Add Numerical Differentiation Support in Clad

Garima Singh | Google Summer of Code 2021
Mentors: Vassil Vassilev and Alexander Penev

Overview

What is Clad?

Clad is an automatic differentiation library implemented as a Clang plugin.

What is Automatic Differentiation (AD)?

Very generally speaking, AD is a set of techniques to evaluate the derivative of a computer program. It computes the exact derivative of a program (if any exists).

Why Numerical Differentiation for an AD library?

Due to some constraints it might be inefficient or even impossible to use AD for a function, this is where numerical differentiation comes in.

Basic Implementation Idea

The formula:

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

For a general case:

$$f'(x_0, x_1, ..., x_n)_{x_i} = \frac{f(..., x_i + h, ...) - f(..., x_i - h, ...)}{2h}$$

The idea:

For multi-argument functions, have a ‘magic’ function to pick and update the correct parameters and forward the rest of the parameters.

Basic Implementation Idea

The implementation:

Implement the ‘magic’ function using templates, parameter packs and index_sequences to pick the correct i^{th} parameter.

This allows us to be very concise and flexible in our implementation (as will be described later). We also have a functionality to print the errors associated with numerical differentiation.

An algorithmic overview of the implementation looks like the following:

```
for each i in args, do:

fx1 := f(updateIndexParamValue(args,
i, sequence)...)

fx2 := f(updateIndexParamValue(args,
i, sequence)...)

grad[i][0] := (fx1 - fx2)/(2 * h)

end for
```

Some Examples: With Clad

For the clad use-case, we synthesize a call to either the “forward” numerical differentiation function or the “reverse” numerical differentiation function.

```
double test_2(double x){
    return std::log10(x);
}
```

The target function

The generated derivative

```
double test_2_darg0(double x) {
    double _d_x = 1;
    return forward_central_difference(std::log10, x, 0, 0, x) * _d_x;
}
```

Some Examples: Standalone

Standalone, the numerical differentiation is capable of a lot. We have the many ways listed below:

- For in-built scalar and non-scalar types. These includes `double`s, `float`s, `double*`, etc.
- Support for user defined types as input (both value and pointer forms). This can be achieved by overloading the `UpdateIndexParamValue` with the special type.
- Can differentiate overloaded operators.
- Can differentiate functors.

Thank You!

You can contact me at garimasingh0028@gmail.com!