New ROOT TFormula class

Lorenzo Moneta (PH-SFT),
Maciej Zimnoch (University of Wroclaw, GSoC)
presented by
Fons Rademakers (PH-SFT)

October 15, 2013
Outline

• **TFormula class in ROOT**
  - Introduction
  - Current functionality and limitations

• **New TFormula class**
  - *Developed by Maciej Zimnoch (Google Summer of Code student 2013)*
  - Using Cling to evaluate expressions

• **Current Status**
  - Performance tests

• **Future improvements**
  - Auto-Differentiation

• **Conclusions**
Introduction

- **TFormula class in ROOT**
  - Class for evaluating mathematical functions provided as expression strings
  - ROOT function class (TF1) derives from TFormula
    - Uses TFormula constructs for making functions from string
  - TF1 is used for fitting and for plotting functions in ROOT

- **Examples of TFormula constructs:**
  - Simple functions
    
    ```
    TFormula("f1", "sin(x)" );
    TFormula("f2", "x^2+2" );
    ```
  - Composition of functions
    
    ```
    TFormula("f3", "f1 + f2" );
    ```
Formula Constructs

– Function with parameters:

\[
\text{TFormula(“f”, “[0] * \sin(x * [1])”);}
\]

– Using predefined functions:

\[
\text{TFormula(“f”, “gaus”);}
\]

• \textit{gaus} is equivalent to: \([0]*\text{TMath::Gaus}(x,[1],[2])\)

\[
\text{TFormula(“f”, “pol2”);}
\]

• \textit{pol2} is equivalent to: \([0] + [1]*x + [2]*x^2\)

– Using any library function:

\[
\text{TFormula(“f”, “ROOT::Math::chisquared_pdf(x,[0])”);}
\]
Current Limitations

- **TFormula** contains customized code to parse the expression string and evaluate it
  - Custom parser (does not use CINT C/C++ parser)
    - CINT too slow to evaluate functions
  - Has been optimized for speed
    - Used for fitting
    - Used in TTreeFormula to query TTree’s
  - Parsing expressions is complex
    - Several 1000’s lines of code
  - Very difficult to extend and maintain
    - E.g. adding new C++11 syntax
Current Limitations (2)

- Dictionary (CINT and now Cling) is used for functions from the library
  - E.g. functions from TMath or ROOT::Math
  - Slow to execute since it function call is wrapped in interpreted code
- TFormula defines some pre-defined functions:
  - gaus, polN, expo, landau
    - Used in the formula as compiled code
    - E.g. "gaus" is much faster than "[0] * TMath::Gaus(x,[1],[2])"
- Pre-defined functions add extra-complexity in the code
  - Difficult to add new ones
Current Limitations (3)

- Do we really need this customized parsing code in TFormula?
• Uses Cling available in ROOT 6
• Replace old parser with the JIT provided by Cling
  – A real C++ interpreter
  – More confident in correctness of results when using a real compiler
  – Better detection of syntax errors
  – Reduce substantially the code size
• Maintain the old functionality for backward compatibility
• Extensible
  – Easy to add new functions
  – Use different variables names than x,y,z
• Scale to large and complex expressions
How Does it Work

- **TFormula** creates a C/C++ functions which is passed to Cling

```c
TFormula("f", "[0] * sin( x * [1] )");
```

```c
Double_t TF__f(Double_t *x, Double_t *p)
{
    return p[0]*TMath::Sin(x[0]*p[1]) ;
}
```

- The created function is now compiled on the fly using the JIT of Cling
Evaluation of TFormula

• Faster evaluation: it is compiled code!
  – No need to have a dedicated parser to analyze and compile the code as in old TFormula

• JIT compilation is done at initialization time, not when evaluating the expression

• The created function is evaluated using its function pointer, which can be retrieved via the ROOT interpreter interface
  – Very small overhead compared to calling the function via MethodCall interface

• Pre-defined functions (gaus) and library functions (TMath::Gaus) are treated in the same way
**TFormula Parsing (Initialization)**

- **TFormula** parsing is now limited to clean up the input expression:
  - interpreting parameter names
    - \([0] \rightarrow \text{par}[0]\)
  - interpreting variables
    - \(x, y, z \rightarrow x[0], x[1], x[2]\)
  - translating pre-defined expressions
    - \(\text{gaus}(0) \rightarrow \text{par}[0]\*\text{TMath::Gaus}(x[0],\text{par}[1],\text{par}[2])\)
- Check validity of expression
- Create the C/C++ function for Cling
- Code is reduced substantially
Advantages of New TFormula

• More flexible code
  – Easy to add new pre-defined functions as shortcuts for user convenience
    • Just one line of code to change to include the translation symbol corresponding to the pre-defined function
    – I.e. what “myfunc” is translated to in C++
  – One can add meaningful parameters directly in the expression
    • TFormula f("f", "A * sin( x * B )");
    • f.SetParameter("A",1); f.SetParameter("B",2);
  – Function dimension is not limited to 4
    • One can define functions with several variables
    • Use f.AddVariable(..)
Performance Tests

• Tests of new **TFormula**:  
  – Formula with 4 variables and 1000 parameters:  
    • Initialization:  
      – Time = 0.1 seconds  
    • 1 million evaluations:  
      – Time = 2.1 seconds

• Using Old **TFormula**:  
  – Using the same expression (1000 parameters)  
    • Initialization  
      – Time = 1.2 seconds  
    • Evaluation:  
      – Fails to evaluate such large expressions  
      – Does not scale for such large formula
Performance Test (2)

- Test of evaluation of new vs old `TF`ormula
  - Time for 1 evaluation (in ns)

<table>
<thead>
<tr>
<th>Expression type</th>
<th>New TFormula v5.99</th>
<th>Old TFormula v5.34</th>
</tr>
</thead>
<tbody>
<tr>
<td>predefined functions gaus(0) + gaus(3)</td>
<td>60 ns</td>
<td>65 ns</td>
</tr>
<tr>
<td>interpreting expression \texttt{TMath::Gaus(...)}</td>
<td>65 ns</td>
<td>400 ns</td>
</tr>
<tr>
<td>formula functions \texttt{exp(-0.5*(x-[1])/[2])^2...}</td>
<td>60 ns</td>
<td>200 ns</td>
</tr>
<tr>
<td>compiled functions double f(double<em>x, double</em>p) { return TMath::Gaus(...)};</td>
<td>50 ns</td>
<td>50 ns</td>
</tr>
</tbody>
</table>
Current Status

• New `TFormula` class is available on github
• It has already been integrated in `TF1` and it can be used for fitting and plotting functions
  – Several remaining issues are being fixed
• Will soon be integrated in the ROOT master
  – Still working on improving:
    • Adding more pre-defined functions
    • Better interface to add new variables and new parameter names in expression
Further Improvements

- Computing Derivatives inside \texttt{TFormula} using Auto-Differentiation (AD)

- AD allows to compute precisely and efficiently the function derivatives
  - Reduces numerical error and reduces computation cost compared to numerical derivatives
  - Extremely useful for fitting/minimization
Automatic Differentiation

- Auto-Differentiation requires to transform the semantic of a program (function)
  - Works by combining values of basic operations using the derivative chain rule

\[
f(x) = g(h(x))
\]
\[
f'(x) = g'(h(x)) \cdot h'(x)
\]
\[
\frac{\partial f}{\partial x} = \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial x}
\]

\[
f(x_1, x_2) = x_1 x_2 + \sin(x_1)
\]

Forward propagation
AD Prototype in Cling

- Prototype implementation of AD as a plug-in for Cling
  - Developed by Violeta Ilieva (Google Summer of Code 2013 student from Princeton University) and Vassil Vassilev (CERN PH-SFT)
  - Uses forward propagation and source code transformation
  - Uses Cling to parse code and create derivative functions

```c
float example_fn(int x, int y, int z) {
    return x + y * z;
}
diff(example_fn, x); // = 1
diff(example_fn, y); // = z
diff(example_fn, 2); // = y
```

```c
float example_fn_derived_x(int x, int y, int z) {
    return 1;
}
float example_fn_derived_y(int x, int y, int z) {
    return z;
}
float example_fn_derived_z(int x, int y, int z) {
    return y;
}
```
Conclusions

• The new `TFormula` leverages the Cling functionality
  – Will be integrated in ROOT 6
  – Provides a more robust and faster evaluation of functions
  – Scales to very large expressions
  – Can be used for building the parametric functions for fitting
  – Integration with `TTreeFormula` and RooFit in the pipeline

• Integrate Auto-Differentiation developments
  – Very interesting and useful technique which can be integrated in ROOT and RooFit to compute derivatives of functions
  – Will speed-up minimization (fitting) of very complex functions
    • E.g. models used at LHC (Higgs combination model)
  • Thanks to Google for allocating us GSoC students!