Making Likelihood Calculations Fast: Automatic Differentiation Applied to RooFit

Garima Singh (Princeton University), Jonas Rembser (CERN), Lorenzo Moneta (CERN), David Lange (Princeton University), Vassil Vassilev (Princeton University)

compiler-research.org

This project was supported in part by the NSF (USA) Grant OAC-1931408 and NSF (USA) Cooperative Agreement OAC-1836650.

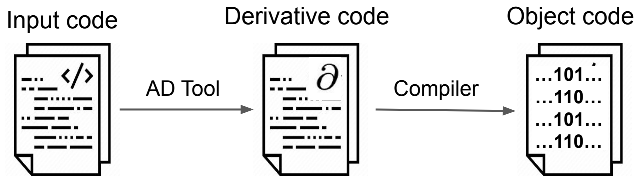
Introduction

Source Code Transformation Based Automatic Differentiation

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

- Faster than numerical differentiation - scales better for problems with large number of parameters.
- More accurate than numerical differentiation - fewer numerical errors!

Source code transformation based AD synthesizes derivative code from the internal representation of the target program.



[Clad](https://github.com/vgvassilev/clad)[1], a compiler based source-code-transformation AD tool. Clad inspects the internal compiler representation of the target function to generates its derivative.

[1] : https://github.com/vgvassilev/clad

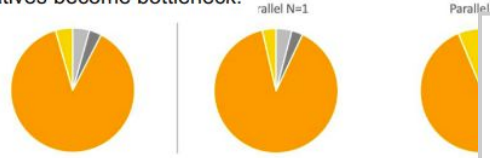
Motivation

Why AD?

- One goal - Make RooFit Faster. Results from a Higgs-combination fit:

	serial old	parallel N=1	parallel N=2	parallel N=4	parallel N=8	parallel N=16
setup_roofit	313	327	315	315	312	327
minuit_init	230	231	231	231	231	231
gradient_calc	6289	7102	3734	1997	1107	879
line_search	323	287	287	287	287	287

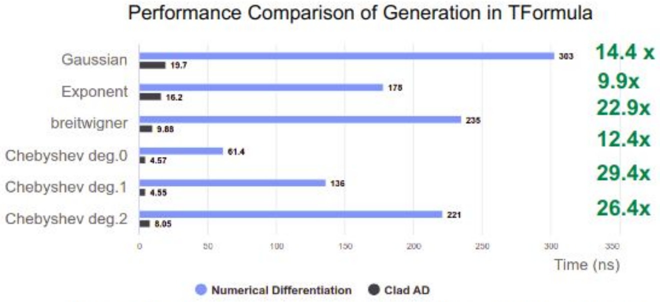
Derivatives become bottleneck!



ICHEP 2022 - Zeff Wolffs - https://agenda.infn.it/event/28874/contributions/169205/a

- Good results, but still use numeric
- Potential next step – use Automati

- We have seen some promising results (in ROOT) already!



TFormula benchmarks of gradient generation time from numerical differentiation and clad AD.

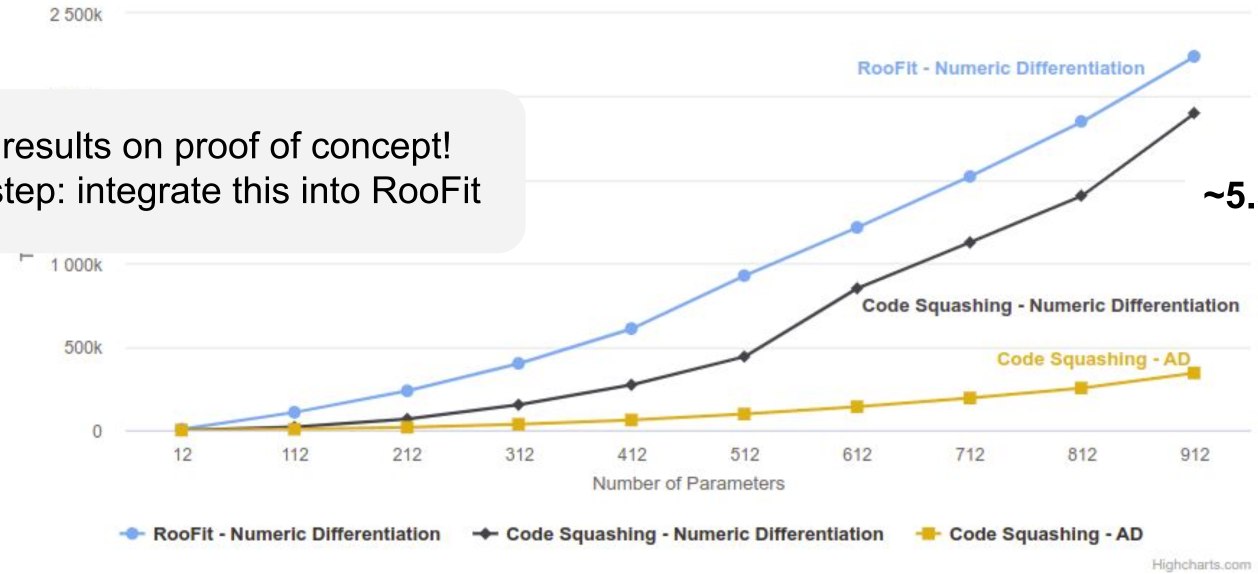
TF1 based benchmarks. TF1 is the TFormula fitting interface for fitting histograms.

Image ref: Automatic Differentiation of Binned Likelihoods With Roofit and Clad - *Garima Singh, Jonas Rembser, Lorenzo Moneta, Vassil Vassilev*, ACAT 2022

Motivation

Okay, but why AD in RooFit????

Performance comparison AD vs numerical differentiation on hf_001 inspired example



Great results on proof of concept!
Next step: integrate this into RooFit

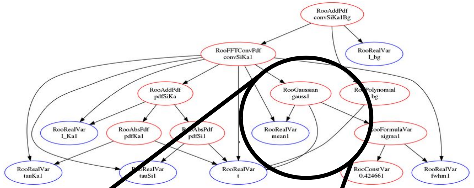
~5.5x speedup

Singh, G., Rembser, J., Moneta, L., Lange, D., & Vassilev, V. (2023). Automatic Differentiation of Binned Likelihoods With Roofit and Clad. *ArXiv [Cs.MS]*. Retrieved from http://arxiv.org/abs/2304.02650

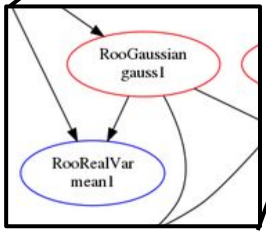
Automatic Differentiation in RooFit

How Does it work?

What that we want to differentiate



A typical RooFit statistical model



Made up of various RooFit objects

Feed to AD tool

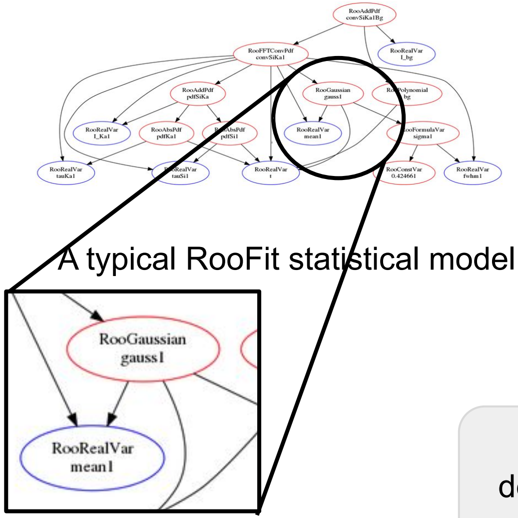
AD Tool

Clad

Automatic Differentiation in RooFit

How Does it work?

What that we want to differentiate



A typical RooFit statistical model

Made up of various RooFit objects

Some way to expose differentiable properties of the graph as code.

C++ code the AD tool can understand

AD Tool

Cla∂

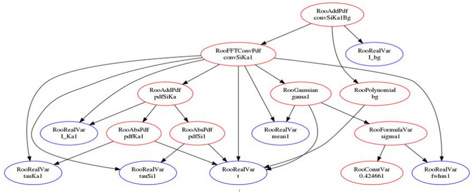
Feed to AD tool

RooFit has an object oriented model which deliberately hides the differential properties of the nodes in favor of ease of use.

Automatic Differentiation in RooFit

How Does it work?

What that we want to differentiate



Define 2 Functions in RooFit

C++ code the AD tool can understand

Stateless function enabling differentiation of each class.

```cpp
double ADDetail::gauss(double x, double mean, double sigma) {
 const double arg = x - mean;
 const double sig = sigma;
 return std::exp(-0.5 * arg * arg / (sig * sig));
}
```

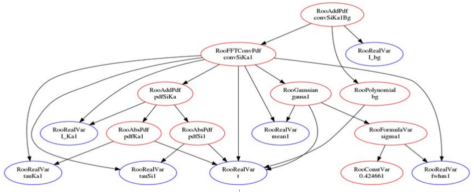
The “glue” function enabling graph squashing.

```cpp
void RooGaussian::translate(...) override {
 result = "ADDetail::gauss(" +
                             _x->getResult() +
                             " ," + _mu->getResult() +
                             " ," + _sigma->getResult() + ")";
}
```

Automatic Differentiation in RooFit

How Does it work?

What that we want to differentiate



Define 2 Functions in RooFit

C++ code the AD tool can understand

`RooGaussian::evaluate()`
The RooFit call to evaluate a gaussian

- Bookkeeping
& caching

`ADDetail::gauss(x, mu, sig)`
The equivalent code generated

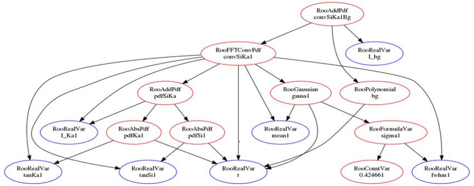
`ADDetail::gauss(x, mu, sig) / ADDetail::gaussIntegral(...)`

The equivalent code generated
(given the class supports analytical integrals)

Automatic Differentiation in RooFit

The Big Picture

What that we want to differentiate



'Squash' the graph into code

`Roo*::translate()`

C++ code the AD tool can understand

C++ code the AD tool can understand

The AD tool

Cla∂

Derivative code of the model!

Current Status

*What Can I Do Right Now?**

```
root[0] RooWorkspace myWS;
root[1] myWS.factory("sum::mu_shifted(mu[0, -10, 10], shift[1.0, -10, 10])");
root[2] myWS.factory("prod::sigma_scaled(sigma[3.0, 0.01, 10], 1.5)");
root[3] myWS.factory("Gaussian::gauss(x[0, -10, 10], mu_shifted, sigma_scaled)");
root[4] RooAbsReal &x = *myWS.var("x");
root[5] RooAbsPdf &pdf = *myWS.pdf("gauss");
root[6] RooArgSet normSet{x};
```

*In ROOT master as of May 2023.

Current Status

*What Can I Do Right Now?**

```
root[6] RooFuncWrapper gaussFunc("myGauss", "myGauss", pdf, normSet);
root[7] gaussFunc.dumpCode();

(double (*)(double *, const double *)) Function @0x7fcfbd2f6000
 at input_line_19:1:
double myGauss(double *params, double const *obs)
{
  const double sigma_scaled = params[2] * 1.5;
  const double mu_shifted = params[0] + params[1];
  const double gauss_Int_x = ADDetail::gaussianIntegral(-10, 10, mu_shifted, sigma_scaled);
  const double gauss = ADDetail::gauss(params[3], mu_shifted, sigma_scaled);
  const double normGauss = gauss / gauss_Int_x;
  return normGauss;
}
```

*In ROOT master as of May 2023.

Current Status

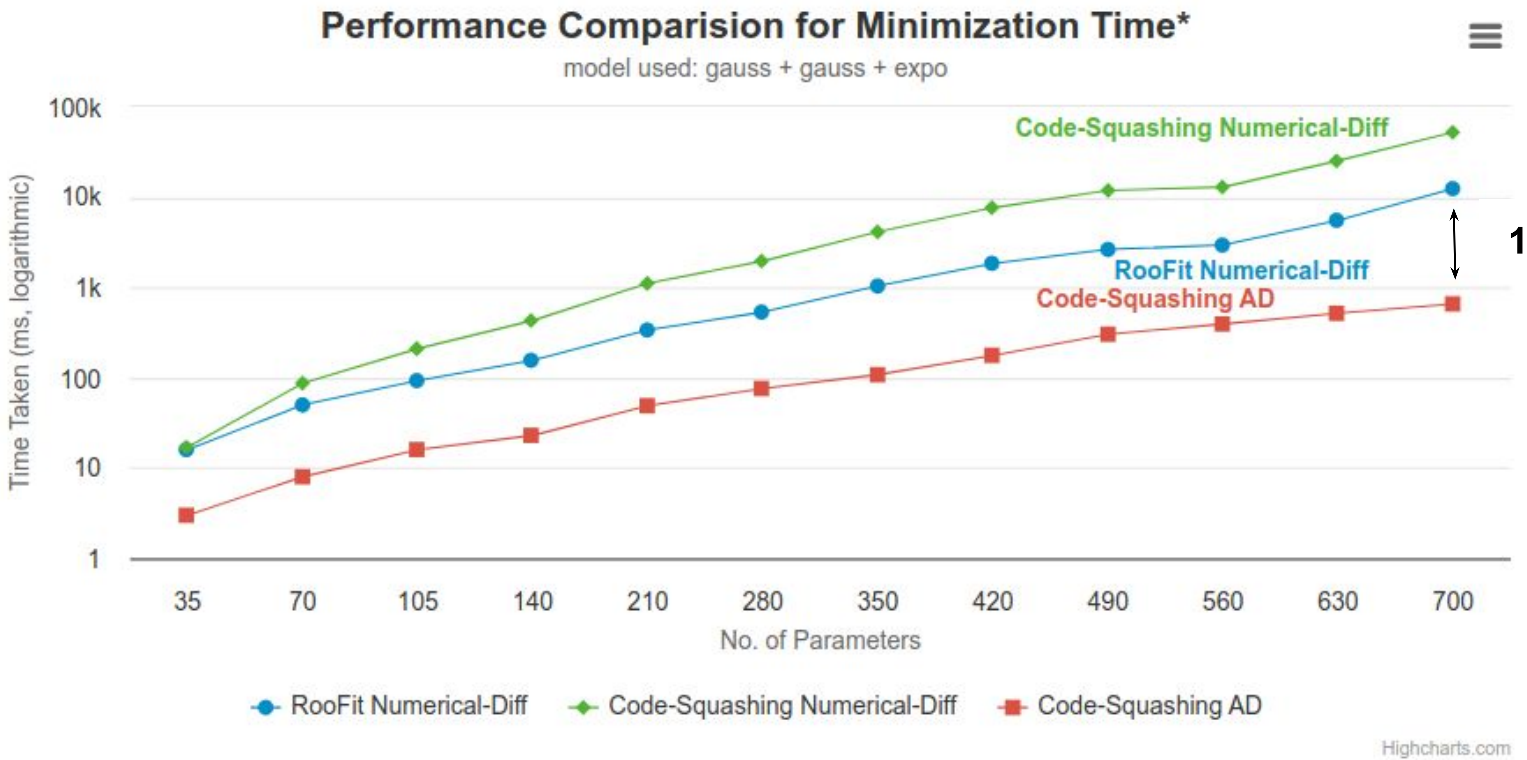
*What Can I Do Right Now?**

```
root[6] RooFuncWrapper gaussFunc("myGauss", "myGauss", pdf, normSet);
root[7] gaussFunc.dumpCode();

(double (*)(double *, const double *)) Function @0x7fcfbd2f6000
 at input_line_19:1:
double myGauss(double *params, double const *obs)
{
  const double sigma_scaled = params[2] * 1.5; "prod::sigma_scaled(sigma[3.0, 0.01, 10], 1.5)"
  const double mu_shifted = params[0] + params[1]; "sum::mu_shifted(mu[0, -10, 10], shift[1.0, -10, 10])"
  const double gauss_Int_x = ADDetail::gaussianIntegral(-10, 10, mu_shifted, sigma_scaled);
  const double gauss = ADDetail::gauss(params[3], mu_shifted, sigma_scaled);
  const double normGauss = gauss / gauss_Int_x; "Gaussian::gauss(x[0, -10, 10], mu_shifted, sigma_scaled)"
  return normGauss;
}
```

*In ROOT master as of May 2023.

Results



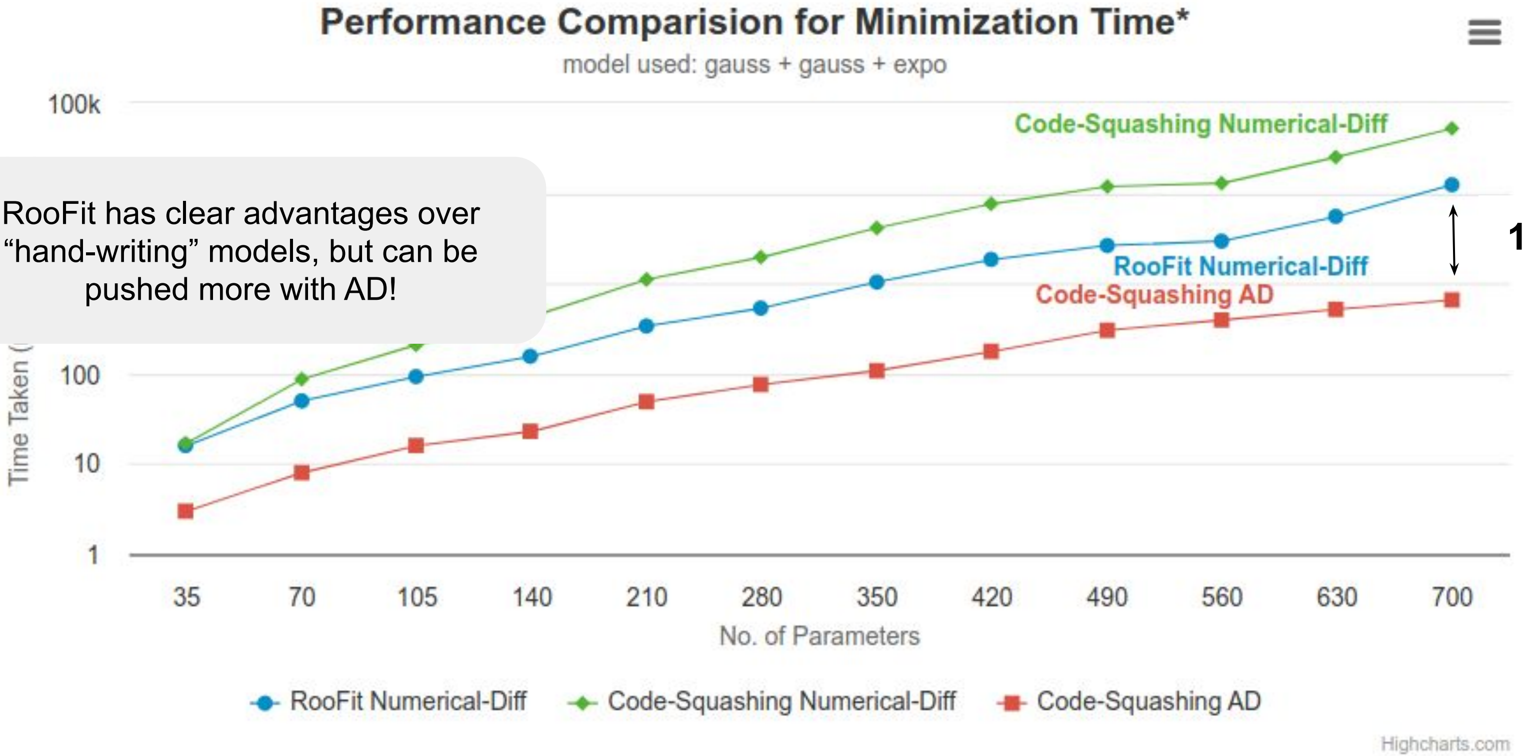
18x Faster!

Tested on ROOT master as of May 2023.

*Excludes the seed generation time, more info - [look here]

Results

RooFit has clear advantages over “hand-writing” models, but can be pushed more with AD!



18x Faster!

Tested on ROOT master as of May 2023.

*Excludes the seed generation time, more info - [look here]

Results

Why??

Configuration (700 params)	**Time / Iteration**	**Total Iterations to Converge**	**Final FCN Value**
Code-Squashing Numerical-Diff	~380 ms	136	659552.2918
RooFit Numerical-Diff	~86 ms	136	659552.2917
Code-Squashing AD	~11 ms	58	659551.9860

Tested on ROOT master as of May 2023.

Results

Why? Code-Squashing vd RooFit (Numerical)

Configuration (700 params)	Time / Iteration	Total Iterations to Converge	Final FCN Value
Code-Squashing Numerical-Diff	~380 ms	136	659552.2918
RooFit Numerical-Diff	~86 ms	136	659552.2917
Code-Squashing AD	~11 ms	58	659551.9860

~ 3.5x Slower time/iteration.

Why? Even Though both use num-diff, RooFit uses complex caching logic, making it faster!

Tested on ROOT master as of May 2023.

Results

Why? Code-Squashing AD vs RooFit Numerical

Configuration (700 params)	**Time / Iteration**	**Total Iterations to Converge**	**Final FCN Value**
Code-Squashing Numerical-Diff	~380 ms	136	659552.2918
RooFit Numerical-Diff	~86 ms	136	659552.2917
Code-Squashing AD	~11 ms	58	659551.9860

~ 8x Faster Derivatives

Why? AD is faster than NumDiff, esp. For large number of params!

Tested on ROOT master as of May 2023.

Results

Why? Code-Squashing AD vs RooFit Numerical

Configuration (700 params)	**Time / Iteration**	**Total Iterations to Converge**	**Final FCN Value**
Code-Squashing Numerical-Diff	~380 ms	136	659552.2918
RooFit Numerical-Diff	~86 ms	136	65955**2.2917**
Code-Squashing AD	~11 ms	58	65955**1.9860**

~ 8x Faster Derivatives

Why? AD is faster than NumDiff, esp. For large number of params!

Faster (and better) Convergence (for large fits)

Why? AD is more numerically stable than NumDiff. Less num error = faster convergence!

Tested on ROOT master as of May 2023.

Conclusion

Summary and Future Work

Our work presents an efficient way to translate complex models such that they can be differentiated using AD. We demonstrate that AD can be used to effectively lower the fitting time for non-trivial models.

- Completely avoid the use of numerical gradients in fits using MINUIT.
- Extend support to cover HistFactory and other parts of RooFit.
- Optimize Clad generated derivatives and further explore how they can be parallelized (OpenMP or CUDA).

Work with experiments to show similar speedups on their production workflows.

The End!

Questions?

https://www.linkedin.com/in/garimasingh28/

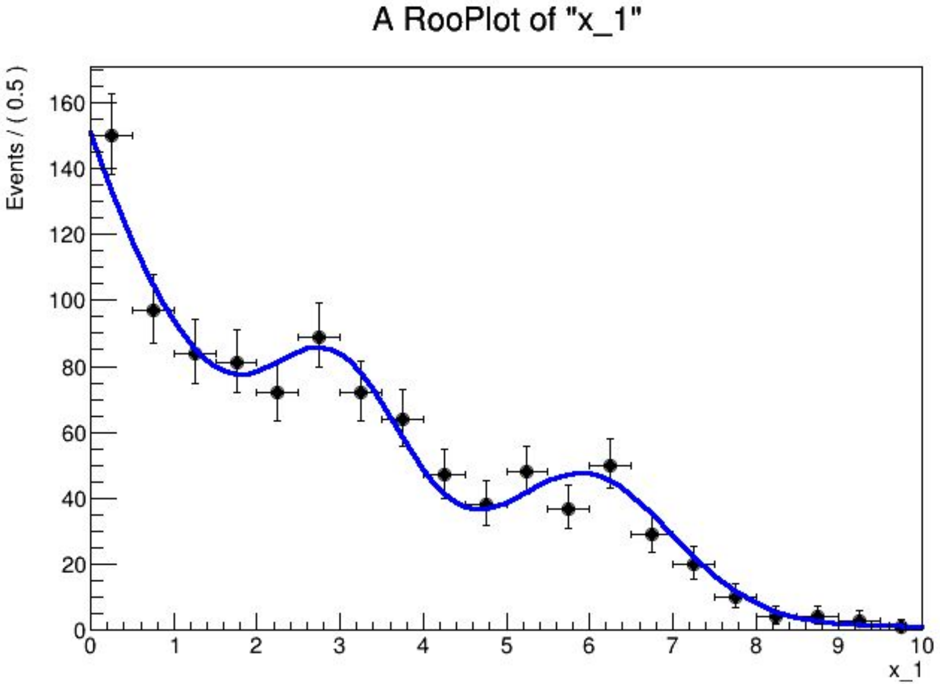
https://github.com/grimmmyshini

garima.singh@cern.ch

Backup

Backup

Model From Benchmarks



Plot for number of channels = 1

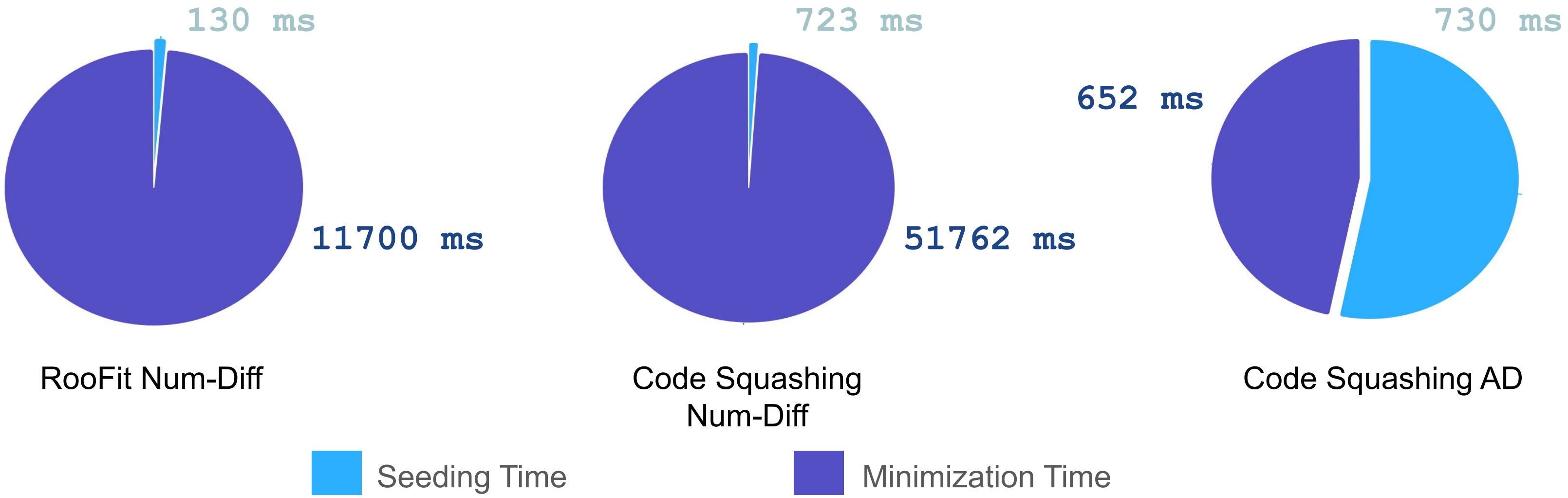
Backup

Model From Benchmarks

```
RooRealVar c("c", "c", -0.5, -0.8, 0.2);
RooExponential expo("expo", "expo", x, c);
// Create two Gaussian PDFs g1(x,mean1,sigma) anf g2(x,mean2,sigma) and their parameters
RooRealVar mean1("mean1", "mean of gaussians", 3, 0, 5);
RooRealVar sigma1("sigma1", "width of gaussians", 0.8, .01, 3.0);
RooRealVar mean2("mean2", "mean of gaussians", 6, 5, 10);
RooRealVar sigma2("sigma2", "width of gaussians", 1.0, .01, 3.0);
RooGaussian sig1("sig1", "Signal component 1", x, mean1, sigma1);
RooGaussian sig2("sig2", "Signal component 2", x, mean2, sigma2);
// Sum the signal components
RooRealVar sig1frac("sig1frac", "fraction of signal 1", 0.5, 0.0, 1.0);
RooAddPdf sig("sig", "g1+g2", {sig1, sig2}, {sig1frac});
// Sum the composite signal and background
RooRealVar sigfrac("sigfrac", "fraction of signal", 0.4, 0.0, 1.0);
RooAddPdf model("model"), "g1+g2+a", {sig, expo}, {sigfrac});
```

Backup

Share of fitting time for 700 parameters



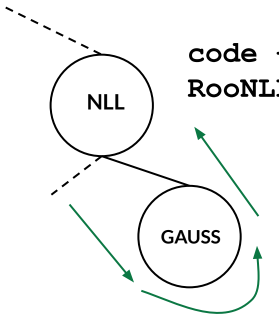
Seeding uses numerical differentiation = Larger times for AD

Possible Fix? Use AD here too!

Seeding: initial parameter scale estimation to get the step size for the minimization.

Backup

How models are translated




```
code +=
RooNLLVar::Translate();
```

```
code +=
RooGaussian::Translate({});
```

The parent node queries the results from the child nodes.

```
// Declare the code
gInterpreter->Declare(code.c_str());
// Get the derivatives of 'code'
gInterpreter->ProcessLine("clad::gradient(code);");
// Use code_grad in wrappers that interface with
// the minimizer.
```

Backup

Clad - Compiler Based AD Tool

Clad, a compiler based source-code-transformation AD tool. Clad inspects the internal compiler representation of the target function to generates its derivative.

```
double absFunc(double x) {
 if (x < 0) return -x;
 else return x;
}
```

`clad::differentiate(absFunc)` →

```
double absFunc_darg0(double x) {
   double _d_x = 1;
   if (x < 0) return -_d_x;
   else return _d_x;
}
```

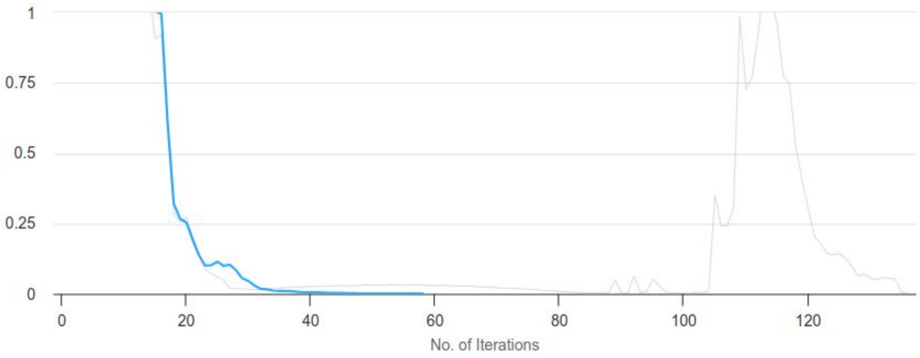
Can be used within Cling[2], the C++ interpreter used with ROOT.

→ Off the shelf JIT compiled Derivatives!

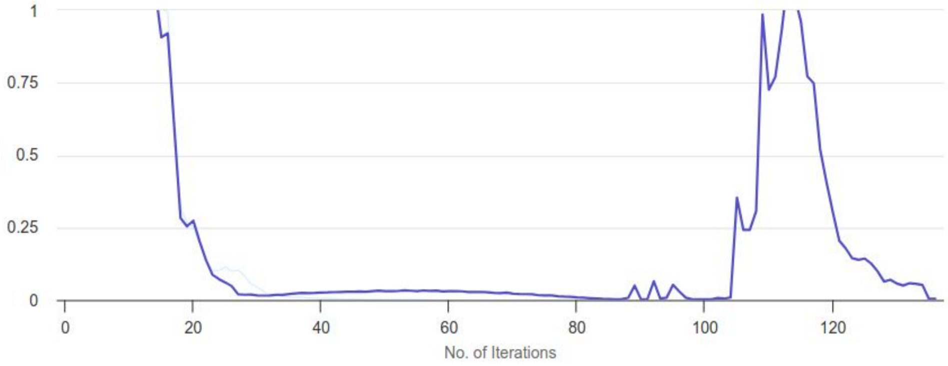
[2] :https://github.com/root-project/cling

Backup

Numerical error and convergence rates: EDM vs Iterations



AD Code-Squashing



RooFit Numerical-Diff
(Without offsetting)

Large number of parameters usually causes numerical issues[3] with minimizations, leading to fluctuation in step sizes and eventually leading to longer or no convergence.

[3] :https://root.cern.ch/root/htmldoc/guides/minuit2/Minuit2.html#convergence-in-mboxmigrad-and-positivedefiniteness