Garima Singh
Manipal Institute of Technology, Manipal
garimasingh0028@gmail.com

Mentors:

Vassil Vassilev
vvasilev@cern.ch

Alexander Penev
alexander_penev@yahoo.com

Add numerical differentiation support in Clad

GSoC 2021

Introduction

In mathematics and computer algebra, automatic differentiation (AD) is a set of techniques to numerically evaluate the derivative of a function specified by a computer program. Automatic differentiation is an alternative technique to Symbolic differentiation and Numerical differentiation (the method of finite differences). Clad[1] is based on Clang which provides the necessary facilities for code transformation. The AD library can differentiate non-trivial functions, find a partial derivative for trivial cases, and has good unit test coverage. In several cases, due to different limitations, it is either inefficient or impossible to differentiate a function. For example, Clad cannot differentiate declared-but-not-defined functions. In that case, it issues an error. Instead, clad should fall back to its future numerical differentiation facilities.

Overview

A brief introduction to numerical differentiation

Numerical differentiation is a technique widely used by smaller computational machines such as scientific calculators to estimate the derivatives of input functions. Usually, these implementations are one of the variants below:

Method of Finite Differences:

Forward difference:

For a function ƒ(x), the derivative ƒ'(x) would be defined as follows:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Backward difference:

Following the same notation, the derivative becomes the following:

$$f'(x) = \lim_{h \to 0} \frac{f(x) - f(x-h)}{h}$$

Central Difference:

Following the same notation, the derivative becomes the following:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

Implementation of the above methods is simple and straightforward, however, the only difficult part is choosing the correct step size (i.e. `h`). Choosing a larger value for `h` may result in grossly overshooting the derivative estimate while choosing a very small value for `h` may lead to a very large rounding error due to the difference. For basic central differences in double precision, the optimal value for `h` is the cube-root of the *machine epsilon(ϵ)* [2]. For forward and backward difference, a formula that balances both of the step size errors is as follows:

$$h = 2\sqrt{\epsilon \left| \frac{f(x)}{f''(x)} \right|}$$

For single-precision calculations, another big issue is the exact representation of either `x + h` or `x`. The big connotation of that is the denominator difference will not be exactly equivalent to `h`, making the derivative estimate suffer.

A possible solution is to explicitly store the independent values and use the difference of those as the denominator as opposed to just `h`, an example is given below:

```
h := sqrt(mach_eps) * x;

xph := x + h;

dx := xph - x;

slope := (F(xph) - F(x)) / dx;
```

Again, we must make sure that this code section is free of lossy compiler optimizations. One way to ensure this in C is to make `xph` volatile.

Clad and numerical diff

As the project problem statement suggests, first we would want to add a dedicated interface for numerical differentiation (let's call it `clad::num_diff`). This would not be any different from `clad::gradient` or `clad::differentiate`. This dedicated interface allows users to utilize Clad's numerical-diff abilities stand alone. This way we also open up avenues to add more numerical diff support to Clad in the future.

The next step would be to generate code of the function derivative in the case that Clad can not derive it. Ideally, Clad replaces the definition of the function to be differentiated (say `func`) with its derivative (say `func_darg0`) whose body is not null. However, in the case that Clad cannot differentiate the function, it emits a warning and returns a null derivative body. One way to avoid this is by replacing the null body with the following:

```
double func_darg0(double x) {
      return clad::NumericalDiff::centralDifference(func);
}
```

Wherein, `NumericalDiff` is the namespace that defines the derivative estimation methods and `centralDifference(func)` is the function evaluated at runtime which essentially returns the result from the equation mentioned here.

Another possible approach is to emit the direct formula into the function derivative as illustrated below:

```
double func_darg0(double x) {
      return (func(x + clad::NumericalDiff::step) - func(x)) /
clad::NumericalDiff::step;
}
```

However, the former approach is far better than the latter in the sense of uniformity and ease of addition of different numerical diff methods. For Clad to operate in the latter mode, we would either need to prewrite each method or somehow figure out a way to generate code for different methods on the fly.

Errors and numerical diff[4]

Since derivatives resulting from numerical methods are mere estimates, it becomes important to know the expected error in the result we provide. The error in the simple forward difference formula is roughly comprised of two types of errors:

1. Theoretical/Truncation Error

   This error stems from the fact that numerical differentiation uses approximation (like a linear approximation of the Taylor series in finite difference methods) and hence is not the most accurate. Estimating the theoretical error is fairly simple, and for the forward difference method, it can be derived by power series expansion as follows:

$$\frac{f(x+h)-f(x)}{h} = \frac{f(x)+f'(x)\cdot h+f''(x)\cdot\frac{h^2}{2}+\cdots-f(x)}{h}$$
$$= f'(x)+\frac{f''(x)}{2}\cdot h+\frac{f'''}{3!}\cdot h^2+\cdots$$

   For small values of `h` the error can be approximated to:

$$\frac{f''(x)}{2}\cdot h$$

   For a more rigorous bound to the truncation error, we can say the following:

$$E(f;a,h) = |f'(a)-\frac{f(a+h)-f(a)}{h}| \le \frac{h}{2} max_{x\in[a,a+h]}|f''(x)|$$

2. Round-off Error

   Due to the nature of computing, choosing a very small value of h (`h → 0`) may result in rounding errors. One case of round-off errors may be if `x + h ≈ x`, in that case, the numerator of the finite difference method becomes 0. That might also be the case if there are floating-point errors in the function itself (here we can make a case of how error estimation is important for even finding numerical derivatives) leading to either incorrect differences or even catastrophic cancellation. There are two main ways to mitigate these errors.

   The first one is to select an appropriate value for `h` such that the roundoff errors are minimized (mentioned in the previous sections). For the forward/backward difference method, we can expect to get a precision of about `no. of precision bits / 2`. For the central difference method, we can expect to get a precision of about `2 * no. of precision bits / 3`.

The second one is to select some other method to approximate the derivative so that we can get more significant digits. A few of which are mentioned in the following sections.

Therefore, summarizing results from this section, we can come to a strict upper bound for the error in the value of the estimated derivative as follows:[5]

$$\frac{h}{2} max_{x \in [a,a+h]} |f''(x)| + \frac{2\epsilon_M}{h} max_{x \in [a,a+h]} |f(x)|$$

Higher-order and complex derivatives

Higher-order differentiation methods are also vital for Clad. There are two aspects to higher-order derivatives, first is higher order methods to approximate the first derivative (covered in later sections) and the second is approximating higher derivatives. We can easily extend the definition of the finite difference method to show the following:[6]

$$f^{(n)} \approx \frac{1}{(2h)^n} \sum_{k=0}^{n} (-1)^k \binom{n}{k} f(x + (n-2k)h)$$

However, this method is usually not the method of choice because of high truncation and roundoff errors for higher derivatives (n > 4). A few other methods with higher accuracy and convergence are mentioned in the following sections.

Another interesting place to look into derivatives is complex-valued derivatives; wherein one can state the following:[8]

$$f'(x) = \frac{\Im(f(x + ih))}{h}$$

This method provides an accuracy of order 2 ($O(h^2)$) and is an obvious better choice as opposed to the vanilla finite difference method. However, this method might prove difficult to implement due to the fact that complex types are not primitive in C/C++ and hence to achieve something like this we have to rewrite the source to accept complex inputs.

Other methods

Higher order methods:

Higher order methods promise higher accuracy as the order increases. These derivatives are “longer” (meaning they are truncated at a later order in the Taylor series) and hence possess lower truncation error. Coefficients of this method can be calculated via simple linear algebra given the sample points; an example of a higher order method for calculating the first derivative with a standard five-point stencil is given below:

$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h}$$

Implementation of this method in Clad should be fairly simple; we can easily pre-store the coefficients for standard stencils and avoid the overhead of matrix multiplication/ other linear algebra operations. However, that will limit the diversity in stencils that can be used and the order of the method.

Cauchy's Integral Formula:

A simple and effective way to generalize calculation of higher order derivatives is done using Cauchy's integral formula as given below:

$$f^{(n)}(a) = \frac{n!}{2\pi i} \oint_\gamma \frac{f(z)}{(z-a)^{n+1}} \mathrm{d}z$$

Where the integral is calculated numerically. Calculation of numerical integrals might pose as an unwanted addition to Clad, however, it might be necessary to employ such methods to be able to get higher order derivatives with acceptable error bounds. The feasibility of implementation of this method in Clad may be experimented with.

Richardson Extrapolation:

This method improves the accuracy of the central difference method. It is a recursive method, with the accuracy of the estimated derivative increasing at every evaluation. The relation for a function $f(x)$ is defined below:

$$D_{m,n} = \frac{4^n D_{m,n-1} - D_{m-1,n-1}}{4^n - 1}$$

Where,

$$D_{0,0} = \frac{f(x+h) - f(x-h)}{2h}$$

And,

$$D_{1,0} = \frac{f(x+h/2) - f(x-h/2)}{2}$$

Which is very similar to the central difference formula; here `n` refers to the method and `m` refers to different multipliers for `h` ($h_m = h/2^m$). Given the high rate of convergence, this method is definitely a contender for implementation, moreover, this method may be extended to estimate higher derivatives with a satisfactory accuracy[7].

A popular computational engine, WolframAlpha, uses this method to numerically estimate derivative values.[3]

Miscellaneous

Non-scalar differentiation:

As of writing of this proposal, Clad does not possess the ability to differentiate non-scalar types such as arrays, matrices or tensors. Numerical differentiation of such types is possible and fairly simple to implement. An example of numerically differentiating a function that takes a non-scalar input (arrays here) is as follows:

$$f'(arr) = \left[D_0, D_1, D_2, \ldots, D_n\right]$$

Where:

$$D_i = \frac{f(arr_i) - f(arr)}{h_i}$$

and $\mathtt{arr}_i$ is the array with a modification of $\mathtt{arr[i]} \mathrel{+}= \mathtt{h}_i$.

This formula of forward finite difference can easily be extended to non-scalar types of higher dimensions too. This feature can be used as a fallback in Clad whenever differentiation with respect to a non-scalar type is requested.

Another interesting thing to note is that one can differentiate virtually any user-defined scalar (or even in some cases non-scalar) type using numerical differentiation as long as we are aware of a suitable step size and the basic arithmetic operations on it are well defined.

Error propagation for accurate estimates:

From the conclusions in this section, we can say that if we are aware of the floating-point errors in the function to be differentiated, we can easily substitute those in for the $2\epsilon_M$ term in the numerator of the round-off error. Hence giving us more accurate estimates of the error in the calculated derivatives. This however introduces an overhead of generating the error estimation code and may only be feasible in the case that the floating-point errors are significantly large.

Goals

1. Implement numerical differentiation support in Clad. It will be available through a dedicated interface.

2. Develop a prototype of configurable error estimation for the numerical differentiation.

Personal Information

Basic Details:

- **Name:** Garima Singh
- **Email:** garimasingh0028@gmail.com
- **Mobile Number:** +91 626 300 8121
- **Time Zone:** India (UTC +5:30)
- **Github:** grimmmyshini
- **University:** Manipal Institute Of Technology
- **Major:** Information Technology
- **Current Year:** 3rd-year Undergraduate (2022 expected)
- **Degree:** Bachelor of Technology
- **Availability:** throughout the coding period

Technical Experience:

I am a part of my university's official AI and robotics student project -- Project Manas, whose primary vision and mission are to build a fully autonomous car suited for Indian road conditions. At project Manas, my primary work involved feature addition and optimization of existing open-source software. As such, I am fairly comfortable with working with abstract ideas and quickly adapting to foreign codebases. I have also worked significantly with mathematics-heavy algorithms such as developing a modified version of an Iterative Closest Point matching algorithm that utilized semantic information from a deep neural net backend (this work was undertaken during my internship at a prominent robotics startup in India -- Swaayatt Robots).

In addition to the above, I am also working on developing an error-estimation framework using Clad, details of which can be found here. Because of this, I have significant knowledge of the codebase and am well acquainted with the different modules and their interoperability. I have been contributing to the project since December 2020 and all my commits can be found here.

Motivation:

Having worked with Clad before, I am aware of the areas which require improvement. This proposal will greatly add to improving the robustness of Clad and will also open new opportunities in the field of error-estimation and lossy-compression. If this project meets completion, clad will not only become more resilient to differentiation failures but may also be used to replace hand-derived code as-is in analysis and simulation software (one such example software is ROOT). Numerical differentiation may also be used to cross-verify the derivatives generated by Clad and their resultant granularity may be analyzed.

Timeline

Phase 1 || June 7th - July 16th

Week 1: Investigate different methods in numerical differentiation and discuss possible implementation design.

Week 2: Code up different methods and test out their feasibility and correctness. This week will mostly involve integrating the numerical diff functionality as a fail-safe for forward mode differentiation. I will mostly be integrating basic methods here, the more complex ones will be added later on.

Week 3: Take this week out to write up a few preliminary tests and work on the documentation. This task may be completed in a few days, so the rest of the days will serve as a buffer for work from the previous week.

Week 4: Start working on an interface for numerical differentiation so that it can be used standalone.

Week 5: Continue on the work from the past week. This week may also be used as a buffer for debugging.

Phase 2 || July 16th - Aug 23rd

Week 6: Analyse the estimation results and build a small module that tracks the error in the calculated values.

Week 7: Inspect complex variable methods, see if they can be extended to differentiate complex types. Also, use this time as a buffer for finishing work from past weeks.

Week 8: This week is saved for research on other methods, one prominent class of methods to look into here might be iterative methods since they are much more stable than other numerical methods.

Week 9: Prepare demos and tests. Take this week to work on documentation and patch missing features if any.

Week 10: week saved to prepare reports/presentations on the work done. May also use this as a buffer to finish up the work from previous weeks.

References

[1] Vassilev, Vassil, Penev, Alexander, & Shakhov, Roman. (2020). Error estimates of floating-point numbers and Jacobian matrix computation in Clad. Zenodo. http://doi.org/10.5281/zenodo.4134097

[2] Sauer, Timothy (2012). Numerical Analysis. Pearson. p.248.

[3] ND—Wolfram Language Documentation

[4] https://people.clas.ufl.edu/kees/files/NumericalDifferentiation.pdf

[5] https://www.uio.no/studier/emner/matnat/math/MAT-INF1100/h10/kompendiet/ Chapter 11

[6] Shilov, George. Elementary Real and Complex Analysis.

[7] Ström, T., Lyness, J.N. On numerical differentiation. BIT 15, 314–322 (1975). https://doi.org/10.1007/BF01933664

[8] Lyness, J. N., and C. B. Moler. “Numerical Differentiation of Analytic Functions.” SIAM Journal on Numerical Analysis, vol. 4, no. 2, 1967, pp. 202–210. JSTOR, www.jstor.org/stable/2949389. Accessed 10 Apr. 2021.