Google Summer of Code 2014

Automatic Differentiation library using Cling

https://github.com/vgvassilev/clad

Mentors: Lorenzo Moneta
Vasil Vasilev

Student: Martin Vasilev

About CLAD

- Clang based c++ plugin
- Provides a tool for applying automatic differentiation
- Applications

CLAD Status

- Supports member functions (including template classes)
- Supports functors
- Supports mixed derivatives
- Supports n-th order derivatives
- Constant folding

Demo

- Functors

```
class SimpleExpression {
    float x, y;
public:
    SimpleExpression(float x, float y) : x(x), y(y) {}
    float operator()(float x, float y) { return x * x + y * y;}
};
```

```
clad::differentiate(&SimpleExpression::operator(), 0);
clad::differentiate(&SimpleExpression::operator(), 1);
```

```
float operator_call_dx(float x, float y) {
  return (1.F * x + x * 1.F) + ((0.F * y + y *
0.F));
}

float operator_call_dy(float x, float y) {
  return (0.F * x + x * 0.F) + ((1.F * y + y *
1.F));
}
```

```
float operator_call_dx(float x, float y) {
    return (x + x);
}

float operator_call_dy(float x, float y) {
    return (y + y);
}
```

Demo

- Class method calls

```
class A {
public:
    float g_1(float x, float y) {  return x*x + y; }
};
```

```
clad::differentiate(&A::g_1, 0);
clad::differentiate(&A::g_1, 1);
```

```
float g_1_dx(float x, float y) {
    return (1.F * x + x * 1.F) + (0.F);
}

float g_1_dy(float x, float y) {
    return (0.F * x + x * 0.F) + (1.F);
}
```

```
float g_1_dx(float x, float y) {
    return (x + x);
}

float g_1_dy(float x, float y) {
    return 1.F;
}
```

Demo

- Template functions

```
template<typename T>
T multiplication(T x) {
    return x * x;
}
```

```
clad::differentiate(multiplication<float>, 0);
clad::differentiate(multiplication<double>, 0);
```

```
float multiplication_dx(float x) {
    return (1.F * x + x * 1.F);
}

double multiplication_dx(double x) {
    return (1. * x + x * 1.);
}
```

```
float multiplication_dx(float x) {
    return (x + x);
}

double multiplication_dx(double x) {
    return (x + x);
}
```

Demo

- Mixed derivatives

```
float f1(float x, float y) {
  return x * x + y * y;
}
```

```
auto f1_dx_clad_func = clad::differentiate(f1, 0);
auto f1_dx_dy_clad_func = clad::differentiate(f1_dx, 1);
```

```
float f1_dx(float x, float y) {
    return (1.F * x + x * 1.F) + ((0.F * y + y
* 0.F));
}

float f1_dx_dy(float x, float y) {
    return ((0.F * x + 1.F * 0.F) + ((0.F *
1.F + x * 0.F))) + ((((0.F * y + 0.F * 1.F) +
((1.F * 0.F + y * 0.F)))));
}
```

```
float f1_dx(float x, float y) {
    return (x + x);
}

float f1_dx_dy(float x, float y) {
    return (0.F);
}
```

Demo

- N-th derivative

```
float test_2(float x, float y) {
  return x * x + y * y;
}
```

```
clad::differentiate<3>(test_2, 1);
```

```
float test_2_dy(float x, float y) {
    return (0.F * x + x * 0.F) + ((1.F * y + y * 1.F));
}

float test_2_d2y(float x, float y) {
    return ((0.F * x + 0.F * 0.F) + ((0.F * 0.F + x * 0.F))) +
((((0.F * y + 1.F * 1.F) + ((1.F * 1.F + y * 0.F)))));
}

float test_2_d3y(float x, float y) {
    return (((0.F * x + 0.F * 0.F) + ((0.F * 0.F + 0.F * 0.F))) +
((((0.F * 0.F + 0.F * 0.F) + ((0.F * 0.F + x * 0.F)))))) + ((((((0.F *
y + 0.F * 1.F) + ((0.F * 1.F + 1.F * 0.F))) + ((((0.F * 1.F + 1.F *
0.F) + ((1.F * 0.F + y * 0.F)))))))));
}
```

```
float test_2_dy(float x, float y) {
    return (y + y);
}

float test_2_d2y(float x, float y) {
    return (1.F + 1.F);
}

float test_2_d3y(float x, float y) {
    return (0.F);
}
```

Demo

- Heat equation $\frac{\partial u}{\partial t} - \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) = 0$

```
double heat_equation_func(double x, double y, double z, double t) {
  return sin(x) + sin(y) + sin(z) + t * t;
}
```

```
bool heat_equation(double x, , double y, double z, double t) {
  float a = 1.4;
  auto heat_equation_dt = clad::differentiate(heat_equation_func, 3)
  auto heat_equation_d2x = clad::differentiate<2>(heat_equation_func, 0);
  auto heat_equation_d2y = clad::differentiate<2>(heat_equation_func, 1);
  auto heat_equation_d2z = clad::differentiate<2>(heat_equation_func, 2);

  return heat_quation_dt.execute(x, y, z, t) == a *
    (heat_quation_d2x.execute(x, y, z, t) + heat_quation_d2y.execute(x, y, z, t)
     + heat_quation_d2z.execute(x, y, z, t));
}
```

Clad in action

- Used in raytracer
  - Calculates normal vectors in tracing

Future work

- Derive function taking vector of points
- Implement second and higher order mixed derivatives

Google Summer of Code

Thank you

https://github.com/vgvassilev/clad