A Python upgrade to the GooFit package for parallel fitting

Henry Schreiner[1,*], *Himadri* Pandey[1], *Michael D* Sokoloff[1], *Bradley* Hittle[2], *Karen* Tomko[2], and *Christoph* Hasse[3]

[1]University of Cincinnati, Cincinnati, Ohio, USA
[2]Ohio Supercomputer Center, Columbus, Ohio, USA
[3]CERN / Technische Universität Dortmund (DE), Dortmund, Germany

Abstract. The GooFit highly parallel fitting package for GPUs and CPUs has been substantially upgraded in the past year. Python bindings have been added to allow simple access to the fitting configuration, setup, and execution. A Python tool to write custom GooFit code given a (compact and elegant) MINT3/AmpGen amplitude description allows the corresponding C++ code to be written quickly and correctly. New PDFs have been added. The most recent release was built on top of the December 2017 2.0 release that added easier builds, new platforms, and a more robust and efficient underlying function evaluation engine.

1 Introduction

High Energy Physics experiments around the world are producing record amounts of data. Existing tools, such as the RooFit fitting framework, provide flexible and powerful abstractions for building distributions to fit that data with, but this power comes at a cost; this is computationally expensive, and often only runs on a single core. Modern architectures provide many cores, as well as new computing paradigms, such as GPUs, that provide significant new potential for high performance computations, but are non-trivial for physicists to use to build distributions in a familiar description style.

GooFit is a high-performance multi-thread and GPU ready framework providing a similar syntax to RooFit [1, 2]. Some comparisons are shown in figure 1 and table 1. GooFit provides composition of model pieces in the same manor as RooFit, but is powered by GPUs. It also provides ready to use models for common Probability Distribution Functions (PDFs) and quantum mechanical amplitudes, as well as specialized 3-body physics models. The GooFit 2.0 release [3] added a simpler build process, making it easy for users to pick up and run GooFit code, and combined work from several forks, providing more physics models, 4-body models, and more features.

The next two releases of GooFit, given in figure 2, brings us to version 2.2, and the work contained in them will be the topic of this discussion. The most notable new change is the addition of fully functioning Python bindings, allowing users to code the composition of PDFs in a dynamic scripting language, opening up new possibilities for quick modeling and interesting interactions with other Python libraries, such as for plotting. Other changes include a new indexing system, which makes amplitudes and PDFs easier to write, and provides some

[*]e-mail: henry.fredrick.schreiner@cern.ch

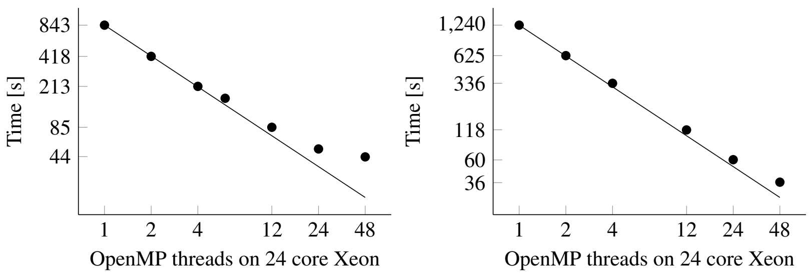

Figure 1. On the left: Fit from an analysis of $D^0 \to \pi^+\pi^-\pi^0$, with 16 time-dependent amplitudes [4]. Original RooFit code took 19,489 s on a single core. 40 free parameters and 100,000+ events. On the right: The “ZachFit”: a fit to a $M(D^{*+}) - M(D^0)$ measurement [5]. 142,576 events in an unbinned log likelyhood fit.

small performance benefits on GPUs. A prototype for a uniform decay language, shared with other packages such as AmpGen [6], provides a powerful new frontend that can be used for user code.

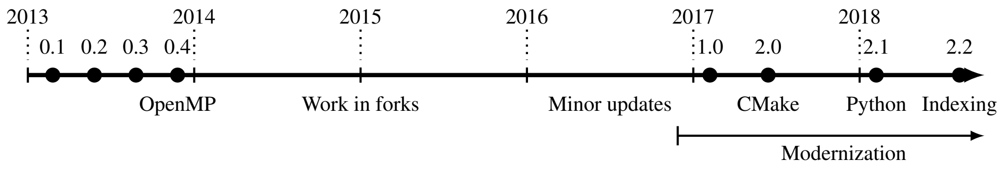

Figure 2. Timeline of GooFit development. Key points are version 2.0: New build system, C++11, and 4-body time dependent analyses support; version 2.1: Python bindings using Pybind11; and version 2.2: new indexing (and lots of Python improvements).

Table 1. GPU/CPU performance comparison. The models are the same as figure 1. From ref. [7].

Number of Cores	Hardware	Pipipi0	ZachFit
2 Cores	Core 2 Duo	1,159 s	738 s
GPU	GeForce GTX 1050 Ti	86.4 s	60.3 s
GPU	Tesla K40	64.0 s	60.3 s
MPI & GPU	Tesla K40 ×2	39.3 s	54.3 s
GPU	Tesla P100	20.3 s	23.5 s

2 Python Bindings

Possibly the most significant new feature in the recent GooFit releases is the support for Python as a composition language. This allows simpler installation, simpler user code, faster development, easier advanced control of the fit, and interoperability with the rest of the Python ecosystem, such as for plotting.

2.1 Building and Installing

To install GooFit, a user now can simply use the Package Installer for Python, `pip`, to install it much like any other package. GooFit will compile from source during the install process so that it can select the maximum level of optimizations supported on your system. It will look for, and automatically use, CUDA and the GPU backend instead of the CPU backend as available. In `pip` version 10, you can simply install GooFit; older versions require you to install the SciKit-build package before installing GooFit.

This Python installer integration was provided using the SciKit-Build package, released by the developers of CMake. It adapts a CMake based build (such as GooFit) to the Python ecosystem, and is one of the reasons support for alternate build systems was added to `pip` 10. Build options, such as explicit CPU/GPU selection, can be passed through to the underlying CMake build system if needed.

In the GooFit repository, 12 of the original 13 GooFit examples have been adapted to the new Python syntax. For many of the examples, ROOT is no longer required in the Python version, being replaced by common Python libraries.

2.2 Syntax

```cpp
#include <goofit/...>
using namespace GooFit;

Observable x{"x", 0, 10};
Variable mu{"mu", 1};
Variable sigma{"sigma", 1, 0, 10};
GaussianPdf gauss{"gauss", &x, &mu, &sigma};
UnbinnedDataSet ds{x};

std::mt19937 gen;
std::normal_distribution<double> d{1, 2.5};
for(size_t i=0; i<100000; i++)
       ds.addEvent(d(gen));

gauss.fitTo(&ds);

std::cout << mu << std::endl;
```

```python
from goofit import *
import numpy as np

x = Observable("x", 0, 10)
mu = Variable("mu", 1)
sigma = Variable("sigma", 1, 0, 10)
gauss = GaussianPdf("gauss", x, mu, sigma)
ds = UnbinnedDataSet(x)

data = np.random.normal(1, 2.5, (100000,1))
ds.from_matrix(data, filter=True)

gauss.fitTo(ds)

print(mu)
```

Figure 3. Example of GooFit 2.2 code in C++ (left) and Python (right). Some includes and main-function code have been suppressed in the C++ version for clarity. This code snippit fits a Gaussian PDF to a Gaussian distribution, and then prints the fit result for the μ parameter to the screen.

The syntax in the Python version of GooFit is very similar to that of the C++ version. In Python, memory is managed for the user, with Python keeping track of refcounts even when the only way to access a GooFit object is through another GooFit object. See figure 3 for a side-by-side example of C++ and Python code for GooFit.

On top of this, some "Pythonizations" have been added; a term popular in the Python community for customizing the bindings to allow the user to write code that looks simple and clear and more Pythonic. An example of such a feature is setting the value of a Variable. In C++, you call methods for setting and getting a variable's values using `setValue` and `getValue`. You can do this in Python as well, but you can also just access or assign to the `value` property on the variable. So if you want to set a variable named `var` to 1.2, you would use either the C++ compatible expression `val.setValue(1.2)` or the more Pythonic expression `val.value = 1.2`. You can also use many of the built in features of Python on GooFit objects directly, such as printing, lengths, indexing, and more.

Several methods have been added to make it easy to convert events to and from Numpy arrays. The user can choose to filter out-of-range values during the conversion.There are also utilities to extract interesting quantities, evaluations, and caches from GooFit into Numpy arrays.

2.3 New features

To simplify the use of GooFit from Python, several new tools were added; these tools are usually available for C++ code as well. Directly evaluating a PDF over the currently set Observable limits and binning can be done with the `evalutatePdf` method. One dimensional Monte Carlo generation on the CPU is added, as well, with the `fillMCDataSimple` method. A new tool was added called `DalitzPlotter`, which makes evaluation and simple MC generation over a 3-body amplitude model easy to do. The existing 4-body Monte Carlo generation, powered by MCBooster [8], was also improved primarily for easy Python access.

Examples were added in Python for this new functionality, using Jupyter notebooks to provide interactive visualization of 1D and Dalitz models.

2.4 Documentation

Documentation for GooFit has been improved and integrated with the Python bindings. Most PDFs have relevant documentation covering the mathematics that have been implemented. This is picked up by the Doxygen parser, and included in the online API documentation for the C++ code. The same documentation is also exported to the Python bindings, and gets rendered into HTML with math intact inside a Jupyter notebook if you display a PDF class. This extraction was done in CMake by selecting the Doxygen style comments, printing them out into new generated headers, and giving that to the Python bindings. No external dependencies are required; all scripting is done directly in CMake.

3 The New Indexing System

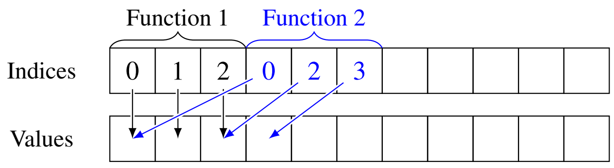

Figure 4. The classic GooFit indexing system. Each function keeps track of a set of indices, along with the total number of indices it "owns" (not shown). GooFit then looks up the values from the given indices; this is internally referred to as GooFit's "double lookup" system. The meaning of indices are hard-coded into the function and mostly the programmers responsibility to correctly handle.

A key feature of GooFit is the indexing system; this is the system GooFit uses to allow the PDF and amplitude code to access shared values. The original system, shown in figure 4, just stores values once, but then uses a double lookup system to find the correct index to read.

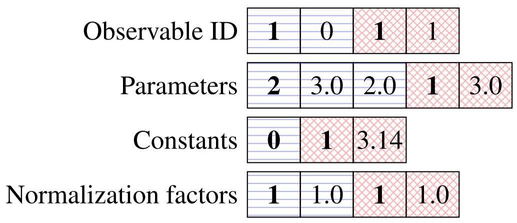

Figure 5. The new GooFit indexing system. Four arrays are stored, and instead of storing a lookup, each parameter is copied in by the underlying system as needed (potentially multiple times). This reduces the lookup cost in some cases.

The new system, shown in figure 5, uses a little preprocessing to store values in a structure that no longer requires a double lookup, saving a significant amount of time in some cases.

The indexing system redesign in GooFit 2.2 targets improved GPU performance. The indexing system in GooFit needs to track the Variables, Observables, and any constants used by any combination of PDFs. This information is packed into four separate buffers. The primary index buffer provides either a constant integer value, or contains the index into one of the three subsequent buffers which contained events, parameters, or floating-precision constants [2]. For a NVIDIA Tesla P100 GPU, 16 blocks of 32-byte sized segments specific to a thread can be cached. Two problems occur with this 32 byte memory segment that affects GPU performance. First is that if only 8 of 32 bytes are used, then the memory transaction is wasteful. Second, the segments may need to be loaded multiple times during the execution of a PDF, which is unsatisfactory for GPU performance. Using the new indexing system improves this access pattern by having memory reads behave coalesced. A side effect of the new indexing system is duplicated memory for a specific Variable or Observable. The performance improvement at the cost of this duplicating memory has a larger impact on performance by providing better memory access coherence for each PDF.

The second improvement with the indexing system provides is less complexity in developing and testing PDFs as well by providing a method for testing that indices specific in PDF construction match with the device function. Helper functions have been added allowing for registered Variables and Observables to also return the index that needs to be used within the device function. This provides a simpler mechanism for creating and debugging new PDFs by verifying indices in the device function.

An additional change provided with the indexing update is the ParameterContainer structure which provides a consistent method for accessing all indexed values. Each device function requires a ParameterContainer reference, and each device function will need to increment the internal structure. Any additional caching techniques are hidden in this structure.

Figure 6 provides a performance comparison between the old indexing system in GooFit 2.0 and the new indexing system found in GooFit 2.2 by calculating a 3-body decay. The results are captured on a Dual Xeon E5-2680 with 28 cores and an NVIDIA Tesla P100 GPU. The complete 3-body decay runtime improved by over 30% with GooFit 2.2 and over 9 million events on the GPU ("Dalitz" example, far right plot in figure 6).

4 A Uniform Decay Language

GooFit has preliminary support for decays written in the DecayLanguage syntax as implemented in AmpGen [6], primarily implemented as a standalone Python package in the SciKit-HEP organization [9].

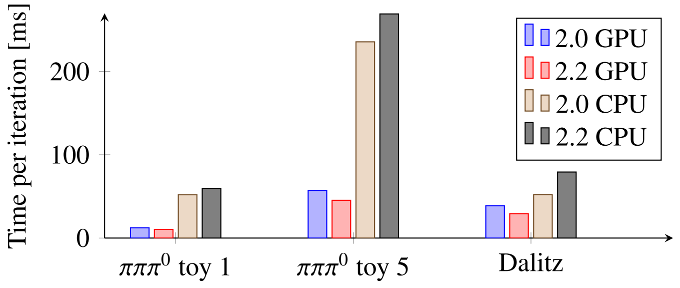

Figure 6. Comparison of GooFit 2.0 vs GooFit 2.2 on a Dual Xeon E5-2680 with 28 cores running at 2.4 Ghz, with an NVIDIA P100 GPU. Performance gains can be seen on the GPUs when upgrading to the new indexing system in GooFit 2.2. A penalty can be seen for the CPU version; CPU optimizations could be added in the future but were not considered a priority for now.

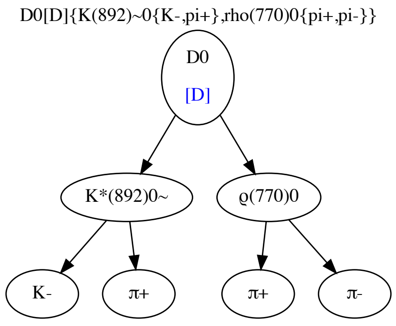

Figure 7. Example of a single $D_0 \to K^-\pi^+\pi^+\pi^-$ decay using DecayLanugage's line display feature on an AmpGen line (syntax of an Ampgen shown at the top).

AmpGen is a tool for compiling amplitudes for use in the LHCb event generator. It is also available as a stand-alone package apart from the LHCb framework. It builds PDFs symbolically in C++, then converts them to a C file that can be compiled into a simple library. However, it also provides a clean, powerful grammar for describing decays. See figure 8 for an example of the syntax.

To make this language a general one that can be used across packages like GooFit, a new decay language parser was built in Python called DecayLanguage framework and released in Beta form. This implements an AmpGen compatible syntax, but is designed to produce code for other systems; the first implementation is provided, and produces C++ GooFit code. It also produces diagrams of lines in a Jupyter notebook, such as the one shown in figure 7.

A comparison between the native C++ GooFit model description file and the AmpGen file for a similar decay, the $D^0 \to K^\mp\pi^\pm\pi^\pm\pi^\mp$ model [10], takes 1,314 lines to describe in GooFit but only 222 lines as AmpGen grammar. The AmpGen grammar maps directly onto the physics description of the model, with one line per decay or parameter, while the GooFit code is tied up in managing memory and building C++ structures. Once converted to GooFit code by DecayLanguage, the resulting code is very similar to the handwritten code, but with helpful comments containing descriptions of the decay, and provides identical performance.

```
1  EventType D0 K0S0 pi+ pi-
2
3  D0{K0S0,rho(770)0{pi+,pi-}} 2 1.000 0.00 2 0.0 0.0
4  D0{K*(892)bar-{K0S0,pi-},pi+} 0 1.740 0.01 0 139.0 0.3
5
6  K(0)*(1430)bar-_mass 2 1.463 0.002
7  K(0)*(1430)bar-_width 2 0.233 0.005
```

Figure 8. Example of the AmpGen syntax. This is for a $D^0 \to K^0_S\pi^+\pi^-$ decay. Line 1 is the event type declaration. Lines 3-4 show example decay channels. Lines 6-8 show parameter definitions. Numbers are in groups of three: the first digit is 0 for free, 2 for fixed. The second number is the value, the third is the uncertainty.

5 Summary

Three key improvements make GooFit 2.1 and 2.2 easier to use and more powerful. First, the new Python bindings allow users to access GooFit from the popular Python interpreter. This makes it easier to compose models quickly, manipulate them from scripts, and interact with the scientific Python software ecosystem. GooFit's internal indexing system was replaced, making it simpler to compose new PDFs and eliminating some common classes of mistakes for new developers. It provided a performance benefit for GPUs, and gives the GooFit core developers more flexibility for further improvements in the future. Finally, the new Decay-Language provides interoperability with the AmpGen package, with more packages possible in the future. This is in early stages, but provides great potential for future developments.

References

[1] W. Verkerke, D.P. Kirkby, eConf **C0303241**, MOLT007 (2003), `physics/0306116`
[2] R.E. Andreassen, W.M. de Silva, B.T. Meadows, M.D. Sokoloff, K.A. Tomko, IEEE Access **2**, 160 (2014), `10.1109/ACCESS.2014.2306895`
[3] R. Andreassen et al., *"GooFit/GooFit" [software]* (October 23, 2018), Release 2.2.1, Zenodo, `https://doi.org/10.5281/zenodo.1469544`
[4] J.P. Lees et al. (BaBar), Phys. Rev. **D93**, 112014 (2016), `1604.00857`
[5] J.P. Lees et al. (BaBar), Phys. Rev. Lett. **111**, 111801 (2013), `1304.5657`
[6] T. Evans et al., *"AmpGen" [software]* (December 1, 2018), `https://gitlab.com/tevans1260/AmpGen`
[7] H. Schreiner, C. Hasse, B. Hittle, H. Pandey, M. Sokoloff, K. Tomko, J. Phys. Conf. Ser. **1085**, 042014 (2018), `1710.08826`
[8] A.A. Alves Jr, M.D. Sokoloff (2017), `1702.05712`
[9] *"The Scikit-HEP Project"*, `http://scikit-hep.org/`
[10] R. Aaij et al. (LHCb), Eur. Phys. J. **C78**, 443 (2018), `1712.08609`