New Developments in Minuit2

L. Moneta, O. Zapata, H. Deminsky

Minuit2

- Minuit
  - Popular minimisation program developed in the 1970s by F. James.
  - It is a Variable metric method (quasi-Newton method) based on the DFP / BFGS update of the inverse Hessian matrix.
  - Work extremely well for fitting (e.g. parameter estimation) and it is has been used extensively in HEP.
  - available in ROOT since the beginning in the TMinuit class.

- Minuit2
  - improved version re-written in C++ classes of Minuit
  - available in ROOT and as a standalone version
    - e.g. used by the iMinuit Python package
  - already in use in the statistical analysis of LHC experiments

Minuit Algorithm

- Start with an initial approximation of inverse Hessian, $H = (\nabla^2 f(x))^{-1}$
  - e.g. use diagonal second derivatives
- Iterate :
  - compute new step direction as $p_k = -Hg$ where $g = \nabla f(x_k)$
  - perform line search for optimal point $x_{k+1} = x_k + \alpha p_k$
    - $s_k = x_{k+1} - x_k$
  - compute the new gradient g at x_{k+1} and $y_k = g_{k+1} - g_k$
  - Update inverse Hessian matrix H_k according to BFGS or DFP update formula

$$\text{BFGS}: H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k} \qquad \text{DFP}: H_{k+1} = H_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k}$$

  - stop iteration when the Expected Distance from the Minimum (EDM) $\rho = g^T H g$ is small
- EDM provides a scale-invariant quantity to tell the convergence of method.
  - This is unique in Minuit!

Advantages of Minuit

- Method work very well, superior to gradient descent methods
  - much less number of iteration to converge
  - approximate Hessian converge to true Hessian at the minimum
  - use regularisation of Hessian by correcting for non-positive defined Hessian
    - add some offset to the diagonal of H to make it positive defined
  - no need to perform matrix inversion at each iteration
  - self-correcting if approximation is not good enough
- Disadvantage:
  - require a fairly good initial Hessian approximation for having a fast convergence
    - second diagonal derivatives are often good enough (define scale)
  - Sensitive to initial parameters, it is a local minimiser, can get stuck in local minimum
  - Sensitive to bad numerical precision in function and gradient calculation
  - Does not scale to problems with huge number of parameters
    - proofed to work to $> \sim 1000$ parameters (e.g Higgs combination fits)
    - will not work for training deep learning models with million of parameters
      - need to use gradient descent in these cases

External Gradient and Hessian

- Minuit computes (by default) numerically the gradient using a 3 points rule and adaptive step size
  - algorithm well-tested and robust
  - Essential having good numerical derivatives when gradient is close to zero (near the minimum) to converge rapidly
- Support for external gradient
  - needed for users exploiting Automatic Differentiation (AD)
- Option to provide external Hessian or only the diagonal of the Hessian (needed for seeding)
  - without providing Hessian, Minuit2 computes it numerically

Other new improvements in Minuit2

- Improved debugging
  - can return all minimisation iteration status
  - can provide a detailed output for each iteration in debug mode
- Possibility to add callbacks which can be called at each iteration
- Thread-safety: Minuit can work in multi-threads if user provided function can
  - support for likelihood or gradient parallelisation
- Addition of new minimization methods:
  - BFGS: use standard BFGS formula instead of the default hybrid mode of using BFGS or DFP formula depending on some conditions

Specialized Algorithms for Fitting

- When minimising Least-square functions:

$$F(\mathbf{x}) = \sum_{k=1}^{K} f_k^2(\mathbf{x}) = \sum_{k=1}^{K} \left(\frac{Y_k - T_k(\mathbf{x})}{\sigma_k} \right)^2 ,$$

$$\begin{aligned} \frac{\partial^2 F}{\partial x_i \partial x_j} &= \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j}\sum_k f_k^2 \\ &= \frac{\partial}{\partial x_i}\sum_k 2 f_k \frac{\partial f_k}{\partial x_j} \\ &= \sum_k 2 \frac{\partial f_k}{\partial x_i}\frac{\partial f_k}{\partial x_j} + \sum_k 2 f_k \frac{\partial^2 f_k}{\partial x_i \partial x_j}. \end{aligned}$$

this can be neglected when residuals f are small

$$\frac{\partial^2 F}{\partial x_i \partial x_j} \approx \sum_k 2 \frac{\partial f_k}{\partial x_i}\frac{\partial f_k}{\partial x_j}.$$

$$H_k \approx \mathbf{J}_k^T \mathbf{J}_k$$

Many algorithms have been developed on this idea (Levenberg-Marquardt method, Fumili,…)

Specialized Fitting Methods

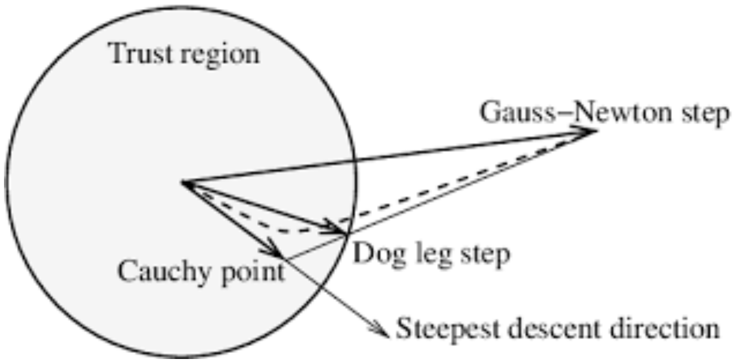
- Hessian can be computed directly from the first derivatives of the model function
  - It is like linear approximation of a non-linear least-square problems
- This approximation is also valid in the case of binned likelihood fits.
  - but not really for standard unbinned maximum likelihood fits
- Advantage:
  - positive defined and easy to calculate (one can use a 2-point rule)
  - faster to converge than standard Minuit/BFGS methods
- Disadvantage:
  - Initial point need to be close enough to the minimum to have the approximation $\mathbf{H}_k \approx \mathbf{J}_k^T \mathbf{J}_k$ valid
  - require a more complex interface, user needs to provide the Jacobian matrix (number of fit points , number of parameters) at each iteration

Fumili Algorithm

- Old algorithm proposed already in 1961 by I. Silin
- Implemented later in the CERN library and made also available to ROOT with TFumili class.
  - It is using the Hessian approximation combined with a trust region method.
    - a multidimensional parallelepiped ("box") is defined around the point and used its intersection with the Newton direction for the next step
    - size of the parallelepiped changes dynamically
      - depending on the function improvements and the expectation from a quadratic approximation.
- Faster than Minuit for least-square/binned likelihood when the starting point is close enough to the solution

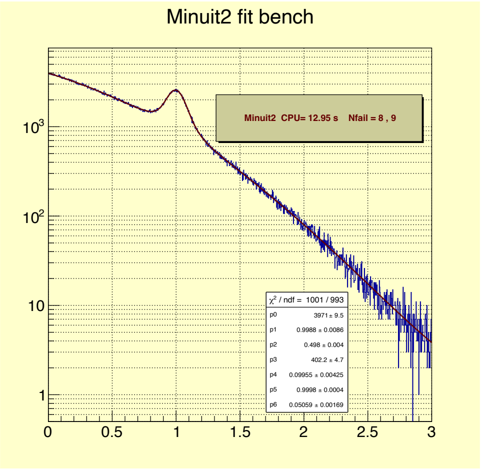
Fumili2

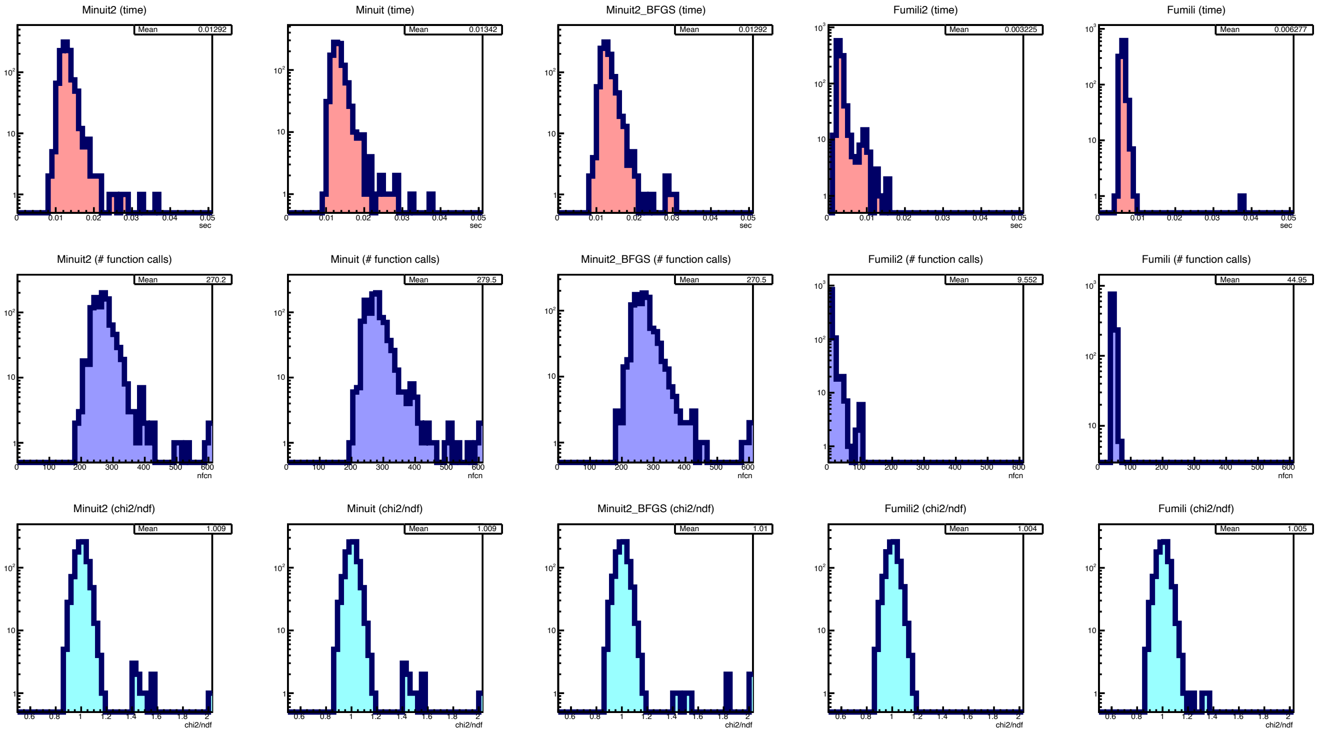
- New implementation of Fumili integrated into Minuit2 library
  - re-using Minuit2 interfaces classes
  - working well for least-square and binned likelihood fits
- Based on trust-region using dogleg step
  - trust region can be scaled using a metric defined by the diagonal of the approximated Hessian



Benchmark Results

- Use a binned likelihood to fit signal peak over some background

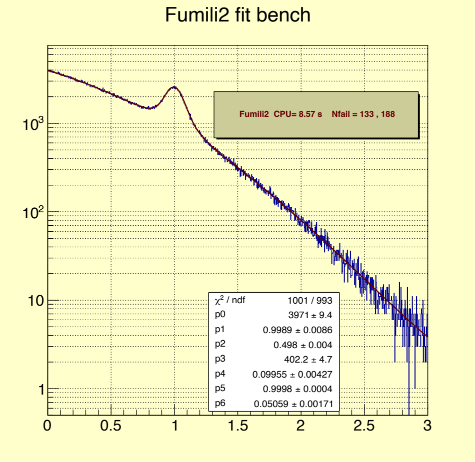


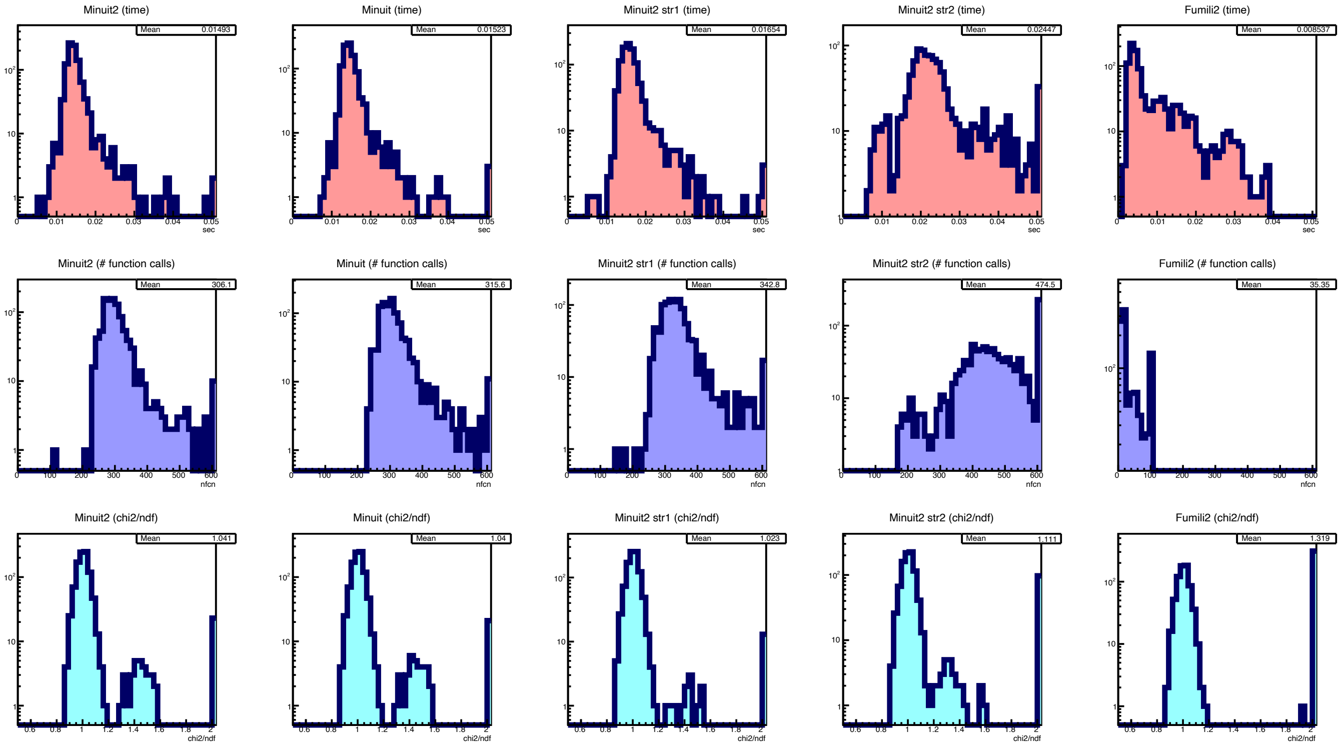


1000 bins - 7 parameters
repeat fit 1000 times with different data and different initial parameter values

Benchmark Results (2)

- Using initial parameters values further away from minimum solution





Using a starting point further away we start to see more fit failures !

ROOT Minimization Interface

- ROOT provides class **`ROOT::Math::Minimizer`** as general interface for minimization
- Current default is TMinuit (old Minuit implementation)
  - plan to switch to use Minuit2 as default in the next release
- implemented by several algorithms
  - TMinuit, Minuit2, Fumili, GSL Minimisers and GSL Fitting algorithms (Levenberg-Marquardt)
  - also simulated annealing and Genetic algorithm
  - RMinimizer (minimiser based on R algorithms)
  - and from Python: `scipy.optimize`

Scipy optimizers

- *O. Zapata* developed an implementation of ROOT::Math::Minimizer using `scipy.optimize`
- `scipy.optimize.minimize` provides several minimization algorithms

scipy.optimize.minimize

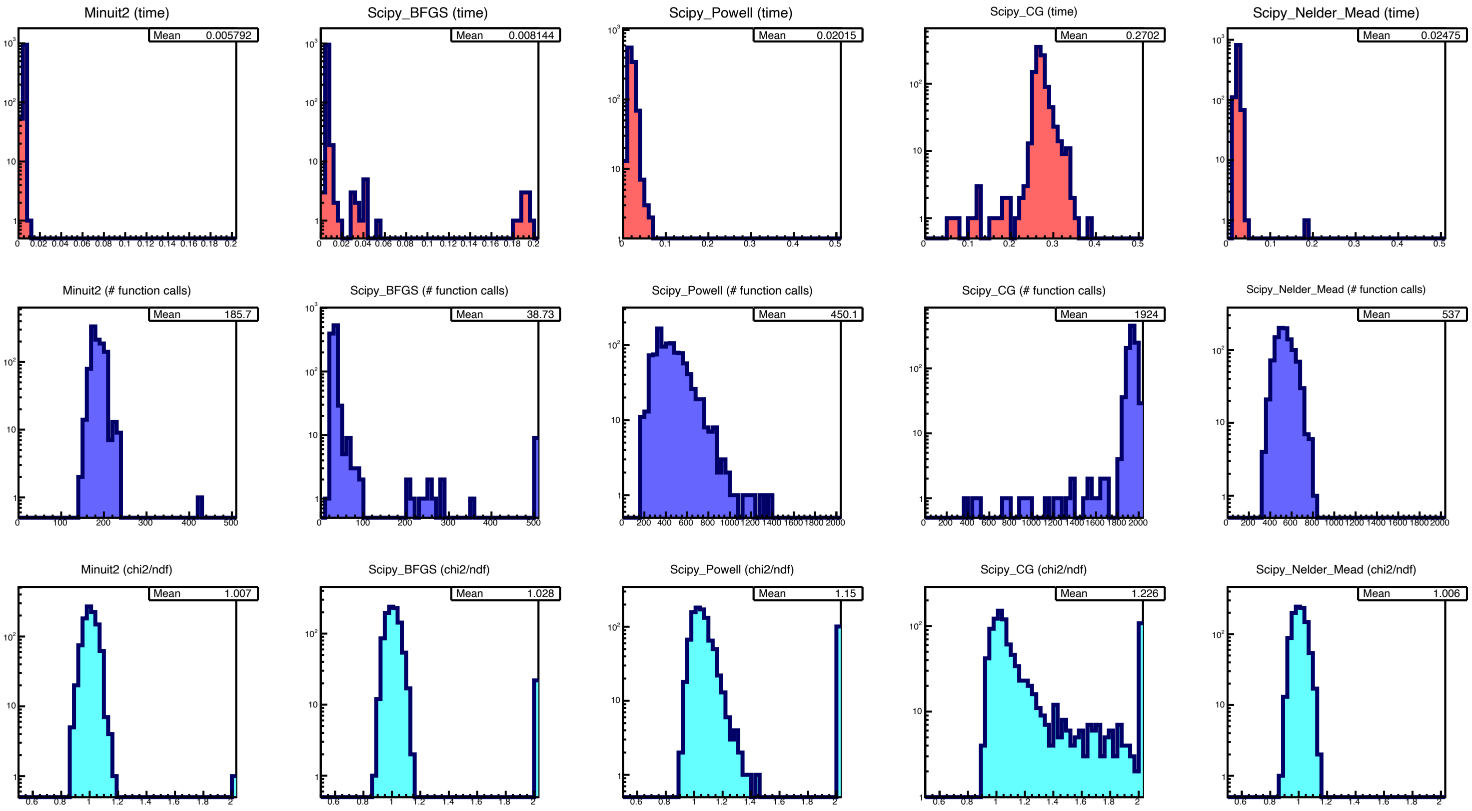
```
scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None,
hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None) #
```

method : ***str or callable, optional***

Type of solver. Should be one of

- 'Nelder-Mead' (see here)
- 'Powell' (see here)
- 'CG' (see here)
- 'BFGS' (see here)
- 'Newton-CG' (see here)
- 'L-BFGS-B' (see here)
- 'TNC' (see here)
- 'COBYLA' (see here)
- 'SLSQP' (see here)
- 'trust-constr'(see here)
- 'dogleg' (see here)
- 'trust-ncg' (see here)
- 'trust-exact' (see here)
- 'trust-krylov' (see here)

Benchmark using Scipy Minimisers



Poor performance of `scipy` with respect to Minuit!

Using Scipy Minimizer interface from *O. Zapata*

iminuit

- Jupyter-friendly Python frontend to Minuit2 C++ library in ROOT
- Part of Scikit-HEP project, developed in sync with ROOT
- Backend in particle and astroparticle physics libraries zfit, pyhf, gammapy, flavio, ctapipe, ...
- Easy to install: *pip install iminuit* installs precompiled binary package on all major platforms
- Comprehensive documentation with many tutorials
- 100 % test coverage

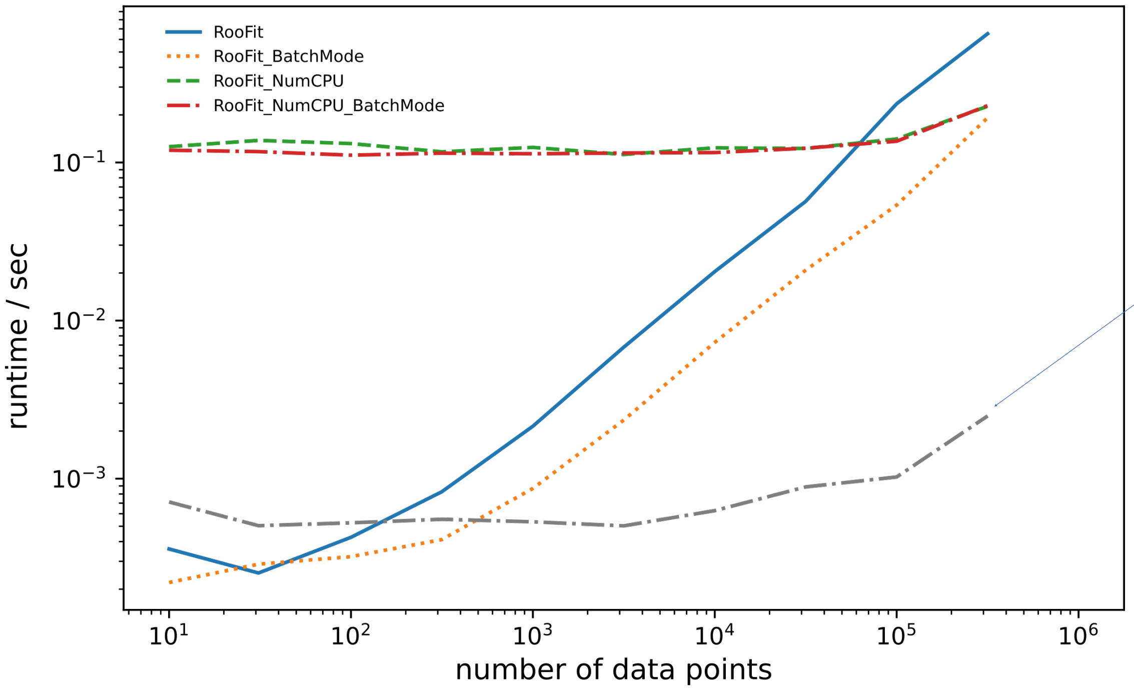
- Batteries included: shipped with common cost functions for statistical fits
    - Binned and unbinned maximum-likelihood
    - **Template fits (new):** including mix of templates and parametric models HD, A. Abdelmotteleb EPJ C 82, 1043 (2022)
    - Non-linear regression with (optionally robust) weighted least-squares
    - Gaussian penalty terms
    - Cost functions can be combined by adding: *total_cost = cost_1 + cost_2*
- Support for SciPy minimisers as alternatives to Minuit’s Migrad algorithm
- Smart visualization of fit results in Jupyter notebooks + **interactive fits**

Example fit with interactive fitting widget

```python
import numpy as np
from scipy.stats import norm
from iminuit import Minuit, cost

truth = 100., 200., 0.3, 0.1, 0.7, 0.2

def scaled_cdf(xe, n1, n2, mu1, sigma1, mu2, sigma2):
    return n1 * norm.cdf(xe, mu1, sigma1) + n2 * norm.cdf(xe, mu2, sigma2)

xe = np.linspace(0, 1)
m = np.diff(scaled_cdf(xe, *truth))
n = np.random.default_rng(1).poisson(m)  # generate random histogram

c = cost.ExtendedBinnedNLL(n, xe, scaled_cdf)
m = Minuit(c, *truth)

m.inter
```

High performance fitting in Python with iminuit

- Using Python not performance bottleneck, if numerical code is accelerated with Numba JIT
- Crucial for high performance: accelerated parallelized SIMD-friendly PDF and accelerated unbinned likelihood function
- Benchmarks for unbinned likelihood fit of normal distribution with parameters μ, σ



- iminuit
- iminuit.cost.UnbinnedNLL
- numba-accelerated normal distribution from numba-stats package
- automatic parallelization and fastmath

Up to 100x faster than RooFit (C++) with NumCPU (parallel computation) and BatchMode (≈ fastmath) options

Conclusions

- Minuit is more than 50 years old but it still the best minimization algorithm for HEP fitting problems
- Minuit2 implementation will be made soon the default in ROOT
  - improved recently by adding Fumili and BFGS
  - add support for external gradient and Hessian (for AD users)
  - improve logging and usability
  - multi-thread-safe if user provided function is
- Python version (*i*minuit) available for the Python user community
- Future work:
  - integrate more trust-region based methods in generic mimimizations
  - implement support for non-trivial parameter constraints