August 2025
galin.bistrev@cern.ch

Supporting Automatic Differentiation in CMS Combine profile likelihood scans

Author: Galin Bistrev
Supervisors: David Lange, Jonas Rembser, Vassil Vassilev

Keywords: Automatic Differentiation, CMS Combine, RooFit, Statistical Modelling

Abstract

One common task in data analysis is the determination of model parameters in order to describe a given dataset. This can be done in various ways, one of which is the discrete profiling method, which was developed as part of the analysis of data at the CMS experiment. In essence, the discrete profiling method is a generalization of the profile likelihood method: while the latter handles continuous nuisance parameters by profiling over them, the former extends the approach to also include discrete parameters that switch between different candidate models. However, datasets are continually growing in size and the number of parameters increases, thus making statistical analysis computationally expensive. Implementation of Automatic Differentiation (AD) would significantly reduce the computational cost of discrete profiling, thus enabling a more efficient data analysis. In the following summer student report, a documentation of the discrete profiling method is provided along with a report on the work done in order to support AD in Combine—the main software for statistical analysis used in the CMS experiment.

1 Automatic Differentiation and Discrete Profiling

In high-energy physics, it is necessary to determine the best-fit model parameters obtained from large datasets. These parameters are often determined using the maximum likelihood method, where models for both signal and background processes—either parametric or empirical models derived from Monte Carlo simulations—are built and fitted to the observed data. However, in some cases, the exact form of these underlying models is not directly inferable. This introduces an additional source of uncertainty: the choice of the functional form. A common way to estimate this systematic uncertainty is to fit several plausible functions and examine the spread in the resulting parameter values. These approaches, however, often involve a degree of arbitrariness. That is why, following the discovery of the Higgs boson, the CMS collaboration developed the *discrete profiling method*, which systematically treats background parametrization while minimizing arbitrariness. This method involves likelihood minimization over models containing thousands of free parameters distributed across hundreds of likelihood components. Efficient minimization requires accurate gradients with respect to all parameters, which can be obtained either numerically or via Automatic Differentiation (AD).

Numerical differentiation computes derivatives by varying one parameter at a time and reevaluating the likelihood. RooFit [1] optimizes this process by caching intermediate results, so that only affected components are recomputed. However, bookkeeping becomes costly as the number of parameters grows, leading to a significant increase in computational time. In contrast, AD evaluates exact derivatives efficiently by applying the chain rule to the function's computation graph, avoiding unnecessary recalculations.

For a function $y = f(g(x))$ decomposed into intermediate steps

$$w_0 = x, \quad w_1 = g(x), \quad w_2 = f(g(x)) = y, \tag{1}$$

AD defines two recursive modes:

$$\frac{\partial w_i}{\partial x} = \frac{\partial w_i}{\partial w_{i-1}} \cdot \frac{\partial w_{i-1}}{\partial x}, \quad \frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial w_{i+1}} \cdot \frac{\partial w_{i+1}}{\partial w_i}. \tag{2}$$

Forward accumulation propagates derivatives from the input outward, while reverse accumulation propagates from the output inward. Reverse mode is particularly efficient when differentiating functions with many input parameters, as is common in high-energy physics analyses.

RooFit implements AD through `Clad` [2], a plugin for the Clang C/C++ compiler. RooFit transforms its models into state-independent C++ code using their `translate` functions and combines them into a single representation of the full model. This combined code is then later compiled using the C++ interpreter Cling and differentiated by Clad, thus providing significant performance gains over numerical differentiation.

The benefit of AD becomes particularly clear in the context of profile likelihood scans. Each scan requires many repeated minimizations, one for each value of the parameter of interest, with the continuous nuisance parameters floating in every fit. Although AD introduces an initial overhead for generating and compiling the gradient code, this cost is incurred only once. The same compiled gradient can then be reused for every minimization in the scan, amortizing the setup time.

The main statistical analysis framework in CMS, `Combine` [3], encodes the probability density for observed data parameterized by $\vec{\Phi} = (\vec{\mu}, \vec{\nu})$, where $\vec{\mu}$ are parameters of interest and $\vec{\nu}$ are nuisance parameters. For primary observables $\vec{x}$ and auxiliary observables $\vec{y}$, the model factorizes as:

$$p(\vec{x}, \vec{y}; \vec{\Phi}) = p(\vec{x}; \vec{\mu}, \vec{\nu}) \prod_k p_k(y_k; \nu_k), \tag{3}$$

and the likelihood for a dataset is:

$$L(\vec{\Phi}) = \prod_d p(\vec{x}_d; \vec{\mu}, \vec{\nu}) \prod_k p_k(y_k; \nu_k), \tag{4}$$

where d indexes the dataset entries. Statistical inference is performed using the profile negative log-likelihood (NLL), $-\ln L(\vec{\mu}, \hat{\vec{\nu}}(\vec{\mu}))$, where $\hat{\vec{\nu}}(\vec{\mu})$ denotes the values of nuisance parameters maximizing the likelihood for fixed $\vec{\mu}$. The profile likelihood plays a central role in statistical inference, since it allows the extraction of parameter uncertainties directly from the likelihood shape. In particular, when the likelihood is non-parabolic—as is often the case in realistic analyses—the profile likelihood provides a reliable way to compute confidence intervals for the parameters of interest (POIs). The discrete profile likelihood extends this concept by incorporating systematic effects from discrete choices of the model, ensuring that such uncertainties are also reflected in the final POI intervals. In Combine, minimization of the NLL is handled by the `CascadeMinimizer`, which utilizes the discrete profiling method. For each discrete configuration, the continuous parameters are minimized in the usual way using RooFit's interface to the `Minuit2` minimizer.

Discrete profiling treats model choice, such as the functional form of the background, as a discrete nuisance parameter. Separate likelihood fits are performed for each candidate model, profiling continuous nuisance parameters in the usual way by minimizing the likelihood with `Minuit2`. At each value of the parameter of interest (POI), the final profile likelihood, in which the floating discrete parameters are also considered, is constructed as the envelope of all the individual profile likelihood curves, where only the continuous parameters are left floating. That is, at each value of the parameter of interest, the minimum value of the NLL across all discrete choices is used. This envelope curve effectively incorporates the uncertainty associated with the discrete choice of a model into the profile likelihood. Another feature of this method is the addition of a correction factor to the NLL to account for the number of free parameters in each model, preventing more complex functions from being favored simply due to their flexibility and ensuring a fair comparison across models of different complexity.

Figure 1 illustrates this principle. The black curve shows the full profile likelihood for a POI named x, the blue curve corresponds to fixing nuisances at their best-fit values, and red dashed curves represent other fixed choices. The green dashed envelope recovers the full profile, including systematic effects.

By treating the choice of model as a source of systematic uncertainty, discrete profiling ensures a consistent handling of uncertainties in statistical inference. However, it is computationally intensive when many nuisance parameters are involved. AD mitigates this by accelerating gradient evaluation during likelihood minimization, improving the efficiency of discrete profiling in Combine. The implementation of AD in Combine was the primary focus of this work.

2 Implementing Automatic Differentiation in CMS Combine via RooFit and Code Generation

In order for AD to be implemented successfully in Combine, the logic of discrete profiling used in the `CascadeMinimizer` must be imported into RooFit's `RooMinimizer`. By doing this, Combine can reuse the AD-enabled minimization of the NLL, performed by `RooMinimizer`, without needing to implement a separate AD mechanism in its own framework.

The first step was to implement the switching of discrete PDF indices needed for the discrete profiling procedure. In `Combine`, this is done by the class `RooMultiPdf`. `RooMultiPdf` is a special type of PDF object, recently introduced in RooFit in the scope of the presented work. This object holds a list of individual PDFs and can switch between them based on a set `RooCategory` index. Each value of the category corresponds to one PDF in the list. The class

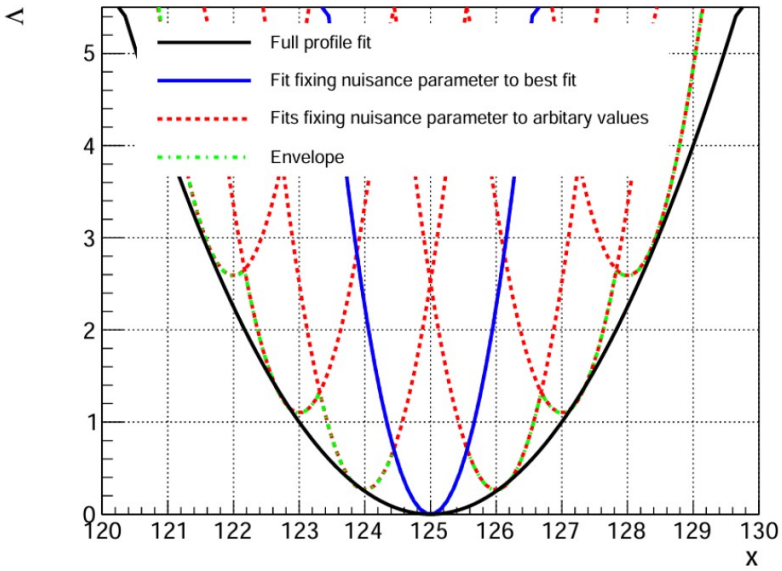


Figure 1: Illustration of the discrete profiling method. The envelope of fixed-nuisance curves (green) reproduces the full profile likelihood (black). Picture taken from [4]

also provides a correction factor that implements the statistical penalty applied during the profiling. By default, the correction to the NLL is 0.5 times the number of free parameters. This is stored in the variable cFactor defined inside the `RooMultiPdf` class. In practice, the value of the correction factor depends on the application and on the amount of data available. During discrete profiling, Combine loops over all combinations of PDF indices in one or more `RooMultiPdf` objects. For each combination, it sets the indices, minimizes the likelihood over the continuous parameters, and records the NLL. To enable the use of this new class for AD-supported discrete profiling, its implementation has been designed to be supported by the RooFit code generation engine, dubbed `codegen`.

After implementing the `RooMultiPdf` class in `codegen`, the algorithm of discrete profiling was ported from `CascadeMinimizer.cxx` to the function `RooMinimizer::fitFCN()` in `RooMinimizer.cxx`. The first feature to be implemented was a class included in the namespace of `RooMinimizer.cxx` called `FreezeDisconnectedParametersRAII`, which automatically identifies and freezes parameters that are disconnected from the likelihood computation graph during minimization. This ensures that the minimizer only varies parameters that actively contribute to the NLL.

The next feature is the function `generateOrthogonalCombinations`. This function creates a list of index combinations by first initializing the base configuration with all indices set to zero and then generating new combinations by changing only one category at a time.

Along with `generateOrthogonalCombinations`, there is also the function `reorderCombinations`. It takes the set of indices generated by `generateOrthogonalCombinations` and reorders them so that the combinations that differ the least from the current best one are evaluated first.

In RooFit, `RooCategory` objects are the nodes in the computation graph that represent discrete parameters. Before the two previously described functions are executed inside `RooMinimizer::fitFCN()`, the code loops over all parameters and collects those `RooCategory` objects that are not set constant into a vector. These correspond to the discrete nuisance parameters that will be profiled during the fit. If no floating `RooCategory` parameters are found, the function falls back to a standard continuous minimization: the minimizer is initialized, disconnected parameters are frozen, and all continuous variables are optimized in the usual way.

After this check, if discrete parameters are present, the algorithm proceeds by generating combinations of their index values via `generateOrthogonalCombinations`. The discrete category indices are then updated and reordered using `reorderCombinations`. Then the continuous parameters are minimized while the discrete indices are temporarily frozen. This ensures that the minimizer only adjusts the continuous parameters for the currently selected combination. Once the minimization is complete for a given combination, the resulting NLL value is stored. The combination is also marked as "tried" to avoid repeating the same evaluation. If the combination produces a lower NLL than any previously tested combination, it is recorded as the current best, after which the whole procedure described above is repeated until no better value for NLL is found.

The implementation can be summarized as follows:

- **Generate combinations and iterate until convergence**:

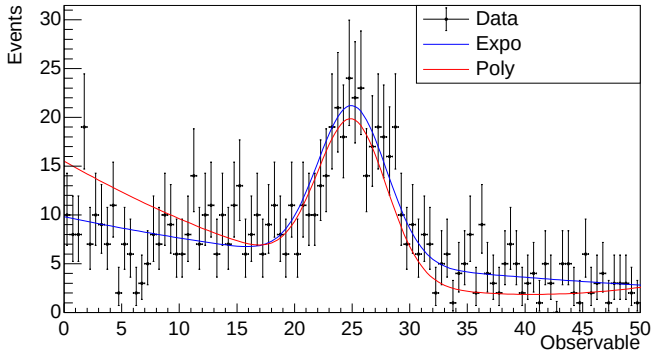
```
    while (improved) {
        improved = false;

        auto combos = generateOrthogonalCombinations(maxIndices);
        reorderCombinations(combos, maxIndices, bestIndices);

        for (const auto &combo : combos) {
            if (tried.count(combo)) continue;
```

- **Apply discrete combination, freeze, minimize, and update best combination**:

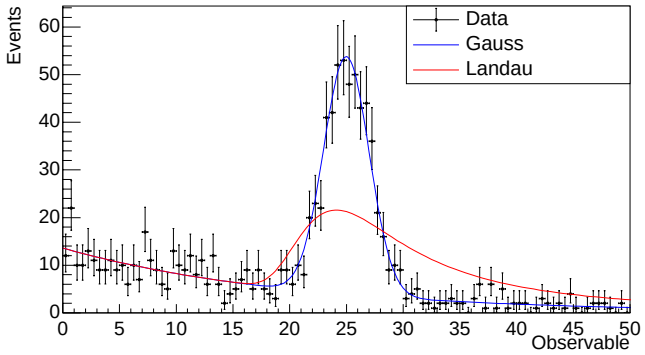
```
for (size_t i = 0; i < nPdfs; ++i) pdfIndices[i]->setIndex(combo[i]);

std::vector<bool> wasConst(nPdfs);
for (size_t i = 0; i < nPdfs; ++i) {
    wasConst[i] = pdfIndices[i]->isConstant();
    pdfIndices[i]->setConstant(true);
}
FreezeDisconnectedParametersRAII freeze(this, *_fcn);
_minimizer->Minimize();

for (size_t i = 0; i < nPdfs; ++i) pdfIndices[i]->setConstant(wasConst[i]);

double val = _minimizer->MinValue();
tried.insert(combo);
nllMap[combo] = val;

if (val < bestNLL) {
    bestNLL = val;
    bestIndices = combo;
    improved = true;
}
```

This implementation ensures that discrete profiling efficiently finds the best combination of discrete nuisance parameters while continuously optimizing the continuous parameters, ultimately identifying the combination that minimizes the total NLL. The underlying assumption in this approach is that the different discrete parameters are only weakly correlated. For this reason, the algorithm explores combinations where only a single discrete parameter differs from the current best choice, updates the best NLL found, and then repeats the procedure. The full code of the described algorithm can be found in `RooMinimizer.cxx`, which is in ROOT's official repository.

A tutorial showcasing the use of the new `RooMultiPdf` object and the discrete profiling method has been developed. It demonstrates the construction of different models composed of multiple `RooMultiPdf` objects across two different categories (Figure 2) with a shared parameter (in that case the mean of the two Gaussians).



(a) Category 0: `RooMultiPdf` combining an exponential and Chebyshev polynomial, mixed with an extra Gaussian via `RooAddPdf` with `frac0`.



(b) Category 1: `RooMultiPdf` combining a Gaussian and Landau, mixed with an extra exponential via `RooAddPdf` with `frac1`.

Figure 2: Visualization of the two categories used in the simultaneous model. Each plot shows data points, the combined model, and a selected PDF component.

From those models, different toy datasets are generated. These datasets are combined into one, which is then used for the construction of an NLL object via the usage of `codegen`:

```
std::unique_ptr<RooAbsReal> nll(simPdf.createNLL(combined,EvalBackend("codegen")));
```

Once the NLL object is created, the continuous parameters are minimized for each discrete configuration using `minim.minimize("Minuit2","Migrad")`. A discrete profiling scan over the mean is then carried out across all PDF index combinations, yielding the result shown in Figure 3. The tutorial is available in the official ROOT repository under the name `rf619_discrete_profiling.C`.

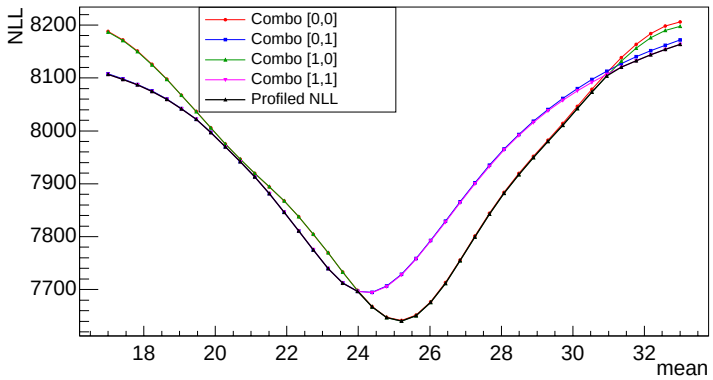


Figure 3: NLL value as a function of the mean value for each combination.

A benchmark was developed to evaluate the performance of discrete profiling using automatic differentiation (AD). In principle, AD can provide a significant speedup for profile likelihood scans, but for this particular benchmark, the observed performance falls short due to unnecessary overhead in the gradient generated by Clad. Further development in Clad is needed to generate fully efficient gradients for RooFit likelihoods.

3 Conclusion

In the present summer student report, the author presented an overview of AD and the discrete profiling method, describing how AD can efficiently compute exact derivatives for models with thousands of parameters, and reported on the implementation of AD within the CMS `Combine` framework.

The discrete profiling method presents a systematic approach for estimating model parameters by minimizing the likelihood across multiple candidate models while simultaneously profiling continuous nuisance parameters, thus reducing arbitrariness in model choice.

Supporting AD of likelihood functions built with the Combine framework is key to optimizing the performance of the discrete profiling algorithm. By implementing the core principle of discrete profiling in `RooMinimizer.cxx`, `Combine` can successfully use the `codegen` backend for NLL evaluation, which is already available within the RooFit framework. This will allow for more efficient data analysis with large numbers of nuisance parameters, which is crucial in high-energy physics.

References

[1] W. Verkerke and D. P. Kirkby, "The RooFit Toolkit for Data Modeling," in Proc. CHEP 2003 and PHYSTAT 2005, La Jolla and Oxford, 2003, pp. 186–189, arXiv:physics/0306116.

[2] G. Singh, J. Rembser, L. Moneta, D. Lange, and V. Vassilev, "Making Likelihood Calculations Fast: Automatic Differentiation Applied to RooFit," EPJ Web Conf. 295 (2024) 06014, https://doi.org/10.1051/epjconf/202429506014

[3] A. Hayrapetyan et al., "The CMS Statistical Analysis and Combination Tool: Combine," Comput. Softw. Big Sci. 8 (2024) 19, https://doi.org/10.1007/s41781-024-00121-4

[4] P. D. Dauncey, M. Kenzie, N. Wardle, and G. J. Davies, "Handling Uncertainties in Background Shapes: The Discrete Profiling Method," JINST 10 (2015) P04015, https://doi.org/10.1088/1748-0221/10/04/P04015