Third MODE Workshop on Differentiable Programming for Experiment Design

Princeton University 24-26 July, 2023

Making Likelihood Calculations Fast Using Automatic Differentiation in RooFit

Garima Singh (Princeton University), Jonas Rembser (CERN), Lorenzo Moneta (CERN), 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

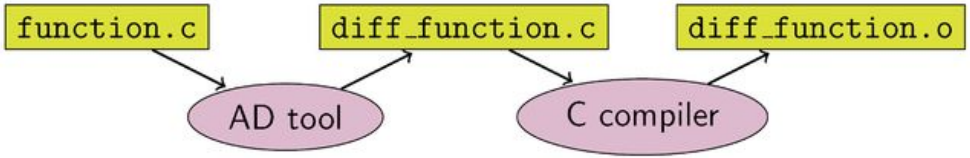
Goal

Add automatic differentiation (AD) to RooFit, a statistical modelling library packed in ROOT.

Methods of Automatic Differentiation

The Two Techniques

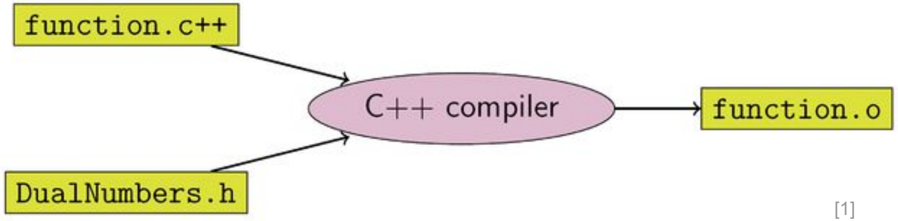
Source Code Transformation AD



[1]

- Synthesize derivative code from the input program automatically.
- **Faster** - allows for easier compiler optimization.
- Eg. Tapenade, Enzyme, **Clad**

Operator Overloading AD



[1]

- Use a new data type and operator overloading to keep track of derivatives as the original program executes.
- **Slower** and requires hand writing annotations and changing data types.
- Eg. PyTorch/TensorFlow, CoDiPack, etc.

[1] : https://en.wikipedia.org/wiki/Automatic_differentiation

An Efficient Method of Differentiation

Compiler-Based Source Transformation AD: Clad

Clad[1], a source code transformation AD tool, implemented as a plugin to the clang compiler. 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;
}
```

- Proximity to compiler allows for more control over code generation.
- Support for a good subset of modern C++ constructs.

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

An Efficient Method of Differentiation

Compiler-Based Source Transformation AD: Clad

Clad also can be used within Cling[3], the C++ interpreter used with ROOT.

```
[2]: double fn(double x, double y) {
       return x*x*y + y*y;
     }
```

```
[3]: auto fn_dx = clad::differentiate(fn, "x");
```

```
[4]: fn_dx.execute(5, 3)
```

```
[4]: 30.000000
```

Binder Tutorial

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

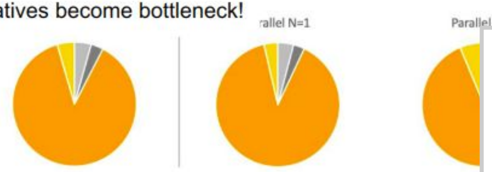
Motivation

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	setup_roofit	327	setup_roofit	315	setup_roofit	315	setup_roofit	312	setup_roofit	327
minuit_init	230	minuit_init	231	minuit_init	231	minuit_init	231	minuit_init	231	minuit_init	231
gradient_calc	6289	gradient_calc	7102	gradient_calc	3734	gradient_calc	1997	gradient_calc	1107	gradient_calc	879
line_search	323	line_search	287	line_search	287	line_search	287	line_search	287	line_search	287

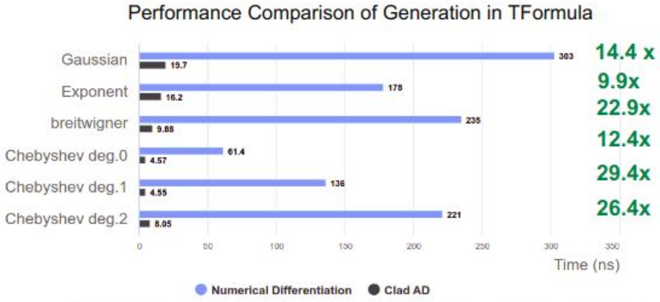
Derivatives become bottleneck!



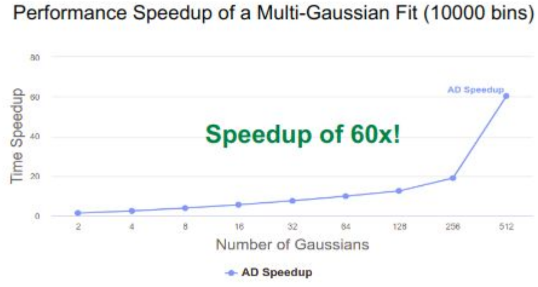
ICHEP 2022 - Zeff Wolffs - https://agenda.infn.it/event/28874/contributions/169205/a

- 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.



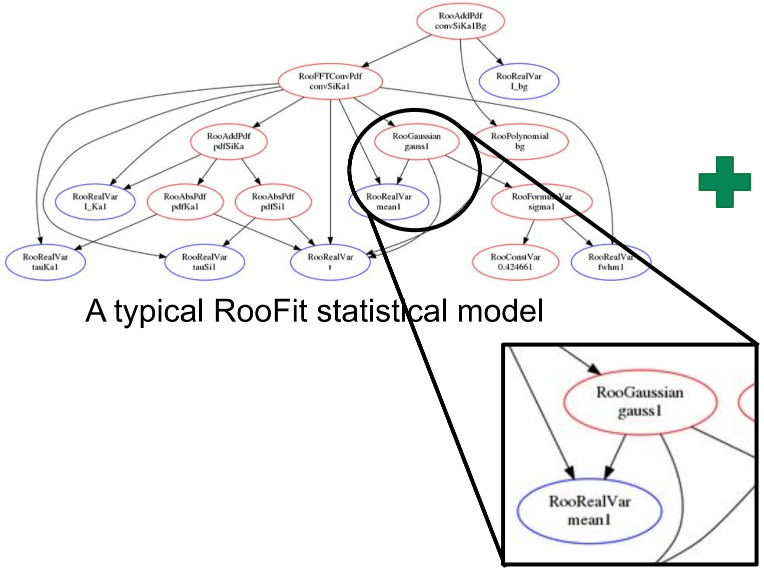
TF1 based benchmarks. TF1 is the TFormula fitting interface for fitting histograms.

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

Automatic Differentiation in RooFit

Sounds easy…

What we want to differentiate



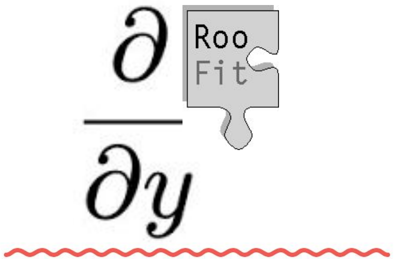
A typical RooFit statistical model

Made up of various RooFit objects

Our AD tool of choice

Cla∂

Differentiable RooFit Models!



Actually, not so simple…

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

Challenges

RooFit represents all mathematical formulae as RooFit objects which are then brought together into a compute graph. This compute graph makes up a model on which further data analysis is run.

Math Notations		RooFit Object
variable	x	RooRealVar
function	$f(x)$	RooAbsReal
PDF	$f(x)$	RooAbsPdf
space point	$\hat{x}$	RooArgSet
integral	$\int_a^b f(x)$	RooRealIntegral
list of space points	$\hat{x}_1, \hat{x}_1, \hat{x}_1 ...$	RooAbsData

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \longrightarrow$$

Gaussian Probability Distribution Function (pdf)

```
//Obj represents f(x) here
RooGaussian obj(x, mu, sigma);
```

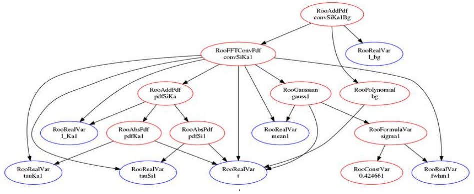
Equivalent Code in C++ with RooFit

Programmers/users know this relationship. But how do we connect these two together when a connection is not obvious in code?

Automatic Differentiation in RooFit

How Does it work?

What that we want to differentiate



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

C++ code the AD tool can understand

C++ code the AD tool can understand

The AD tool

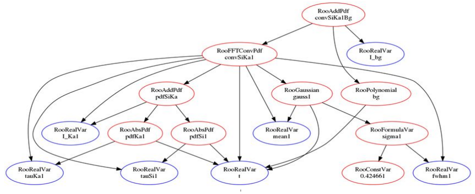
Clad

Derivative code of the model!

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.

```
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));
}
```

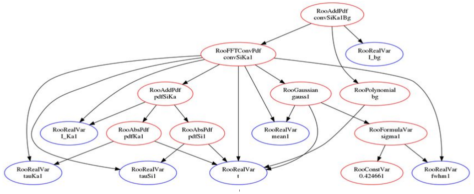
The “glue” function enabling graph squashing.

```
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

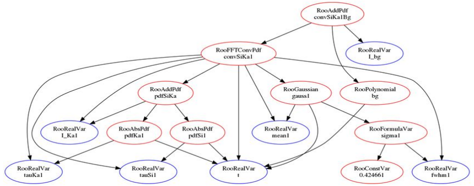
`ADDetail::gauss(x, mu, sig) / ADDetail::gaussIntegral(...)`

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

Automatic Differentiation in RooFit

The Big Picture

What that we want to differentiate

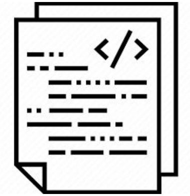


'Squash' the graph into code

`Roo*::translate()`

C++ code the AD tool can understand

C++ code the AD tool can understand



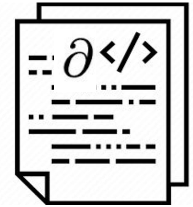
+

The AD tool

Clad

=

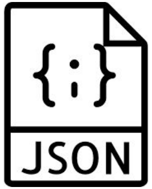
Derivative code of the model!



Automatic Differentiation in RooFit

Interlude: JSON to C++?

JSON model configuration



An independent translation module

TranslateModel.so

TranslateModel.h

Model translated to C++ code

Results

- A HistFactory example (binned pdfs based on template histograms)

Out of RooFit, POC

- A basic RooFit example with binned fit of analytical shapes

In RooFit

- A WIP ATLAS HistFactory Benchmark

In RooFit

Results

- A HistFactory example (binned pdfs based on template histograms)

Out of RooFit, POC

- A basic RooFit example with binned fit of analytical shapes

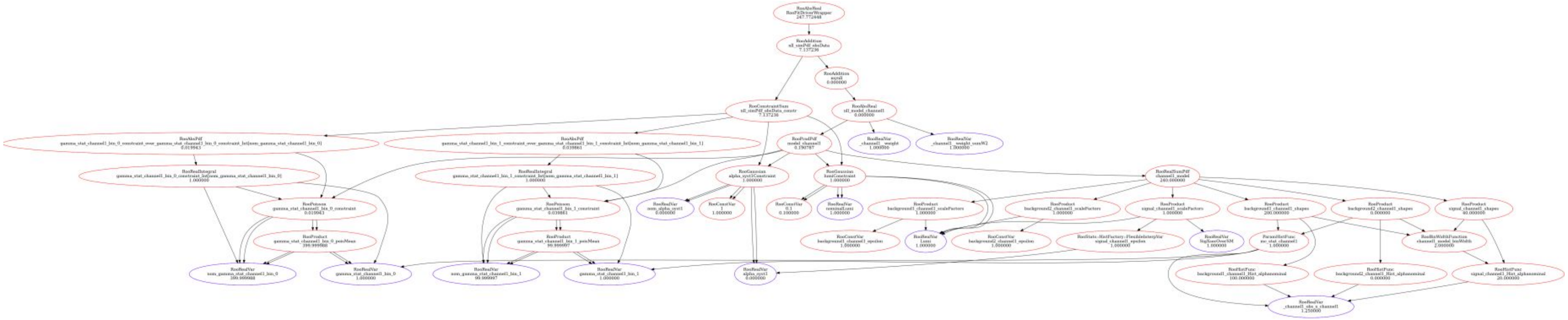
In RooFit

- A WIP ATLAS HistFactory Benchmark

In RooFit

Results

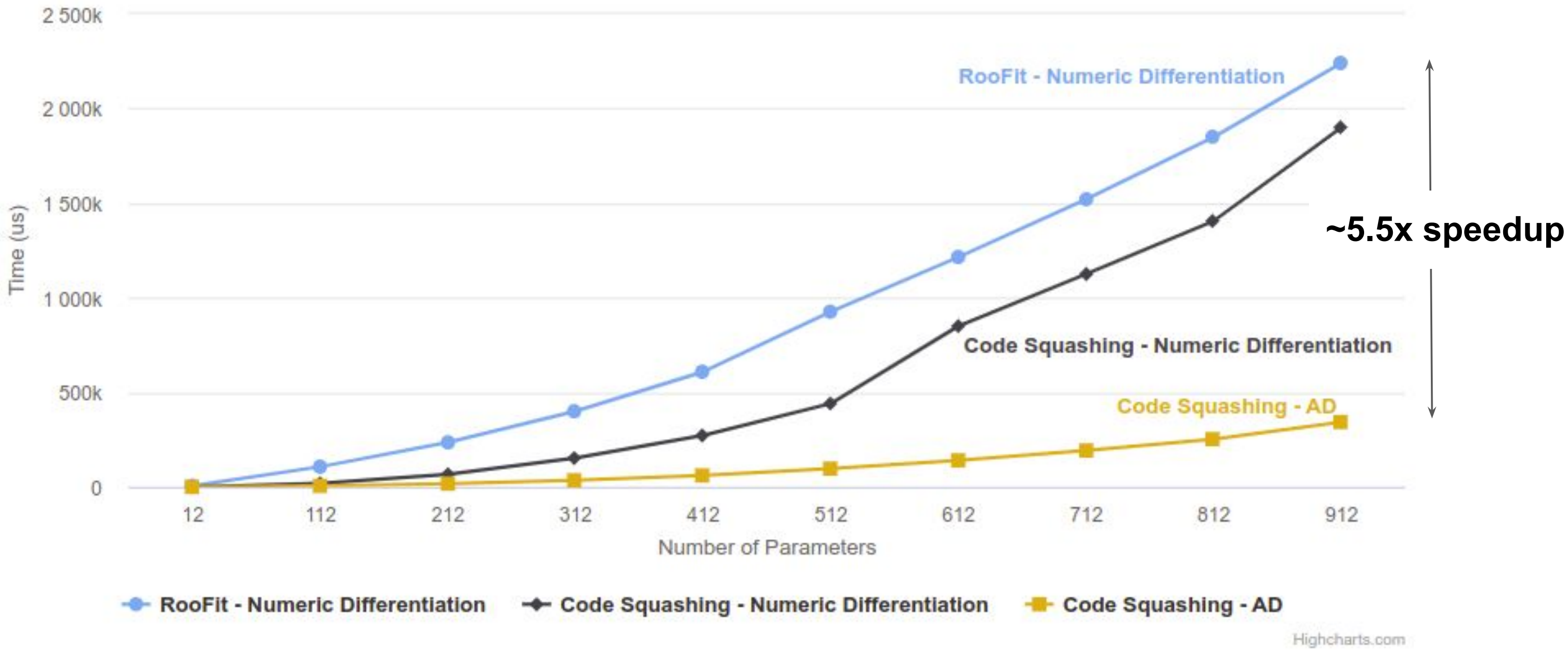
The POC HistFactory Model



An example histogram fitting model with 2 bins and 2 channels, with 3 samples per channel. Based on the hf_001 example.

Results

The POC HistFactory Model



Tested on ROOT v6.26.

Results

- A HistFactory example (binned pdfs based on template histograms)

Out of RooFit, POC

- A basic RooFit example with binned fit of analytical shapes

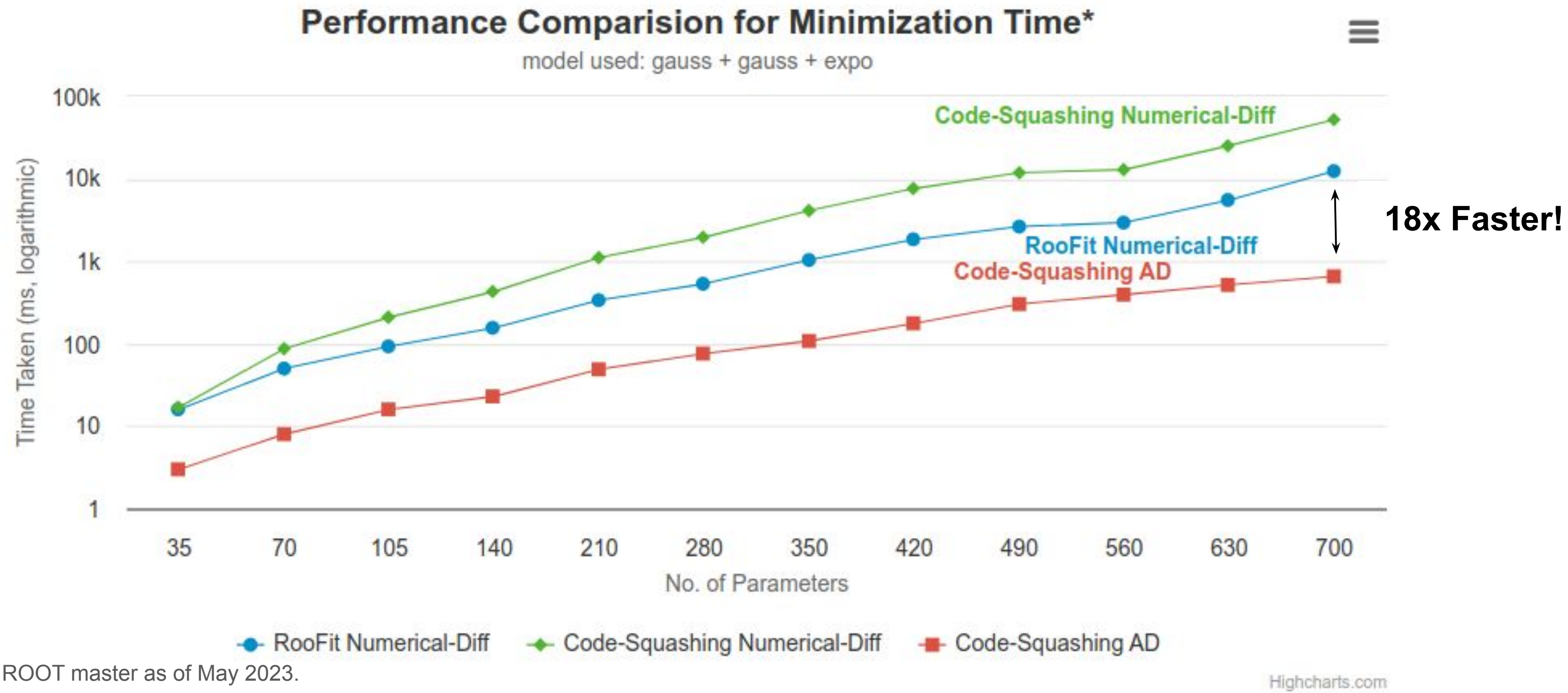
In RooFit

- A WIP ATLAS HistFactory Benchmark

In RooFit

Results

The Real RooFit Example

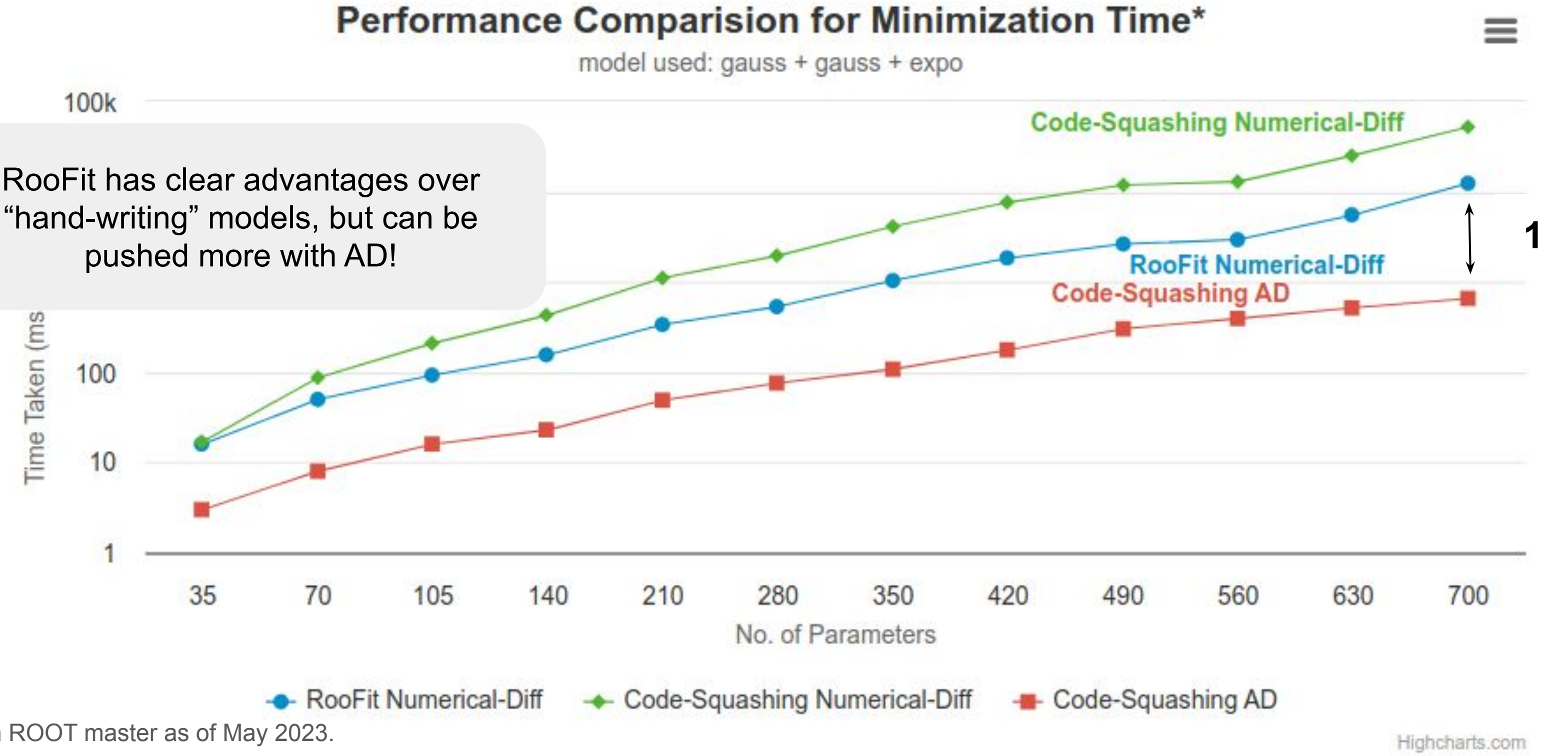


Tested on ROOT master as of May 2023.

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

Results

The Real RooFit Example



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

- A HistFactory example (binned pdfs based on template histograms)

  Out of RooFit, POC

- A basic RooFit example with binned fit of analytical shapes

  In RooFit

- A WIP ATLAS HistFactory Benchmark

  In RooFit

Results

WIP: ATLAS HistFactory Benchmark

No. Of Channels	FItting Time (s)*		AD Speedup
	RooFit Numerical-Diff	Code-Squashing AD	
1	0.03	0.01	**2x**
5	1.19	0.26	**3.5x**
10	2.22	0.36	**5.2x**
20	7.38	1.17	**5.3x**

Link to paper: https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PAPERS/HIGG-2018-51/

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

Results

WIP: ATLAS HistFactory Benchmark

We are still investigating issues with JIT-ing in ROOT and also working on reducing these times.

	JIT Time in ROOT (s)*	Compile Time (g++ 10, s)	Compile Time (clang-13, s)
-O0	~16	1.15	0.82
-O1	~17	4.46	6.00
-O2	~17	9.24	8.57
-O3	~17	10.69	8.88

* For a **non optimized** channel in the benchmark, For a partly optimized one, the time taken is < 1 sec

Results

WIP: ATLAS HistFactory Benchmark

	JIT Time in ROOT (s)*	Compile Time (g++ 10, s)	Compile Time (clang-13, s)
-O0	~16	1.15	0.82
-O1	~17	4.46	6.00
-O2	~17	9.24	8.57
-O3	~17	10.69	8.88

Now < 1 sec, **potential for much more!**

* For a **non optimized** channel in the benchmark, For a partly optimized one, the time taken is < 1 sec

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.

Conclusion

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

Conclusion

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

Conclusion

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

Future Work

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

Takeaways From a CS Person

an inexperienced :)

Takeaways From ~~a~~ CS Person

High usability but impacted performance. **Low Risk**

Develop New AD Tools

Ideal world! Amazing performance and usability.

Don't Change ANYTHING

Complete Rewrite

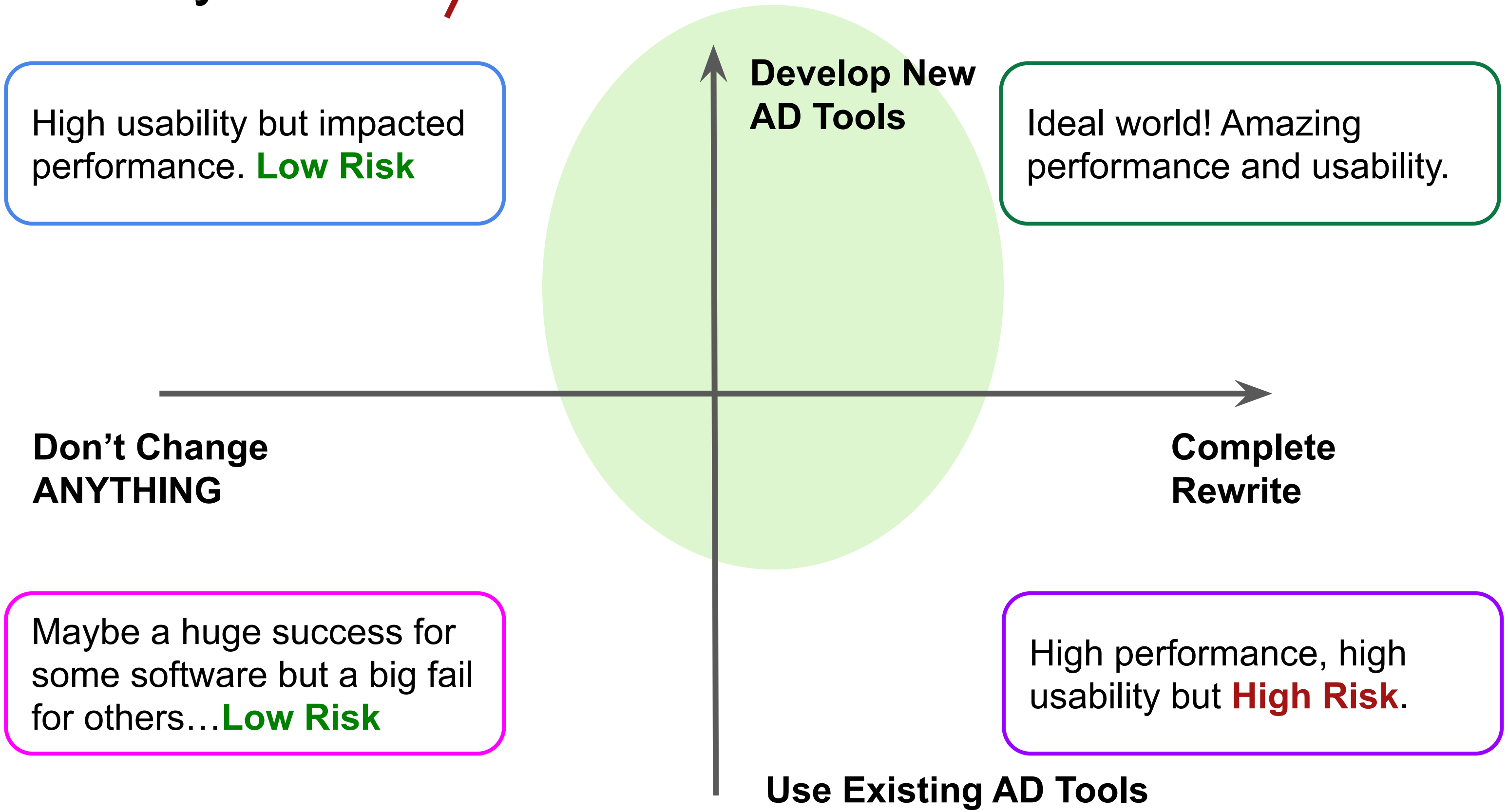
Maybe a huge success for some software but a big fail for others…**Low Risk**

High performance, high usability but **High Risk**.

Use Existing AD Tools

an inexperienced :)

Takeaways From a CS Person



an inexperienced :)

Takeaways From a CS Person

High usability but impacted performance. **Low Risk**

Develop New AD Tools

Ideal world! Amazing performance and usability.

- Likely to be most successful.
- New AD tools (or modifications to current ones) -> encode domain specific optimizations!
- Can't avoid rewrites…
- **MORE COLLABS WITH CS!!!!**

Don't Change ANYTHING

Maybe a huge success for some software but a big fail for others…**Low Risk**

Use Existing AD Tools

The End!

Questions?

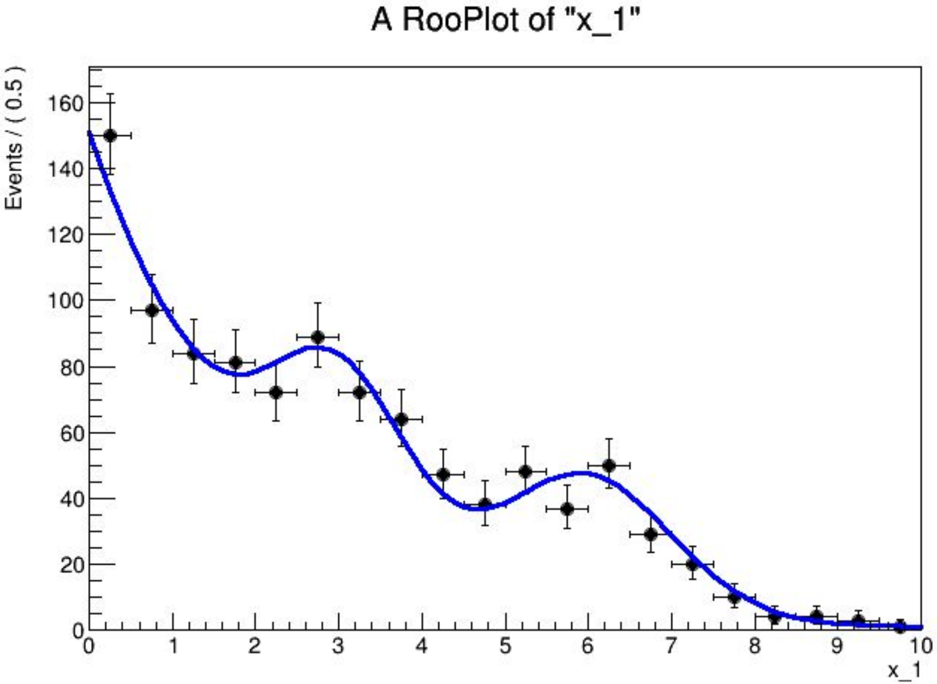
https://www.linkedin.com/in/garimasingh28/

https://github.com/grimmmyshini

garima.singh@cern.ch

Backup

Model From Benchmarks



Plot for number of channels = 1

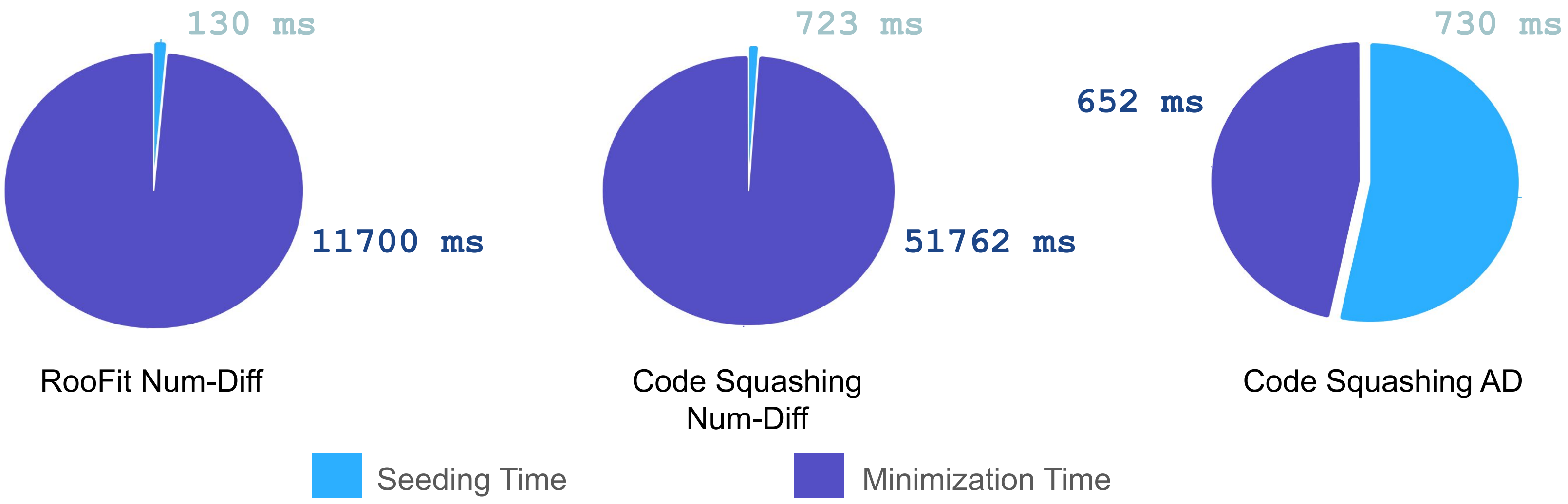
Backup

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



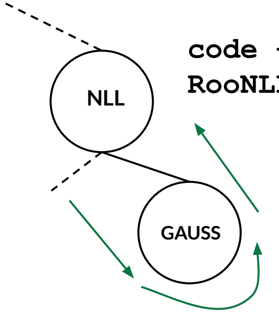
Seeding uses numerical differentiation = Larger times for AD

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();
```

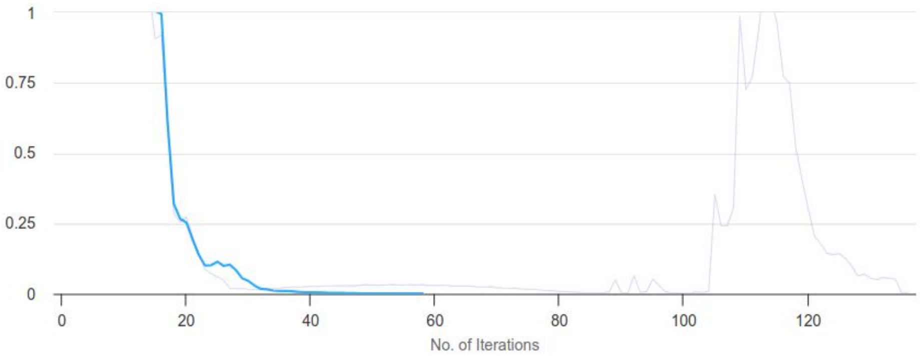
```
code +=
RooGaussian::Translate({});
```

The parent node queries the results from the child nodes.

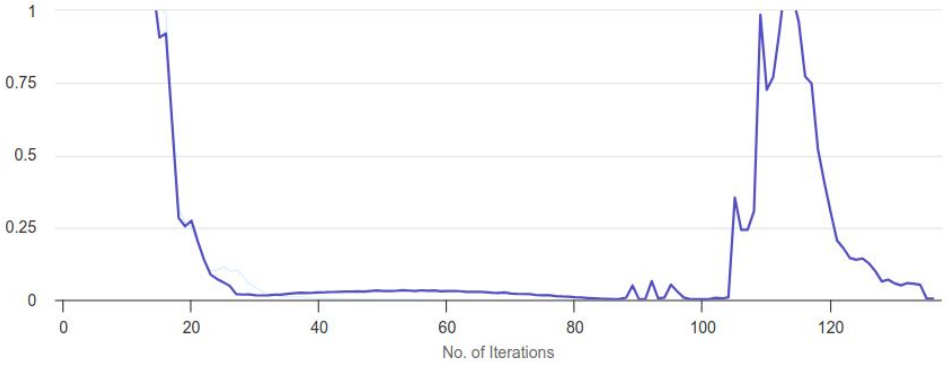
```
// Declare the code
gInterpreter->Declare(code.c_str());
// Get the derivatives of 'code'
gInterpreter->ProcessLine("clad::gradient(code);");
// Use code_grad in wrappers that interface with
// the minimizer.
```

Backup

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

Backup

Why is RF faster in once benchmark but not the other?

The granularity of the RooFit computation graph that represents a HistFactory model is too high. It caches the result of relatively simple operations, so the caching logic is more expensive than re-evaluating the model.

However, these results inspired us to do some optimizations in HistFactory, so by now RooFit should be again on par with code squash num-diff or even better!

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

Introduction

Source Code Transformation Based Automatic Differentiation

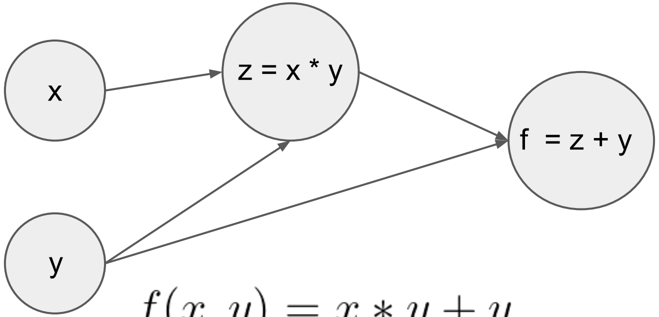
Automatic Differentiation (AD) is a set of techniques to evaluate the exact derivative of a computer program.

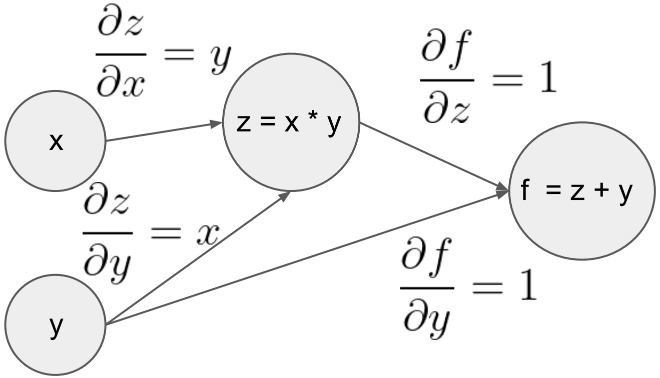
- Faster than numerical differentiation - scales better for problems with large number of parameters.
- More accurate than numerical differentiation - fewer numerical errors!

Introduction

What is Automatic Differentiation?

Simply put, it's a way for computers to differentiate computer programs. AD applies the chain rule of differential calculus throughout the semantics of the original program.





$$f(x, y) = x * y + y$$

$$f'(x, y)_x = y \quad f'(x, y)_y = x + 1$$

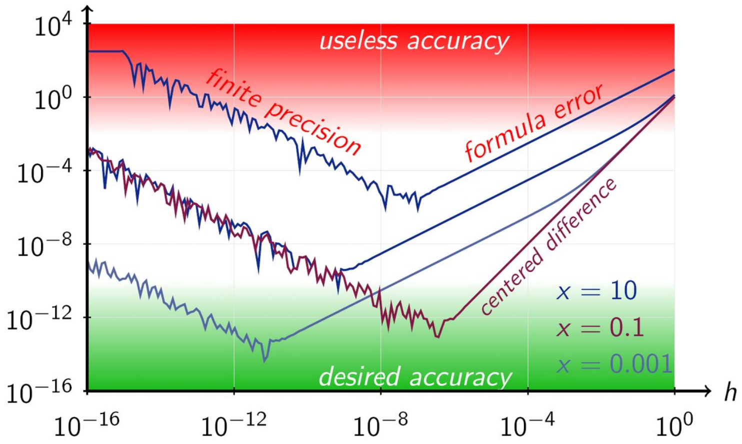
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} * \frac{\partial z}{\partial x} = y \quad \frac{\partial f}{\partial y} = (\frac{\partial f}{\partial z} * \frac{\partial z}{\partial y}) + \frac{\partial f}{\partial y} = x + 1$$

Introduction

Why AD over numerical differentiation?

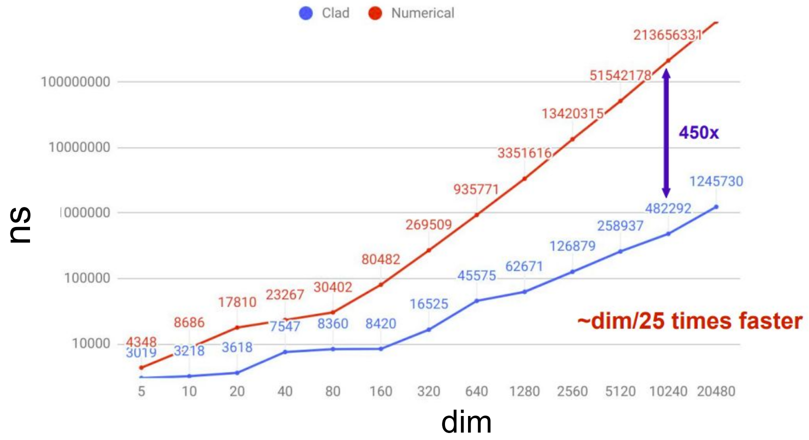
- Calculates exact derivatives of programs, free from numerical errors.
- More performant for functions with high number of parameters.

Difficulty in choosing step size due to numerical error



https://commons.wikimedia.org/wiki/File:AbsoluteErrorNumericalDifferentiationExample.png

Comparison between Clad's AD and numerical diff



https://www.researchgate.net/publication/346917467_Automatic_Differentiation_in_ROOT

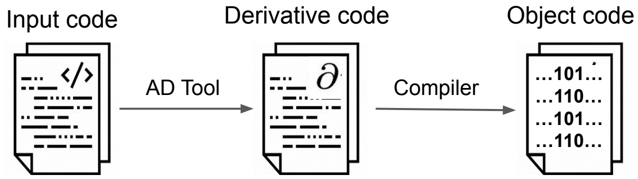
Introduction

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.



Automatic Differentiation in RooFit

Anatomy of a Translate Function

Object to manage the code squashing and derivative generation. Provides a bunch of utility functions for code squashing.

```
void RooGaussian::translate(ADDetail::CodeSquashContext &ctx) const
{
      // Build a call to the stateless gaussian.
      std::string const& xName = ctx.getResult(&x.arg());
      std::string const& muName = ctx.getResult(&mean.arg());
      std::string const& sigName = ctx.getResult(&sigma.arg());
      std::string const& ResName = "ADDetail::gauss(" + xName + ", " + muName + ", " + sigName + ")";
      ctx.addResult(this, ResName);
}
```

A function to query the string representing the result of the input RF variable.

Assigns the class a string that represents its result in the squashed code.

Motivation

Why AD in RooFit?

Usual RooFit is performant even with numerical-diff because of its complex caching logic.

However, even if this caching would be done at a very granular level, it has lots of overhead from virtual calls and bookkeeping, which is why we expect AD to be superior.