FERMILAB-TM-2731-SCD

# Optimizing the `LArSoft GaussHitFinder` Module

Michael H.L.S. Wang[1], Giuseppe Cerati[1], and Boyana Norris[2]

[1]Fermi National Accelerator Laboratory, Batavia, IL 60510
[2]University of Oregon, Eugene, OR 97403

**Abstract**

The steps taken to optimize the `GaussHitFinder` module found in the `LArSoft` toolkit are documented in detail. By replacing the ROOT-based multi-Gaussian fitter with a custom standalone version, speedups of over 8× are achieved.

## 1 Introduction

The work described in this document is part of a broader effort to optimize and modernize the common LArTPC reconstruction code found in the `LArSoft` toolkit. It focuses on the `GaussHitFinder` module because profiling revealed it as one of the major contributors to CPU usage, and it was also a fairly easy target in terms of the relative gains achievable for a given amount of effort. To allow optimization studies without having to run the entire `LArSoft/ART` framework, the very first step taken was to create a standalone version of the code that is both algorithmically identical to, and reproduces, exactly, the results of the implementation in the `LarSoft` framework. All optimization work was then conducted on this standalone version.

Our approach in optimizing code is to begin with purely algorithmic improvements without resorting to exotic architectural features. Only if, after implementing a reasonably efficient algorithm, we still find that there is room or need for more optimizations, do we begin to look into exploiting hardware features that could further improve performance. In what follows, we begin by describing in detail what the standard implementation of `GaussHitFinder` does. After that, we then provide a detailed account of the algorithmic improvements applied to the code and describe the results.

## 2 Detailed Description of the `GaussHitFinder` module

The purpose of `GaussHitFinder` is to find peaks associated with wire hits and determine the parameters of these peaks from the deconvoluted waveforms from the wires in each plane of a Liquid Argon TPC. It consists of three major stages which involve hit candidate finding, merging of hits that are close in proximity, and fitting Gaussian functions to the waveforms to determine the peak parameters. We describe each stage in detail below.

### 2.1 Hit Candidate Finding: `findHitCandidates`

The first step in `GaussHitFinder` is to identify hit candidates. It does this by looking at all ADC readings from a single wire that lie within a given time window of the waveform which is required to have a minimum of five ticks. Starting with the maximum ADC reading within this window, which it requires to be above a given threshold for that particular wire plane, it scans backwards in an attempt to find either the rising edge of a unipolar pulse or a valley between two such pulses. Prior to performing this backward scan, it verifies first that there are at least three ADC readings upstream of the maximum, otherwise, it moves to the forward scanning stage which we will describe later. Going back to our discussion of the backward scan, it looks for a set of three consecutive ADC readings where the central one is bracketed by a downstream one with a value greater than its own and an upstream one with a value greater than or equal to its own. This search is continued all the way to the very beginning of the waveform until a valley or rising edge is found. When this is satisfied, the code recursively calls itself, but this time using a truncated

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

range of the waveform having the same starting point as the previous definition but which now ends at the third point of the moving window in which the valley or rising edge was detected.

After it returns from this first recursive call, it proceeds to the forward scanning stage in which it searches for a valley or falling edge downstream of the peak. The procedure is similar to the backward scan except in reverse. Prior to the forward scan, it checks to see if there are at least three ADC readings downstream of the maximum. If so, it looks for a triplet of consecutive ADC readings where the central one is bracketed by an upstream one that is greater in value and a downstream one that is greater or equal in value. This search is conducted all the way to the end of the waveform until something is found, in which case, having identified both a leading and trailing edge, a new entry is added to the list of hit candidates. The mid-points of the triplets in which the leading and trailing edges were found are taken as the first and last tick, respectively, of the pulse waveform associated with the hit. The locations of these two ticks and that of the peak, including their associated ADC values, and an estimate of the half-width of the pulse are all stored in the hit candidate list. Next, the code recursively calls itself a second time. In this recursive call, a truncated waveform is passed, which begins at the third point of the moving window in which the valley or edge was detected, and ends at the same point as the previous definition of the waveform.

In this recursive manner, all sub-pulses with peaks above the specified threshold for the wire plane are found, located, and added separately to the list of hit candidates.

### 2.2 Merging Candidate Hits: `MergeHitCandidates`

After all the hit candidates are found, this next step tries to merge consecutive hits that are close enough to each other into a single group before passing it to the next stage where hit parameters are extracted by fitting Gaussian functions to the waveform. This way, parameters for the entire group of hits can be determined with a single fit to a sum of Gaussian functions. Two consecutive hits are considered close enough for merging if the first tick of the downstream hit is less than two ticks away from the last tick of the upstream hit. A whole sequence of hits in which this condition is satisfied between adjacent hits are merged into a single group. As soon as an incoming hit's first tick is more than two ticks away from the previous one, it it used to start a new group of merged hits.

### 2.3 Finding Peak Parameters: `findPeakParameters`

As soon as all hit candidates are merged into groups, these groups are then passed on to the peak parameter finding stage. All this stage does is to fit the merged group of hits simultaneously to a sum of Gaussian functions with three independent parameters per function. One Gaussian function is associated with each hit candidate in the merged group. The initial values of the amplitude, mean, and width ($\sigma$) of each Gaussian are estimated from the peak ADC value, its tick position, and the approximate width, respectively, of the hit stored in the hit candidate list. A constrained fit is performed in which the parameters are allowed to vary within a limited range. The amplitude is allowed to vary from its initial value within 0.1 to `AmpRange` (2.0) times its initial value. The mean is allowed to vary from its initial value in either direction within `PeakRange` (2.0) times the estimated hit width, unless the range exceeds the first or last tick of the merged waveform, in which case those limits are imposed. The width is allowed to vary from its initial value from a lower limit of `MinWidth` (0.5 tick) or 10% of the estimated hit width, whichever is larger, up to an upper limit of `MaxWidthMult` (3.0) times the estimated hit width. The goodness of fit parameter, $\chi^2$/DOF is calculated and required to be less than or equal to the maximum value of a `double` type in `C++`.

## 3 Algorithmic Improvements to the `GaussHitFinder` Module

As mentioned in the introduction, a standalone version of the `GaussHitFinder` module was created for the optimization studies. Inline assembly code was inserted that allowed reading the Time Stamp Counter (TSC) registers of the `x86` CPU at various points in the code, for the purpose of measuring the CPU cycles spent in different sections. It was found that the `findPeakParameters` section of the module consumed 98% of the total CPU cycles. Based on this result, the entire algorithmic optimization effort was targeted at this portion of the code.

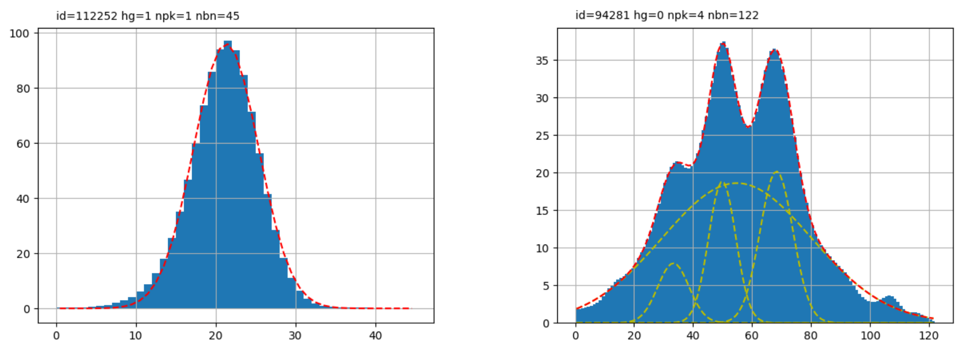

Figure 1: The `findPeakParameters` function fits single or multiple Gaussian functions to the deconvoluted waveform. These plots demonstrate the ability of the new version of the fitter described in this section to peform this task. In the example on the right, the maximum number of peaks is set to 4 and there are also thresholds below which peaks are not considered.

The first thing we attempted was to simply replace the `ROOT/Minuit`-based fitting in `findPeakParameters` with the `levmar` library [1], which consists of C/C++ implementations of the Levenberg-Marquardt optimization algorithm . The goal in trying `levmar`, which is derived from the 80's-era, `FORTRAN`-based `MINPACK` from ANL [2], was to get a quick idea of what gains could be expected by simply switching to a different fitting library. Since the gains were minimal ($\sim$10-20%), we decided to write our own fitting routine from scratch to allow maximum flexibility in optimizing the code. We also stayed with the choice of the Levenberg-Marquardt algorithm since this is a robust and efficient algorithm that has stood the test of time.

## 3.1 Brief Description of Marquardt's Method

The Marquardt algorithm [3] (also referred to as Levenberg-Marquardt), combines the strengths of the method of steepest descent and a Taylor series expansion of the fitting function near the minimum. The former is reliable in that it always finds the minimum no matter what the starting parameters are, but it can be inefficient. The latter, on the other hand, is fast but works reliably only in the vicinity of the minimum. Recall that a least-squares fit amounts to solving the matrix equation given by $\boldsymbol{\beta} = \boldsymbol{\alpha}\ \boldsymbol{\delta}\mathbf{a}$ for the parameter increments $\boldsymbol{\delta}\mathbf{a}$, where $\boldsymbol{\beta}$ is a vector representing the negative gradient of $\chi^2$ and $\boldsymbol{\alpha}$ is the curvature matrix. The Marquardt method combines the best of both worlds by introducing a damping parameter $\lambda$ and modifying the matrix equation to:

$$\boldsymbol{\beta} = (\boldsymbol{\alpha} + \lambda\mathbf{I})\ \boldsymbol{\delta}\mathbf{a}$$

where $\mathbf{I}$ is the identity matrix and, therefore, $\lambda$ only affects the diagonal elements of the curvature matrix. When it is large, the diagonal elements dominate and $\boldsymbol{\delta}\mathbf{a}$ is in the direction opposite that of the $\chi^2$ gradient. When it is small, things revert back to the case derived using a Taylor series expansion. In other words, by adjusting this parameter, one can toggle between the method of steepest descent and the Taylor series method. In Marquardt's original prescription, one starts out with a suitable value for the damping parameter, like $\lambda = 0.01$, and solves the matrix equation for the parameter increments. The basic idea is that if the calculated $\chi^2$ decreases, then $\lambda$ is reduced by some factor $\nu > 1$. On the other hand, if $\chi^2$ increases, $\lambda$ is successively increased by the same factor until $\chi^2$ starts to decrease. A typical value for this factor is $\nu = 10$. See References [4] and [5] for good introductions to Marquardt's method.

## 3.2 Implementation of the Fitting Algorithm

All the code described in this section is available online [6]. The main steering routine for the steps described in this section is the function `mrqdtfit`. As in the standard framework implementation, the deconvoluted waveform is fitted to a multiple Gaussian function

$$G(x) = \sum a_{1i} e^{-\frac{1}{2}\left(\frac{x-a_{2i}}{a_{3i}}\right)^2}$$

where the sum runs over the number of found peaks. Figure 1 shows some examples of the waveform fitted to this function by the new fitter described in this section. Starting with the initial parameter estimates described in Section 2.3, the values of the Gaussian fitting function are calculated at each TDC tick. The residuals or difference, $\Delta y = y^d - y(x)$, between the measured ADC values and the function values are also calculated. These are all done in the function `fgauss`. After this, the initial $\chi_0^2$ is calculated in `cal_xi2` and the derivatives of the fitting function with respect to all parameters are evaluated analytically in `dgauss`. Then, all elements of the $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ matrices are calculated in `setup_matrix` according to:

$$\beta_k = \sum \left\{ \frac{1}{\sigma_i^2} \left[ y_i^d - y(x_i) \right] \frac{\partial y(x_i)}{\partial a_k} \right\}$$

$$\alpha_{jk} = \sum \left[ \frac{\partial y(x_i)}{\partial a_j} \frac{\partial y(x_i)}{\partial a_k} \right]$$

which is based on a first order expansion of the fitting function and where the sums in $i$ run over all data points. In practice, all the $\sigma_i$'s are set to unity, as was done in the standard framework implementation of `GaussHitFinder`.

Once all of the setup steps above are completed, the matrix equation is solved for the parameter increments using Gaussian elimination in `solve_matrix`. The initial value of the $\lambda$ is chosen to be 0.1% of the largest diagonal element of $\boldsymbol{\alpha}$. To save time, a full matrix inversion is not done, and only those calculations necessary to determine the parameter increments are carried out. The first step is to form an augmented $n \times (n+1)$ matrix out of the symmetric $n \times n$ $\boldsymbol{\alpha}$ matrix and the $\boldsymbol{\beta}$ vector. The idea is to diagonalize the $\boldsymbol{\alpha}$ matrix to solve for $\boldsymbol{\delta}\mathbf{a}$. The technique of partial pivoting is also used in order to prevent the diagonal elements from vanishing and to increase numerical stability. The procedure goes through the rows sequentially, starting from the top. As we go through each row, the diagonal element in that row is compared with all other elements in the same column below it to find that with the largest magnitude. In case the largest value is in a row other than the one under consideration, the two rows are swapped.

The fit parameters are then incremented by the solution to the matrix equation. Using the updated parameters, the function values, residuals, and $\chi^2$ are re-evaluated with `fgauss` and `cal_xi2`. To update the damping parameter, we follow the more sophisticated strategy described in Reference [7] rather than Marquardt's original scheme. This new strategy updates the damping parameter in a smooth manner resulting in less erratic behavior and faster convergence. In this strategy, a factor $\nu$ is initially set equal to 2. Then, a *gain factor* is calculated according to

$$\rho = \frac{2\left(\chi_0^2 - \chi^2\right)}{\boldsymbol{\delta}\mathbf{a}^T \left(\lambda \boldsymbol{\delta}\mathbf{a} + \boldsymbol{\beta}\right)}$$

and $\lambda$ is updated based on whether $\rho < 0$ or $\rho > 0$. If $\rho < 0$, the updated parameters are not accepted and the previous values are restored. The damping parameter $\lambda$ and factor $\nu$ are updated according to $\lambda \to \nu\lambda$, followed by $\nu \to 2\nu$. Using the updated value of $\lambda$, the curvature matrix $\boldsymbol{\alpha}$ is re-determined. The matrix equation is solved again for the parameter increments and the entire procedure described in this paragraph is repeated as long as $\rho < 0$.

On the other hand, when $\rho > 0$, the new parameters are accepted, the damping parameter is updated to $\lambda \to \lambda \max\left\{\frac{1}{3}, 1 - (2\rho - 1)^3\right\}$, and $\nu$ is restored to $\nu = 2$. Starting with these updated values, more iterations of the entire procedure described in this section are performed until the change in $\chi^2$ drops below a preset value or the number of iterations exceeds a preset number of trials. When this point is reached, the final set of parameters is used to calculate the full inverse matrix $\boldsymbol{\alpha}^{-1}$ and the parameter errors are determined from the diagonal elements.

## 3.3 Results of the New Marquardt-based Fitter

The new Marquardt-based fitter described above was tested on an input file consisting of over 100k samples. Figure 2 shows that the new fitter is able to reproduce the reconstructed times of the standard ROOT-based fitter quite well. Figure 3 shows distributions of the CPU cycles for both the ROOT-based fitter and the new Marquardt-based fitter. The new fitter is roughly 8× faster than the ROOT-based fitter.

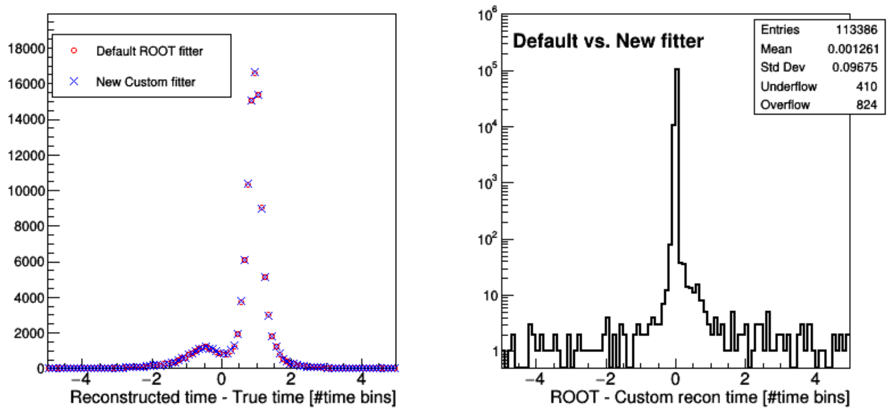

Figure 2: The plot on the left shows the difference between the reconstructed and true time at which the peaks occur. It is shown for both the ROOT-based fitter used in the standard framework version of the code and the new custom Marquardt-based fitter. The plot on the right compares the reconstructed times of the ROOT-based and new Marquardt-based fitters.

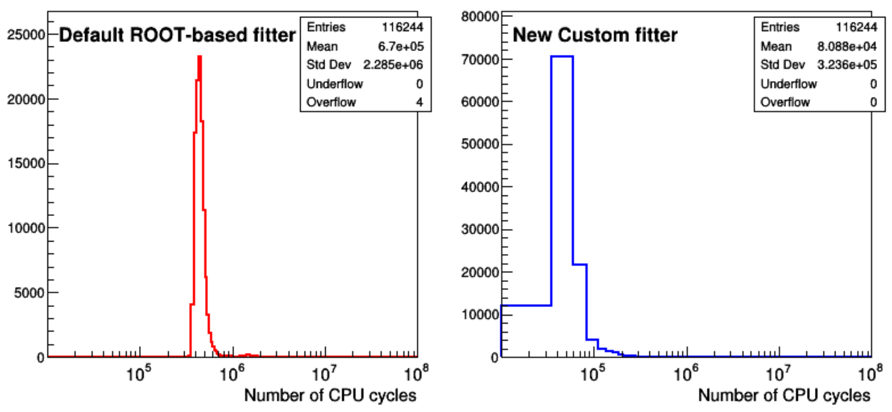

Figure 3: The plots above show the number of CPU cycles it takes to peform the `findPeakParameters` function using the standard ROOT-based fitting (left) and the new Marquardt-based fitting (right). The speedup achieved with the new fitter is over 8×.

## 4 Acknowledgements

We wish to acknowledge Edward P. Hartouni of Lawrence Livermore National Laboratory for introducing us to the Levenberg-Marquardt algorithm many years ago and for sharing his very valuable insight on this algorithm and its implementation.

## References

[1] Lourakis, Manolis. "Levmar : Levenberg-Marquardt Nonlinear Least Squares Algorithms in C/C++." http://users.ics.forth.gr/~lourakis/levmar/.

[2] More, J. J., B. S. Garbow, and K. E. Hillstrom. *User Guide for MINPACK-1.* Report no. ANL-80-74. Argonne National Laboratory. 1980.

[3] Marquardt, Donald W. "An Algorithm for Least-Squares Estimation of Nonlinear Parameters." *Journal of the Society for Industrial and Applied Mathematics* 11, no. 2 (1963): 431-41. doi:10.1137/0111030.

[4] Bevington, Philip R., and D. Keith Robinson. *Data Reduction and Error Analysis for the Physical Sciences.* 3rd ed. McGraw-Hill Education, 2003.

[5] Press, William H., Saul Arno Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing.* A Cambridge: Cambridge University Press, 2016.

[6] Wang, M., G. Cerati, and B. Norris. "lartpcalgo-optimize GitHub repositiory." https://github.com/mhlswang/lartpcalgo-optimize.

[7] Nielsen, Hans Bruun. *Damping Parameter in Marquardt's Method.* Technical paper no. IMM-REP-1999-05. Lyngby, Denmark: Technical University of Denmark. 1-31.