New Developments in Minuit2

L. Moneta, J. Rembser O. Zapata

Introduction

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.
- Works 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 same algorithm (MIGRAD)
- Available both in ROOT and as a standalone version
- Being used in the statistical analysis of LHC experiments
- Default minimizer in ROOT since latest release, 6.32
- **iMinuit** : python package built on top of Minuit2
  - used in large astroparticle physics experiments

Characteristics of Minuit

- **Works very well,** superior to gradient descent methods
  - Much less number of iterations to converge
  - No need to perform matrix inversion at each iteration
  - Approximate Hessian converges to true Hessian at the minimum
  - Regularisation when Hessian is not positive defined
    - add offset to the diagonal of H to make it positively defined
  - Self-correcting if the Hessian approximation is not good enough
- Disadvantages:
  - Sensitive to initial parameters, it is a local minimiser and can get stuck in local minima
  - Sensitive to bad numerical precision in function and gradient calculation
  - Does not scale to problems with a huge number of parameters
    - proven to work to $> \sim 1000$ parameters (e.g Higgs combination fits)
    - will not work for training deep-learning models with millions of parameters
      - need to use gradient descent in these cases

External Gradient and Hessian

- Minuit requires the function gradient at each iteration
  - computed by default numerically using a 3 points rule and adaptive step sizes
    - well-tested and robust method
    - essential to having good precision when the gradient is close to zero (near the minimum) to converge rapidly
- Support for external gradients provided by user
  - needed for users exploiting Automatic Differentiation (AD)
- **New**: Option in Minuit2 to provide external Hessian or only the diagonal of the Hessian (G2) for seeding
  - without providing Hessian, Minuit2 computes G2 numerically
  - using initial user steps is often not good (need good estimates)

New improvements in Minuit2

- Improved debugging
  - can log and return to user all minimisation iteration states
  - can provide a detailed output of each iteration (in debug mode)
- Possibility to add users callback functions at each iteration
- Thread-safety: Minuit2 can work in multi-threads if user provided function can
  - support for likelihood or gradient parallelisation
- Addition of new minimization methods:
  - **BFGS**: use only standard BFGS formula instead of the default mode of using both BFGS or DFP formula depending on some conditions

New Strategy 3

- Added a new strategy (**strategy=3**) thanks to Will B.
  - Similar behaviour as strategy 1 and 2, but with improved Hessian computation
    - use diagonal Hessian for seeding as in strategy 1
    - use same number of cycles (iterations) as in strategy 2
  - Compute off-diagonal Hessian elements using central derivatives (5-point rule: 3 extra function evaluation)
- This gives improved precision in Hessian in case of fits with large statistics
  - Avoid the problem of having a non-positive defined Hessian after the minimization

Specialized Algorithms for Fitting

- When minimising Least-square functions:

$$F(x) = \sum_{k=1}^{n} f_k^2 = \sum_{k=1}^{n} \left(\frac{y_k - T_k(x)}{\sigma_k} \right)^2$$

Hessian $$H_{ij} = \frac{\partial^2 F(x)}{\partial x_i \partial x_j} = \sum_{k=1}^{n} 2 \frac{\partial f_k \partial f_k}{\partial x_i \partial x_j} + 2 f_k \frac{\partial^2 f_k}{\partial x_i \partial x_j} \approx \sum_{k=1}^{n} 2 \frac{\partial f_k \partial f_k}{\partial x_i \partial x_j}$$

$$\Rightarrow \quad \mathbf{H} \approx \mathbf{J}^T \mathbf{J}$$

this can be neglected when residuals f are small

Neglect second derivatives of model function: linearisation

- Many algorithms have been developed on this approximation:
  - e.g. Levenberg-Marquardt (GSL), Fumili, …

Likelihood Fits

- **For likelihood functions:**

$$\mathscr{L}(x) = -\sum_{k=1}^{n} \log f(y_k|x) \quad \text{and} \qquad H_{ij} = \frac{\partial^2 \mathscr{L}(\theta)}{\partial\theta_i \partial\theta_j} = -\sum_{k=1}^{n} \frac{\partial}{\partial x_i}\left(\frac{1}{f_k}\frac{\partial f_k}{\partial x_j}\right) = \sum_{k=1}^{n} \frac{1}{f_k^2}\frac{\partial f_k \partial f_k}{\partial x_i \partial x_j} - \sum_{k=1}^{n} \frac{1}{f_k}\frac{\partial^2 f_k}{\partial x_i \partial x_j}$$

  - the linear approximation is not always valid!

- **For binned likelihood fits**, can write the likelihood as

$$\mathscr{L}(x) = -\sum_{k=1}^{n} \log P(n_k|\mu_k(x)) = -\sum_{k=1}^{n} \log \frac{e^{-\mu_k(x)}\mu_k(x)^{n_k}}{n_k!}$$

and after removing constant terms

$$\mathscr{L}(x) = \sum_{k=1}^{n} \left(\mu_k(x) - n_k \log \mu_k(x)\right)$$

$$H_{ij} = \frac{\partial^2 \mathscr{L}(\theta)}{\partial\theta_i \partial\theta_j} = \sum_{k=1}^{n} \frac{\partial}{\partial x_i}\left(\frac{\partial \mu_k}{\partial x_j} - \frac{n_k}{\mu_k}\frac{\partial \mu_k}{\partial x_j}\right) = \sum_{k=1}^{n} \frac{n_k}{\mu_k^2}\frac{\partial \mu_k \partial \mu_k}{\partial x_i \partial x_j} - \sum_{k=1}^{n} \frac{(n_k - \mu_k)}{\mu_k}\frac{\partial^2 \mu_k}{\partial x_i \partial x_j} \approx \sum_{k=1}^{n} \frac{n_k}{\mu_k^2}\frac{\partial \mu_k \partial \mu_k}{\partial x_i \partial x_j}$$

$$\mathbf{H} \approx \mathbf{J}^T\mathbf{J}$$

this can be neglected
it is like a residual f_k

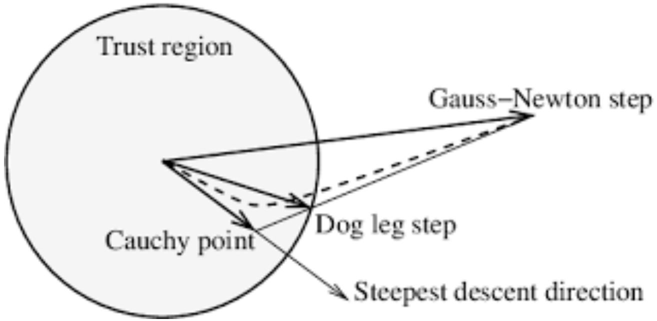
- **The same algorithms used for least-square fitting can be used !**

Specialized Fitting Methods

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

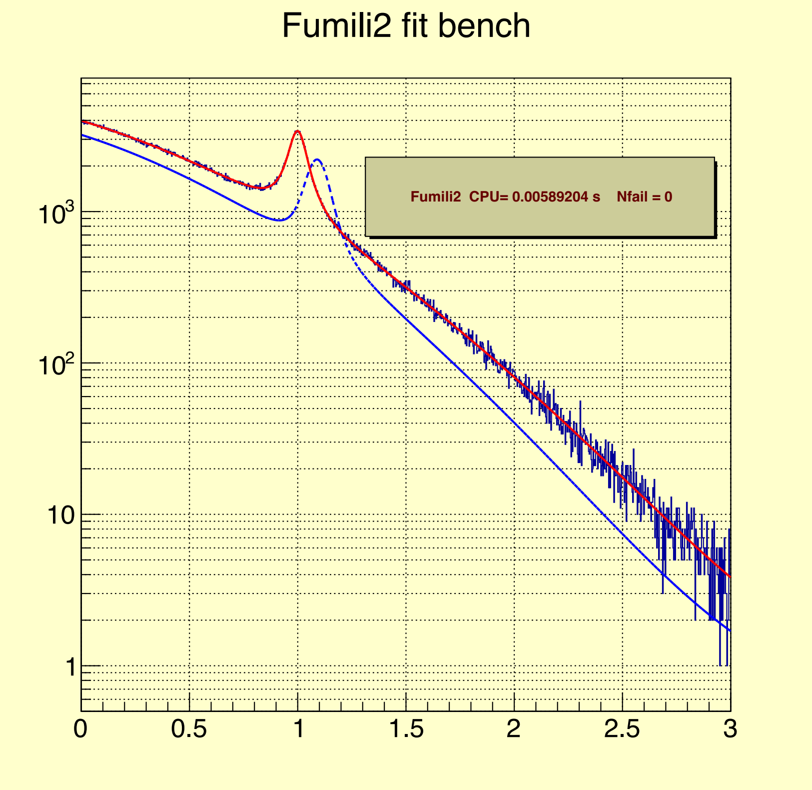
New Fumili Algorithm

- New implementation of Fumili algorithm: **Fumili2**
  - original algorithm from I. Silin implemented in the Cernlib and `TFumili` class
- It is integrated into Minuit2 library
  - re-using Minuit2 interfaces classes
  - working for both 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



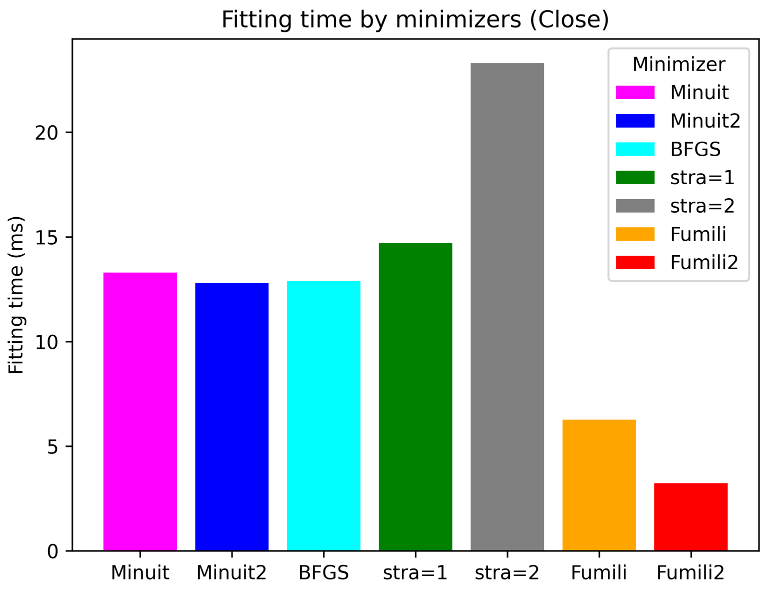
Benchmark Results

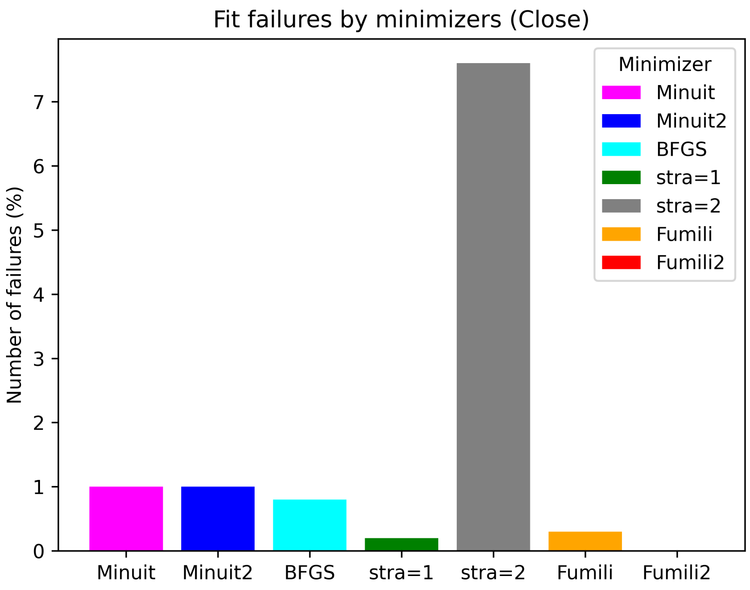
- Use a binned likelihood to fit signal peak over some background in a histogram
  - 1000 bins
  - 7 parameter fits performing numerical convolution
  - repeat fit 1000 times with different data and different initial random parameter values
    - not too far from the minimum



Benchmark Results

- Binned likelihood fit to signal peak over some background

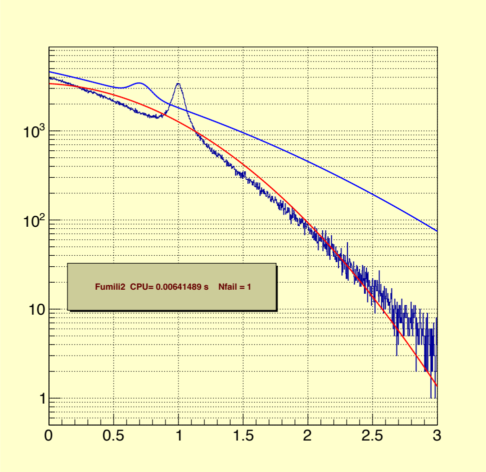




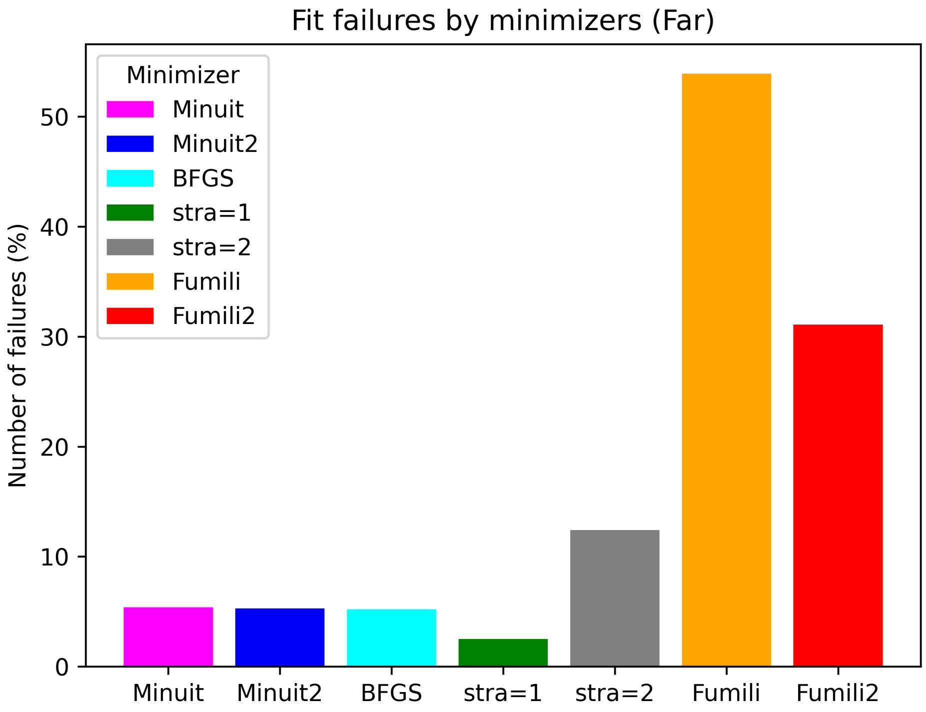
- New Fumili algorithm (**Fumili2**) works very well !

Benchmark Results (2)

- **With initial parameters values further away from minimum**



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
  - `TFumili`
  - GSL minimisers and fitters algorithms (Levenberg-Marquardt)
  - Simulated annealing and Genetic algorithm
  - R-Minimizer : minimiser based on algorithms from R
  - and now from Python: `scipy.optimize`

Scipy optimizers

- New implementation of ROOT::Math::Minimizer using `scipy.optimize` (from *O. Zapata)*
- `scipy.optimize.minimize` provides several minimization algorithms

scipy.optimize.minimize

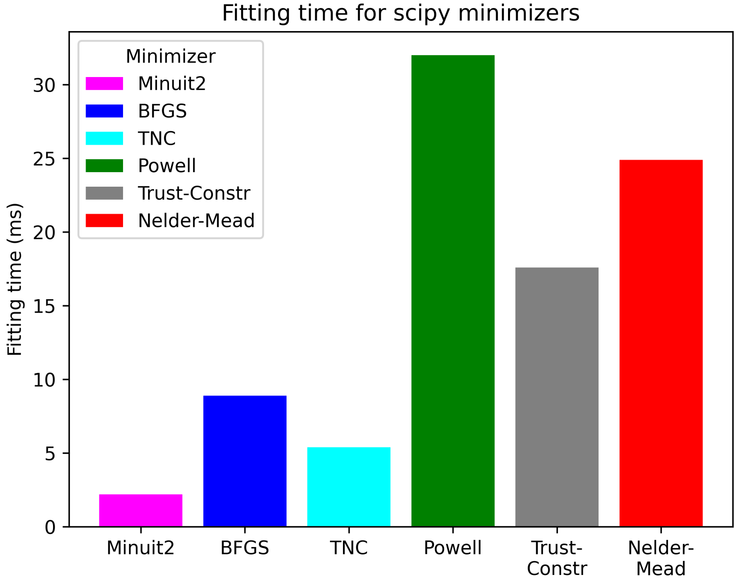
```
scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None,
hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None) #
```

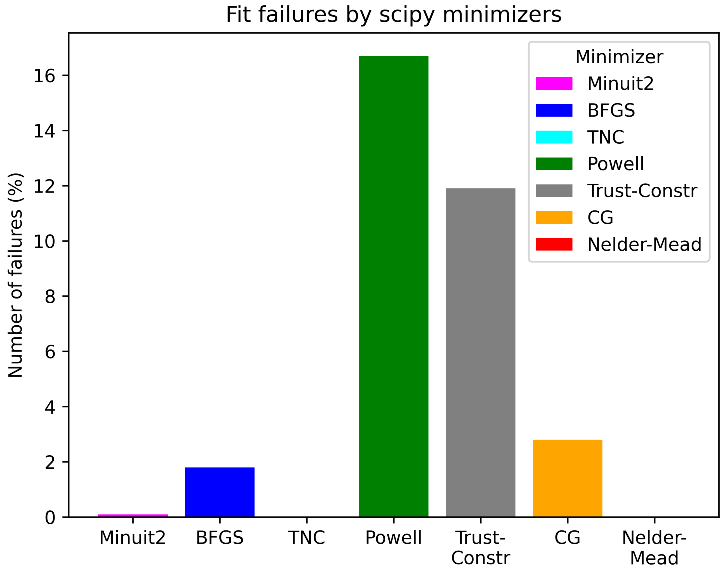
method : *str or callable, optional*

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 with Scipy





Time for CG is > 600 ms

- Varying performance of `scipy` minimisers
  - Minuit2 performs better!
- Fitting using AD
  - without providing gradients `scipy` optimisers perform worse
    - *e.g. number of failures for TNC is more than 80%*

Conclusions

- Minuit is more than 50 years old but it seems to be still the best minimization algorithm for HEP fitting problems
- New algorithm (Fumili2) for least-square and binned likelihood fit
- Recent improvements in Minuit2:
  - support for external gradient and Hessian (for AD users)
  - improve logging and usability
  - new strategy 3 for fixing some issues with high statistics fits
- Minuit2 is now the default minimiser in the latest ROOT version (6.32)
- Future work:
  - implement support for non-trivial parameter constraints

References

- Minuit2:
  - Users guide
  - Minuit Tutorial on Function Minimization (*F. James*)
- ROOT Minimisers
  - ROOT::Math::Minimizer
- scipy:
  - scipy.optimize.minimize documentation
  - scipy ROOT interface
- iMinuit
  - https://iminuit.readthedocs.io/en/stable/

Backup Slides

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

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

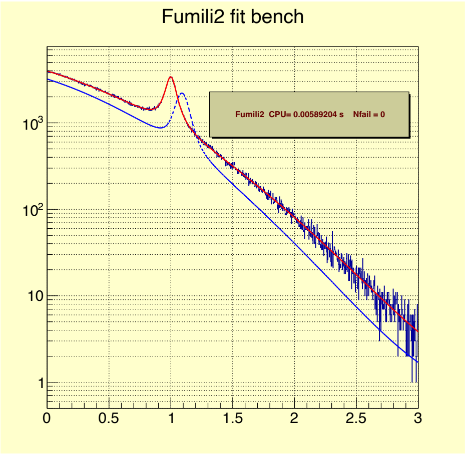
  - stop iteration when the Expected Distance from the Minimum (EDM) $\rho = g^T H g$ is small
- EDM provides a scale-invariant quantity to tell the convergence of method.
  - This is unique in Minuit!

Fumili Algorithm

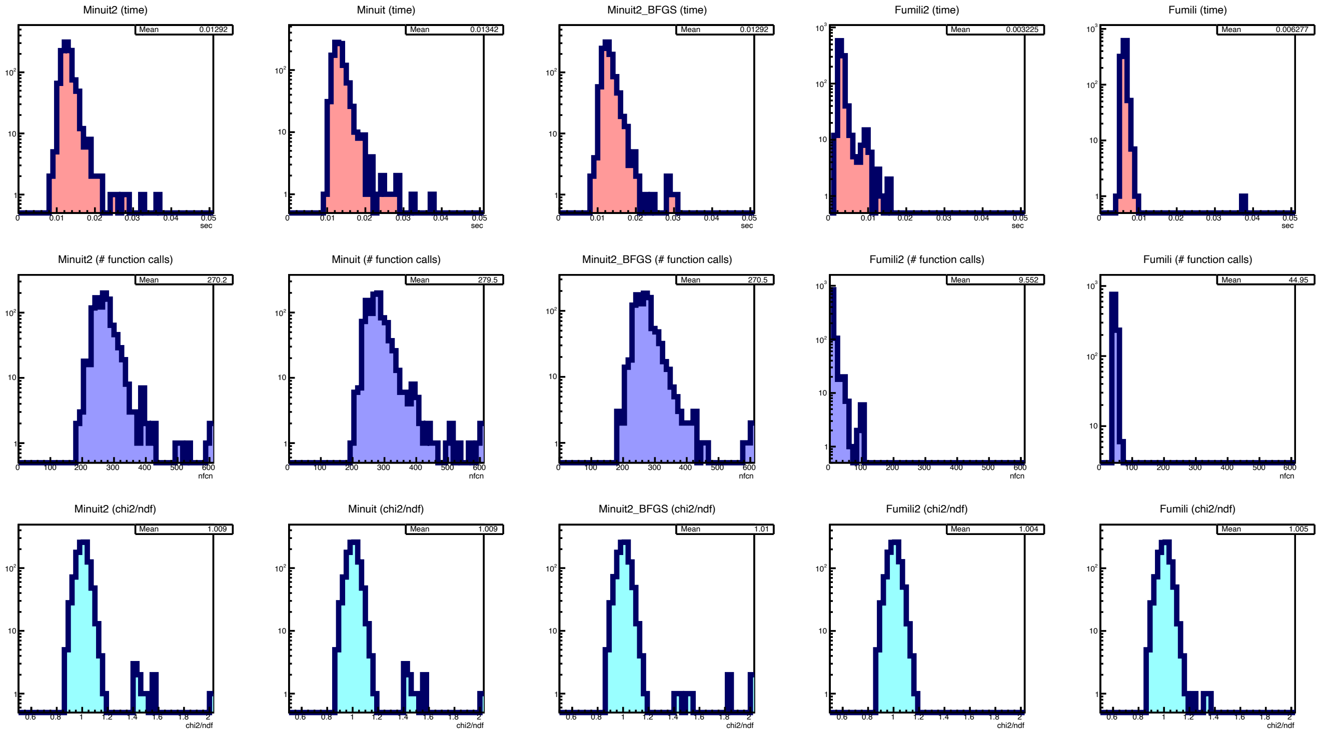
- Old algorithm proposed already in 1961 by I. Silin
- Implemented later in the CERN library and made also available to ROOT with TFumili class.
    - It uses the Hessian approximation combined with a trust region method.
        - a multidimensional parallelepiped ("box") is defined around the point 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 fits when the starting point is close enough to the solution

Benchmark Results

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

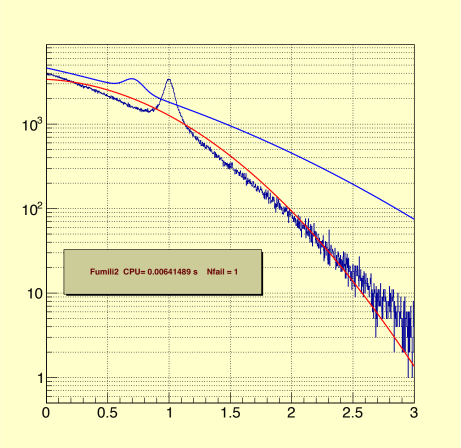


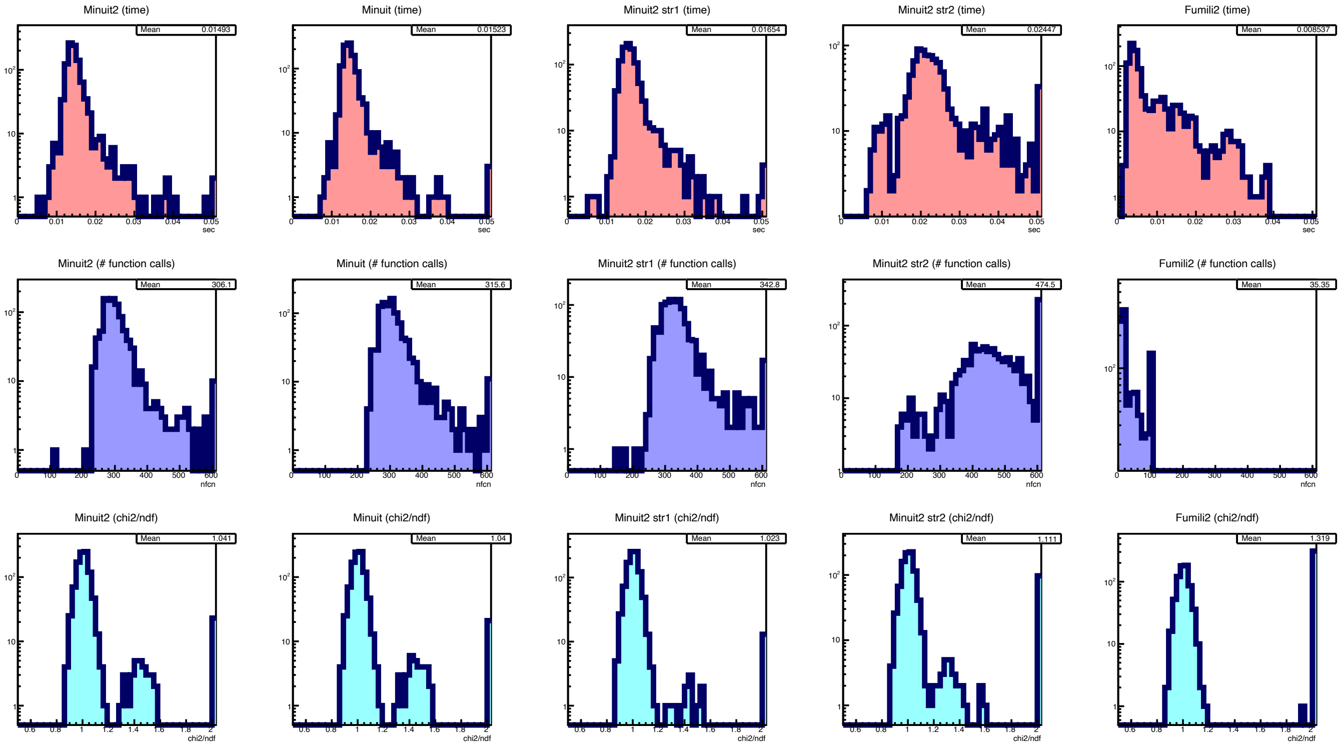
1000 bins - 7 parameters repeat fit 1000 times with different data and different initial parameter values



Benchmark Results (2)

- Using initial parameters values further away from minimum solution

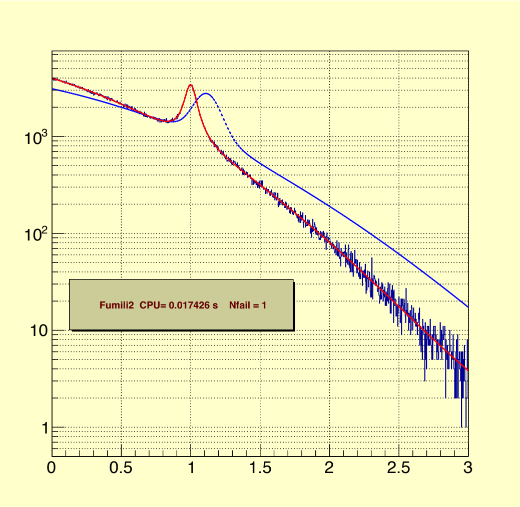




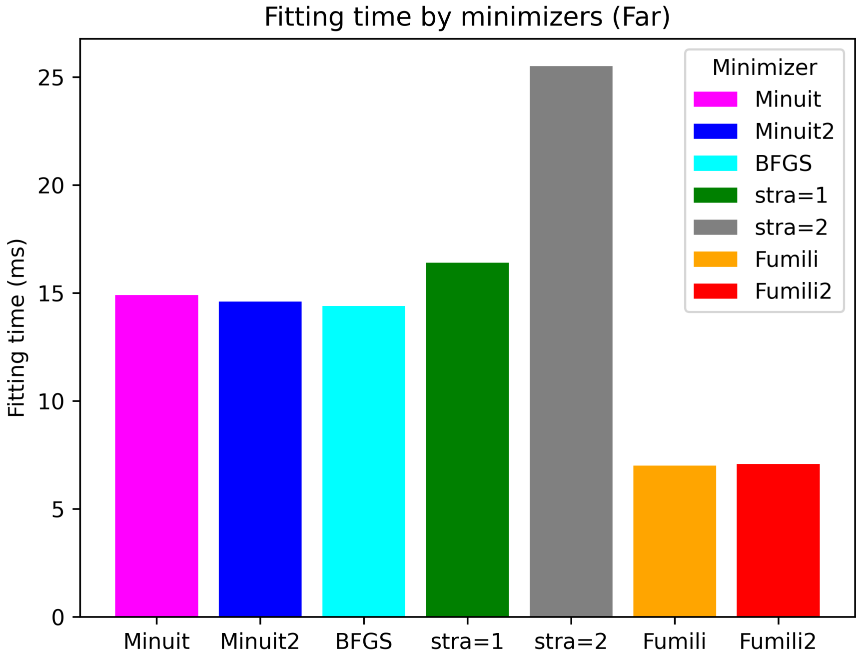
Using a starting point further away we start to see more fit failures !

Benchmark Results (2)

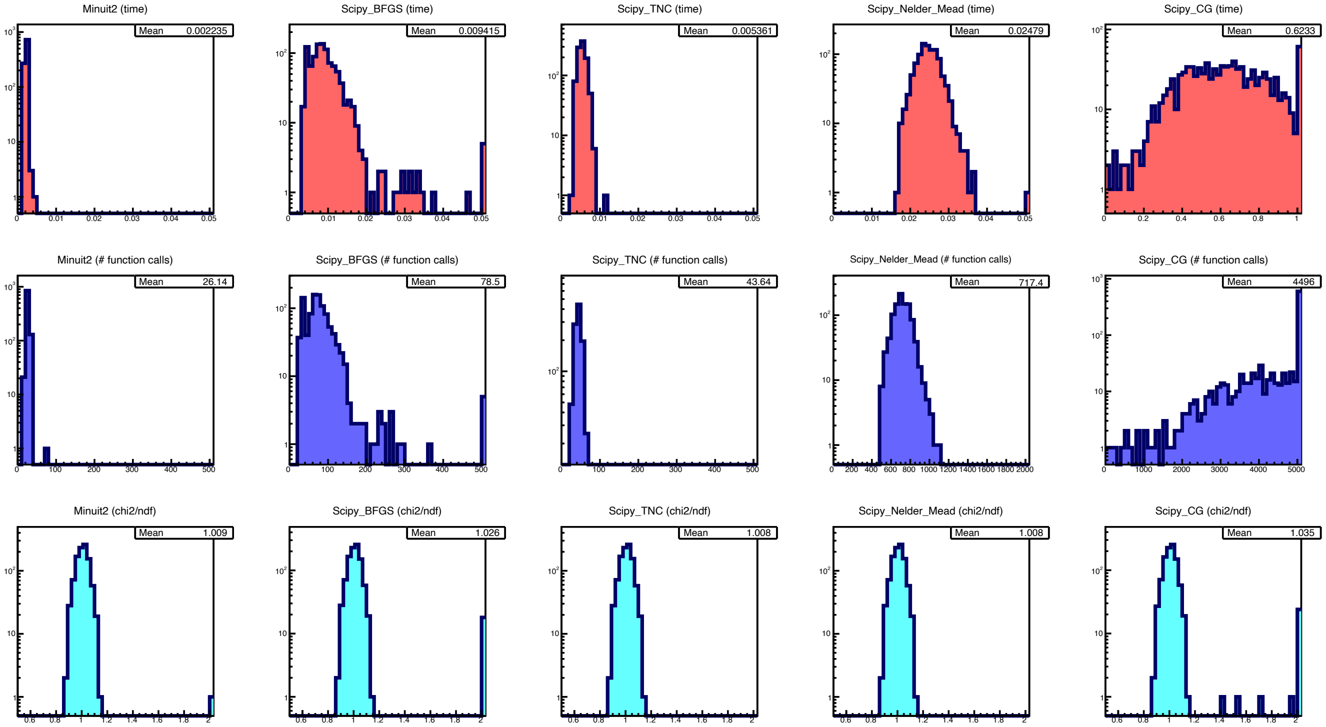
- Using initial parameters values further away from minimum solution



Using a starting point further away we see also longer fitting time



Benchmark using Scipy Minimisers



Poor performance of `scipy` with respect to Minuit!

Using Scipy Minimizer interface from *O. Zapata*

Scipy using Numerical Derivatives

- Fitting time and failures in Scipy with numerical gradients