Progress Report

March-April 2022

Agenda

- Solved and inspected multiple basic examples with RooFit.
- Conceptualized the code generation approach for RooFit.
  - Worked on converting a Histfactory mode to a n-ary tree like structure that helps parse and construct the code
  - Built dummy RooFit classes that can be used to perform code generation.
- Presented the Error Estimation Framework at SIAM UQ’22
- Worked on multiple issues/features for the FP Error Estimation Framework.
  - Error printing for `clad::estimate_error`.
  - Build default `std::abs` cast over errors + other abstractions over clang expression generation.
  - Error estimation of nested function calls - In Progress.

RooFit Examples with Clad

- The initial part of the work was to check if clad can differentiate simplified RooFit models.
- As such, we manually coded the C++ equivalent of a basic RooFit model and also a class that represented the squashed model.

```
double nll002(double lumi, double sigXsecOverSM, double
gamma_stat_channel1_bin_0, double gamma_stat_channel1_bin_1) {
  for (std::size_t iSample = 0; iSample < nSamples; ++iSample)
{
     // calculate mu
  }
  // calculate nll
  return nll;
}
```

```
auto nll002_gradient = clad::gradient(nll002,
                                      "lumi,sigXsecOverSM, \
                                       gamma_stat_channel1_bin_0, \
                                       gamma_stat_channel1_bin_1");
```

RooFit Examples with Clad

```
class HFLikelihoodWrapper final : public RooAbsReal {
protected:
  void evaluateGradient(double *out) const override {
     out[0] = 0;
     out[1] = 0;
     out[2] = 0;
     out[3] = 0;
     nll002_gradient.execute(_lumi, _mu, _gamma0, _gamma1, &out[0], &out[1], &out[2], &out[3]);
  }
};
```

- Lastly, we call minimize with the object of the squashed class.
- Two major issues with minimization with clad produced gradients:
  - Minuit1 does not converge correctly - possibly because it relies on the step size provided by first order numerical derivatives. The step size by that point becomes a good estimate.
  - With Minuit2, the convergence is correct but it takes longer. This could be attributed to the line search that is performed for minimizations

RooFit Examples with Clad

- The biggest challenge is to use AD on software that is really optimized for using numerical derivatives. As such the AD approach ends up suffering from overhead incurred by the workarounds.

- All this work is documented on the following github repository: https://github.com/guitargeek/roofit-clad-work

Code-Gen with RooFit

- Model the compute graph with an N-ary tree, model the RooFit classes with a **`translate`** function that translates the corresponding RooFit object to C++ code.
- All this generation is fairly static in nature, only the inputs/results to each **`translate`** function change.

```
std::string NTree::getCode() {
  std::string code = "", global = "";
  getCodeRecur(this, code, global);
  return global + code;
}
```

```
void NTree::getCodeRecur(NTree* head, std::string&
code, std::string& global) {
   for(auto it : head->child) {
     getCodeRecur(it, code, global);
   }
   code += head->data->translate(head, global);
}
```

Code-Gen with RooFit

```
class ExRooHistFunc : public ExRooReal{
public:
 // . . .
 std::string translate(NTree *head, std::string &globalScope) override {
   if (init) {
     globalScope += "double " + name + "[" + std::to_string(nBins) + "]" + initalizer + ";\n";
     if (unroll){
       globalScope += "double binBoundaries[" + std::to_string(nBins + 1) + "];\n";
       std::string x = head->child[0]->data->getResult();
       unrollGetBins(x, bin, globalScope);
     }
     init = false;
   }
   result = name + "[" + bin + "]";
   return "";
 }
};
```

Code-Gen with RooFit

```
class ExRooHistFunc : public ExRooReal{
public:
 // . . .
 std::string translate(NTree *head, std::string &globalScope) override {
   if (init) {
     globalScope += "double " + name + "[" + std::to_string(nBins) + "]" + initalizer + ";\n";
     if (unroll){
       globalScope += "double binBoundaries[" + std::to_string(nBins + 1) + "];\n";
       std::string x = head->child[0]->data->getResult();
       unrollGetBins(x, bin, globalScope);
     }
     init = false;
   }
   result = name + "[" + bin + "]";
   return "";
 }
};
```

`getResult` returns the variable a RooFit object represents, used to propagate results/input between nodes.

Code-Gen with RooFit

```
class ExRooHistFunc : public ExRooReal{
public:
 // . . .
 std::string translate(NTree *head, std::string &globalScope) override {
   if (init) {
     globalScope += "double " + name + "[" + std::to_string(nBins) + "]" + initalizer + ";\n";
     if (unroll){
       globalScope += "double binBoundaries[" + std::to_string(nBins + 1) + "];\n";
       std::string x = head->child[0]->data->getResult();
       unrollGetBins(x, bin, globalScope);
     }
     init = false;
   }
   result = name + "[" + bin + "]";
   return "";
 }
};
```

`globalScope` stores variable declarations so that they can be placed before the generated code.

Code-Gen with RooFit

```
class ExRooHistFunc : public ExRooReal{
public:
 // . . .
 std::string translate(NTree *head, std::string &globalScope) override {
   if (init) {
     globalScope += "double " + name + "[" + std::to_string(nBins) + "]" + initalizer + ";\n";
     if (unroll){
       globalScope += "double binBoundaries[" + std::to_string(nBins + 1) + "];\n";
       std::string x = head->child[0]->data->getResult();
       unrollGetBins(x, bin, globalScope);
     }
     init = false;
   }
   result = name + "[" + bin + "]";
   return "";
 }
};
```

If an object specifies “code”, we build and return it so that it can be added to the function body.

We also store the variable represented by this class’s object in the result variable.

Code-Gen with RooFit

```
ExRooAbsReal SF1("SigXsecOverSM");
ExRooConst SF2(1), SF3(1);
ExRooHistFunc sig(true, "ibin", "sig", "{20, 10}"),
              bgk1(false, "ibin", "bgk1", "{100, 0}"),
              bgk2(false, "ibin", "bgk2", "{0, 100}");
ExRooAbsReal X("x");
ExRooRealSum MU("mu", {&sig, &bgk1, &bgk2}, {&SF1, &SF2, &SF3});

NTree *head = new NTree(/*build a nested tree*/);
```

```
std::string code = "double nll(double x, double SigXsecOverSM) { \n"
                           + head->getCode()
                           + " return " + head->data->getResult() + ";\n}\n";
std::cout << code;
gInterpreter->Declare(code.c_str());
gInterpreter->ProcessLine("#include \"clad/Differentiator/Differentiator.h\"");
gInterpreter->ProcessLine("auto df = clad::gradient(nll, \"SigXsecOverSM\");");
gInterpreter->ProcessLine("df.dump();");
```

Code-Gen with RooFit

```
double nll(double x, double SigXsecOverSM) {
double sig[2]{20, 10};
double binBoundaries[3];
int ibin= 0;
while (binBoundaries[ibin] < x) {
   ibin++;
}
double bgk1[2]{100, 0};
double bgk2[2]{0, 100};
double mu = 0;
mu += sig[ibin] * SigXsecOverSM;
mu += bgk1[ibin] * 1.000000;
mu += bgk2[ibin] * 1.000000;
return mu;
}
```

- We are able to also calculate the derivative of the generated code using ROOT cling.
- Waiting on a discussion with Jonas to figure out how to move forward.

SIAM UQ 2022

- Talk went well, the recording is up here: https://siamuq22.us2.pathable.com/meetings/virtual/5MFQ2KnbzDMfH4TWA

- Possible collaboration opportunities, needs some patches/improvements to the framework.

Misc. Clad Features

- Working on adding the following features to the error estimation framework.
  - Default-ing the `SetError` implementation. **Done**
  - Build default `std::abs` cast over errors + other abstractions over clang expression generation. **Done**
  - Error estimation of nested function calls. **Blocked**
    - The DiffMode does not persist over nested function calls.
    - Also the plugin is system needs modification everytime I add some new feature.
    - Also multiple test failures on debug builds due to failed assertions, might be a good idea to check them out.
  - Error printing for `clad::estimate_error`. **Queued**

Future Plans

- Figure out next steps with Jonas for the RooFit + AD work.
- Try getting unblocked on the clad issues.
- Also develop a small example/benchmark to understand the overhead of the error estimation code generation on the usual gradient generation.