What the new RooFit can do for your Analysis

Stephan Hageboeck (CERN, EP-SFT) for the ROOT team

Introduction

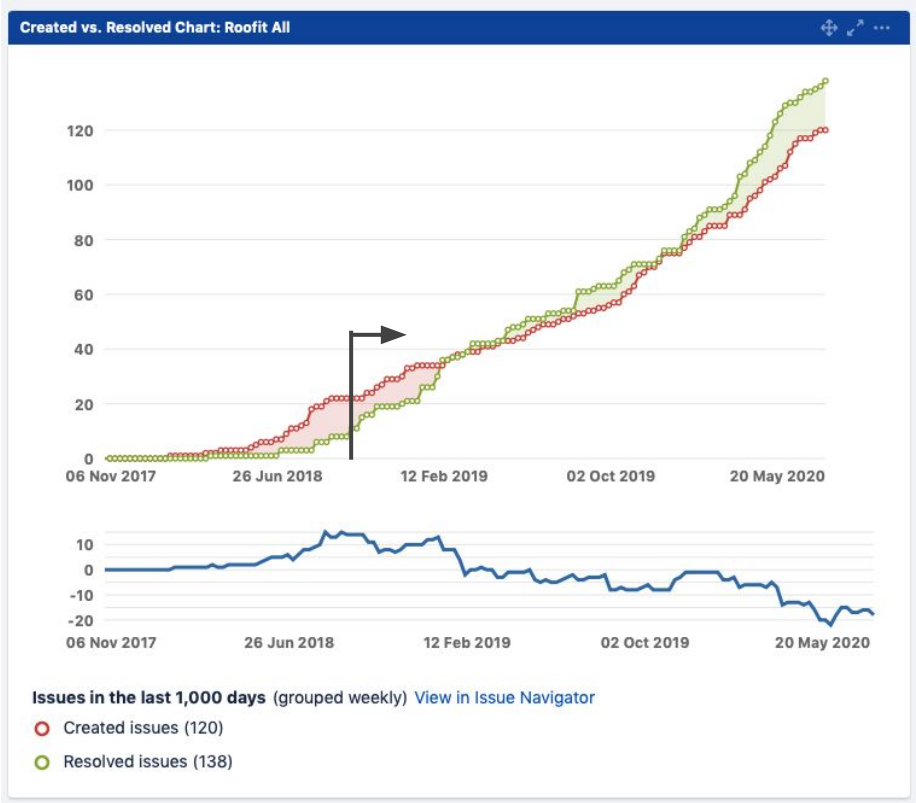
- RooFit: Development started before ~2005 until ~2011, not changed much in recent years
- End of 2018: Joined CERN's SFT group to resume development

Challenges:

- Old interfaces + old bugs
- Communication with users: What do they need?
- Speed:
  - More events need to be processed (Run 3 >= 5x more data?)
  - Higher statistics → allow for more complex models
  - Goal: significant speed up (> 5x in one thread)

1. State of Affairs

- 10/2018: 200 bugs, 15 improvements being tracked
  - Crash when '.q'
  - Not able to open ROOT 5 workspaces
- Today: 74 bugs + 30 improvements being tracked sft.its.cern.ch/jira/
  - Priority? Still relevant?
  - You can vote RooFit's JIRA



2. Collections, Interfaces, Usability

2. Collections, Interfaces & Usability

- RooArgSet / RooArgList were internally based on a linked list
  - Clumsy iterators, slow access
- Now based on std::vector
- **No need to update old code**
- But:
  - Shorter, more readable, STL-like code
  - **Iterates 25% faster**
  - Typical RooFit workflows between 3 and 21% faster

ROOT 6.16

```
TIterator* it = pdf.getParameters(obs)->createIterator();
RooAbsArg* p;
while ((p=(RooAbsArg*)it->Next())) {
  p->Print();
}
delete it;
```

ROOT 6.18

C++

```
for (const auto p : *pdf.getParameters(obs))
  p->Print();
```

Python

```
for p in pdf.getParameters(obs):
  p.Print()
```

2. Interfaces & Usability: Python

- New PyROOT has landed
- STL-like C++ interface:
  Automatic generation of Python iterators

ROOT 6.16

```
it = pdf.getParameters(obs).createIterator();
p = it.Next();
while p is not None:
  print(p.GetName, p.getVal())
  p = it.Next();
```

ROOT 6.22

```
for p in pdf.getParameters(obs):
  print(p.GetName(), p.getVal());
```

Pythonisations

- (Short) Python code that steers C++ backend
- Summer student project approved
  But: COVID-19
- Requests? → RooFit's JIRA

Pythonisations:

- `getattr(workspace, 'import')(...)` → `workspace.Import(...)`
- `collection.size()` → `len(collection)`
- `pdf.fitTo(data, ROOT.RooFit.Range("sideband"))` → `pdf.fitTo(data, range="sideband")`

COVID

2. Collections, Interfaces & Usability

- Modernised category classes
- Replace class of {char[256]; int} by int
- Will enable faster loading and computations (WIP)
- Saves memory in datasets:

```
root [0] sizeof(RooCatType)
(unsigned long) 288
root [1] sizeof(int)
(unsigned long) 4
```

ROOT 6.20

```
RooCategory cat("cat", "Lep. mult.");
cat.defineType("0Lep", 0);
cat.defineType("1Lep", 1);
cat.defineType("2Lep", 2);
cat.defineType("3Lep", 3);

TIterator* typeIt = cat.typeIterator();
RooCatType* catType;
while ( (catType =
  dynamic_cast<RooCatType*>(typeIt->Next()) )
  != nullptr) {
  std::cout << catType.GetName() << ", "
    << catType.getVal() << std::endl;
}
delete typeIt;
```

ROOT 6.22

```
RooCategory cat("cat", "Lep. mult.");
cat.defineTypes(
  {"0Lep", "1Lep", "2Lep", "3Lep"},
  { 0,      1,      2,      3});

for (const auto& name_idx : cat) {
  cout << name_idx.first << ", "
    << name_idx.second << endl;
}
```

- **No need to update old code**

2. Interfaces & Usability

Expert Level

- For people who write their own PDFs:
  Proxies are now type safe

```
// Header
RooTemplateProxy<RooAbsPdf> pdf;

// .cxx
pdf->fitTo(...)
```

ROOT 6.22

[ROOT 6.22 release notes]

```
// Header
RooRealProxy realProxy;

// .cxx
RooAbsArg* absArg = realProxy.absArg();
RooAbsPdf* pdf = dynamic_cast<RooAbsPdf*>(absArg);
// Should work, but maybe someone stored wrong
// object. Add some checking code.
if (pdf != nullptr) {
...
}
pdf->fitTo(...);
```

ROOT 6.20

2. Interfaces & Usability

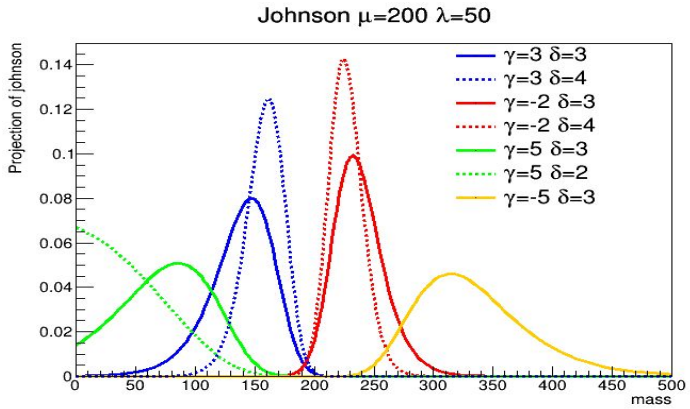
- LHCb: Johnson PDF missing for typical workflows
- Citeable reference
- Faster, checks parameters
- Accurate analytic integrals
- Also new: RooHypatia2

```
//signal model
//Johnson
RooAbsPdf *johnson = new RooGenericPdf( "johnson", "(mass>M_thr)*delta_j"
    "/(sigma_j*TMath::Sqrt(TMath::Pi()))"
    "*TMath::Exp(-0.5*(gamma_j+delta_j*TMath::ASinH((mass-mean_j)/sigma_j))*"
    "(gamma_j+delta_j*TMath::ASinH((mass-mean_j)/sigma_j)))"
    "/TMath::Sqrt(1+(mass-mean_j)*(mass-mean_j)/(sigma_j*sigma_j))",
    RooArgSet( mass, mean_j, sigma_j, gamma_j, delta_j, M_thr ) );
```

ROOT 6.18

```
RooRealVar delta("delta", "delta", 2., -1., 10.);
RooJohnson johnson("johnson", "johnson", mass, mu, lambda, gamma, delta);
```

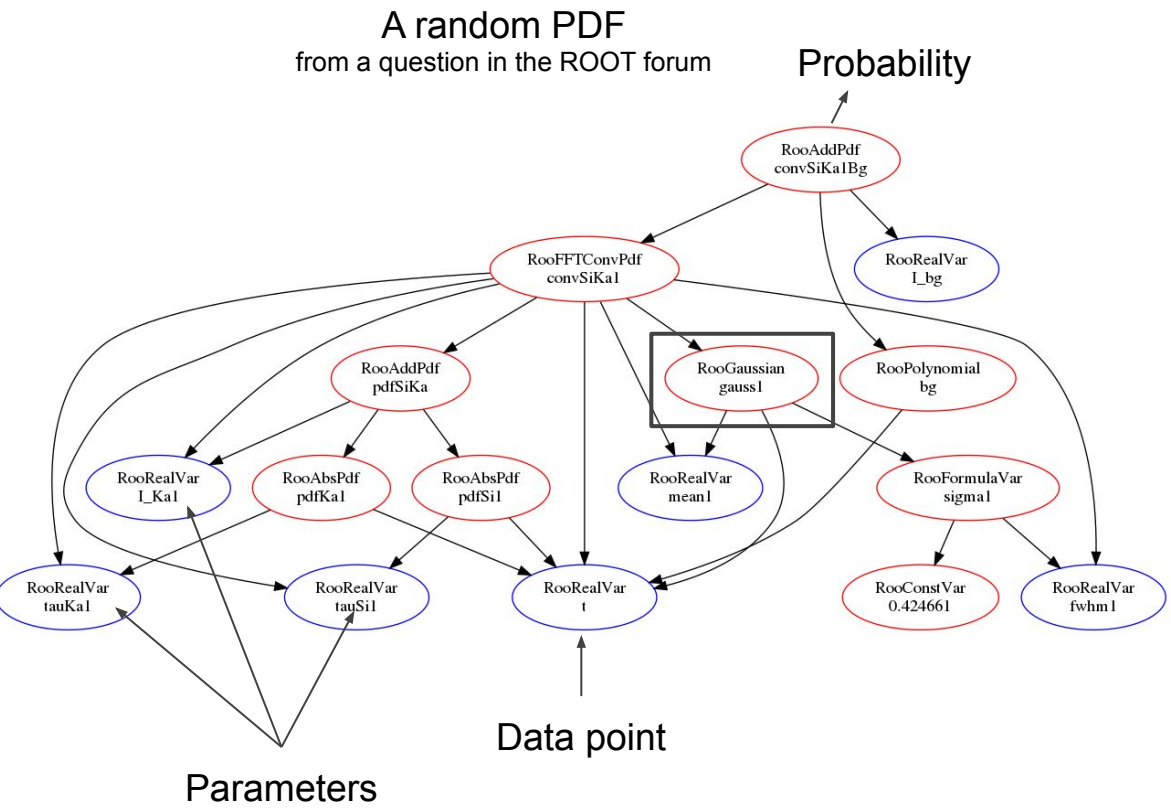
NB: Factor 2 missing here.
No harm done, though, since RooFit integrates numerically to normalise.



RooJohnson docs

3. Speed
Batched Computations & Vectorisation

3. Speed: How RooFit Computes



RooFit has a problem:

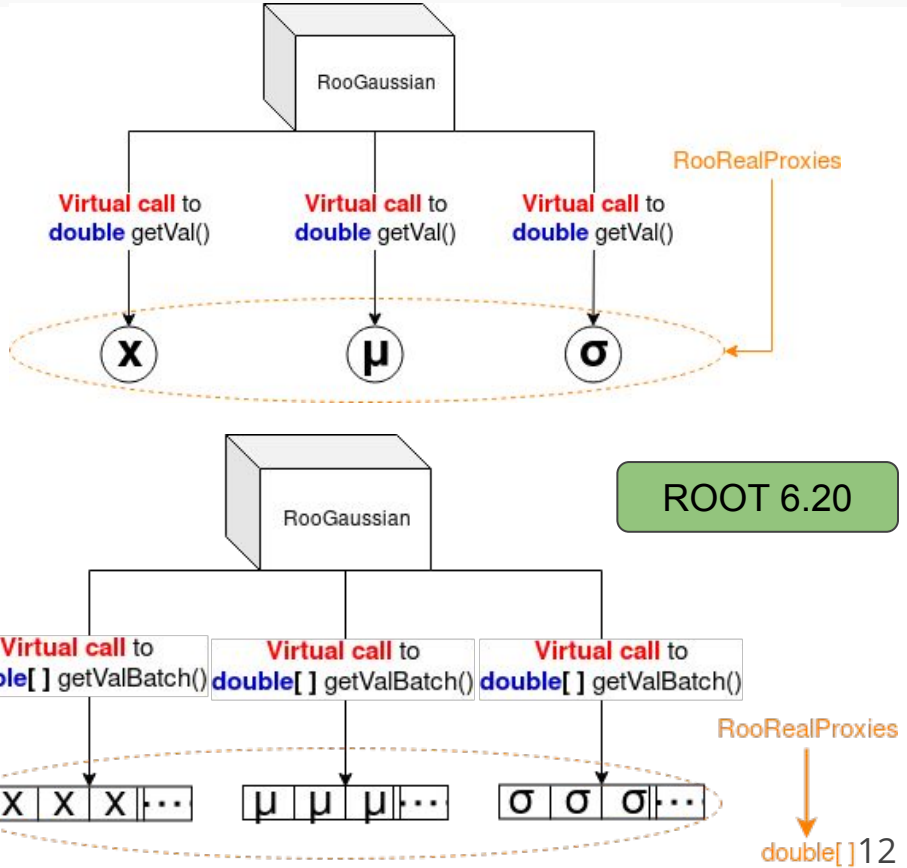
1. Load a single data point into variables
2. Walk whole expression tree (minus cached branches)
3. Obtain one probability. Repeat at 1. with next data point.
   - Simple profiling: 50% L1/L2 cache misses
   - No chance to vectorise computations

3. Speed: Scalar vs Batch Evaluation

```
double RooGaussian::evaluate() const
{
  const double arg = x - mean;
  const double sig = sigma;
  return exp(-0.5*arg*arg/(sig*sig));
}
```

- Completely reorganised internal data storage in ROOT 6.20

```
for (int i = 0; i < n; ++i) {
  const double arg = x[i] - mean[i];
  const double halfBySigmaSq = -0.5 /
                (sigma[i] * sigma[i]);
  output[i] = vdt::fast_exp
            (arg*arg * halfBySigmaSq);
}
```

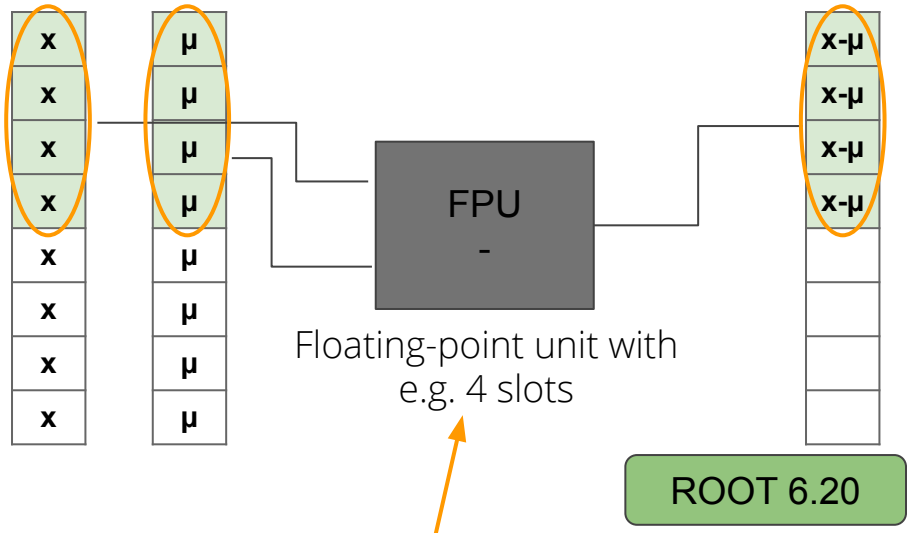


3. Vectorisation: Exploit SIMD

- When running loops on contiguous data, can exploit vectorised instructions

  `result = x[0..3] - μ[0..3]`
- Caveat:
  - Need to recompile for specific CPU architectures
- RooFit can use fast, inlinable, vectorisable math functions

  VDT

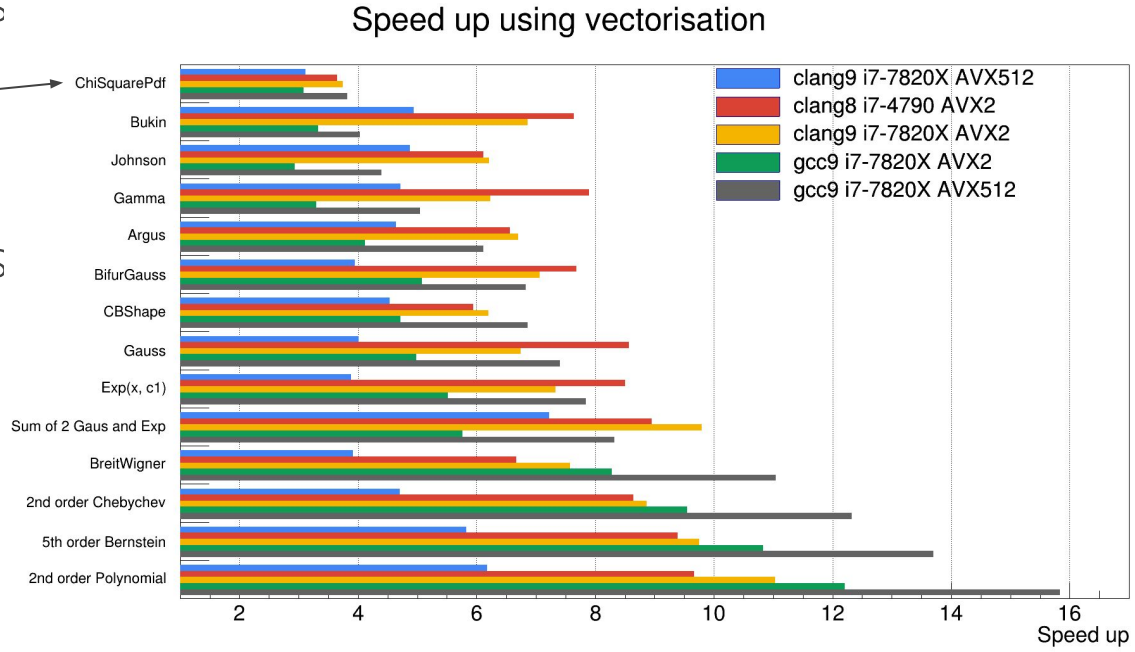


Modern CPUs support:

- 2 doubles (SSE),
- 4 (AVX, 2011 / AVX2, 2013)
- 8 (AVX512, 2015)

3. Vectorisation: Exploit SIMD

- Better cache locality & less function calls:
  Speed up of 3x
- Depending on CPU and compiler, speed up ranges from 4x to 16x
- Caveats:
  - Only for built-in PDFs
  - ROOT often not compiled with AVX or SSE instructions



Demo

4. Outlook

4. Outlook

WIP

Binned fits (e.g. "HistFactory"):

- Not yet supporting fast batch computations
  - Expect significant speed up

GPUs:

- Redesign of data structures: Done
- Vectorisable computation kernels → GPU kernels: ~ Done
- Infrastructure to submit GPU computations and collect results: TODO
- Technical student project starting in September

Vectorisable kernel for Gaussian distribution:

```
for (int i = 0; i < n; ++i) {
  const double arg = x[i] - mean[i];
  const double halfBySigmaSq = -0.5 /
                (sigma[i] * sigma[i]);
  output[i] = rf_fastExp
             (arg*arg *
halfBySigmaSq);
}
```

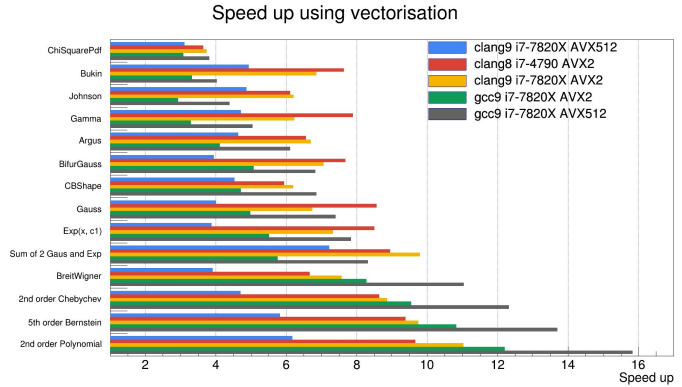
Timeline of What Happen(s/ed) with RooFit

1. Fix annoying/pressing issues — ROOT 6.16
2. Transform RooArgSet/List from LinkedList → std::vector — ROOT 6.18
   - More memory friendly, 25% faster to iterate/allocate/destroy/access
   - Easier to use
3. Provide frequently used PDFs centrally
   - E.g. RooJohnson, RooHypatia
4. Batched function evaluation — ROOT 6.20
   - Old RooFit manages data inefficiently
   - Factor 3 faster when loading data more efficiently
5. Vectorise function evaluation
   - Factor 4 - 16 faster evaluation
6. Nicer interfaces (Python, Categories, RooFit proxies) — ROOT 6.22
7. Fast function evaluation for binned fits — WIP
8. GPUs

https://sft.its.cern.ch/jira/browse/ROOT-9815

Summary

- RooFit is evolving again
- I work to make RooFit:
  - Faster
  - Easier to use
  - More stable
- More ideas in pipeline:
  - Fast batch computations & vectorisation for binned fits
  - Nicer interfaces
  - GPU offloading
  - RNTuple as storage backend to allow for faster bulk reading
  - Likelihood gradient parallelisation (developed at NIKHEF)



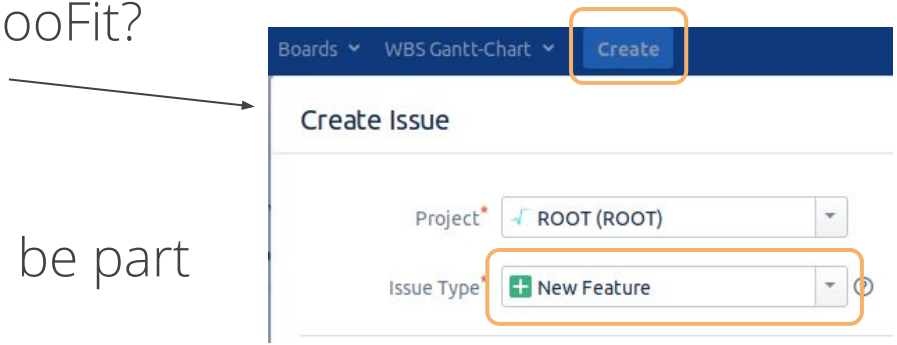
- Check out the tutorials! There are notebooks
- And the release notes (also for 6.20, 6.18 ...)

Useful Links

Useful Links

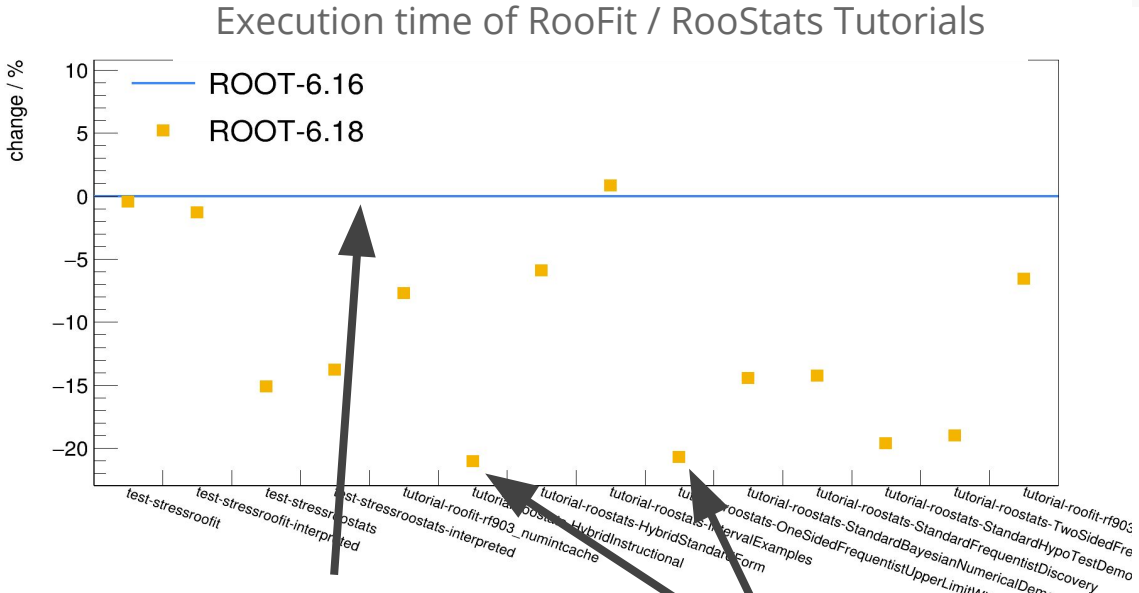
- RooFit tutorials (Recommended! There are notebooks!)
- RooFit documentation
- Release notes
- Test the faster batch mode:
  - `auto result = pdf.fitTo(*data, RooFit::BatchMode(true), RooFit::Save());`
  - NB: No vectorisation by default. Will talk to LHCb computing coordinators.
- Feature request? https://sft.its.cern.ch/jira/
  - A new PDF that should go into RooFit?
  - Ideas for Python interfaces?
- Tricky problems? https://root-forum.cern.ch/c/roofit-and-roostats
- Think that a certain workflow should be part of ROOT's tests?

stephan.hageboeck@cern.ch



Backup

2. Faster, More Intuitive Collections



ROOT-6.16

```
TIterator* paramIter = paramList.createIterator() ;
RooAbsArg* param ;
while((param = (RooAbsArg*)paramIter->Next())) {
  _paramList.add(*param) ;
}

delete paramIter ;
```

ROOT-6.18
20% faster

```
for (const auto param : paramList) {
  _paramList.add(*param) ;
}
```

- STL-like iterators are 25% faster
- Test impact on RooFit / RooStats tutorials with a few critical iterators replaced
- **20% faster**
- More iterators replaced over time

Modernising Code that uses RooFit

When compiling ROOT with `R__SUGGEST_NEW_INTERFACE`, deprecation warnings are triggered:

- Requested during ROOT user's workshop 2018
- Flags functions/classes whose use is discouraged, but won't be fully deprecated
- For developers to modernise code

C Compare Viewer

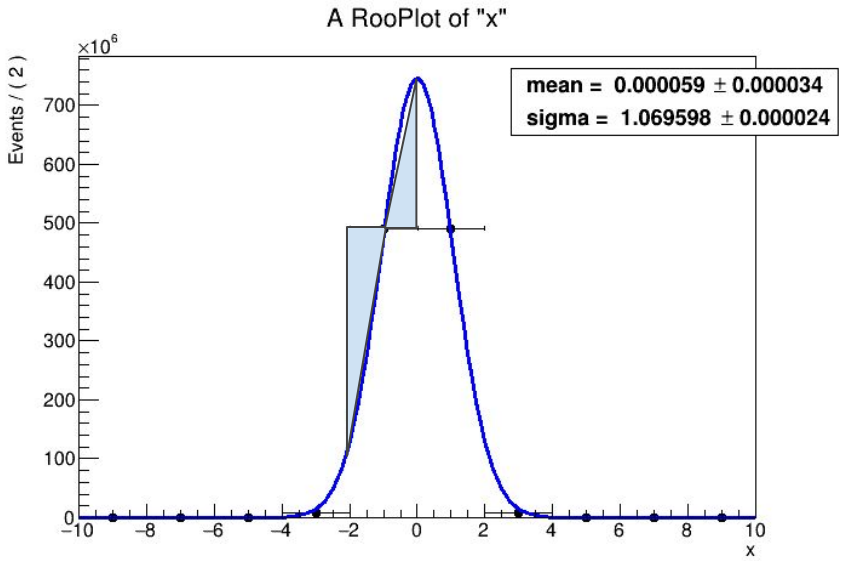
Local: RooAbsCollection.h

```
108  Bool_t overlaps(const RooAbsCollection& otherColl) const ;
109
110  /// \deprecated TIterator-style iteration over contained elements. Use begin() and end() or
111  /// range-based for loop instead.
112  inline TIterator* createIterator(Bool_t dir = kIterForward) const
113  R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
114    // Create and return an iterator over the elements in this collection
115    return new RooLinkedListIter(makeLegacyIterator(dir));
116  }
117
118  /// \deprecated TIterator-style iteration over contained elements. Use begin() and end() or
119  /// range-based for loop instead.
120  RooLinkedListIter iterator(Bool_t dir = kIterForward) const
121  R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
122    return RooLinkedListIter(makeLegacyIterator(dir));
123  }
124
125  /// \deprecated One-time forward iterator. Use begin() and end() or
126  /// range-based for loop instead.
127  RooFIter fwdIterator() const
128  R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
129    return RooFIter(makeLegacyIterator());
130  }
131
```

Better Sampling of PDFs

WIP

- Binned fits not yet supporting fast batch computations
  - Expect significant speed up
- Better sampling of PDFs for binned fits
  - RooFit uses PDF evaluated at bin centre when fitting to binned data
  - Can be inaccurate if 2nd derivative large
  - Batch evaluations will allow to increase accuracy (≈ numeric integral)



Testing the Fast BatchMode

When updating a PDF, run the the following tests:

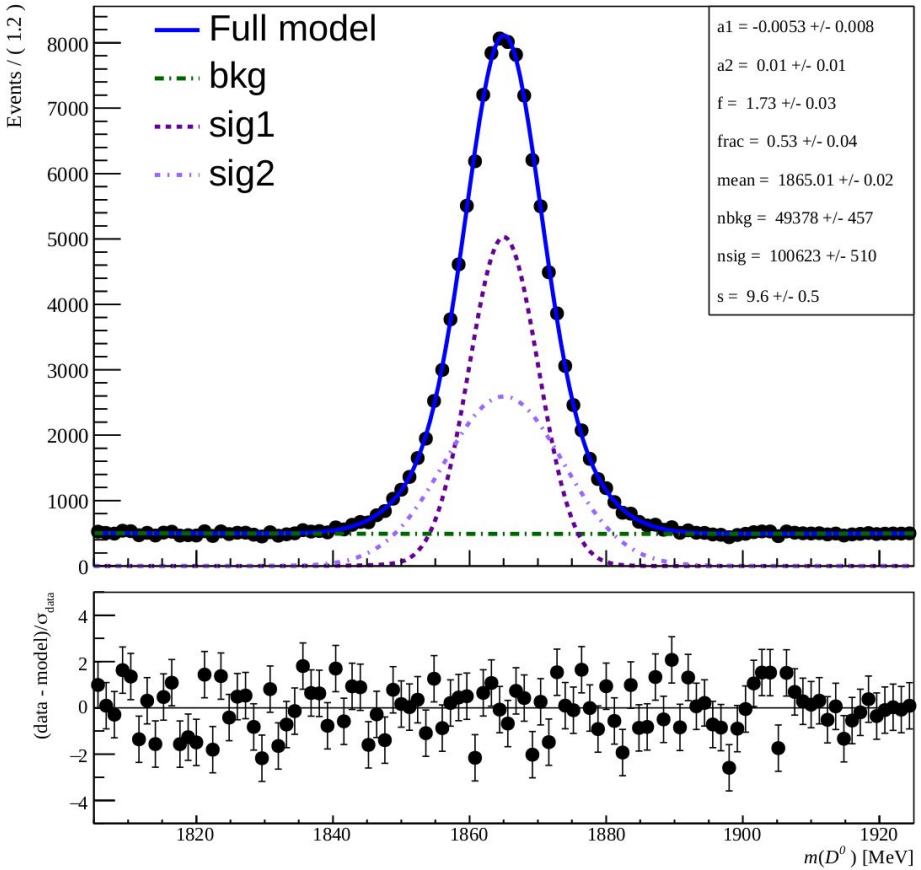
Test	Rel. Accuracy
Bare function values	1.E-13
Probabilities	1.E-13
Log-probabilities	2.E-13
Fit with standard RooFit algorithm	Parameters within 1.5 σ
Fit with batch algorithm	Parameters within 1.5 σ
Compare scalar/batch parameters	Parameters agree to 1.E-5 Correlation coeff to 1.E-4

Additionally:

- Started to collect "official" LHCb examples to test speed and accuracy
- **LHCb1**: D0 mass fit (Python)
- **LHCb2**: Johnson + 6 Gauss (C++)

NB: These run on various platforms like e.g. several Linux distributions, Mac, different compilers …

Real-World Example: "LHCb1"



- Second LHCb real-world fit problem
- Model with 1.5E6 events

$$N_{sig}(f*Gauss + (1-f)*CB) + N_{bkg}\ \Sigma a_i * Cheby_{2,i}$$

- Results:
  - Fit time 409s → 63s
  - Parameters agree to 1.E-5 *relative* precision (except a1: 2.E-4)
    - I.e., agreement better than 0.001*statError