RooFit: model serialization to JSON and other news

Jonas Rembser (CERN, EP-SFT)

7 October 2022, Belle II Data Preservation Workshop

Introduction

- **RooFit**: C++ library for statistical data analysis in ROOT
  - Model specification and fitting to data (baseline RooFit)
  - Implements common statistical tests (RooStats)
  - Includes tools to specify complex binned models (HistFactory)

- Recent development focused on:
  - **Performance** boost (preparing for larger datasets of **HL-LHC**)
  - More **user friendly** interfaces and high-level tools

- **Topics** of today:
  - Quick overview on **recent RooFit highlights**
  - RooFit model **serialization to JSON/YAML**
  - RooFit and **automatic differentiation (AD)**
  - Statistical model **preservation**

(cc) BY

RooFit development areas

In which areas does RooFit evolve (besides bugfixes)?

Vectorization

Gradient parallelization

Higher-level interfaces

GPU Implementation

Fit precision and correctness

Pythonizations

Automatic differentiation

Testing and benchmarking

Interoperability

Targeted optimizations for expensive workflows

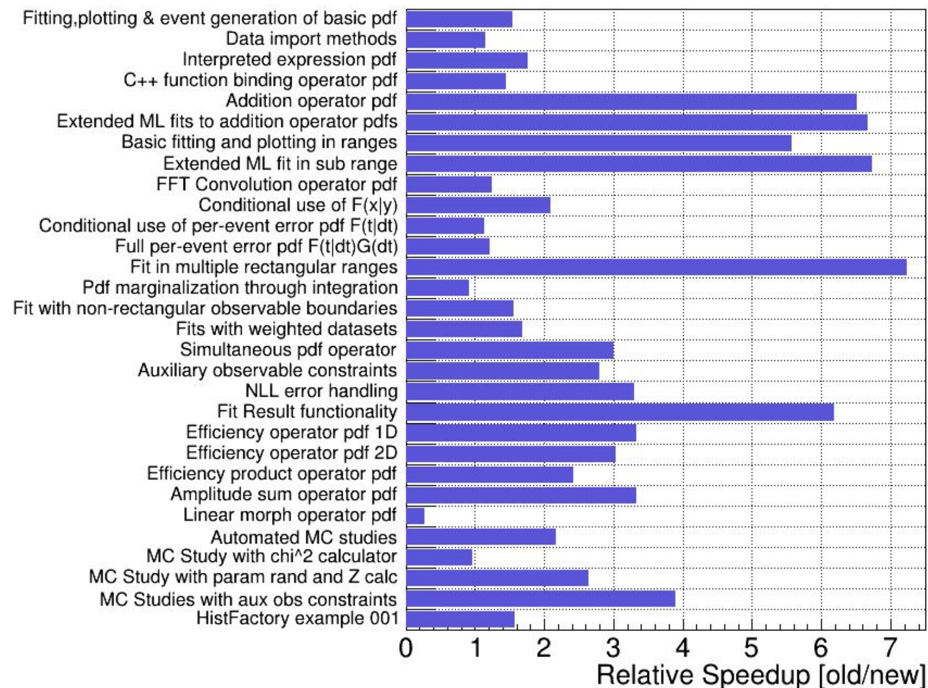
Performance optimization ⟷ User interface and experience

- Not all areas are covered with the same level of activity
- Some areas started to be covered only recently (*automatic differentiation, interoperability*)

(cc) BY

New RooFit computation backend

- New computation backend for likelihood fits ("***BatchMode***") that makes use of vectorization and other optimizations
- Easy to enable in your fits:
  - `pdf.fitTo(data, BatchMode(true))`
- **Significant speedup** of likelihood minimization in most RooFit tutorials (up to 7x, see plot on the right)
  - By next release (*6.28*) hopefully for all of them
- Please **try it out** and open GitHub issues if your fit results are wrong or the fit became slower



(cc) BY

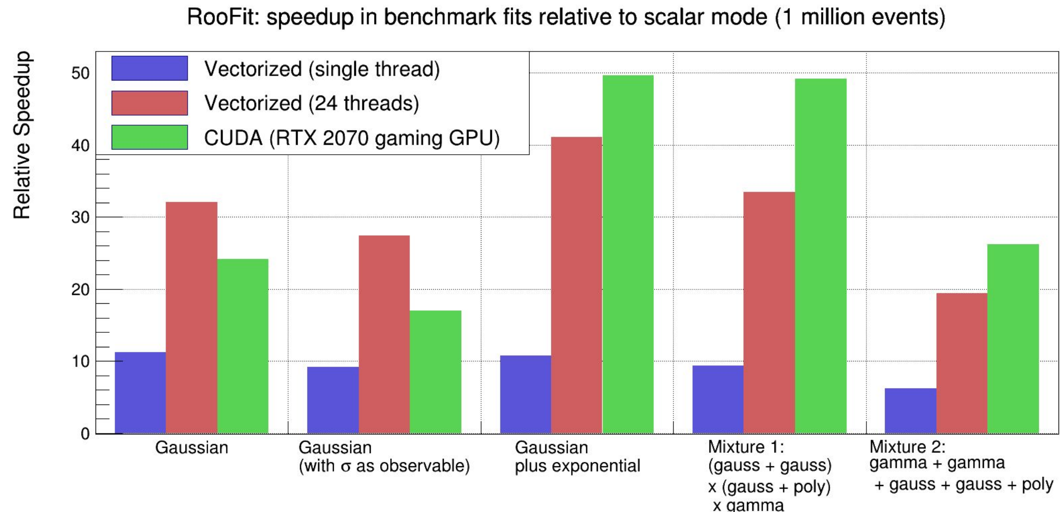
Status of RooFit's BatchMode

- Architecture-specific accelerator libraries for key functions
  - Optimal one loaded at runtime, given current architecture
  - Now also includes **GPU version**! Try it out with `pdf.fitTo(model, BatchMode("cuda"))`
- Multithreading via `ROOT::EnableImplicitMT()`

- **Huge speedup** for unbinned fits with many events

- For large computation graphs with few events, BatchMode still has larger overhead than recursive evaluation

- Maybe this is exactly the kind model class Belle II uses?



(cc) BY

RooFit pythonizations

- PyROOT bindings **more pythonic** in *6.26*
- Now you can for example:
  - use **Python keyword arguments** instead of RooFit command arguments
  - pass around **Python sets or lists** instead of `RooArgSet` or `RooArgList`
  - pass **Python dictionaries** to functions that take `std::map<>`
  - implicitly convert floats to `RooConstVar` in `RooArgList/Set` constructors
- All pythonizations are [documented]
- Some Pythonizations to help with C++/Python lifetime issue
  - Still there are memory leaks when returning owning pointers
- *See also this [ROOT meeting presentation]*

Example code from the [rf316_llratioplot.py] tutorial showcasing the pythonizations:

```python
# Create background pdf poly(x)*poly(y)*poly(z)
px = ROOT.RooPolynomial("px", "px", x, [-0.1, 0.004])
py = ROOT.RooPolynomial("py", "py", y, [0.1, -0.004])
pz = ROOT.RooPolynomial("pz", "pz", z)
bkg = ROOT.RooProdPdf("bkg", "bkg", [px, py, pz])

# Create composite pdf sig+bkg
fsig = ROOT.RooRealVar("fsig", "signal fraction",
                       0.1, 0., 1.)
model = ROOT.RooAddPdf("model", "model",
                       [sig, bkg], [fsig])

data = model.generate((x, y, z), 20000)

# Make plain projection of data and pdf on x observable
frame = x.frame(Title="Projection on X", Bins=40)
data.plotOn(frame)
```


(cc) BY

RooFit with NumPy, Pandas, and RDF

- ROOT *v6.26* **new converters** between NumPy arrays/Pandas dataframes and **RooDataSet/RooDataHist**
  - No translation from RooDataHist to dataframe because histograms are in general multi-dimensional
  - Tutorial in Python

- New `RooRealVar.bins()` function to get RooFit **bin boundaries** as NumPy array

- Creating **RooFit datasets** from **RDataFrame**
  - Works for both `RooDataSet` and `RooDataHist`
  - Weighted filling still needs to be implemented
  - Tutorial in C++ and Python

Example of exporting RooDataSet to Pandas:

```
from ROOT import RooRealVar, RooCategory, RooGaussian

x = RooRealVar("x", "x", 0, 10)
cat = RooCategory("cat", "cat",
                  {"minus": -1, "plus": +1})

mean = RooRealVar("mean", "mean",
                  5, 0, 10)
sigma = RooRealVar("sigma", "sigma",
                   2, 0.1, 10)

gauss = RooGaussian("gauss", "gauss",
                    x, mean, sigma)


data = gauss.generate((x, cat), 100)

df = data.to_pandas()
```

	x	cat
0	6.997865	-1
1	7.211196	-1
2	3.198248	1
3	5.015824	1
4	7.782388	1
...	...	...
95	6.878027	-1
96	0.475900	1
97	4.451101	-1
98	3.481015	-1
99	4.010105	-1

100 rows × 2 columns

CC BY

Parallelized gradient calculation

- For many parameters, most fitting time is spent for the **numeric gradient computation** (re-evaluation after varying each parameter one at a time)
- Distributing the **gradient calculation over multiple processes** is a very general way to speed up fitting (see ACAT 2019 presentation)
- Gradient parallelization is part of ROOT *6.26*
- It comes together with **new likelihood classes** with improved performance for parallelization over entries

More info in this talk from ICHEP 2022

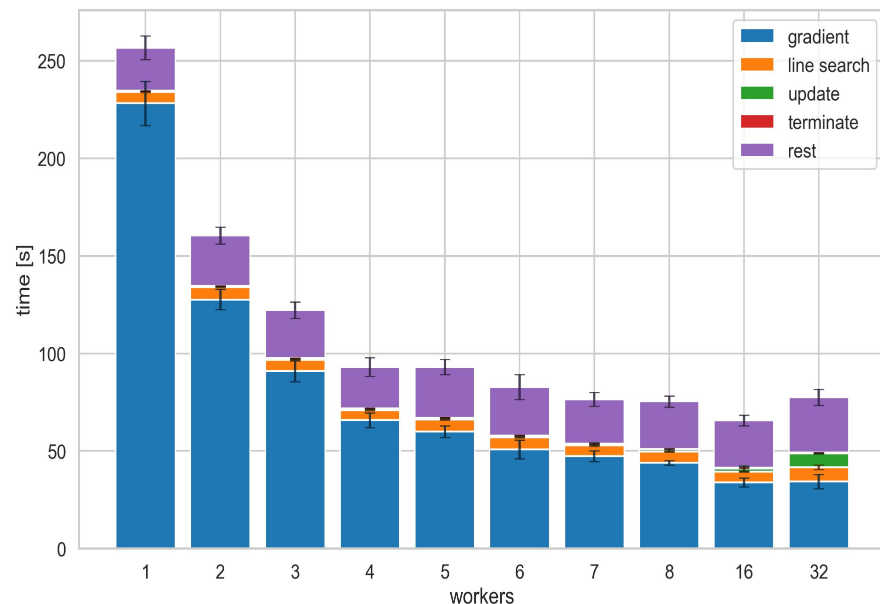


Figure from the ACAT 2019 presentation showcasing the scaling of the gradient parallelization for an ATLAS Higgs combination fit

(cc) BY

Motivation for RooWorkspace ⇄ JSON/YAML

- The push towards publishing likelihoods is getting stronger

- pyhf has been extremely successful in attracting users
- important reason (among others): ability to define models in a declarative language
  - pyhf JSON is readable, editable, and feature-complete!
- however, limited to HistFactory use-case
  - no “complicated” models, only stacks of homogeneously binned histograms in non-overlapping regions

A **round-trip-capable**, **human-readable declarative** format for statistical models was missing.

(cc) BY

RooWorkspace ⇄ JSON/YAML

- Model-building tools require descriptive languages to define the model
- **JSON** or **YAML** is a well-readable standard industry format
- The new RooFit *(6.26)* includes a **new** [`RooJSONFactoryWSTool`] to **import/export** `RooWorkspaces` to JSON or YAML
- This can ease interoperability also with other statistics frameworks such as **pyhf** an **zfit**
- Serialization standard referred to as **HS3** *(HEP statistics serialization standard)*

Example on the right: *JSON for Gaussian signal with **`RooArgusBG`** background*


```
"pdfs": {
        "signal": {
            "type": "Gaussian",
            "x": "mes", "mean": "sigmean", "sigma": "sigwidth"
        },
        "background": {
            "type": "ARGUS",
            "mass": "mes", "resonance": 5.291,
            "slope": "argpar", "power": 0.5
        },
        "model": {
            "type": "pdfsum",
            "summands": [
                "signal",
                "background"
            ],
            "coefficients": [
                "nsig",
                "nbkg"
            ],
            "tags": [
                "toplevel"
            ]
        }
},
"variables": {
        "mes": { "value": 5.25, "min": 5.2, "max": 5.3 },
        "sigmean": { "value": -5.28, "min": 5.2, "max": 5.3 },
        "nsig": { "value": 200, "min": 0, "max": 10000 },
        "argpar": { "value": -20, "min": -100, "max": -1 },
        "nbkg": { "value": 800, "min": 0, "max": 10000 }
}
```


More info in this talk on the ROOT users workshop 2022

(cc) BY

The Implementation: RooJSONFactoryWSTool

- **Extensible system** to manage import and export of functions, pdfs and variables
- Two-layer approach:
  - Possibility to **plug in C++ code** for import/export of specific RooFit objects
  - For simpler classes, **define mapping of JSON keys** to RooFit constructor arguments (*import expressions*) or the `RooAbsProxies` (*export expressions*)
- More on how to do this in the [doxygen page]

Import Expressions

```
"Gaussian": {
    "class": "RooGaussian",
    "arguments": [
        "x",
        "mean",
        "sigma"
    ]
},
"Poisson": {
    "class": "RooPoisson",
    "arguments": [
        "x",
        "mean"
    ]
},
```

Export Expressions

```
'RooGaussian": {
    "type": "Gaussian",
    "proxies": {
        "x": "x",
        "mean": "mean",
        "sigma": "sigma"
    }
},
'RooPoisson": {
    "type": "Poisson",
    "proxies": {
        "x": "x",
        "mean": "mean"
    }
},
```

RooWorkspace to JSON:

```
tool = ROOT.RooJSONFactoryWSTool(myworkspace)
tool.exportJSON("myworkspace.json")
```

JSON to RooWorkspace:

```
ws = ROOT.RooWorkspace("somename")
tool = ROOT.RooJSONFactoryWSTool(ws)
tool.importJSON("myworkspace.json")
```


(cc) BY

Standardizing the top-level JSON content

Going beyond the RooWorkspace, a JSON workspace standard should fulfill these criteria:

1. An arbitrary number of **likelihoods and models** should be storable
2. **Combinations** of different likelihoods (binned and unbinned) should be very easy!
3. The JSON should be **easily manipulable** by different tools and also by hand

What can be written to the JSON should probably include:

- PDFs
- Functions
- Likelihoods/loss functions
- Data
- Parameters/parameter lists
- Parameter "snapshots"
- Metainfo
- ...?

Ongoing **discussions** with **ATLAS users, pyhf** and **zfit developers** to explore possibility for a common standard, dubbed the **HEP statistics serialization standard**

This means the **JSON format** from the RooJSONFactoryWSTool is **not stable yet**

(cc) BY

Automatic differentiation (AD) in RooFit

- **Gradient** of RooFit model essential for minimization
  - RooFit uses numeric derivatives, varying one parameter at the time
  - Using analytic gradients is much more efficient for many parameters
  - We can use **automatic differentiation** techniques to get these gradients

- No code merged yet, but we investigate the following code towards AD:
  - **C++ code generation** form RooFit model to one C++ function **and automatically differentiate** with **clad** (or other source-code transformation fools like Enzyme AD)

(cc) BY

AD for binned likelihoods from HistFactory

Many binned likelihoods follow a similar pattern:

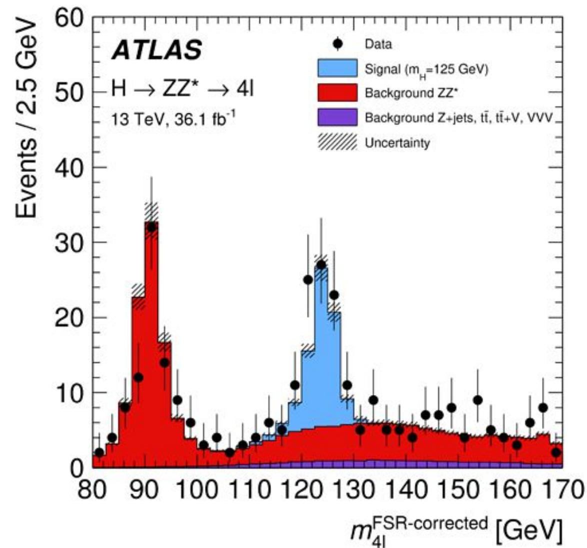
$$L(\vec{n}, \vec{a} \mid \vec{\eta}, \vec{\chi}) = \underbrace{\prod_{c \in \text{channels}} \prod_{b \in \text{bins}} \text{Pois}(n_{cb} \mid \nu_{cb}(\vec{\eta}, \vec{\chi}))}_{\text{product of Poisson terms}} \underbrace{\prod_{\chi \in \vec{\chi}} c_\chi(a_\chi \mid \chi)}_{\text{constraints}}$$

$\vec{n}$: data, $\vec{a}$: auxiliary data

$\vec{\eta}$: unconstrained parameters

$\vec{\chi}$: constrained parameters

Example binned likelihood with one channel: Higgs to 4 leptons



HistFactory is a higher-level tool to build such likelihoods in RooFit.

Good model class for showing AD in RooFit:

- many parameters
- rich computation graph
- few normalization integrals

(cc) BY

Preliminary Results

Explicit Computation Graphs: An Example HistFactory Model

An example histogram fitting model with 2 bins and 2 channels, with 3 samples per channel. Based on the hf_001 example.

(cc) BY

Preliminary Results

```
double nll(double *in)
{
  double nomGammaB1 = 400;
  double nomGammaB2 = 100;
  double nominalLumi = 1;
  double constraint[3]{RooPoisson::poisson(nomGammaB1, (nomGammaB1 * in[0])),
                       RooPoisson::poisson(nomGammaB2, (nomGammaB2 * in[1])),
                       RooGaussian::gauss(in[2], nominalLumi, 0.100000)};
  double cnstSum = 0;
  double x[2]{1.25, 1.75};
  double sig[2]{20, 10};
  double binBoundaries1[3]{1, 1.5, 2};
  double bgk1[2]{100, 0};
  double binBoundaries2[3]{1, 1.5, 2};
  double histVals[2]{in[0], in[1]};
  double bgk2[2]{0, 100};
  double binBoundaries3[3]{1, 1.5, 2};
  double weights[2]{122.000000, 112.000000};
  for (int i = 0; i < 3; i++) {
    cnstSum -= std::log(constraint[i]);
  }
// cont…
```

Constraints defined as calls to their respective ‘evaluate’s.

Constraint sum.

```
// cont..
  double mu = 0;
  double temp;
  double nllSum = 0;
  unsigned int b1, b2, b3;
  for (int iB = 0; iB < 2; iB++) {
    b1 = RooHistFunc::getBin(binBoundaries1, x[iB]);
    b2 = RooHistFunc::getBin(binBoundaries2, x[iB]);
    b3 = RooHistFunc::getBin(binBoundaries3, x[iB]);
    mu = 0;
    mu += sig[b1] * (in[3] * in[2]);
    mu += (bgk1[b2] * histVals[iB]) * (in[2] * 1.000000);
    mu += (bgk2[b3] * histVals[iB]) * (in[2] * 1.000000);
    temp = std::log((mu));
    nllSum -= -(mu) + weights[iB] * temp;
  }
  return cnstSum + nllSum;
}
```

Translated RooProducts.

NLL

(cc) BY

Automatic Differentiation in RooFit

Performance comparison AD vs numerical differentiation on hf_001 inspired example

Tested on ROOT v6.26.

(cc) BY

RooFit and analysis preservation

- RooFit has a few things going for it in the analysis preservation department:
  - The `RooWorkspace` is widely used in the community
  - **Few** software **dependencies** (only ROOT, which has only few dependencies itself)
  - We make **backwards** compatibility a **priority** (...at least since a few years)
  - No hurdles for fixing preservation issues upstream (as it's a HEP community project)

- But there are also problems:
  - Model **specification tightly connected** with **implementation**
  - The correct schema evolution of all RooFit objects is a big burden

The **JSON/YAML serializatio**n and **C++ code generation** approach can be useful to *overcome these shortcomings*!

Maybe we will have something

(cc) BY

Summary

- RooFit is **evolving** steadily
  - Support and development from **ROOT team** at CERN and **external contributors**
- Highlights of the recent version *6.26* are the **BatchMode** and the **Pythonizations**
  - as well as the JSON serialization explained in more detail
- Status of the **JSON** to Workspace tool:
  - Support for most PDFs and functions, easily extensible by users
  - Standardization of JSON structure still in progress (**HS3 project**)
- Future developments will focus on **automatic differentiation**
  - Approach of differentiable **C++ code generation** could benefit analysis preservation
  - Still at early R & D stage, don't expect much of this in upcoming ROOT 6.28 yet

(cc) BY

Links

Recent RooFit **presentations**:

- Talk at ACAT 2021
- RooFit talk at the ROOT users workshop
- RooFit talk at ICHEP 2022
- Automatic differentiation in ROOT (September 2022)

To get more info on the **JSON serialization**:

- The HEP statistics serialization standard GitLab repository
  - You can download the PDF as an artefact of the CI pipeline
- The landing page for the JSON tool in the RooFit documentation
- The RooFit tutorial that explains how to import a model from JSON
- For developers: the RooFit HS3 README file explaining how to extended model support

(cc) BY