Recent Improvements of the ROOT Fitting and Minimization Classes

Lorenzo Moneta
(CERN, PH-SFT)

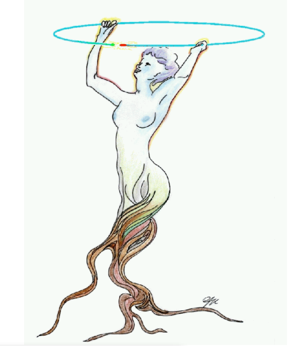
ROOT
An Object-Oriented Data Analysis Framework

ROOT Math Work Package:
R. Brun, D. Gonzalez-Malin, A. Kreshuk, L.M., E. Offermann
W. Verkerke(RooFit), K. Cranmer (RooStat), TMVA team and many other contributors

Outline

- ROOT Math and Statistical classes
- Recent developments in ROOT Math classes
- Fitting Improvements
  - Fitting GUI Editor
  - New Fitting classes
- Minimization
- New Statistical classes (*RooStats*)
- Conclusions

Structure of ROOT Math/Stat Libraries



High Level Analysis Libraries
- RooStats
- RooFit
- TMVA

Fitting and Minimization
- Quadp
- Linear Fitter
- TFumili
- Minuit2
- TMinuit

Statistical Libraries
- Limit classes
- MLP

Extra Libraries
- Unuran
- FFTW
- Spectrum
- Foam

Linear Algebra
- SMatrix
- TMatrix

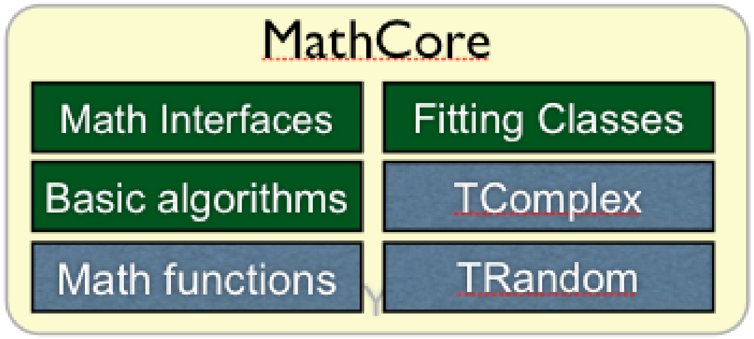
Physics Vectors
- GenVector
- TVector3/TLV

MathCore
- Math Interfaces
- Fitting Classes
- Basic algorithms
- TComplex
- Math functions
- TRandom

MathMore
- Random Numbers
- Extra algorithms
- Extra Math functions
- GSL

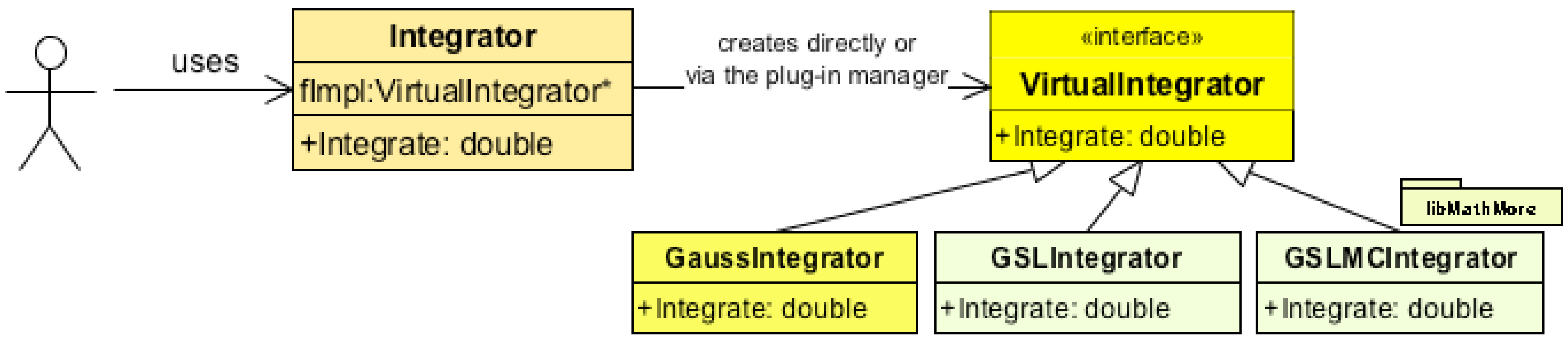
Recent MathCore Developments

- Provide in *libMathCore* all basic Math functionality
- Improve basic functions (provided by `TMath` namespace):
  - use STL for some algorithms acting on data sets or array (sorting, location of min/max, etc..)
    - provide also a template interfaces (to avoid duplications)
  - implement mathematical and statistical functions using new set of implementations
    - better numerical precision and in some cases better CPU efficiency
- New numerical algorithm classes
  - classes implementing integration, derivation, root finders
  - provide also complementary implementation in *libMathMore*
    - transparent mechanism to use the alternative implementation via the plug-in manager
- New fitting classes

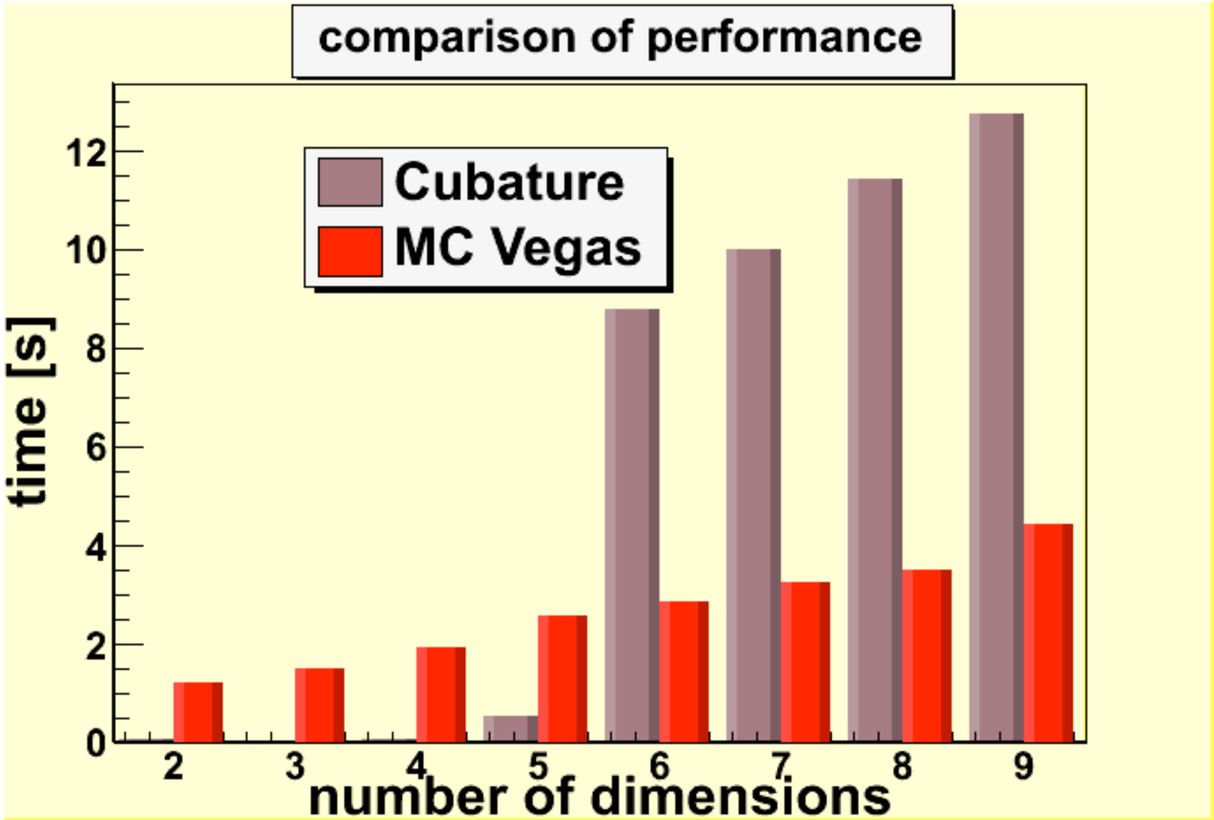


Numerical Algorithms

- Developed common interface for numerical algorithms
  - single entry point for multiple implementations
  - example: integrator class




```
using namespace ROOT::Math;
//multidim integrand function
double func( const double* x, const double *p);
....
// Functor class to wrap user function in interface
Functor f(func,dimension);
// adaptive cubature method
IntegratorMultiDim ig(IntegrationMultiDim::kADAPTIVE);
double v1 = ig.Integral(f,xmin,xmax);

// MC method (VEGAS) loaded from MathMore library
IntegratorMultiDim ig(IntegrationMultiDim::kVEGAS);
double v2 = ig.Integral(f,xmin,xmax);
```

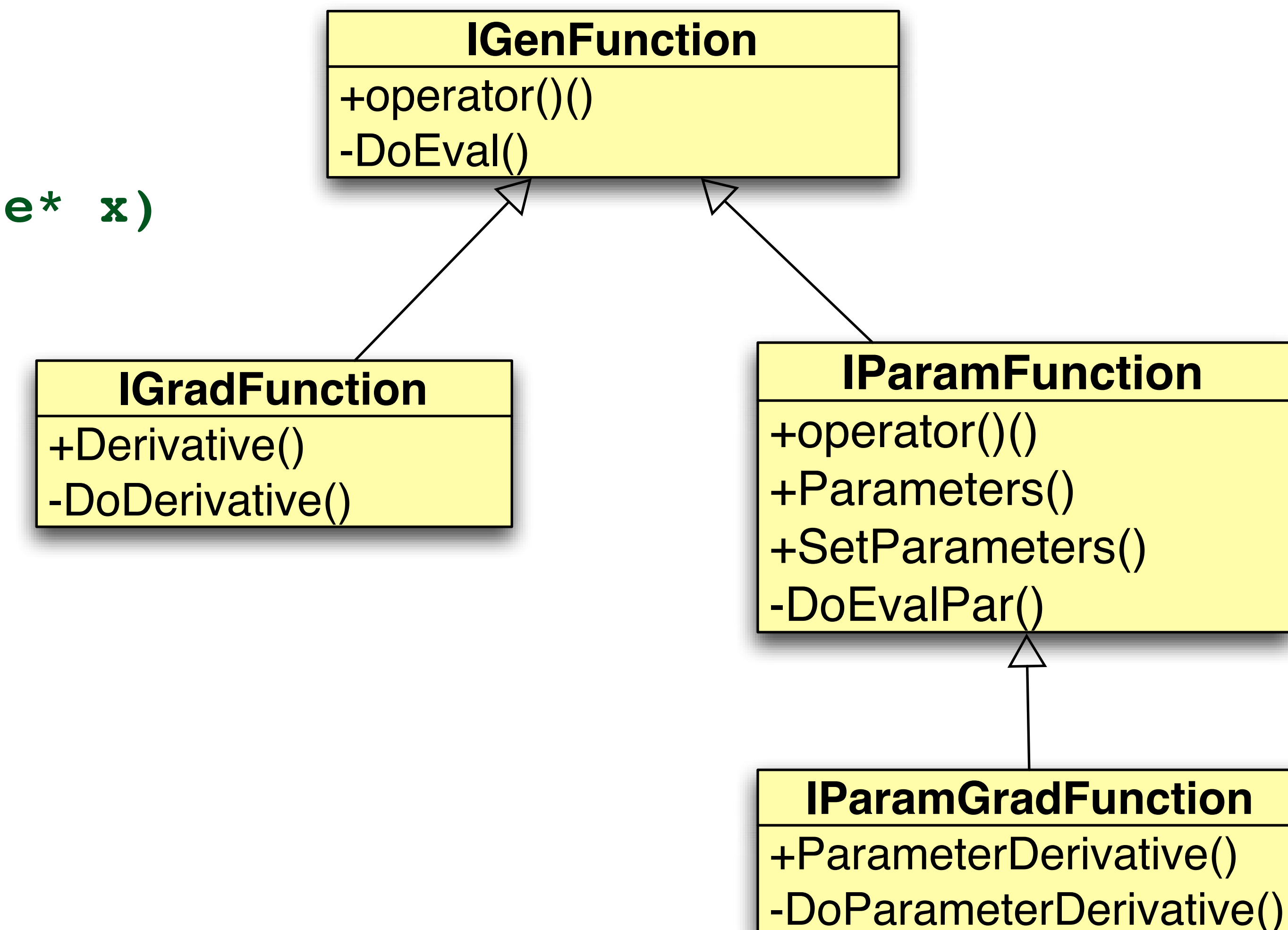


Numerical Algorithms

- Numerical algorithms present in the *MathCore* or *MathMore* library (implemented using the GNU SCientific Library):
  - **Numerical Derivation**
    - central evaluation (5 points rule) and forward/backward
  - **Numerical Integration**
    - gauss integration method
    - one dim. adaptive integration for finite and infinite intervals, singular functions and with Cauchy principal value.
    - multi-dimensional adaptive integration
    - multidimensional MC integration based on PLAIN, VEGAS and MISER
  - **Root Finders**
    - various bracketing and polishing algorithms using derivatives
  - **Minimization**
    - Golden section and Brent algorithm for 1D
    - conjugate gradient and BFGS algorithms for multi-dimensions
    - simulated annealing
    - non linear least square solver with Levenberg-Marquardt algorithm
  - **Interpolation**
    - linear, polynomial, cubic and Akima spline
  - **Chebyshev polynomials** (for function approximation)

Function Interfaces

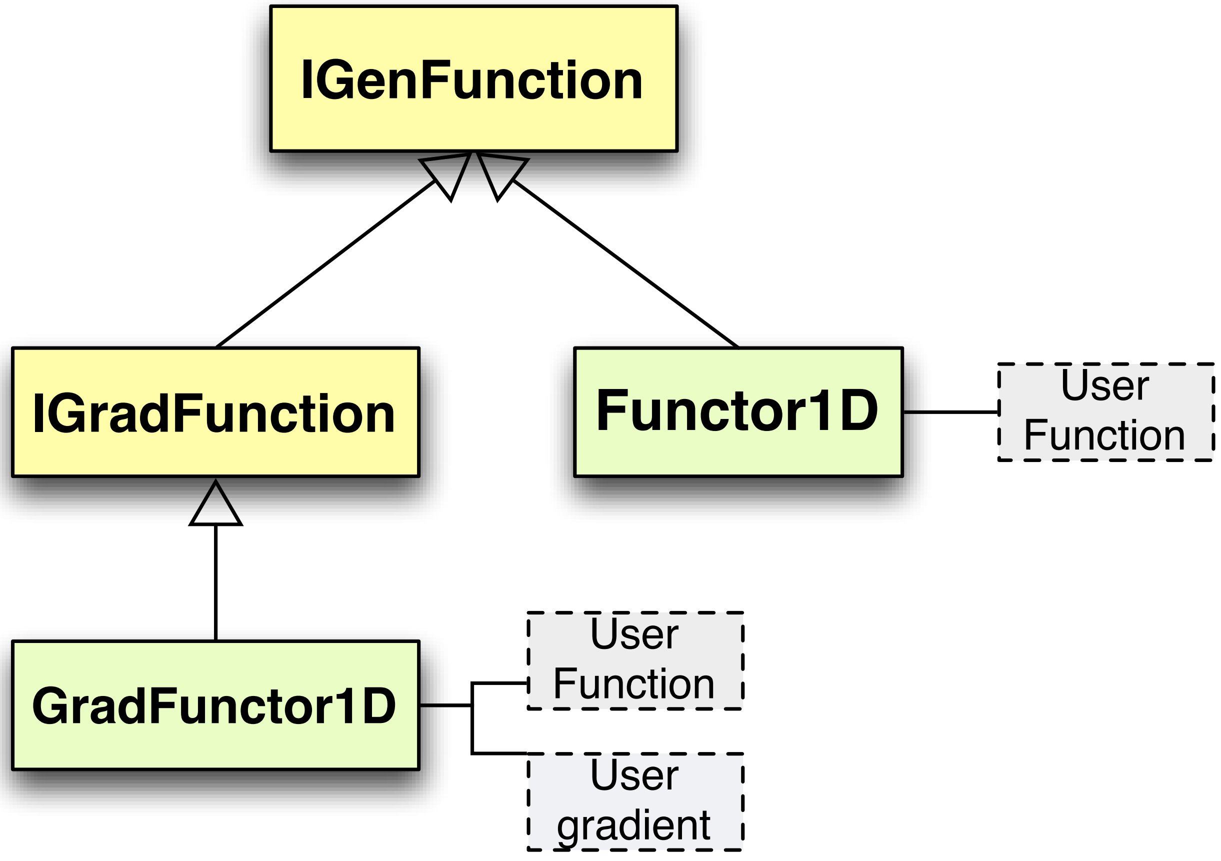
- Minimal function interface used by all numerical algorithms
- Abstract classes for one and multi-dimensional functions
  - simple interface for generic function evaluation
    - `double operator()(double x)`
    - `double operator()(const double* x)`
  - interfaces for functions providing analytical derivatives
    - for algorithms requiring function derivatives (minimization)
  - parametric function interfaces (used in fitting)
    - provide methods for setting/ retrieving parameter values
  - parametric gradient functions
    - provide gradient with respect to the parameter



Functor classes

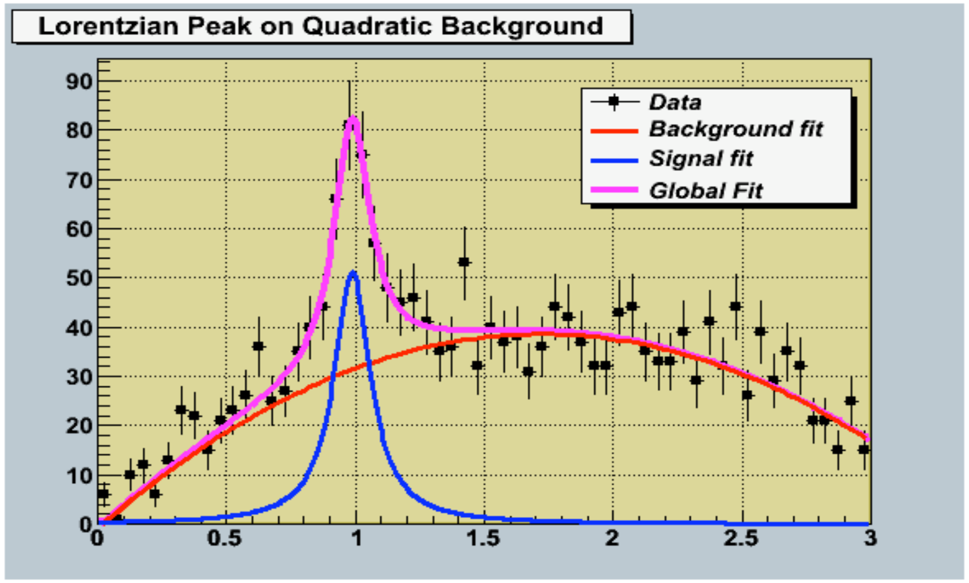
- Classes to wrap C++ callable object or class member function with the right signature
- Use functors to wrap any provided user function in the right interface

```
using namespace ROOT::Math;
// user function
double func( double x);

// Functor class to wrap a callable object
Functor1D f1(func);

// Functor to wrap member function
//  MyClass::Eval(x)
MyClass * myclass = new MyClass(...);
Functor1D f2(myclass, &MyClass::Eval);

// user derivative function
double deriv( double x);

// wrap function and derivative in
// a gradient interface
GradFunctor1D gf1(func,deriv);

// wrap class member functions
GradFunctor1D gf1(myclass, &MyClass::Eval,
                  &MyClass::Deriv)
```



Fitting in ROOT

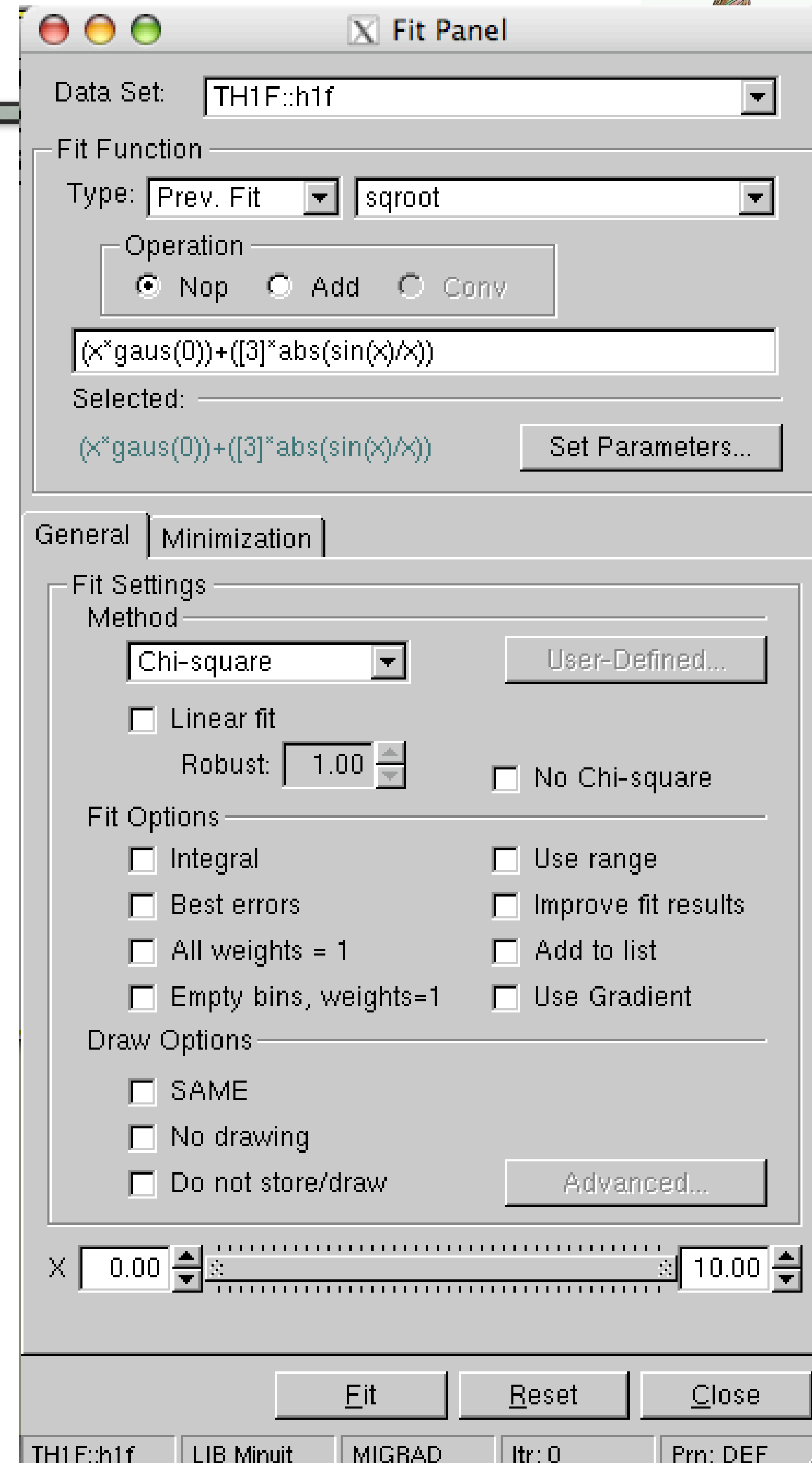
- Fit directly ROOT data classes (Histograms, Trees, Graphs)
- Various options available:
  - binned fits (`TH1::Fit`, `TGraph::Fit`)
    - Least Square fit (default)
    - Likelihood fits (`h1.Fit(func,"L")`)
  - unbinned likelihood fit (`TTree::UnbinnedFit`)
  - user defined model functions
  - user defined objective function
    - via option "U" and using `TVirtualFitter::SetFCN(myfcn)`
  - choice of minimization methods
    - *Minuit, Fumili, Minuit2, Fumili2, GSL*
  - automatic use of linear fits in case of linear functions (`"polN"`)
  - robust fit option (outlayer removal) in case of linear fits



- NEW: GUI (FitPanel) for driving the fit
  - available for all data objects (including the `TTree`)
- NEW: Improved ROOT fitting system and re-implemented the Fit methods for the data classes

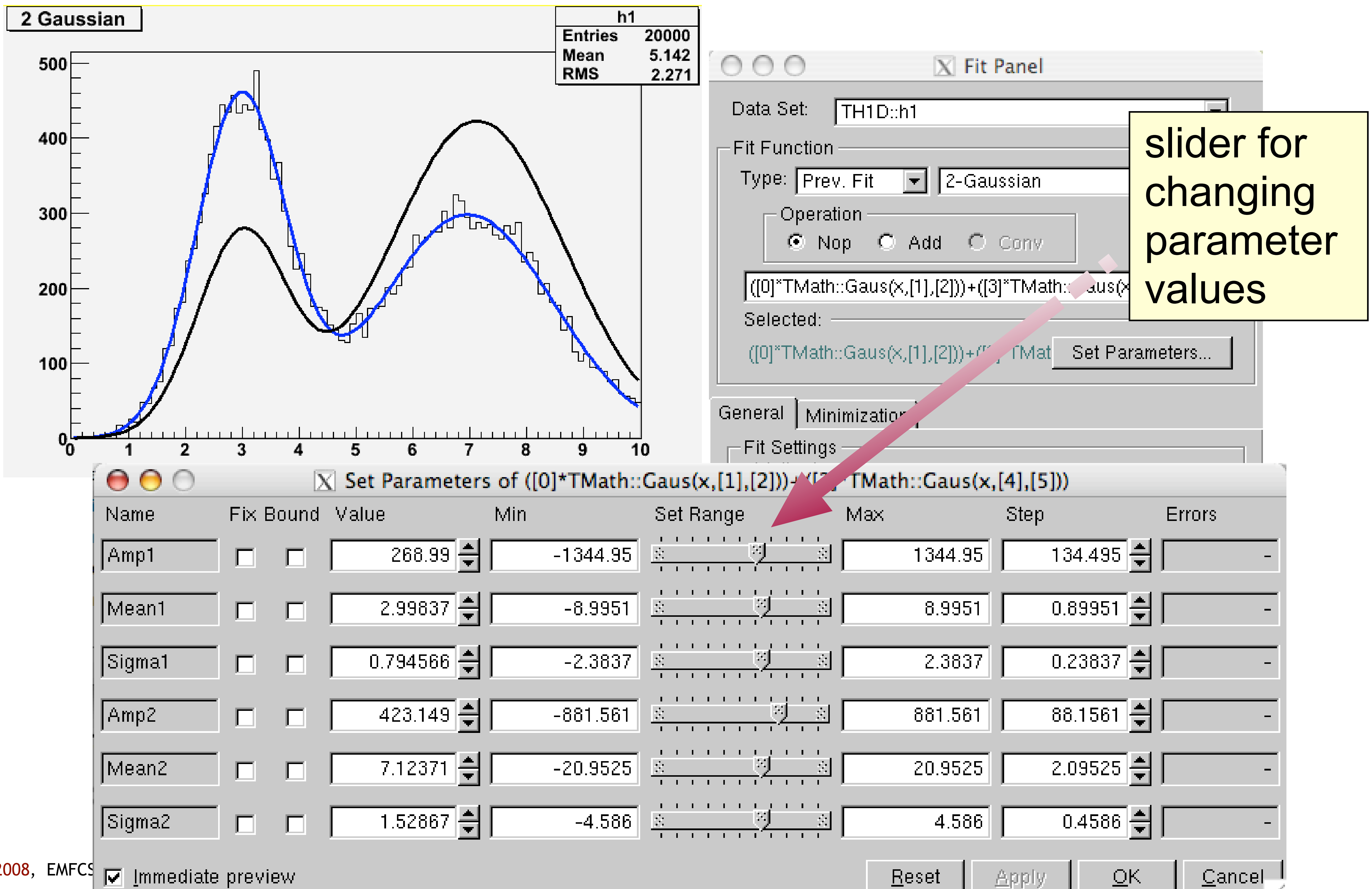
Fit Panel

- GUI for fitting all ROOT data analysis objects
  - easy function selection, fit options and methods
  - allow use of different minimizers (via a separate panel)
  - function parameters settings and control
- New improvements (by D. Gonzalez Maline)
  - include multi-dim histograms and graphs
  - unbinned fitting of TTree data
    - with selection of tree data and cuts
  - direct selection of data objects available in the current directory



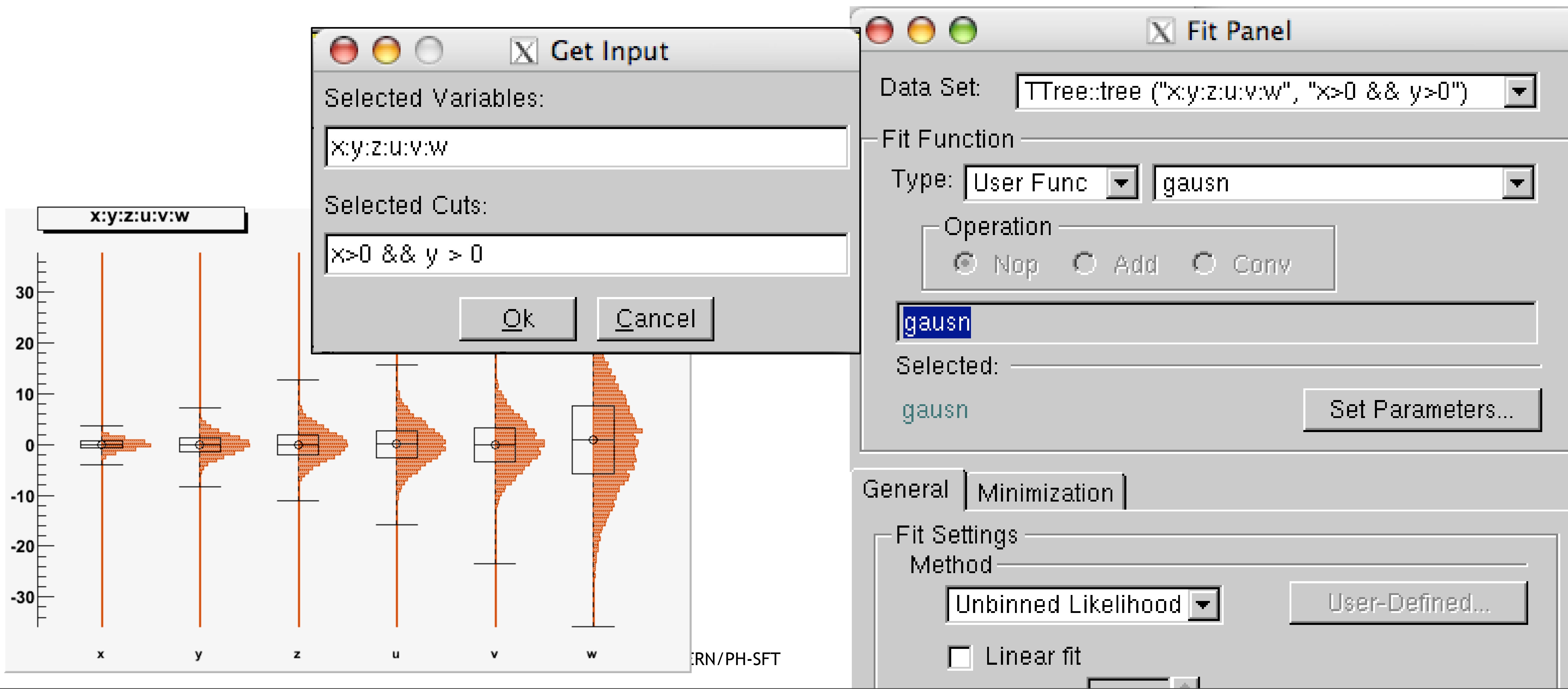
Fit Parameters Control

Immediate preview of the changed function



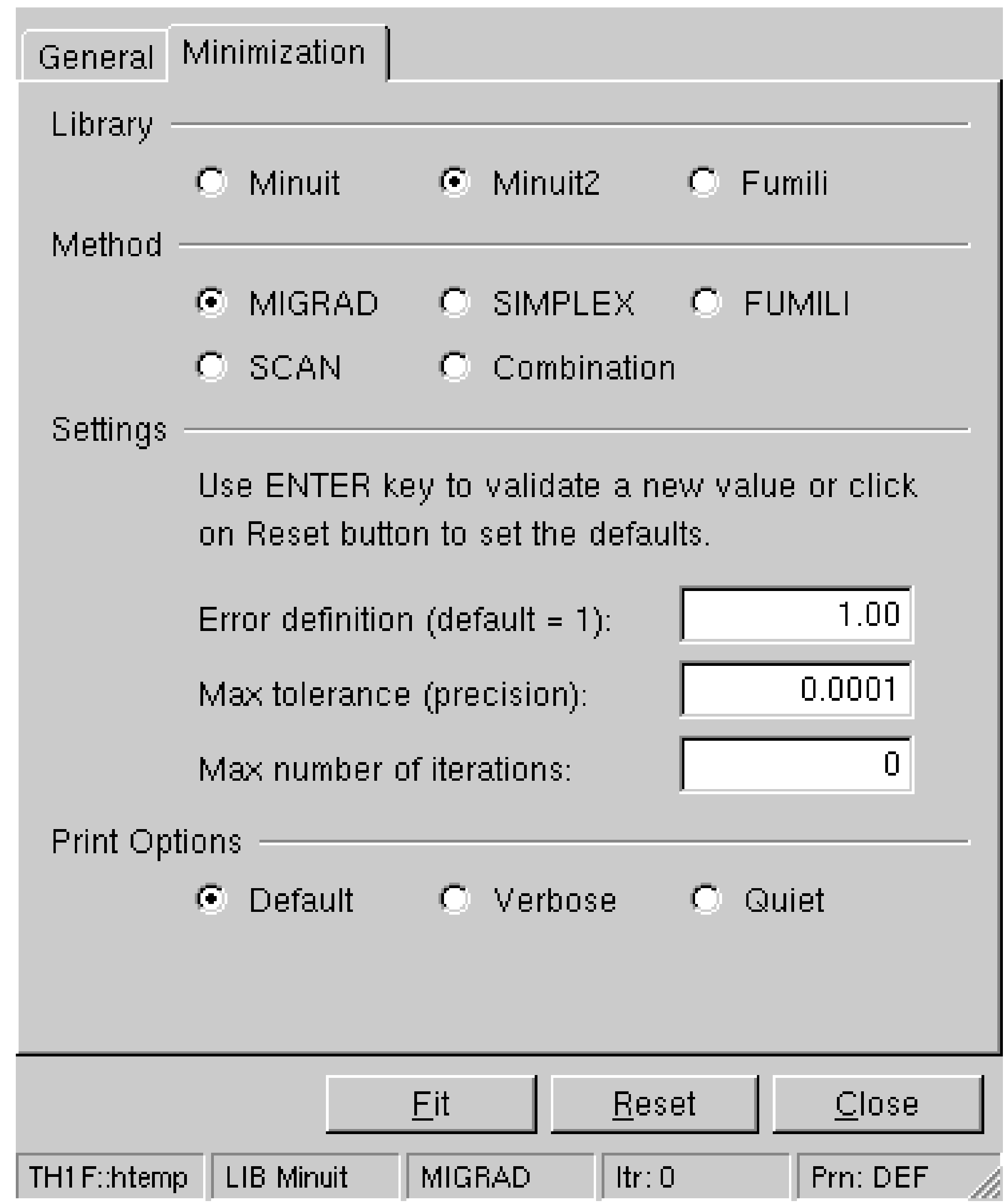
TTree Fitting

- Fit of `TTree` data sets
  - possible to select tree variables to be used in the fit
  - selection cut
- Unbinned Fit of un-limited number of variables



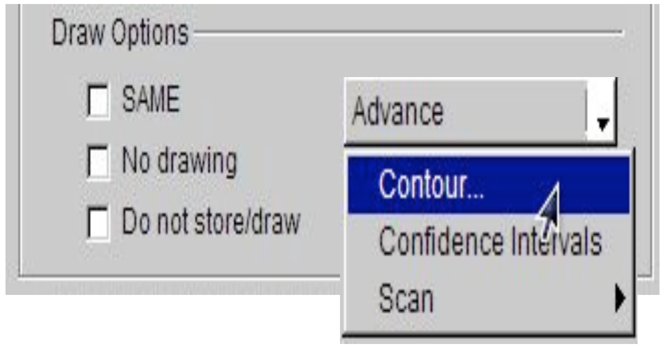
Minimizer Control

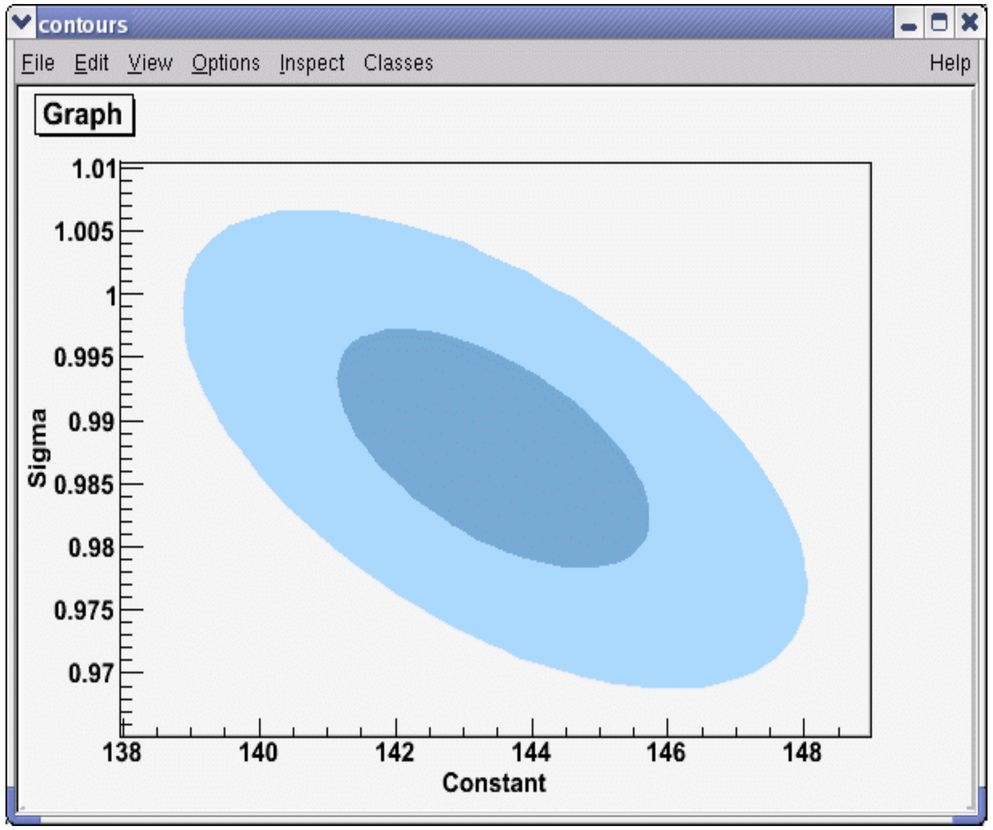
- Interactive library selection
  - *Minuit*, *Minuit2*, *Fumili*, etc.
- Choice of minimization algorithm
  - *Migrad*, *Simplex*, etc.
- Control of minimizer parameters
  - Tolerance, Maximum number of iterations, etc..
- Print Options
  - Default, Verbose, Quite

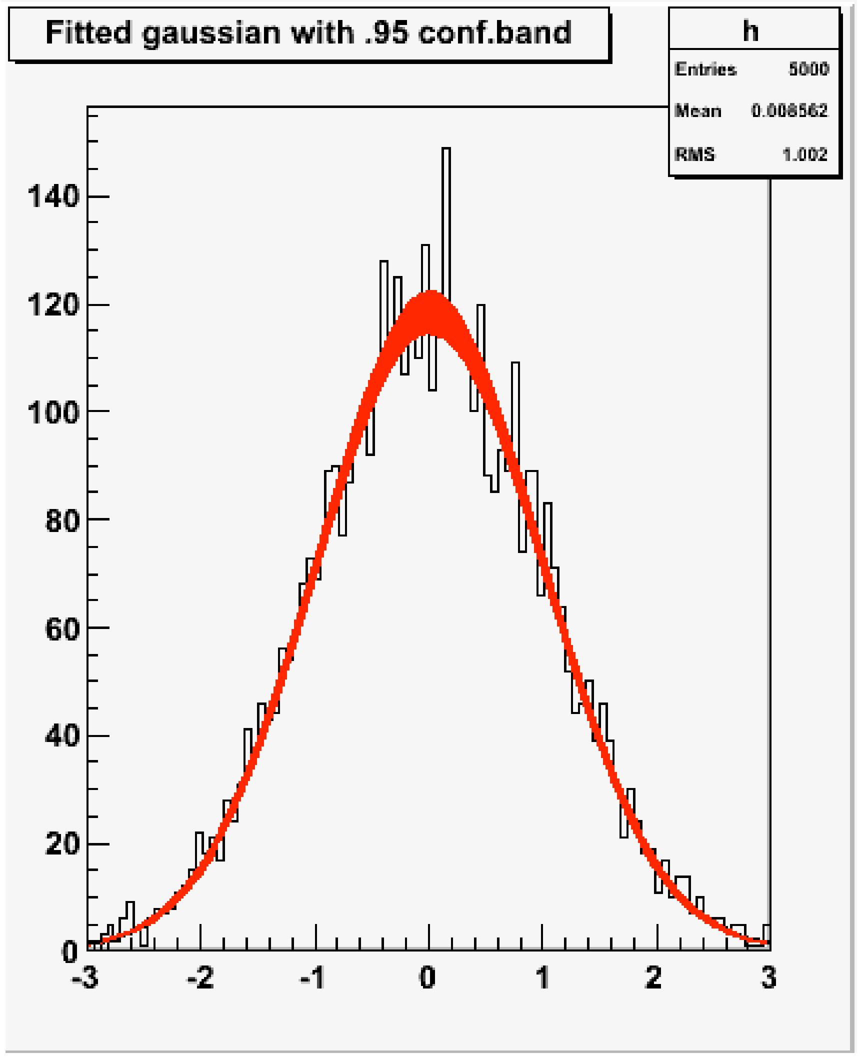


New Coming Features

- Advance drawing options from GUI
  - contour plots
  - scan plot of likelihood
  - confidence intervals plots







Fitting Improvements

- Recently re-engineered fitting and minimization classes
  - maintained full backward compatibility
  - old interface (`TVirtualFitter`) implemented using new classes
- Feature of new fitting classes:
  - separate Fitter and Minimization interfaces
    - `Fitter` class and `Minimizer` interface (multiple implementations)
    - possible to use (and mix) the various minimization engines
  - decouple fitting from data source (`BinData` and `UnBinData` classes)
  - fit result object which can be stored and retrieved (`FitResult`)
    - keep full result information (parameter, errors, covariance matrix, etc..)
  - minimal and generic function interfaces for model functions (pdf) and objective (minimization) functions
    - easier for user to plug-in their functions (i.e. pdf classes from RooFit)
  - support for parallel fits (usable in a multi-threads environment)

Example of Fitting

- Example of fitting via new classes
  - `BinData` class
    - fillable from ROOT objects (i.e. TH1) or simple data arrays
  - `Fitter` class configurable via the `FitConfig` class
  - Fit model functions in a defined interface (`IParamFunction`)
  - Have also interface for objective functions and used by the minimizer (`IMultiGenFunction`)
  - Produce `FitResult` class
    - keep all fit result information
    - provide methods for retrieving it

```
fitresult.Parameters();
fitresult.Errors();
fitresult.CovMatrix(i,j);
```

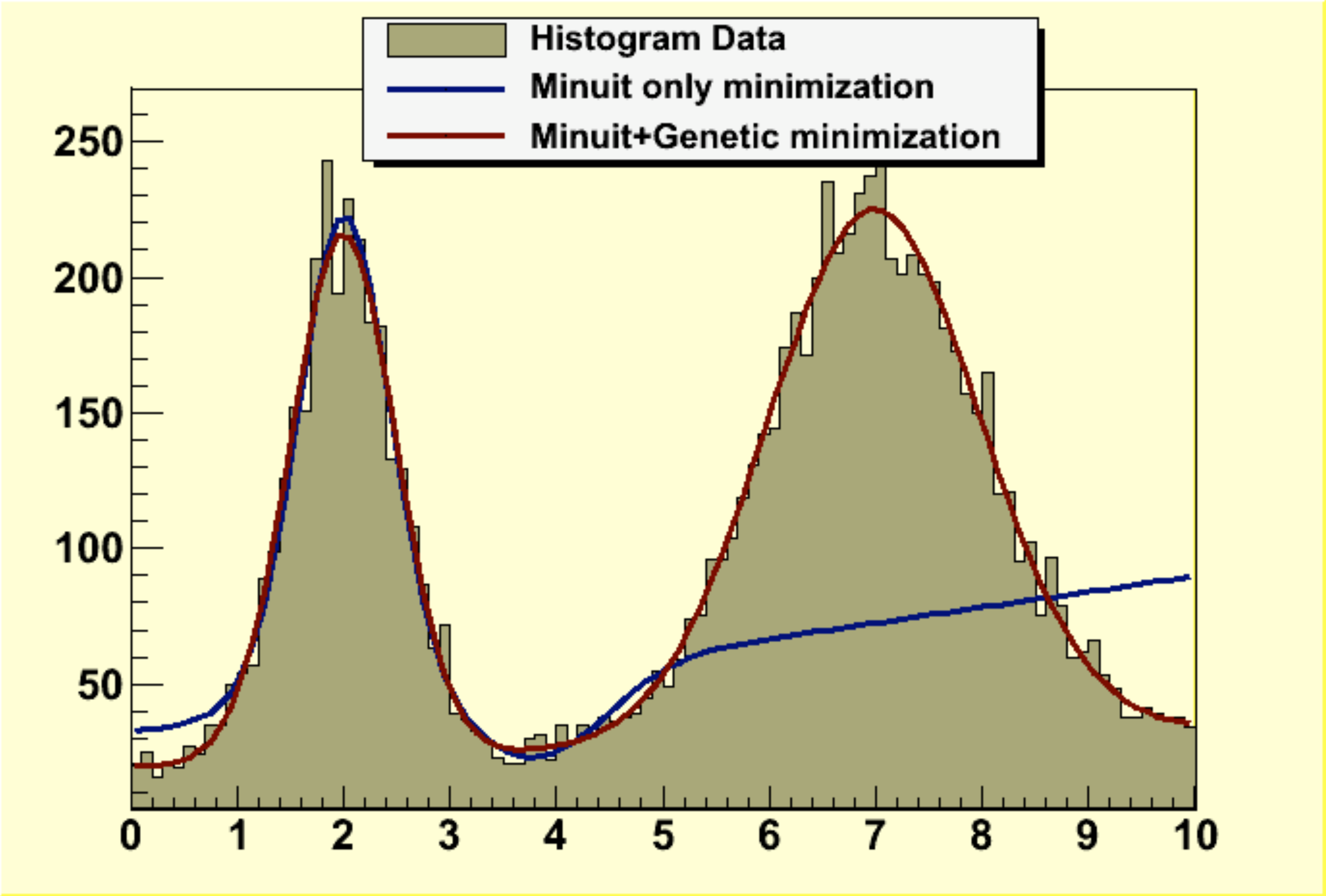
```
// fit inputs
TH1 * h1    = .....
TF1 * f1    = .....

ROOT::Fit::BinData data;
// fill the data set from the histogram
ROOT::Fit::FillData(d,h1);

// create wrapped parametric function for
// requested model function interface
ROOT::Math::WrappedTF1 func(*f1);

// create fitter class
ROOT::Fit::Fitter fitter;

// set minimizer and configuration
fitter.Config().SetMinimizer("Minuit2");

//perform the fit using least square
bool ret = fitter.Fit(data,func);

//retrieve optionally the fit results
if (ret) fitter.Result().Print();

// fit using a user defined objective
// function implementing required
interface
ROOT::Math::IGenFunction mySumSquare(d,f);

ret = fitter.FitFCN(mySumSquare);
```

Function Minimization

- Common interface class (`ROOT::Math::Minimizer`) for all ROOT minimizer implementations.
- Existing currently in ROOT as independent plug-in's:
  - Minuit (based on class `TMinuit`, direct translation from Fortran code)
  - Minuit2 (new C++ implementation with OO design)
  - Fumili (only for least-square or log-likelihood minimizations)
  - GSL minimizers
    - conjugate gradient algorithms (Fletcher-Reeves, BFGS)
    - Levenberg-Marquardt (only for minimizing least square functions)
  - Linear for least square functions (direct solution, non-iterative method)
  - Genetic minimizer (based on algorithm implemented in TMVA)
- Easy to extend and plug-in new Minimizer engines
  - i.e. minimizer from NagC, or Opt++, etc..
- Possible to combine minimizers
- Easy usable outside fitting scope (general optimization problems)

Minimizer Combination

- Example: fit using Minuit in combination with genetic minimizer (used algorithm from *TMVA*)
  - Minuit can be stuck in a local minimum
  - Using in combination with genetic minimizer can find the global minimum



Parallel Minimization

- Prototyped parallelization of MIGRAD algorithm
  - each Migrad iteration consists of:
    - computing function value and gradient
    - computing step by searching for minimum along the Newton direction
    - if satisfactory improve calculation of Hessian matrix, *H*
    - invert to get new matrix $V = H^{-1}$
    - repeat iteration until expected distance from minimum smaller than required tolerance
  - In case of many parameters (> 10) and complex function evaluation, gradient calculation dominates the process:

$$\nabla_i(x) = \frac{\partial f}{\partial x_i} \approx \frac{f(x_i + \delta x_i) - f(x_i - \delta x_i)}{2\delta x_i}$$

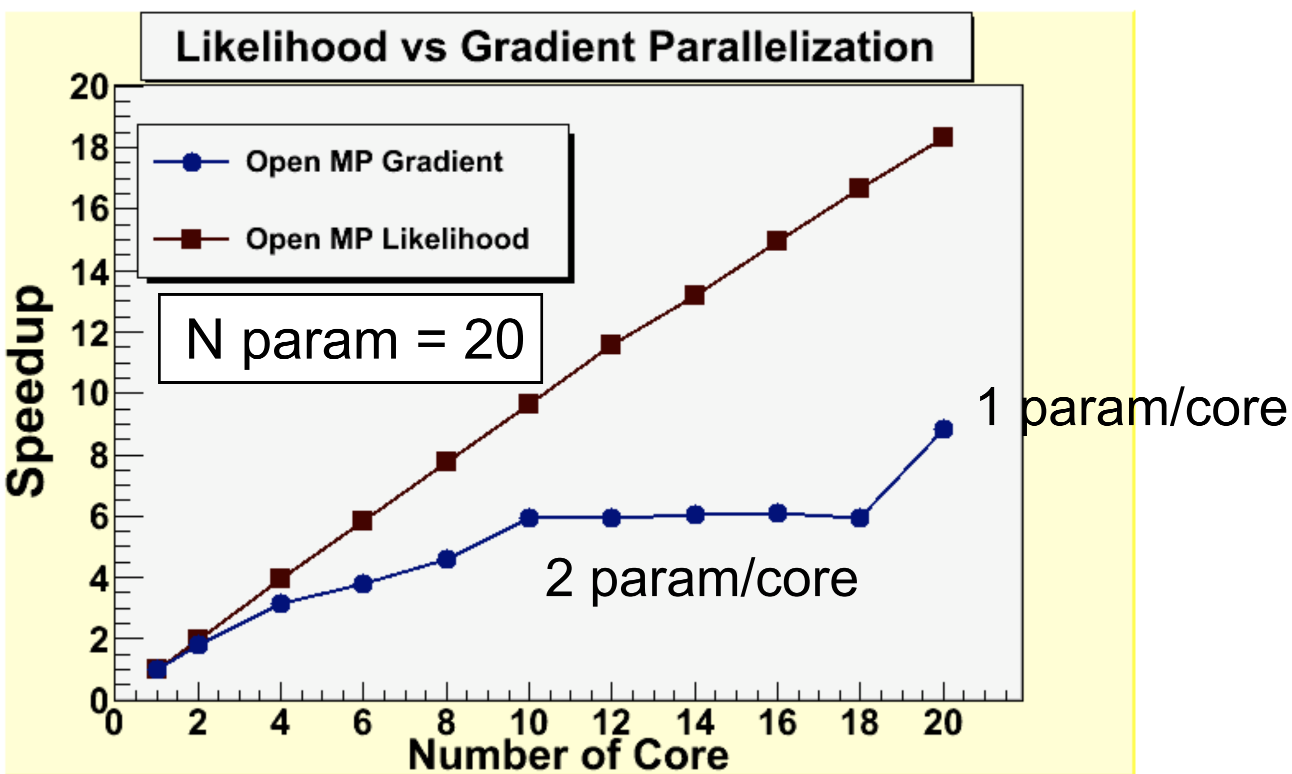
    - al least 2 * NDIM function evaluation are needed
  - Parallelize calculations in Minuit2 by using a thread for each partial derivative
  - Use OpenMP (multi-thread) or MPI (multi-processes) (see Alfio's presentation)
    - distribute in one of the next ROOT release

Parallel Fitting and Minimization

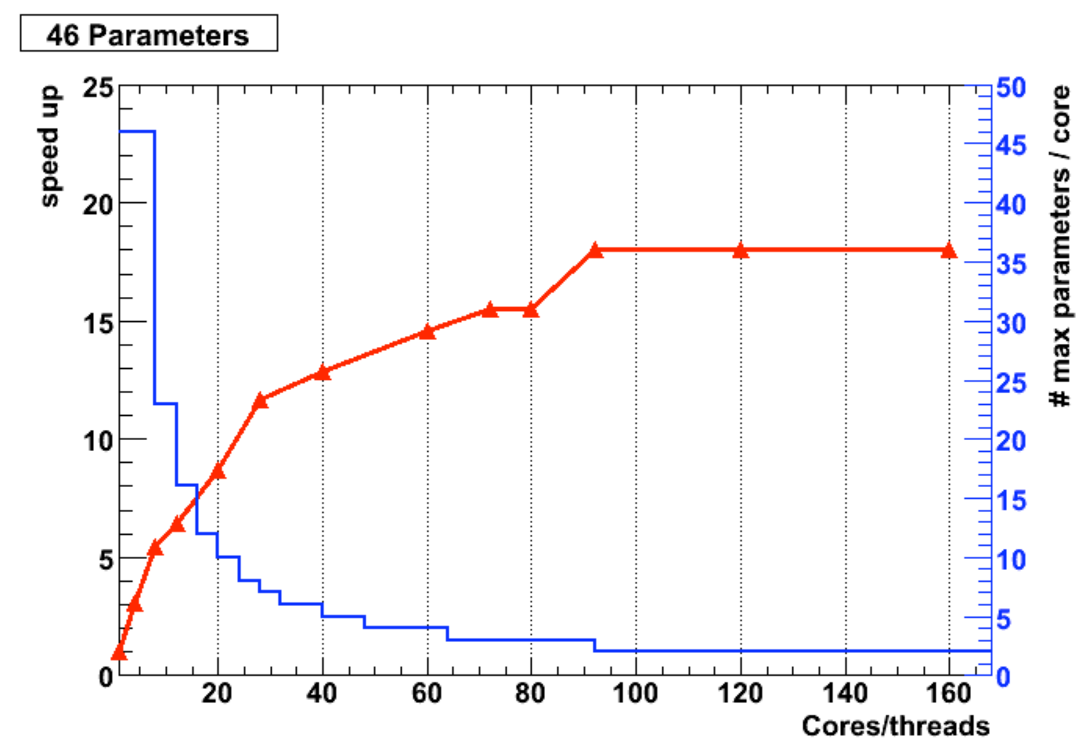
- Minuit parallelization is independent of user code
- Log-likelihood parallelization (splitting the sum) is more efficient
  - more demanding on thread safety of provided code

$$\ln L = \sum_i \ln f(x_i, \theta)$$

- Example: unbinned fit with 20 parameters



complex BaBar fitting provided by A. Lazzaro and parallelized using MPI



- Can have combination on both
  - parallelization via multi-thread in a multi-core CPU
  - multiple process in a distributed computing environment

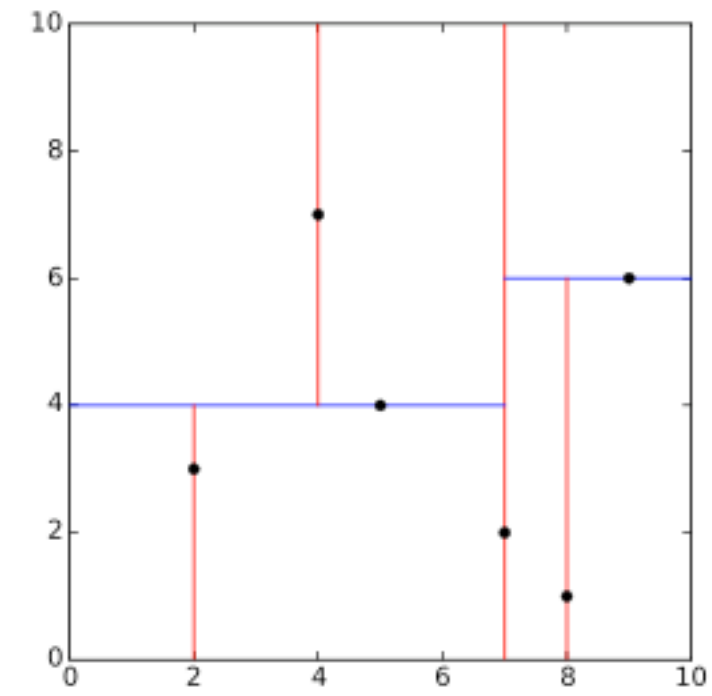
kd-Tree

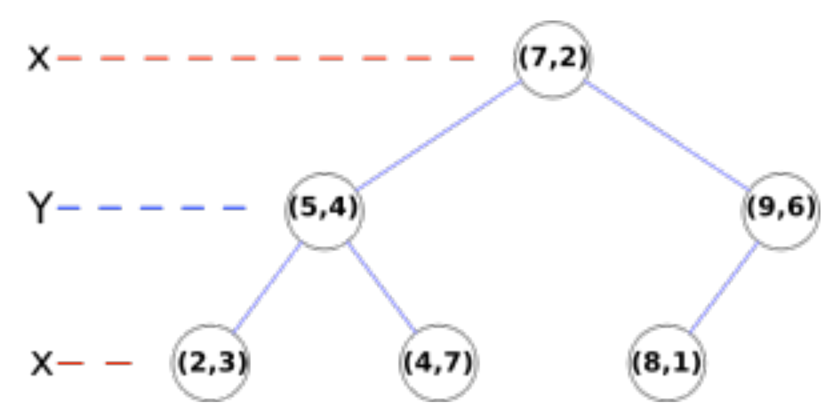
- `kd-tree:` space partitioning data structure for organizing n-dim. points
- Optimized for range or closest point searches

- New class in ROOT for representing kd-tree: `TKDTree`

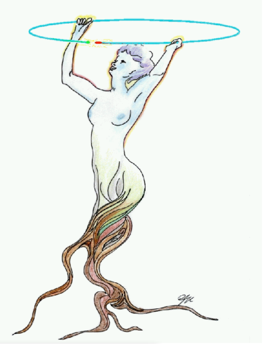
(from M. Ivanov, A. Bercuci and A. Kreshuk)

  - very efficient and optimized implementation to minimize memory consumption
  - can be used for classification of multi-dimensional data (Nearest Neighbor algorithm)
  - also for density estimation (combined with kernel methods) and non-parametetric regression





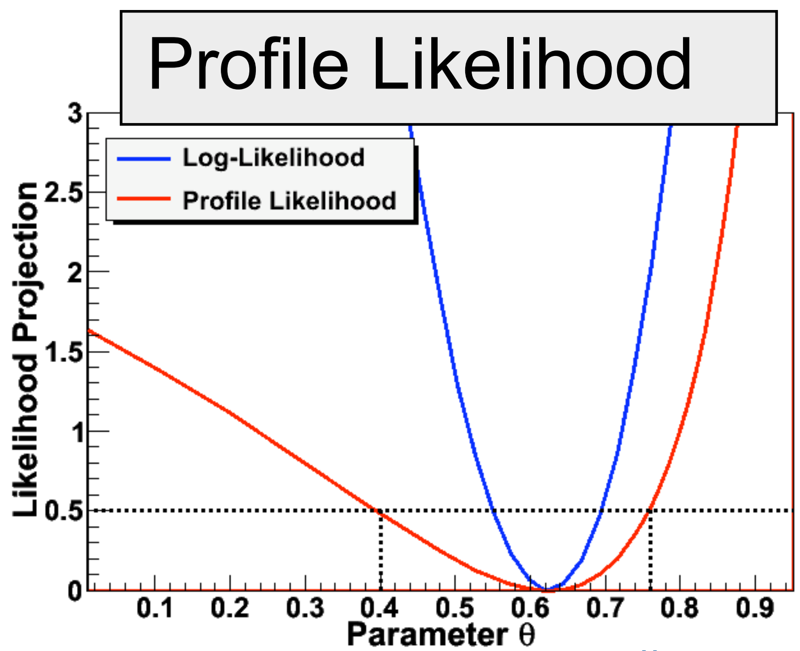
RooStats

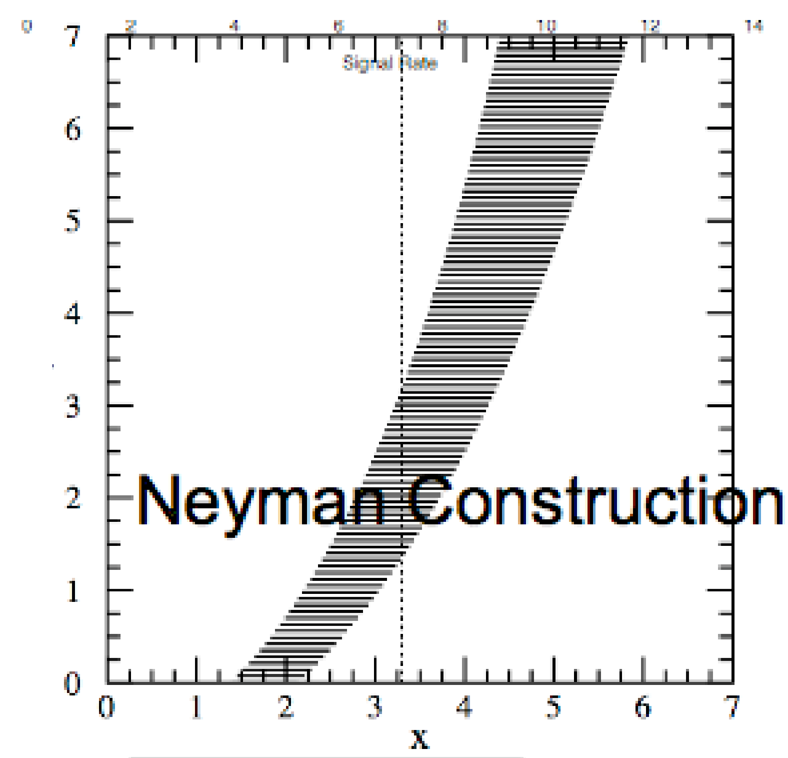


- New statistical framework with tools aimed for discoveries
  - significance, limit, confidence levels
  - common framework for calculations using various tools and techniques
    - convenient for comparing different methods
  - provide possibility of combination of results of multiple measurements
  - support for both frequentists and bayesian statistical tools
    - also within these classes several techniques exist
    - useful for comparing differing methods
      - study coverage, incorporation of systematic errors, etc..
  - all methods basically start with probability density function (pdf) and/or the likelihood function
- Contributions from both ATLAS and CMS

RooStat Methods

- Natural to build on top of *RooFit* toolkit
  - *RooFit* modeling of is very convenient
    - no static notion of parameter and observable
    - can work naturally with both frequentist and bayesian methods
- Major statistical techniques aimed to be in *RooStat* for limits, confidence interval calculations and hypothesis testing:
  - Profile likelihood ratio
  - Neyman constructions
  - Bayesian posterior





$$L(b|Y) = \frac{L(Y|b)L(b)}{L(Y)}$$

Bayes Theorem

$$\ell = \frac{L(x|\theta_{r0}, \hat{\hat{\theta}}_s)}{L(x|\hat{\theta}_r, \hat{\theta}_s)}$$

Workspace Concept

- Maintain a complete description of all the model
  - with possiblity to save entire model in a ROOT file
- New WorkSpace class in RooFit (`RooWorkSpace`)
  - probability density functions
  - (multiple) data sets
  - model configuration
- All information will be available for further analysis and combination of results
  - full likelihood for bayesian analysis
  - probability density functions for frequentists analysis
- Common format for combining and sharing physics results

Conclusions

- Large collection of math and statistical tools available in ROOT
  - working recently on improving the overall quality and better usability
  - improving modularity and trying to remove duplications
    - example: improved fitting and minimization classes
- Considerable efforts from external contributors in developing tools for physics analysis
  - complex fitting (*RooFit*)
  - multivariate analysis (*TMVA*)
  - new statistical framework for discovery, confidence levels and result combination (*RooStats*)
- Important to ensure the correctness of tools we are using
  - large effort on improving validation and test suites
  - good documentation for reference and maintainability
- Need continuously the feedback from users

Documentation

- Online reference documentation
  - class description with *THtml* and *Doxygen*
    - see http://root.cern.ch/root/html520/MATH_Index.html
- ROOT User guides (from http://root.cern.ch/root/doc/RootDoc.html)
  - New Math chapter in the ROOT User Guide (chapter 13)
  - *RooFit* User Guide (new recent version)
  - *TMVA* User Guide
- *Minuit2* User Guide
  - see http://www.cern.ch/mathlibs/documents/minuit/mnusersguide.pdf
- Maintain inventory of all Math functions and algorithms with links to ROOT online doc and CERNLIB
    - http://root.cern.ch/twiki/bin/view/ROOT/MathTable

References

- *ROOT* User Guide: http://root.cern.ch/root/doc/RootDoc.html
- *ROOT* reference guide: http://root.cern.ch/root/htmldoc/ClassIndex.html
- *MathCore* online doc: http://www.cern.ch/mathlibs/sw/MathCore/html/index.html
- *MathMore* online doc: http://www.cern.ch/mathlibs/sw/MathMore/html/index.html
- *Minuit2* online doc: http://www.cern.ch/mathlibs/sw/Minuit2/html/index.html
- *RooFit* homepage: http://roofit.sourceforge.net/
- *TMVA* homepage: http://tmva.sourceforge.net/
- Histogram comparison paper: http://arxiv.org/abs/physics/0605123
- *SPlot* paper: http://arxiv.org/abs/physics/0402083
- *UNURAN* homepage: http://statmath.wu-wien.ac.at/unuran/
- ROOT Talk Forum (for support, requests and discussions)
  - a thread is available for only Math and Statistical topics
- ROOT Savannah for reporting bugs