Supporting Automatic Differentiation in CMS Combine profile likelihood scans

Galin Bistrev (CERN Summer Student)

Supervisors:Jonas Rembser ,Vassil Vassilev , David Lange

Compiler Research Meeting – 25.09.2025

Likelihoods in High Energy Physics

Likelihoods describe the probability of observing data given model parameters (POIs + nuisance parameters).

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

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

Profile likelihoods optimize nuisance parameters for each fixed value of the parameters of interest.

$$-\ln L(\vec{\mu}, \hat{\hat{\vec{\nu}}}(\vec{\mu}))$$

The **profile NLL shape** is used to extract uncertainties and build confidence intervals.

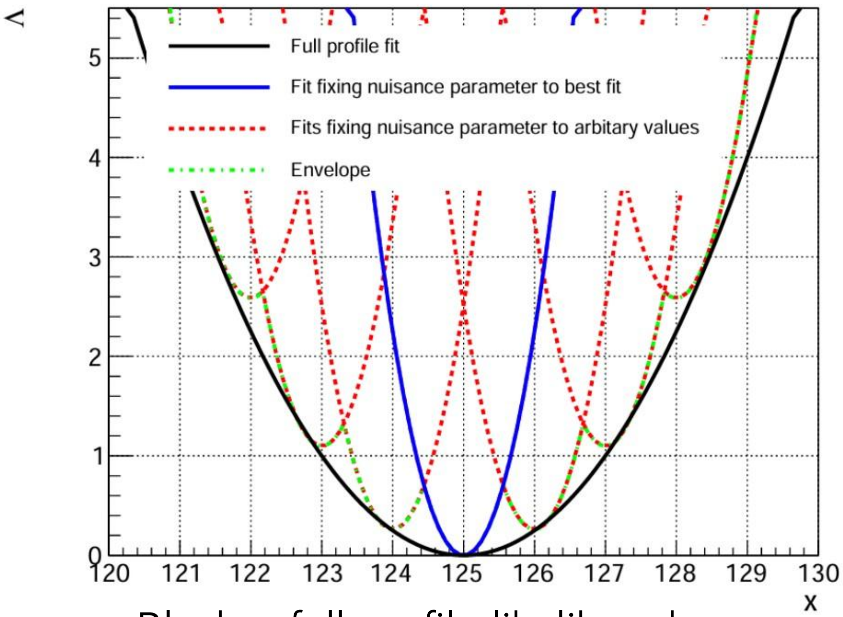
$\vec{x}$	primary observables	$\vec{\mu}$	parameters of interest
$\vec{y}$	auxiliary observables	$\vec{\nu}$	nuisance parameters

Discrete Profiling in CMS Combine

Discrete profiling: treats model choice (e.g. background function) as a **discrete nuisance parameter**, building the **profile likelihood as an envelope** over NLL curves from all models.

This method incorporates a correction factor in the NLL, proportional to the number of free parameters in each model, to prevent a more complex models from being favored solely due to their flexibility and to ensure a fair comparison across models of varying complexity

The method captures **systematic uncertainty from model choice** but is computationally heavy; **Automatic Differentiation (AD)** would accelerate minimization in Combine.



Black = full profile likelihood, blue = best-fit nuisances, red = fixed alternative choices, green = envelope curve including systematics

Automatic Differentiation in RooFit

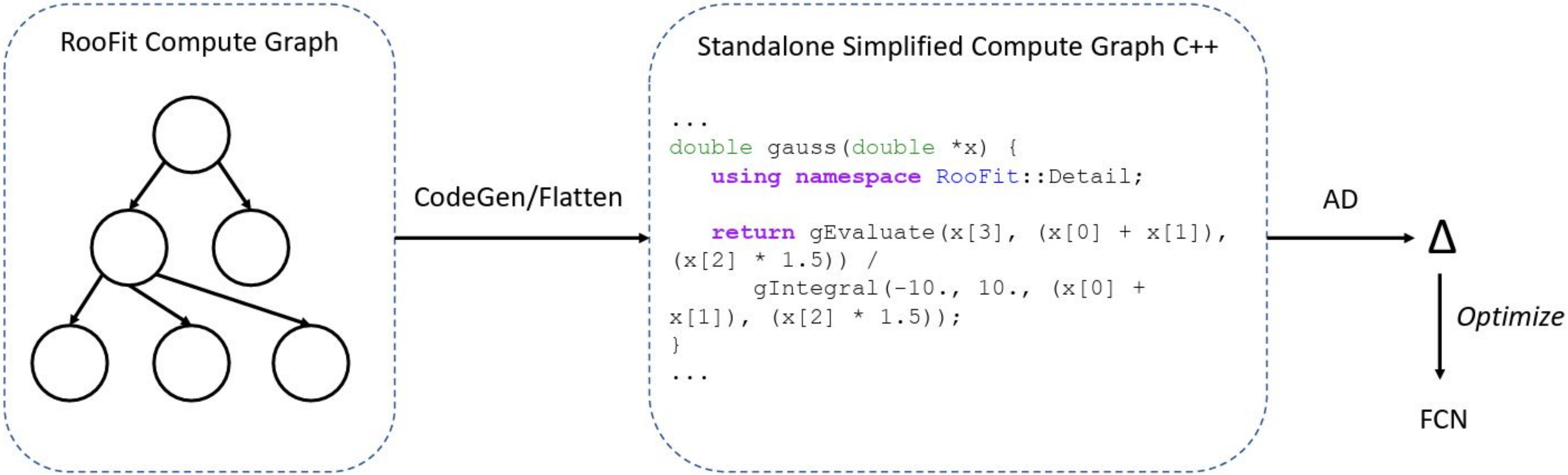
Automatic Differentiation (AD) - set of techniques to evaluate the exact derivative of a computer program:

- employs the chain rule to decompose the compute graph into sequence of elementary operations
- Each scan requires many minimizations with floating nuisances; AD adds a one-time code generation overhead, but the compiled gradient is reused in all fits, amortizing the cost

Numerical Differentiation - set of techniques to approximate the derivative of a function using finite differences:

- replaces derivatives with difference quotients
- varying one parameter at a time and reevaluating the likelihood.

Automatic Differentiation in RooFit

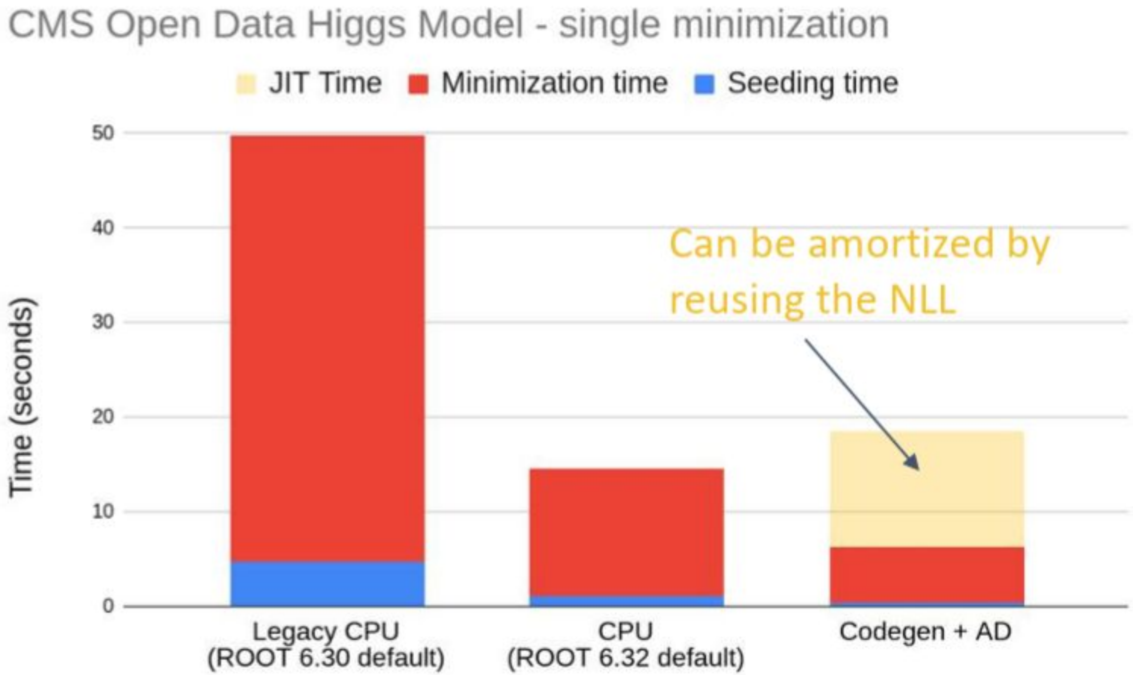


```
...
double gauss(double *x) {
   using namespace RooFit::Detail;

   return gEvaluate(x[3], (x[0] + x[1]),
(x[2] * 1.5)) /
      gIntegral(-10., 10., (x[0] +
x[1]), (x[2] * 1.5));
}
...
```

```
pdf.fitTo(data, RooFit::EvalBackend("codegen"))
pdf.createNLL(data, RooFit::EvalBackend("codegen"))
```

Automatic Differentiation in RooFit



01-October-2024 2 V. Vassilev et al -- Automatic Differentiation in RooFit – CMS ML Town Hall

Support for AD in CMS Combine-RooMultiPdf

Discrete profiling - main minimization algorithm in Combine - CascadeMinimizer.cxx

To enable AD in Combine, the discrete profiling logic from CascadeMinimizer is integrated into RooFit's RooMinimizer, allowing Combine to reuse AD-enabled NLL minimization without duplicating mechanisms.

Statistical penalty applied during the profiling. By default, the correction to the NLL is 0.5 times the number of free parameters.

First step - importing the class RooMultiPdf - switching of discrete PDF indices

```
inline RooAbsPdf *getCurrentPdf() const {
return getPdf(getCurrentIndex()); }
```

Returns the currently active Pdf based on the RooCategory index

```
inline RooAbsPdf *getPdf(int index) const { return
static_cast<RooAbsPdf *>(c.at(index)); }
```

Returns the PDF at the specified position in the internal list. The index is explicitly provided to select a particular PDF.

```
double cFactor = 0.5;
```

Support for AD in CMS Combine-Codegen implementation

The next step is to design the implementation to be supported by the RooFit code generation engine, dubbed "codegen"- **generate optimized C++ code** for evaluating a RooMultiPdf

In MathFunc.h

```
inline double multipdf(int idx, const double* pdfs) {
    return pdfs[idx];
}
```

multipdf()- utility function used in RooFit for handling RooMultiPdf objects during **code generation**. Its purpose is to **select the value of a PDF from an array of precomputed PDF values** based on the active discrete index.

In CodegenImpl.cxx

```
int numPdfs = arg.getNumPdfs();
```

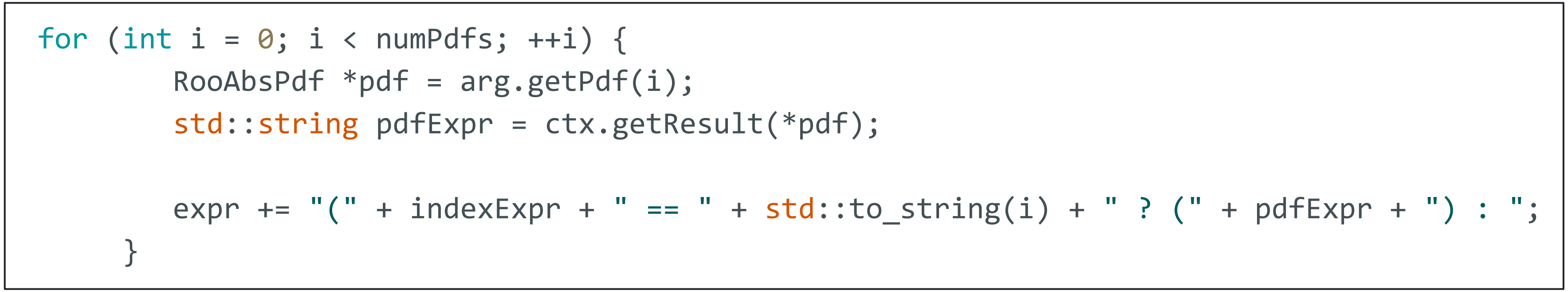
```
  if (numPdfs > 2) {
      ctx.addResult(&arg,
ctx.buildCall(mathFunc("multipdf"),
arg.indexCategory(), arg.getPdfList()));

      std::cout << "MathFunc call used\n";
 }
```

For RooMultiPdf objects with more than two PDFs, the code tells RooFit to use the **multipdf()** function to select the active PDF in the generated C++ code, making it compatible with AD.

Support for AD in CMS Combine-Codegen implementation

Alternatively for numPdfs < 2 , we use ternary expressions for more efficient generation instead of **multipdf()**



```
for (int i = 0; i < numPdfs; ++i) {
        RooAbsPdf *pdf = arg.getPdf(i);
        std::string pdfExpr = ctx.getResult(*pdf);

        expr += "(" + indexExpr + " == " + std::to_string(i) + " ? (" + pdfExpr + ") : ";
     }
```

The final expr string is a C++ representation of the RooMultiPdf that will choose the correct PDF depending on the current discrete index.

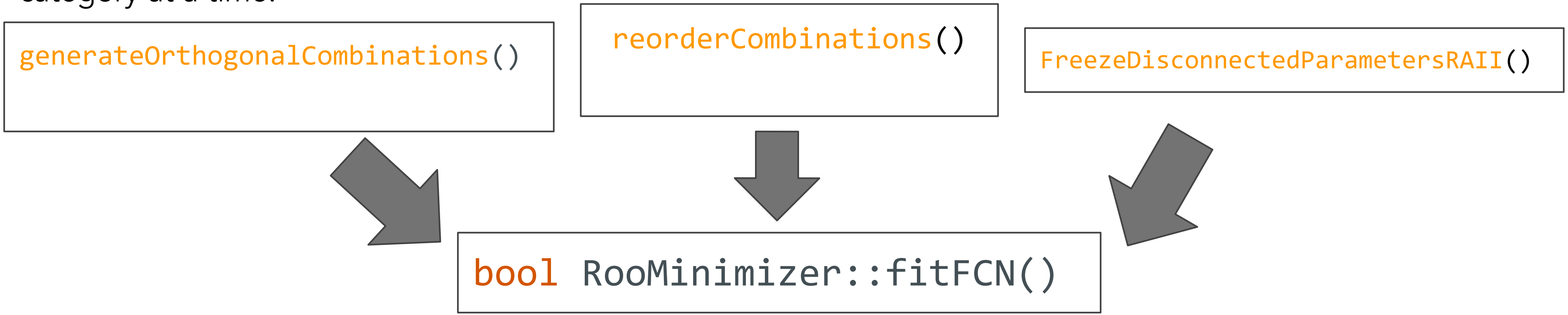
Support for AD in CMS Combine - Minimization logic

In order to import the discrete profiling algorithm in RooFit key pieces of logic and code from CascadeMinimizer.cxx and utils.cc from Combine have been imported into RooMinimizer.cxx.

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.

Adjusts each combination of indices generated by `generateOrthogonalCombinations()` by shifting indices such that the central combination is the one with the currently knows best NLL value.

Freezes parameters that are disconnected from the likelihood computation graph (sigma for gauss , lambda for expo , etc.)



`generateOrthogonalCombinations()`

`reorderCombinations()`

`FreezeDisconnectedParametersRAII()`

`bool RooMinimizer::fitFCN()`

the internal function-callable likelihood evaluator used during minimization

Support for AD in CMS Combine - Minimization logic

```
RooMinimizer const *minimizer
```

```
RooAbsMinimizerFcn const &fcn
```

input

```
FreezeDisconnectedParametersRAII()
```

Freeze disconnected parameters

```
generateOrthogonalCombinations()
```

input to

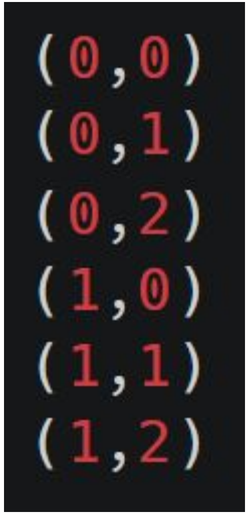
```
std::vector<int> maxValues {2,3};
```

0,1

RooMultiPdf number

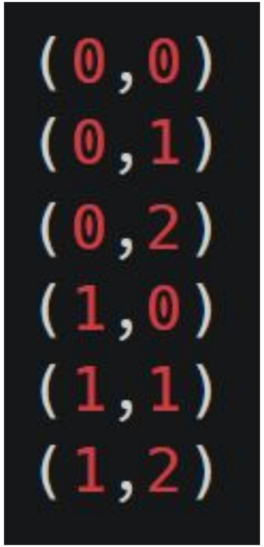
0,1,2

sub-Pdf number



```
(0,0)
(0,1)
(0,2)
(1,0)
(1,1)
(1,2)
```

Support for AD in CMS Combine - Minimization logic



```
(0,0)
(0,1)
(0,2)
(1,0)
(1,1)
(1,2)
```

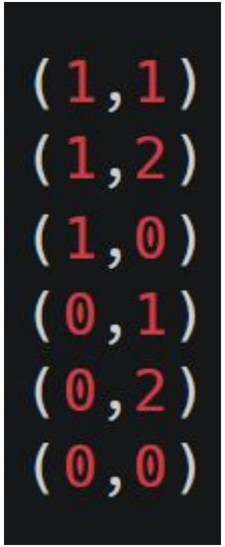
stores

```
std::vector<std::vector<int>> &combos
```

input

```
reorderCombinations()
```

For example best NLL value is (1,1)



```
(1,1)
(1,2)
(1,0)
(0,1)
(0,2)
(0,0)
```

input

```
const std::vector<int> &max
```

maximum size for each index

```
const std::vector<int> &base
```

bias the next set of combinations toward the current best combo

Support for AD in CMS Combine - Minimization logic

Before discrete profiling - Non-constant RooCategory objects are collected as discrete nuisances to profile. If none exist, a standard continuous minimization is performed on the remaining free variables.

```
  if (nPdfs == 0) {
      coutI(Minimization) << "[fitFCN] No discrete parameters, performing continuous minimization only" << std::endl;
      FreezeDisconnectedParametersRAII freeze(this, *_fcn);
      bool isValid = _minimizer->Minimize();
      if (!_result)
         _result = std::make_unique<FitResult>();
      fillResult(isValid);
      if (isValid)
         updateFitConfig();
      return isValid;
   }
```

Support for AD in CMS Combine - Minimization logic

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()`. The combination is also marked as “tried” to avoid repeating the same evaluation:

```
while (improved) {
   improved = false;
   auto combos = generateOrthogonalCombinations(maxIndices);
   reorderCombinations(combos, maxIndices, bestIndices);

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

Support for AD in CMS Combine - Minimization logic

For every possible combination of PDF choices, the algorithm sets the indices accordingly. Then it freezes the category parameters so they don’t change during minimization, runs the minimizer to obtain the NLL value, and restores the categories to their original state. The minimized NLL value is stored for this combination. If it improves on the current best NLL, the best value and its corresponding indices are updated (pseudo code):

```
for each combination of PDF indices:
    set PDF indices according to combination

    //Freeze categories during continuous minimization
    store current constant state of each index
    set all índices constant

    freeze disconnected parameters
    run minimization

    restore original constant state of indices

    val = minimized NLL value
    save val for this combination

    if val < bestNLL:
        bestNLL = val
        bestIndices = combination
```

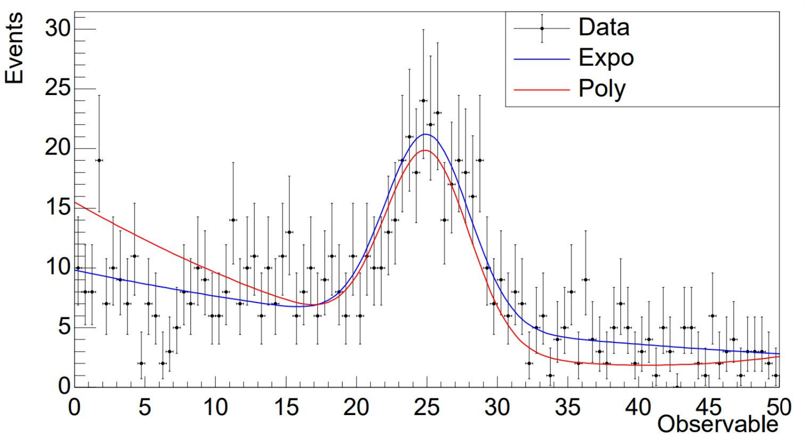
Support for AD in CMS Combine - Results

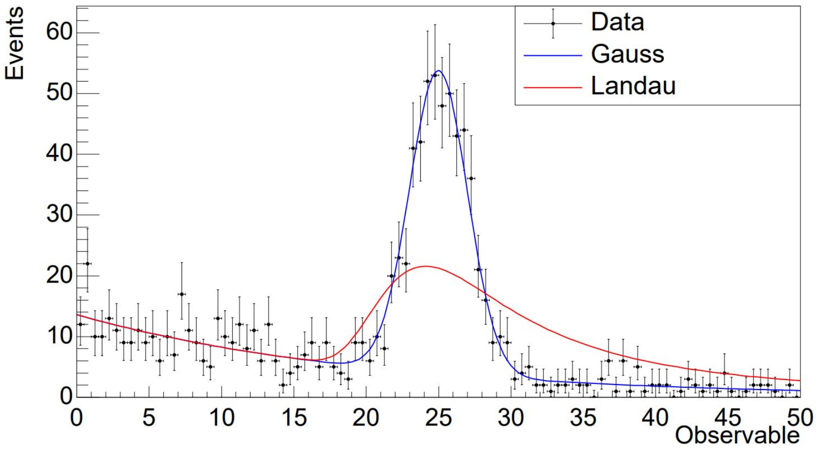
A tutorial in RooFit was created showcasing the construction of different models composed of multiple RooMultiPdf objects across two different categories with a shared parameter (mean)

Tutorial: rf619_discrete_profiling.C
rf619_discrete_profiling.py

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

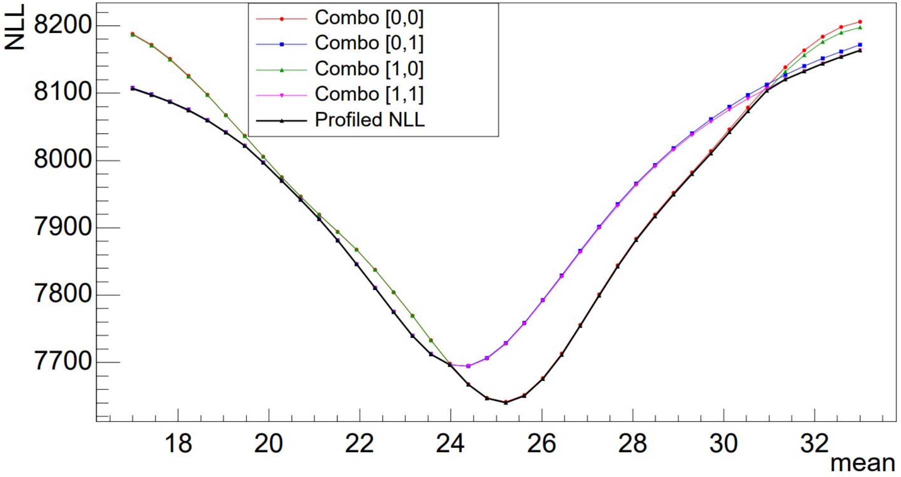
Category 0 (top): RooMultiPdf combining an exponential and Chebyshev polynomial, mixed with an extra Gaussian via RooAddPdf with frac0.





Support for AD in CMS Combine - Results

In the tutorial a discrete profiling scan over the mean is then carried out across all PDF index combinations.The black curve on the figure shows the combination that gives the best NLL for a given mean value.



```
std::unique_ptr<RooAbsReal> nll1(simPdf.createNLL(*data, EvalBackend("codegen")));
RooMinimizer minim(*nll1);
```

```
minim.minimize("Minuit2", "Migrad");
```

Builds the negative log-likelihood function from the model and the data, then hands that function to a minimizer object **minim()** that will adjust the model parameters to find the best fit.

Support for AD in CMS Combine - Conclusions

Conclusions:

- The method was successfully implemented in the RooFit environment and it works with codegen. It is also documented in [CDS]()

- Benchmark shows that observed performance falls short due to unnecessary overhead in the gradient generated by Clad.

An issue that shows the benchmark along with the problem in Clad is presented at:https://github.com/vgvassilev/clad/issues/1521

Assignment for Petro Zaritsky

Huge overhead when differentiating loops with independent iterations #1521

The conceptual solution in the issue involves moving the loop body into a separate function (`double roo_inner_func()`) and computes loop-independent constants outside. This lets Clad differentiate only a single iteration, reducing overhead and making AD much faster. In RooFit, however, this is hard to implement due to many constants defined outside the loop.

Support for AD in CMS Combine - Future work

- Further benchmarking is needed in order to estimate the advantage of AD.
- Additional optimization of Clad , would be needed so that there is no unnecessary overhead .
- The logic implemented in RooMinimizer should be tested for different models in order to see minimizer response.
- Extend doxygen documentation of RooMinimizer to describe treatment of discrete
parameters.
- Test if the implementation of discrete profiling works also inside CMS Combine , replacing their implementation in CascadeMinimizer.cxx

Thank you !