1}^{N_l} \left[(N - m) * \prod_{j=m}^{N} n_j \right]}{\prod_{j=1}^{N} n_j} \tag{27}$$

We can estimate this number for a case in which N=14, $n_j = 1 \quad j \in \{1 \ldots 7\}$ and $n_j = 2 \quad j \in \{8 \ldots 14\}$. In other words half of the coordinates have two hits and the other half of them have one hit.[3] This example has a total of 64 combinations, with $< N - m >= N - 1.875$. With this approach the number of calculations can be greatly reduced. The actual saving depends on the typical occupancy of roads.

Most of this information is also described in slides 7 to 23 of this talk [5].

References

[1] A. Annovi et al. The fast tracker processor for hadronic collider triggers. In *IEEE Trans. Vol. 48.*

[2] S. Belforte et al. Silicon vertex trigger technical design report. Technical report, 1995. CDF/DOC/TRIGGER/PUBLIC/3108.

[3] E. Brubaker et al. The Fast Track Processor Performances for Rare Decays at the ATLAS Experiment. *IEEE Trans. Nucl. Sci.*, 55:145–150, 2008. `doi:10.1109/TNS.2007.913476`.

[4] Cramer's rule. Available from: `http://en.wikipedia.org/wiki/Cramer's_rule`.

[3]For simplicity we assume that the pixel layers are among those with single hits.

[5] Less calculations for the track fitter (protected). Available from: `http://indico.cern.ch/getFile.py/access?contribId=1&resId=0&materialId=slides&confId=58598`.