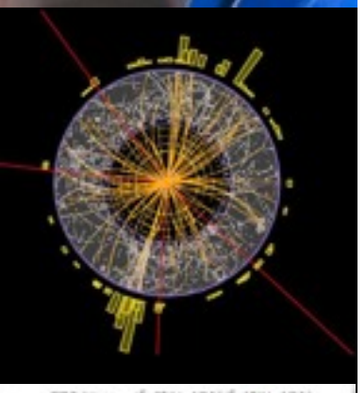
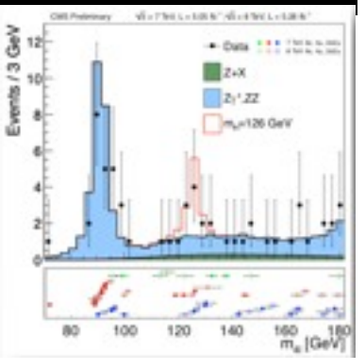
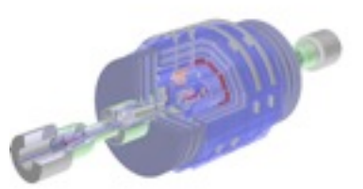




# New Developments in ROOT Math Libraries


*L. Moneta, I. G. Bucur, PH-SFT*

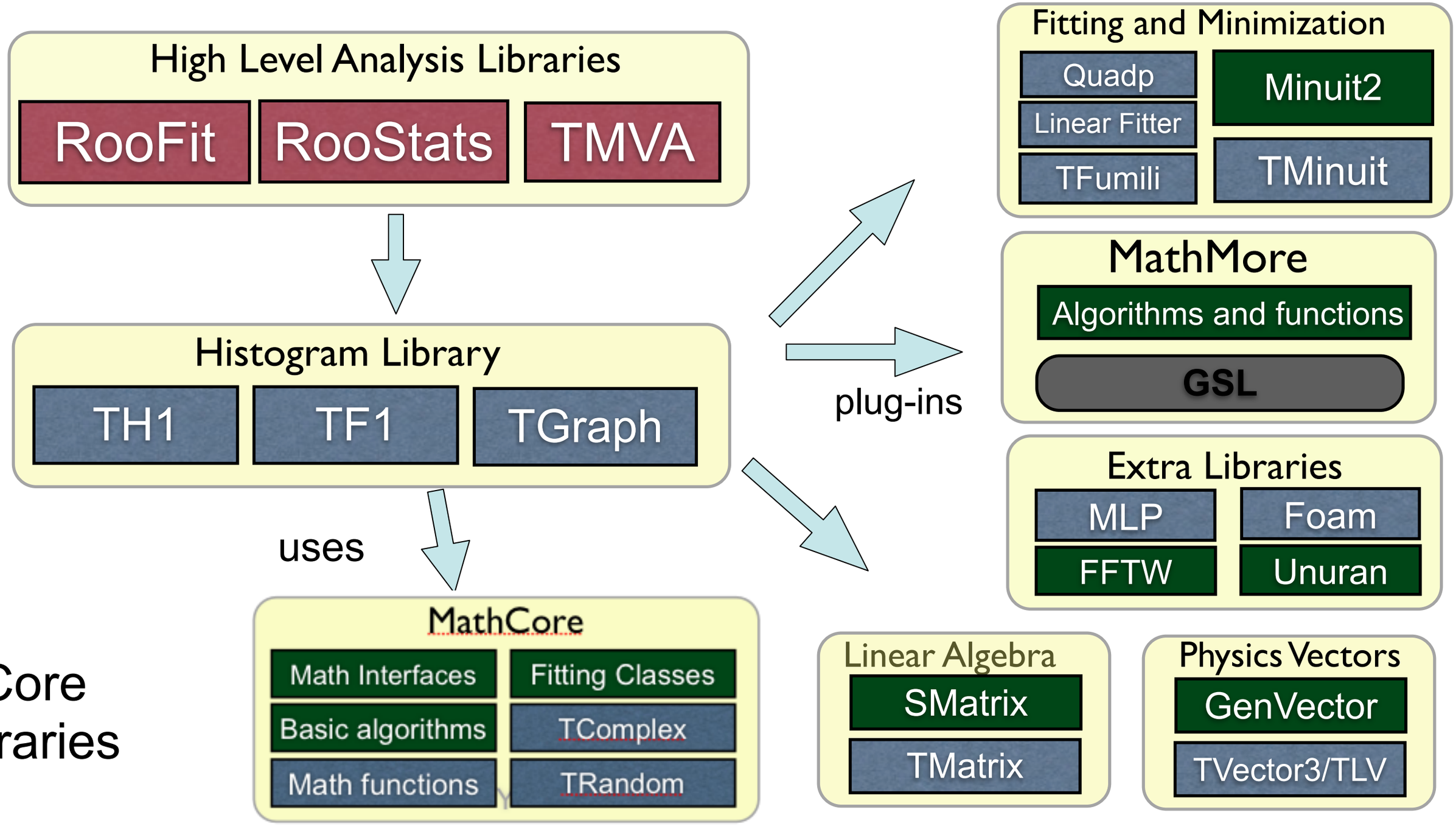
Lorenzo Moneta
root.cern.ch

# Outline

- Introduction
- New developments in core mathematical libraries
  - Fitting and Minimization
- Improvements in Histogram Library
  - new classes (`TEfficiency, TKDE, TKDTreeBinning`)
- Vectorization
  - integration of Vc library in matrix and vector classes
- Future developments
- Conclusions

# ROOT Math/Stat Libraries

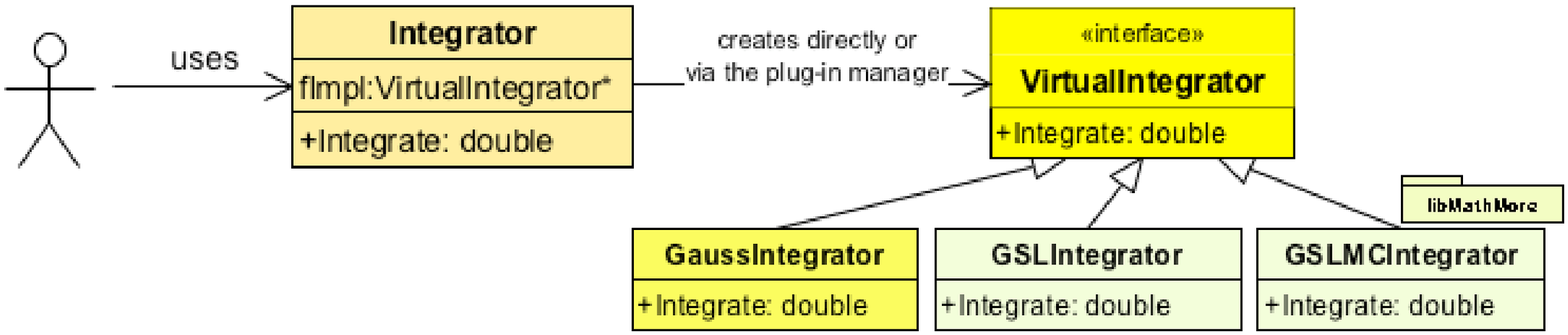
- Large set of mathematical libraries and tools needed for event reconstruction, simulation and statistical data analysis




Core libraries

# Numerical Integration

- Single entry point for multiple implementation:
  **`ROOT::Math::Integrator`**




- Different implementation can be selected at run-time using ROOT plug-in manager
- Integration algorithm and its options can be set using the class `ROOT::Math::IntegratorOneDimOptions` (or `IntegratorMultiDim`)
- Example: this is used in `RooStats::BayesianCalculator`

# New Fitting Classes

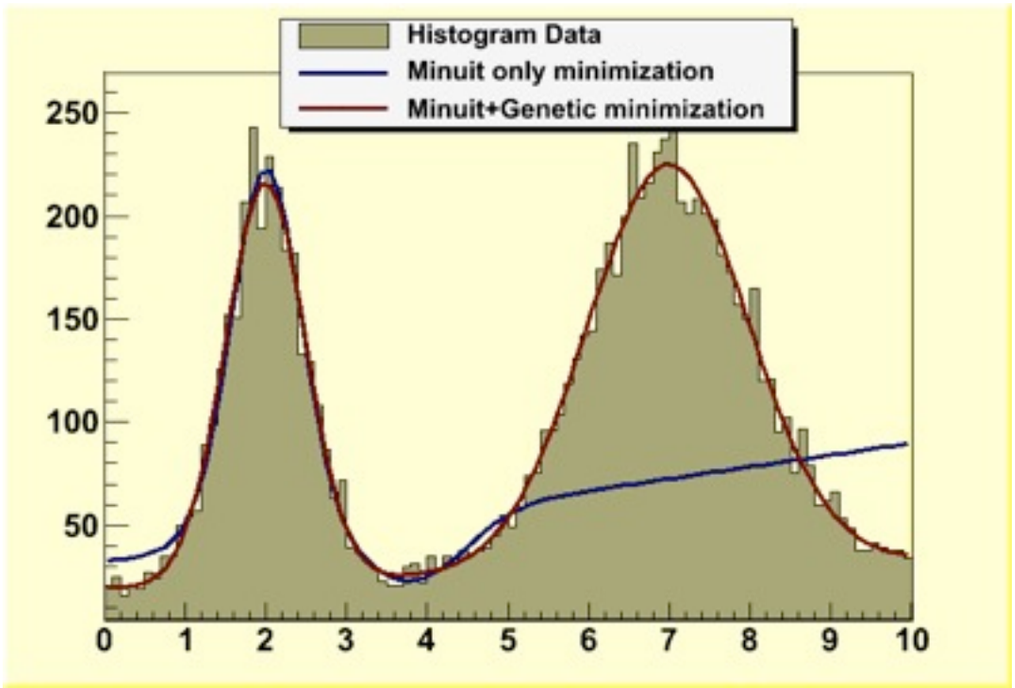
- New fitting classes replacing `TVirtualFitter/TFitter` classes (since v. 5.20)
  - Separate interfaces for fitting and minimization
    - `Fitter` class and `Minimizer` interface (multiple implementations)
    - possible to use (and mix) the various minimization engines
  - Decouple fitting from data source (`BinData` and `UnBinData` classes)
    - same fitting code for all ROOT objects (`TH1,TGraph,TTree,...`)
  - 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)
    - no more static `TVirtualFitter` objects

# Function Minimization

- Common interface class (`ROOT::Math::Minimizer`) for function minimization
- Existing implementations available as plug-ins:
  - **Minuit** (based on class `TMinuit`, direct translation from Fortran code)
    - with Migrad, Simplex, Minimize algorithms
  - **Minuit2** (new C++ implementation with OO design)
    - with Migrad, Simplex, Minimize and Fumili2
  - **Fumili** (only for least-square or log-likelihood minimizations)
  - **GSLMultiMin**: conjugate gradient minimization algorithm from GSL (Fletcher-Reeves, BFGS)
  - **GSLMultiFit**: Levenberg-Marquardt (for minimizing least square functions) from GSL
  - **Linear** for least square functions (direct solution, non-iterative method)
  - **GSLSimAn**: Simulated Annealing from GSL
  - **Genetic**: based on a genetic algorithm implemented in TMVA
- Easy to extend and plug-in new implementations
  - e.g. minimizer based on NagC exists in the development branch
    (see https://root.cern.ch/svn/root/branches/dev/mathDev/math/mathnag)
- See tutorial fit/NumericalMinimization.C on how to use the Minimizer interface

# Function Minimization

- `Minimizer` interface used for fitting in ROOT (by `ROOT::Fit::Fitter`) and also RooFit/RooStats (via class `RooMinimizer`)
- Control of minimization options and type of minimizer using the `ROOT::Math::MinimizerOptions` class
  - to change the minimizer for fitting:
    - `ROOT::Math::MinimizerOptions::SetDefaultMinimizer("Minuit2");`
  - e.g. to change the tolerance:
    - `ROOT::Math::MinimizerOptions::SetDefaultTolerance(1.E-6);`
  - several other options also available: (some specific to the minimizer)
- Possible to combine minimizers
  - e.g. use first Genetic and then Minuit to find the global minimum



# Minuit2

- Object-Oriented version of Minuit  introduced in ver. 5
  - same functionality but with fixes and improvements
    - single side parameter limits
    - added Fumili algorithm
    - better tools to debug minimization with capability to retrieve (if needed) all information at each iteration
      - added in 5.34  a trace object which can be customized by user and produce histograms and graphs to monitor iterations
    - support parallelization in gradient calculation with multi-threads (OpenMP) or multi-process (OpenMPI)
- Used now for complex fits in RooFit/RooStats (e.g. Higgs combination fits with 400 parameters)
  - found to be more robust and able to converge faster (with less function calls)

# Fit Result class

- New **`TFitResult`** class available since 5.26
  - returned from the **`TH1::Fit`** or **`TGraph::Fit`** using **`TFitResultPtr`**
    - need to use option “S” otherwise just the fit status (`int`) is returned
  - **`TFitResult`** contains all fit result information:
    - parameters, error, covariance matrix, Minos errors, minimizer status, etc..

```
// return a smart pointer to TFitResult using option “S”
TFitResultPtr r = h1->Fit("gaus","L S");
double fmin =  r->MinFcnValue(); // minimum of fcn function

const double * par = r->GetParams();  // get fit parameters
const double * err = r->GetErrors();  // get fit errors
TMatrixDSym covMat = r->GetCovarianceMatrix();
TMatrixDSym corMat = r->GetCorrelationMatrix();

r->Print(“V”);  // full printout of result
```

```
****************************************
Minimizer is Minuit / Migrad
MinFCN                =     21.9837
Chi2                  =     32.3656
NDf                   =          47
Edm                   =  1.54911e-11
NCalls                =          63
Constant              =     79.3879   +/-   3.07479
Mean                  =   0.0118002   +/-   0.0317826
Sigma                 =     1.00505   +/-   0.0224758      (limited)
```

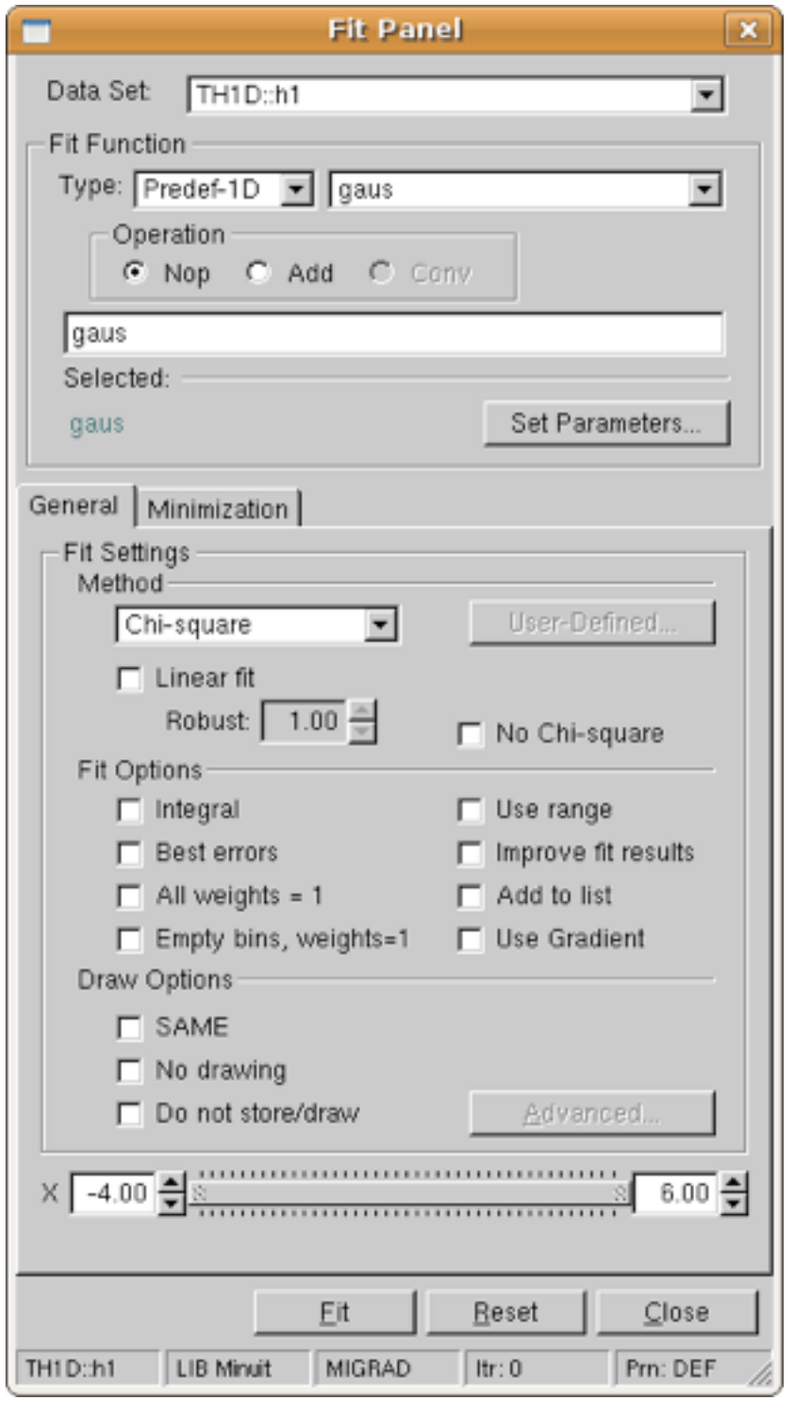
```
Covariance Matrix:
                  Constant       Mean       Sigma
Constant            9.4544  -4.299e-06   -0.039903
Mean             -4.299e-06  0.0010101  5.4604e-08
Sigma             -0.039903  5.4604e-08  0.00050517

Correlation Matrix:
                  Constant       Mean       Sigma
Constant                 1 -4.3991e-05   -0.57739
Mean             -4.3991e-05          1  7.644e-05
Sigma             -0.57739   7.644e-05          1
```

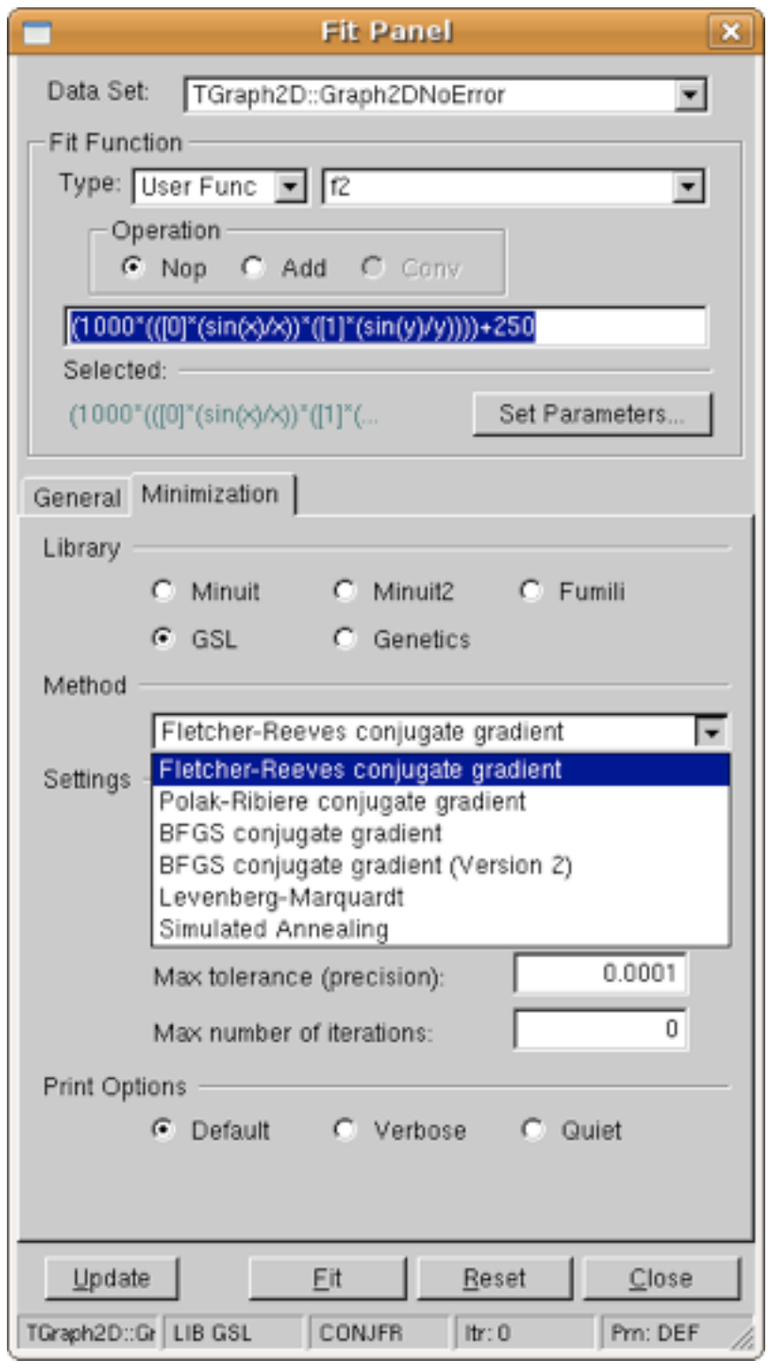
# Other Fitting Improvements

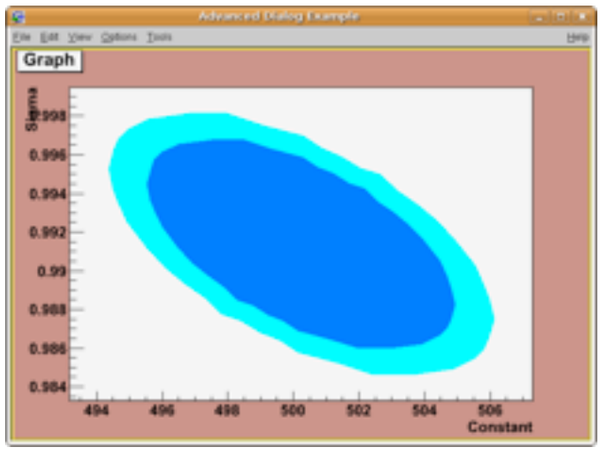
- Support for likelihood fits of weighted histograms (Option `"LW"`)
  - correct error estimation in weighted likelihood fits
- Support for extended un-binned likelihood fits
  - need to deploy as a new option in `TTree::UnbinnedFit`
  - could be used also to fit histogram with buffer (information on all entries is available)
- Support for Pearson Chi-square fit of histograms (Option `"P"`)
  - using expected errors instead of observed ones
- Ability to create complex fitting function objects (`TF1`) using functors
- Add Chebyshev  polynomials to list of pre-defined functions (`cheb0,1,....9` )
  - orthogonal polynomial (better parametrization for fitting)

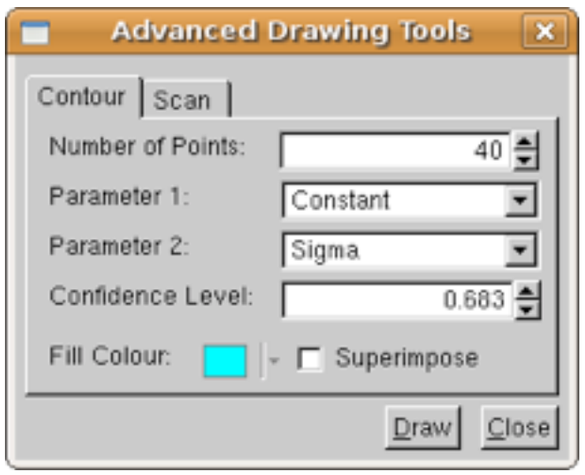
# Fit Panel GUI

- Possible to fit ROOT objects using GUI



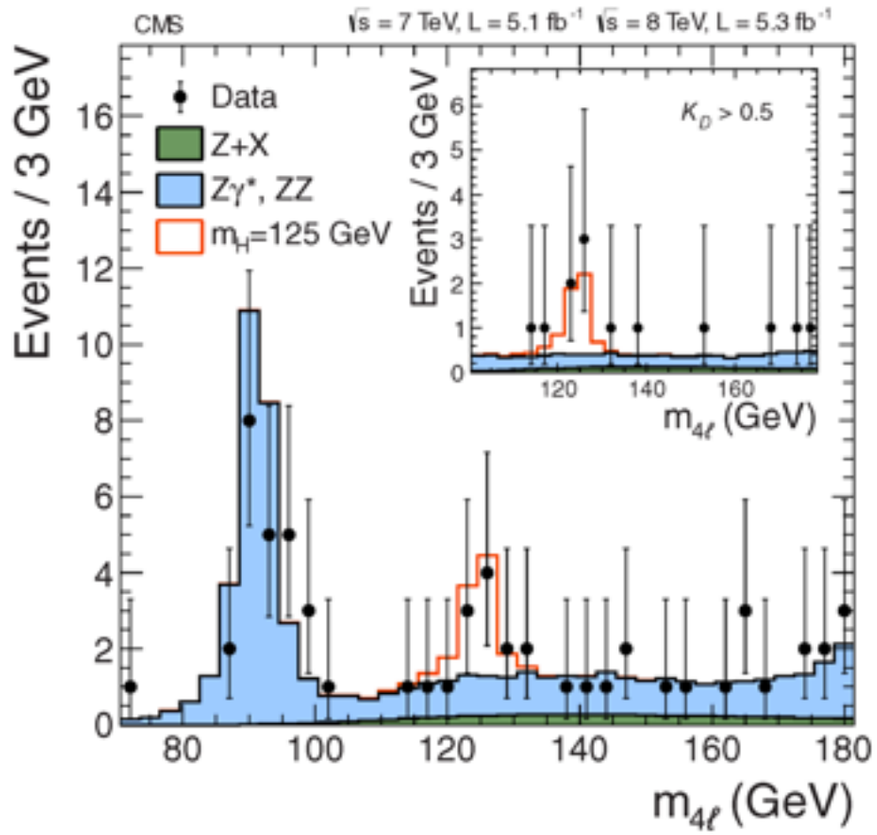



# Improvements in Histogram Library

- Improve histogram classes
  - improved support for weighted histogram
    - support negative weights
    - can perform likelihood fits of weighted histograms
    - automatic enable storage of weight square (Sumw2) when filling with weights (from ROOT ver. 6)
- Ability to plot Poisson errors (asymmetric errors: standard for low statistics)
  - `TH1::GetBinErrorLow` and `TH1::GetBinErrorUp`
  - to enable call `SetBinErrorOption(kPoisson)`

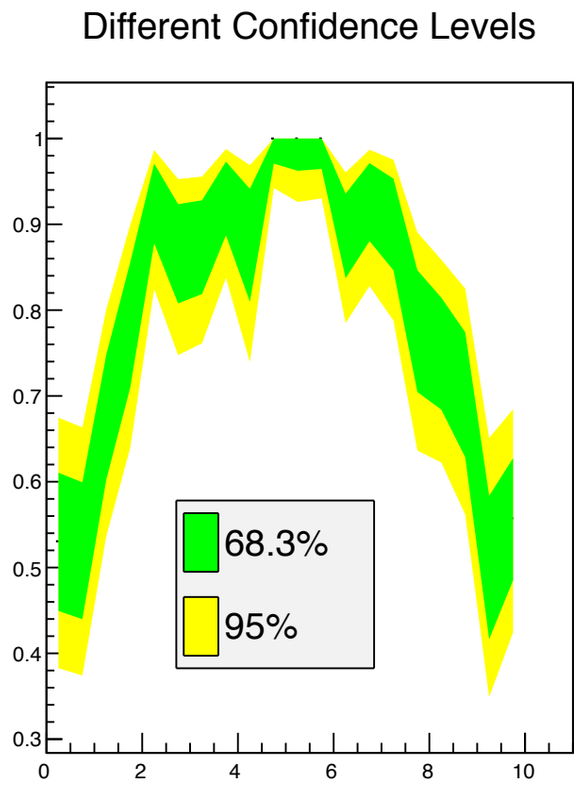


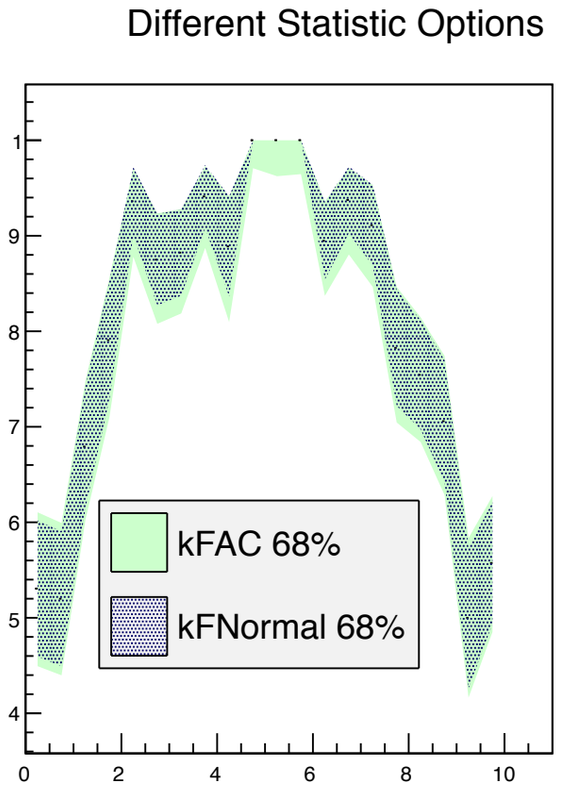
# Histogram Improvements

- Fix in `TAxis::SetRange` to correctly deal with underflow/overflows
- Various fixes in labels histograms
- Support independent axis extensions
- Improve performances of many histogram functions and operations (e.g. `TH1::Merge`)
- Dedicated class for division of histograms (`TEfficiency`) to compute correct statistical errors available since ROOT version 5.28
  - statistics on histogram resulting from division described by Binomial distributions

# TEfficiency

- **`TEfficiency`** provides possibility to estimate and draw intervals at different confidence level and statistics option
- Support also for 2D and 3D
- Can be created from 2 TH1's
- Possible to fill directly **`TEfficiency`**
  - **`eff.Fill(true, x);`**
  - **`eff.Fill(false, x);`**




- **`TEfficiency::Fit`** : binned maximum likelihood fit using a binomial probability for each bin

$$\max L(k_i|N_i,p_i) = \prod_i \frac{n_i!}{(n_i-k_i)!k_i!} f_i^{k_i}(1-f_i)^{n_i-k_i} \quad with \quad f_i = f(\epsilon_i,\vec{p})$$

# New Class for Kernel Estimation

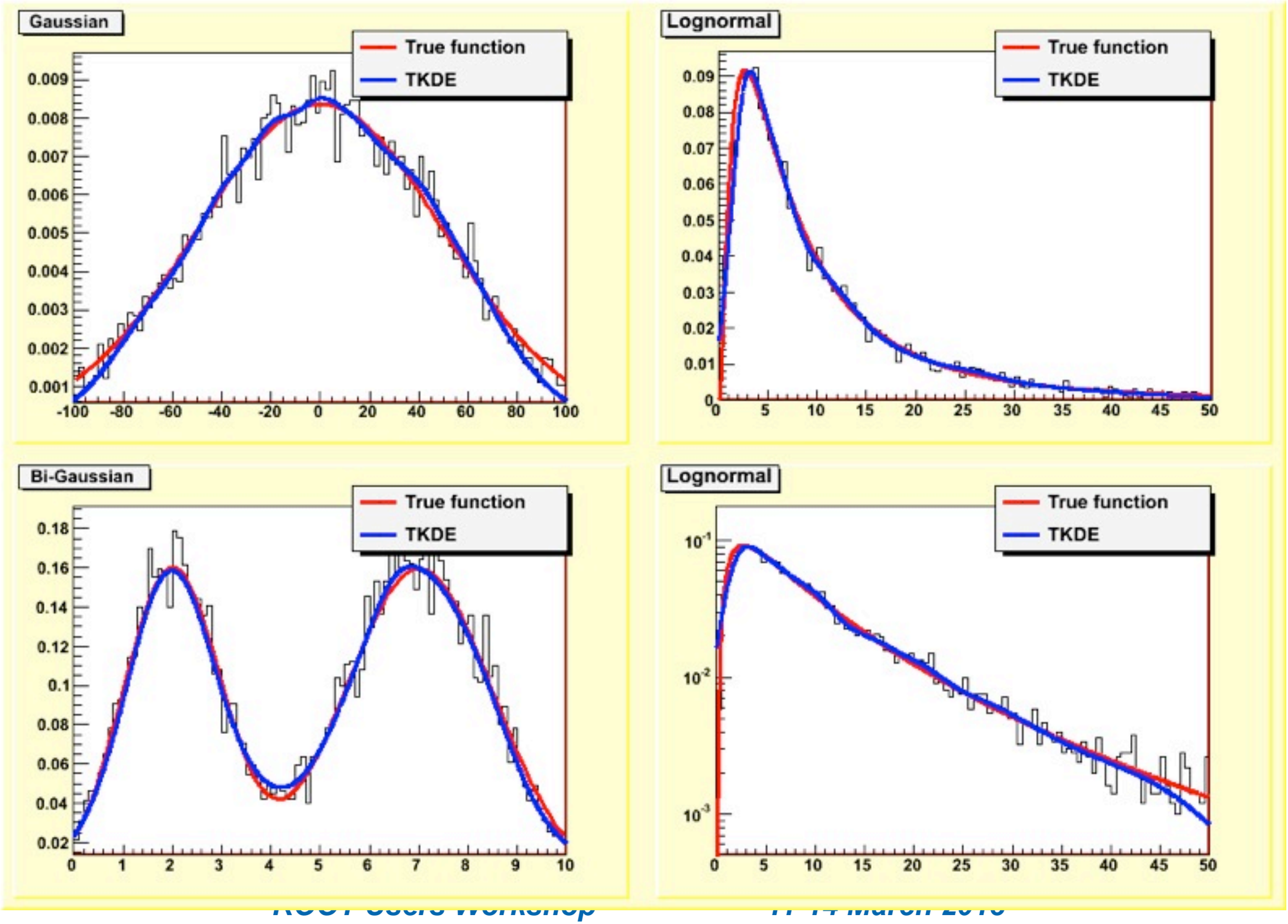
- Class TKDE: non-parametric density estimation using kernels

$$\widehat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

  - support for various kernel (default is Gaussian) but also Epanechnikov, Bi-weight and Arc-cosine kernels
  - support for adaptive bandwidth (better for multi-modal distribution and for describing peaks and tails)
  - can provide both full result or interpolated one for fast evaluation
  - can support data binning for efficient bandwidth computation in the adaptive case
  - can compute error and bias on the estimate density
- Plan to extend to multi-dimensions using *kd-tree* as data storage (`TKDTree`)

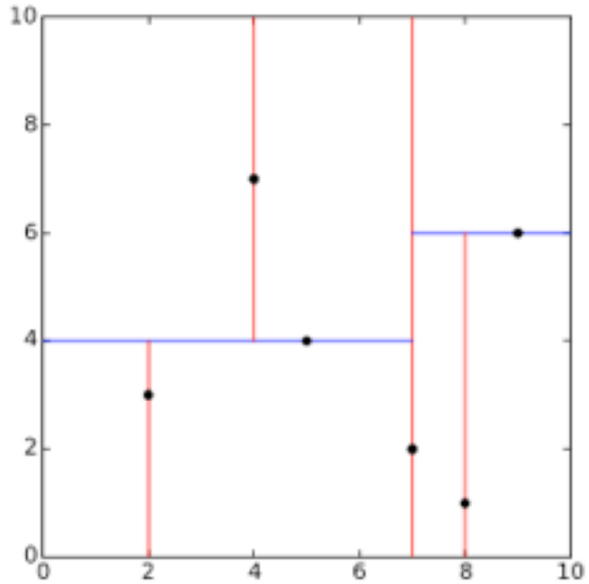
# Examples of TKDE

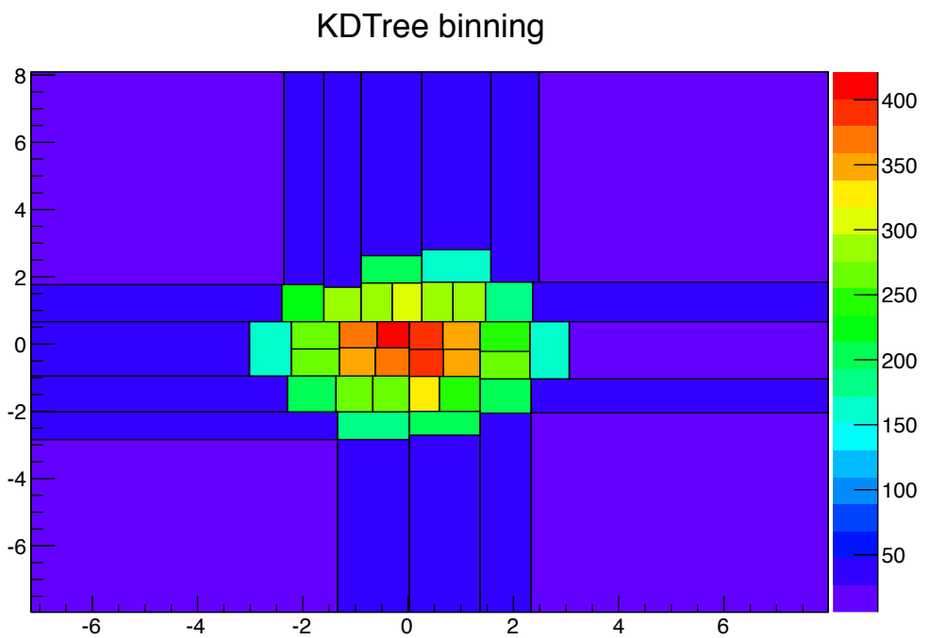
- Example: gaussian, bi-gaussian and log-normal



# TKDTreeBinning

- Class for binning multi-dimensional data using a kd tree (TKDTreeBinning)
  - Automatic binning of data
    - every bin will have same entries or same weight (content)
  - Efficient multi-dimensional histogram
  - Can be used to compare MC and data (make bins with MC and compare with the data)
  - Could be used for non-parametric density estimation using kernels



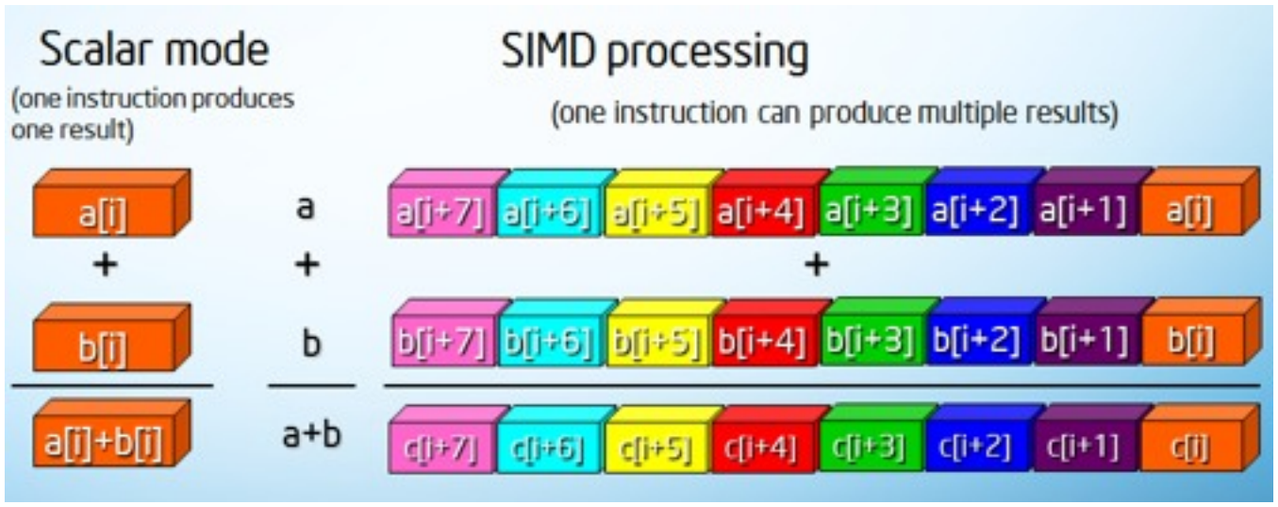



# New Histogram Classes

- **`THnSparse`**
  - multidimensional histograms when only a small fractions of bins are filled
  - memory allocated only for bins which are filled
  - more efficient already than a TH3 when not all-bins are filled
  - implement projections and fitting
- **`THn`**
  - multidimensional histogram when a large fraction of bins are filled
  - use lots of memory (often better to use a TTree)

# Vectorization

- SIMD processing for performing many operation in parallel
  - size of registers depending on architectures
    - SSE : 128 bits : 2 double's or 4 float's
    - AVX:  256bit  : 4 double's or 8 float's)




- Input data must be organized in vectors to perform operations simultaneously
- Automatically compiler can auto-vectorize loops
  - possible if no iteration dependency and no branching
- Alternatively use of intrinsic (but code will look like assembly)

# Vc Library

- C++ wrapper library around intrinsic for using SIMD
  - *developed by M. Kretz (Goethe University Frankfurt)*
  - *used at GSI and BNL, STAR experiment (J. Lauret)*
  - minimal overhead by using template classes and inline functions
- Provides vector classes (**`Vc::float_v, Vc::double_v`**) with semantics as built_in types
  - one can use **`double_v`** as a **`double`**
  - all basic operations between doubles are supported (**+,-,/,***)
  - provides also replacement for math functions

    (**`sqrt, pow, exp, log, sin,...`**)
- Possible to exploit vectorization without using intrinsic and with minimal code changes
  - e.g. replace `double` → `double_v` in functions
  - easy to do in classes or functions templated on the value type
    - e.g ROOT classes in `GenVector` (3D or Lorentz vectors) or in `SMatrix`

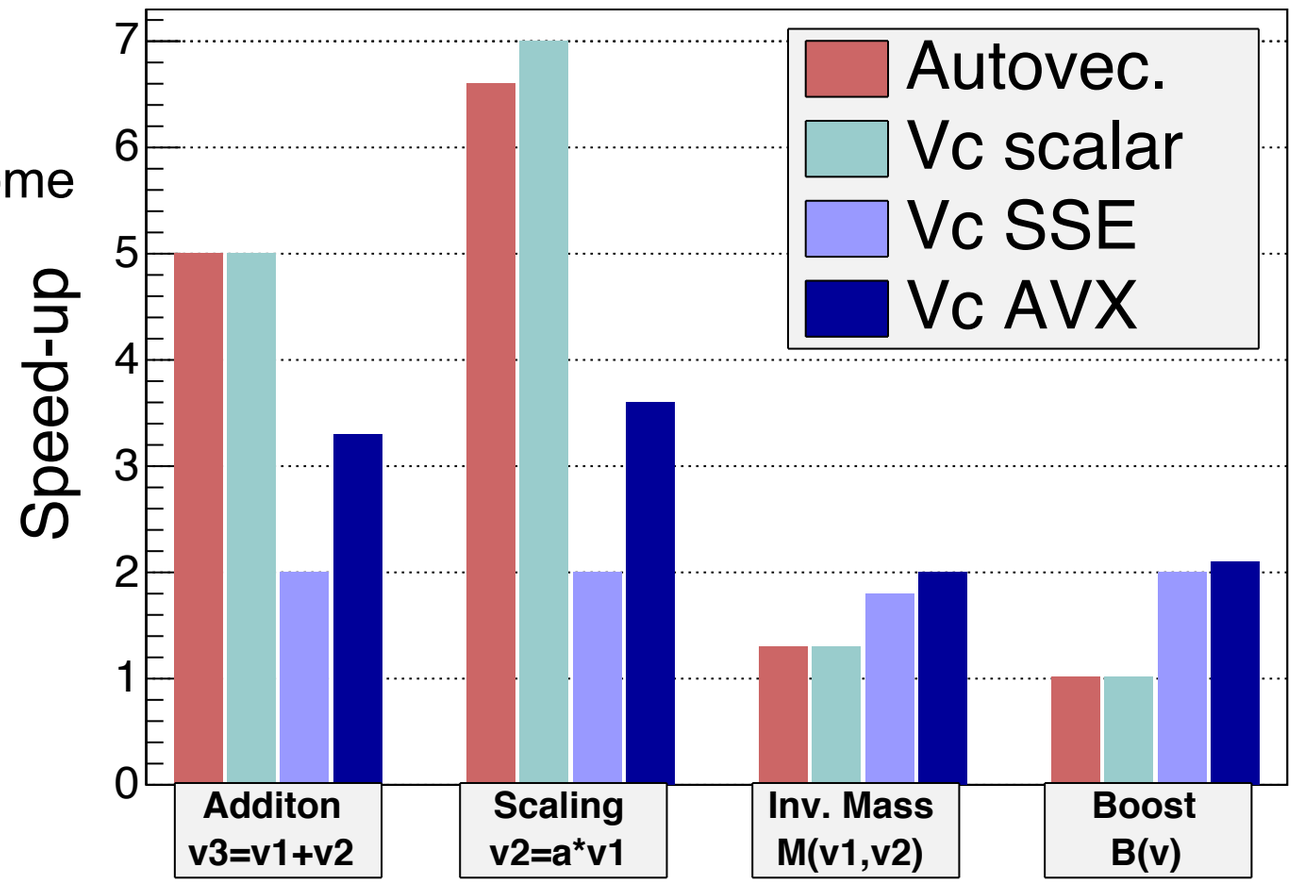
# Evaluation of Vc in ROOT

- Try to use Vc in ROOT to vectorize operation on a list of object (physics vectors, matrices, ...) and not within the object
  - `LorenzVector<PxPyPzE4D<double> >` → `LorenzVector<PxPyPzE4D<Vc::double_v> >`
  - `SMatrix<double, N1, N2 >` → `SMatrix<double_v, N1,N2>`
- Loop on list of objects (vectors, matrices) is reduced by size of `double_v` (`NITER = NITER / double_v::Size`)
  - do not attempt parallelization within objects
- Tested on some basic operation (using double types)
  - Addition of vectors, scaling, invariant mass
  - matrix operations, matrix inversions
  - Kalman filter test
- Test using different compilation flags and Vc implementations
  - `VC_IMPL = Scalar, SSE, AVX`
- Compare results with auto-vectorization
  - compiling using `gcc 4.7.2` with `-mavx -O3 -fast-math`
  - reference is code compiled with `-O2`

# LorentzVector Operations

- Measure speed-up versus scalar version (-O2)

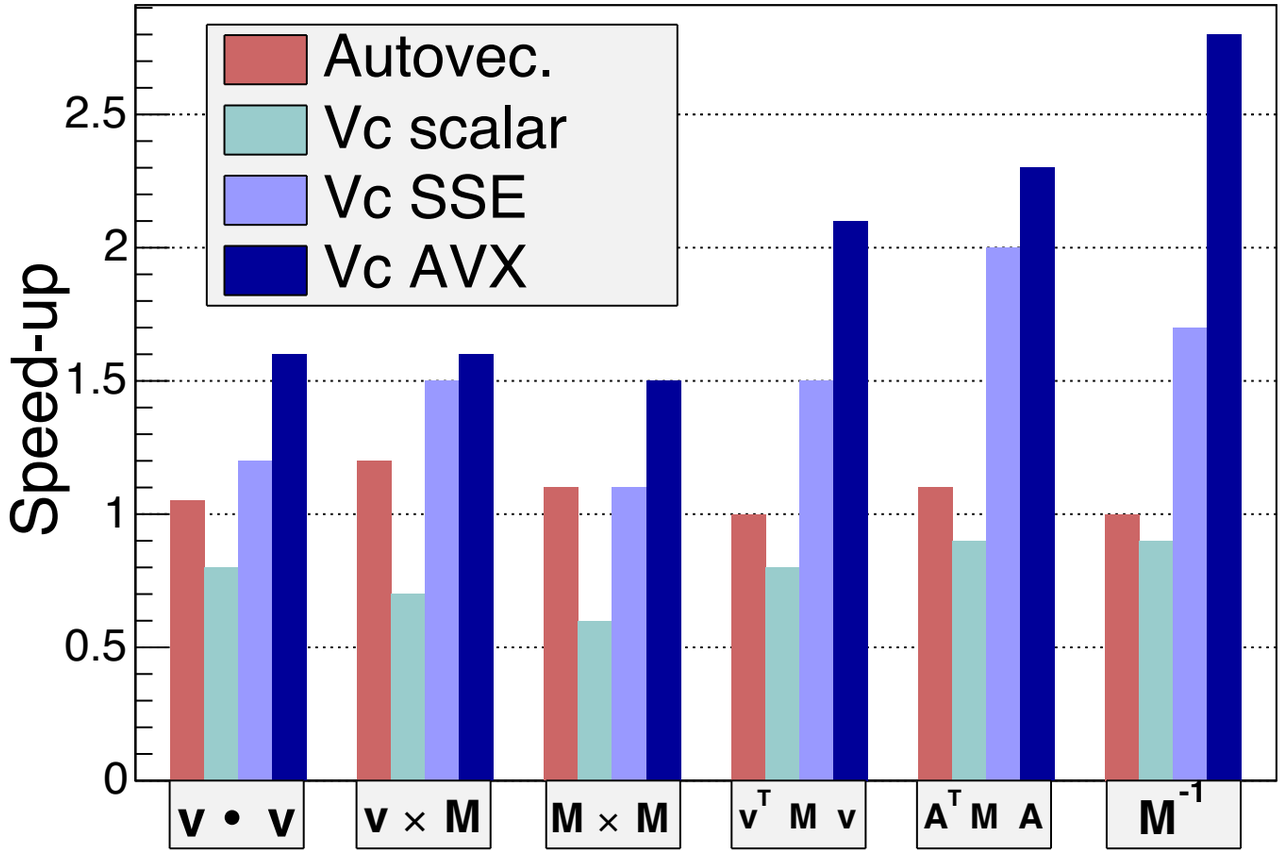
Larger speed-up probably due to some extra compiler optimizations




- Auto-vectorization works well in simple operations, but not for more complex ones (e.g. when one needs to call math. functions)
- Vector lists must be not too large to avoid cache effects (used N=100)

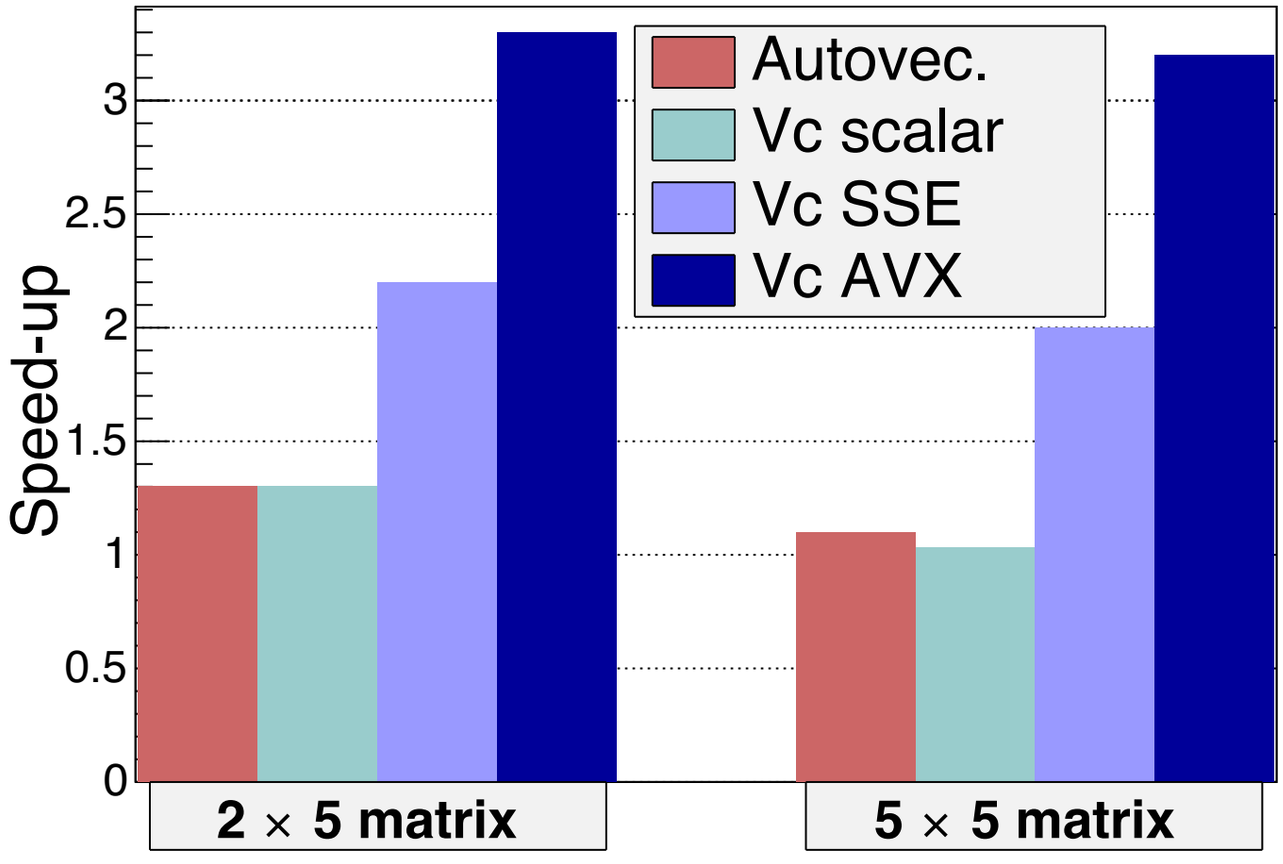
# SMatrix Operations

- Perform operations in SMatrix/SVector using `Vc::double_v` instead of `double`
  - speed-up obtained for processing operations on a list of 100 `SMatrix<double,5,5>` and `SVector<double,5>`



# Kalman Filter Test

- Typical operation in track reconstruction
  - very time consuming
    - inversion + several matrix-vector multiplications




- Clear advantage with Vc
  - SMatrix code can works using `double_v` as value_type
  - good boost in performance in an already performant code (5-10 times faster than CLHEP)

# Vectorization in Fitting

- Vectorize chi-square calculation in fitting ROOT histograms
  - *work performed by M. Borinsky (summer student 2012)*

$$\chi^2 = \sum_i \frac{(y_i - f_{a,b,\ldots}(x_i))^2}{\sigma_i^2}$$

- Required change in data set layout and in functions
  - from array of structure to structure of arrays for input data
  - vectorized function interface (TF1)

```
1 double func( double x, double* p )
{
3     return exp( - p[0] * x );
}
```

Listing 1: Old callback function for TF1

```
void func ( double* x, double* p, double* val )
2 {
      for ( i in range )
4         val[i] = exp( - p[0] * x[i] );
}
```

Listing 2: New vectorizable callback function for TF1

# ROOT Fitting

- Observed performance gain from new data structure and from vectorization using VDT library

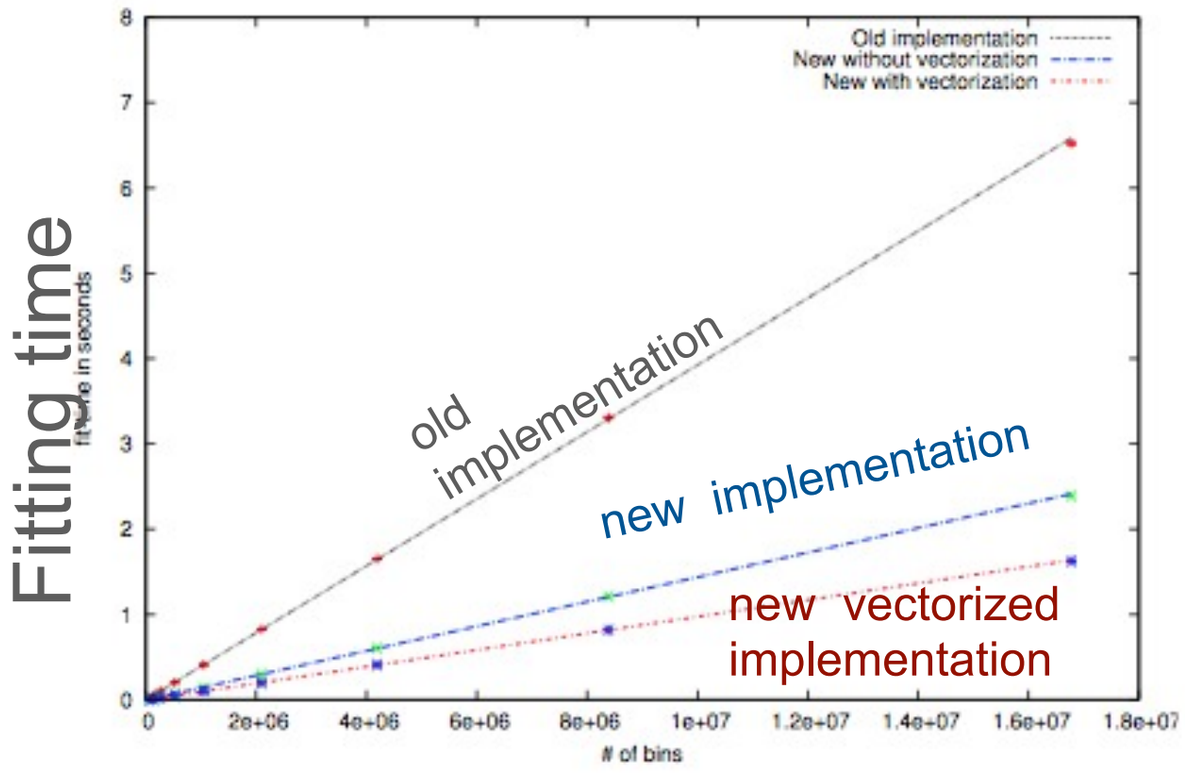



Figure: Performance with and without vectorization

Performance gains on AVX (E5-2690), gcc 4.7

old ⇒ new : 2.7x

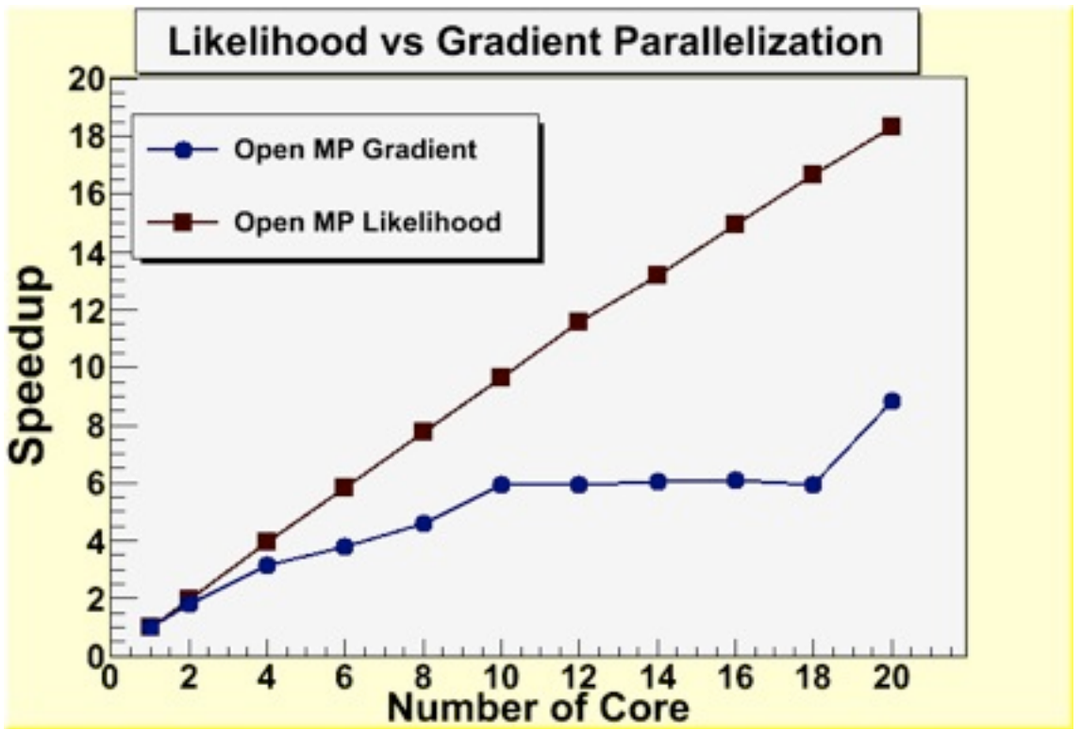
new ⇒ vec: 1.5x

Total speed-up: 4.0x

- Test also using Vc: similar speed-up results (~ 3.5x) but with less code changes
  - would be easy if fit function interfaces are templated

# Fitting Parallelization

- Parallelization in Minimization (Minuit) in the gradient calculation ( independent of user code)
- Log-likelihood parallelization (splitting the sum)
  - more efficient but it is at lower level (often in user provided code)

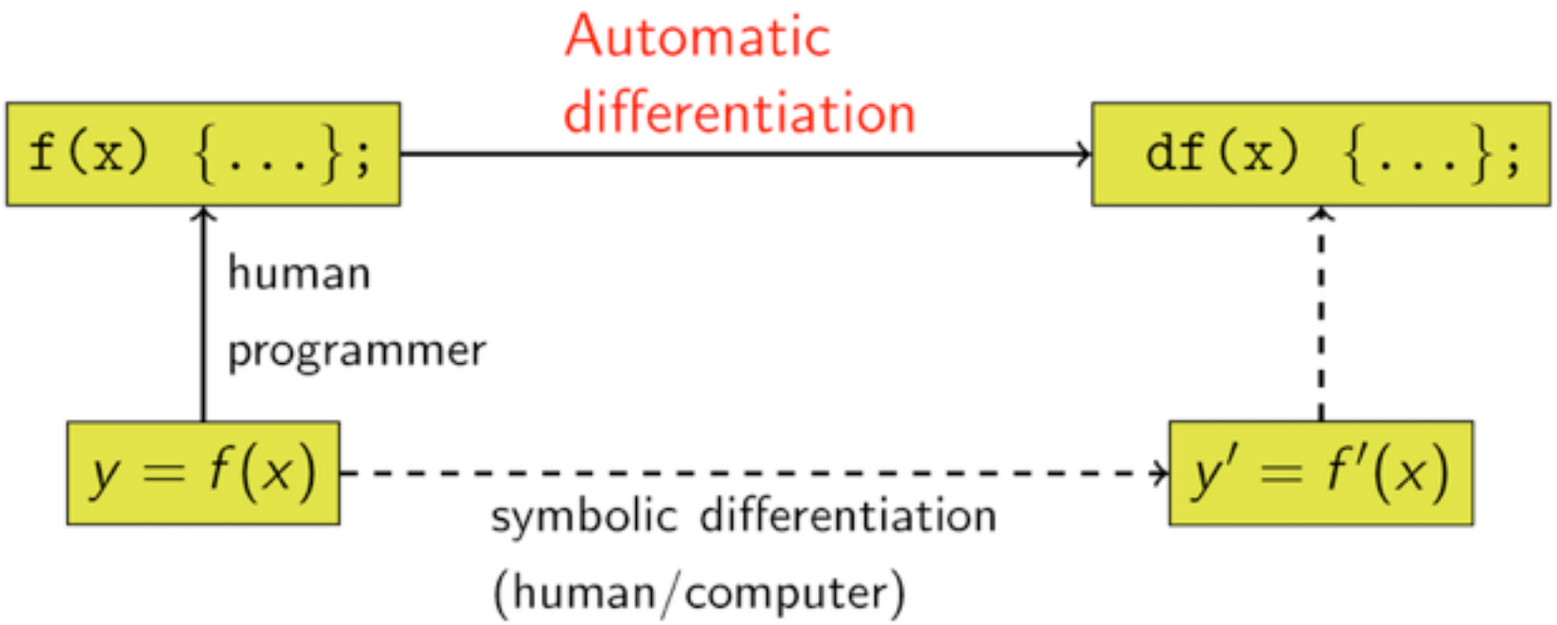



- OpenMP (multi-threads)
- MPI for multi-process in a cluster

Example: unbinned fit with 20 parameters

# Future Developments

- New `TFormula` class based on Cling
- Improve building of complex fit functions:
  - make it easier for fitting
  - add capability of using models from RooFit (saved in workspaces)
  - use C++11 capabilities (e.g. std::function, lambda,...)
- Implement auto-differentiation using Cling/LLVM



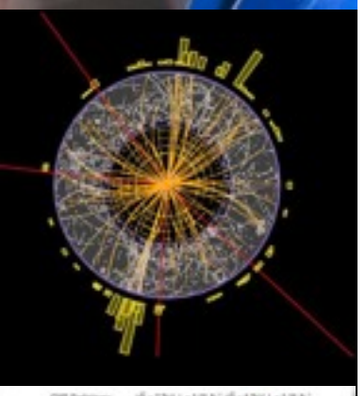
# Future Developments

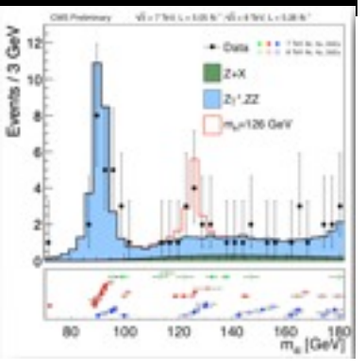
- Minimization:
  - improve support for problems with large number of parameters (case of sparse hessian matrix)
  - constraint minimization
- Deploy multi-threads for fitting
  - need to decide on the technology (OpenMP / Intel TBB)
  - investigate also parallel numerical integration
- Improve histogram and function (TF1) classes
  - investigate having a better inheritance hierarchy
  - need to evaluate if possible and how to maintain backward compatibility
    - TH1 is the most used class in ROOT

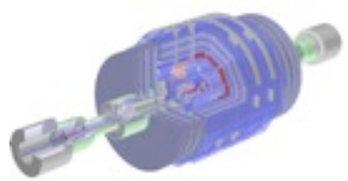
# Conclusions

- Large collection of math and stat tools available in ROOT
  - improved overall quality
    - more tests, studied and improved performance whenever possible
  - improving modularity
    - common interfaces for functions and algorithms
  - improve usability (e.g. new classes like `TFitResult`)
  - new classes useful for data analysis (`TEfficiency, TKDE )`
  - need to update now the user documentation
- Investigating parallelization and vectorization of Math libraries
  - deploy vectorization using Vc for matrix and vector classes
  - implement multi-threads for fitting
- Developed advanced tools for physics analysis
  - complex fitting (*RooFit*)
  - multivariate analysis (*TMVA*)
  - framework for statistical calculations (*RooStats*)

➡*see separate presentations*





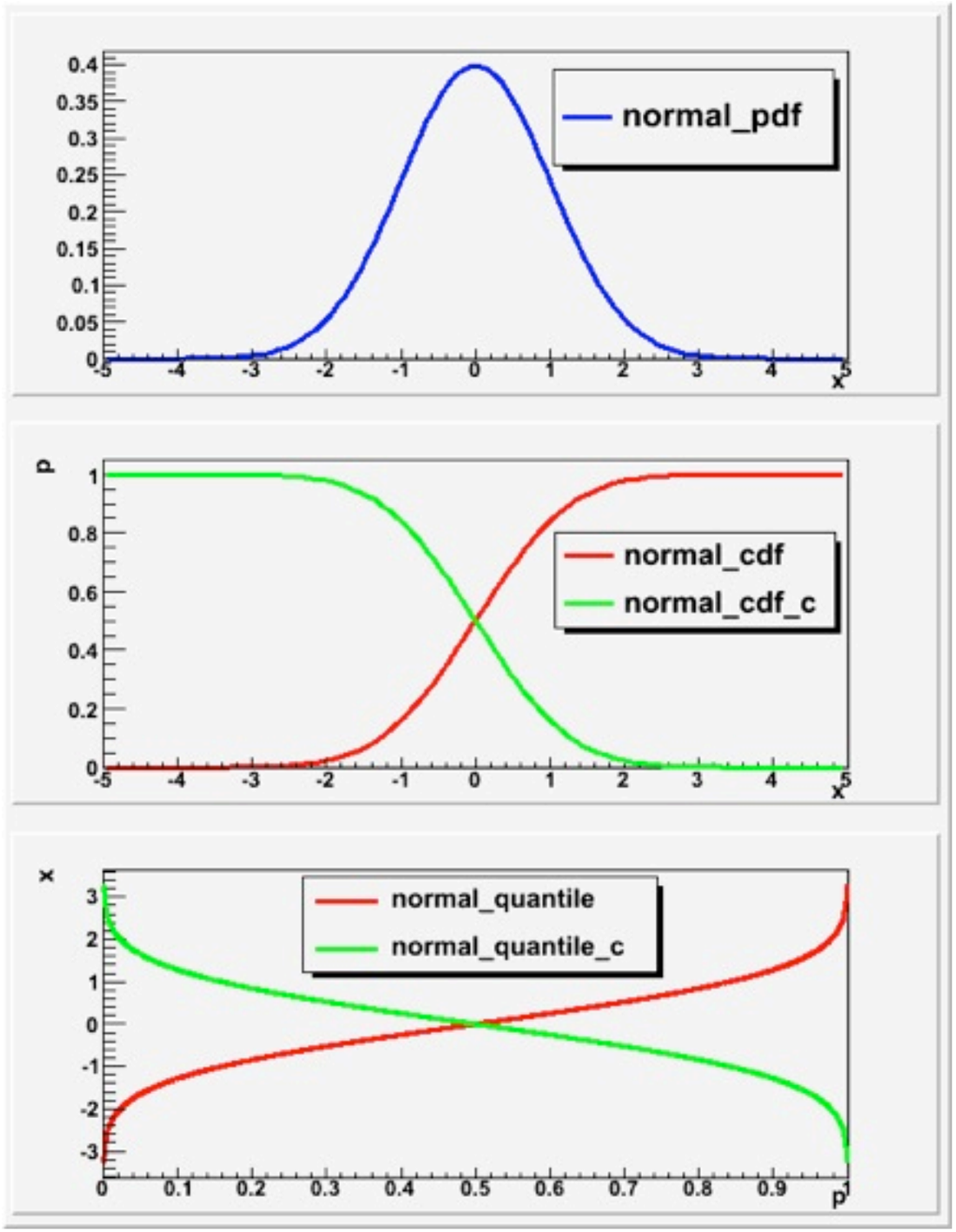



# Back-up Slides

- Statistical Functions in ROOT Math
- Description of numerical algorithm
- Extra classes introduced recently in ROOT
  - `ROOT::Math::DistSampler`
  - `ROOT::Math::GoFTest`
- More on parallelization and vectorization
- Documentation

# ROOT::Math Statistical Functions

- Statistical Functions are provided in a coherent naming scheme
  - probability density functions (pdf)
    - i.e. normal distribution:
      - `normal_pdf( x, sigma, mu)`
  - cumulative distributions (cdf)
    - lower tail: `normal_cdf (x, sigma, mu)`
    - upper tail: `normal_cdf_c (x, sigma, mu)`
  - inverse of cumulative distributions (quantiles)
    - inverse of lower cumulative
      - `normal_quantile (z, sigma)`
    - inverse of lower cumulative
      - `normal_quantile_c (z, sigma)`
- All major statistical distributions available
  - *normal, lognormal, Landau, Cauchy, $\chi^2$, gamma, beta, F, t, poisson, binomial, etc..*
- Defined as free functions in `ROOT::Math` namespace



# Numerical Algorithms in Math Libraries

- Numerical algorithms (in the *MathCore*, *MathMore* or other libraries):
  - **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 (e.g. Brent) and polishing algorithms using derivatives
  - **Minimization (one-dim)**
    - Golden section and Brent algorithm for 1D
  - **Minimization (Multi-dim)**
    - Minuit and Minuit2 (with Migrad and Simplex)
    - conjugate gradient and BFGS algorithms (from GSL)
    - simulated annealing and genetic algorithms
    - Fumili and Fumili2
    - non linear least square solver with Levenberg-Marquardt algorithm
  - **Interpolation**
    - linear, polynomial, cubic and Akima spline

# 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);
```

# DistSampler class

- New interface class in version 5.28 for random generation of  data according to a generic distribution
  - implemented using UNU.RAN and Foam

```
using namespace ROOT::Math;
....
DistSampler * sampler = Factory::CreateDistSampler("Unuran");
// set the sampling distribution
sampler->SetFunction(user_function);
// init with algorithm name
sampler->Init("TDR");
for (int i = 0; i< n;++i) {
   // sample 1D data
   double x = sampler->Sample1D();
   // sample for multi-dimensional data
   const double * xx = sampler->Sample()
   .......
}
```

    - can also generate directly a data sets (binned or unbinned)
  - plan to use it in RooFit for implementing **`RooAbsPdf::generate`**

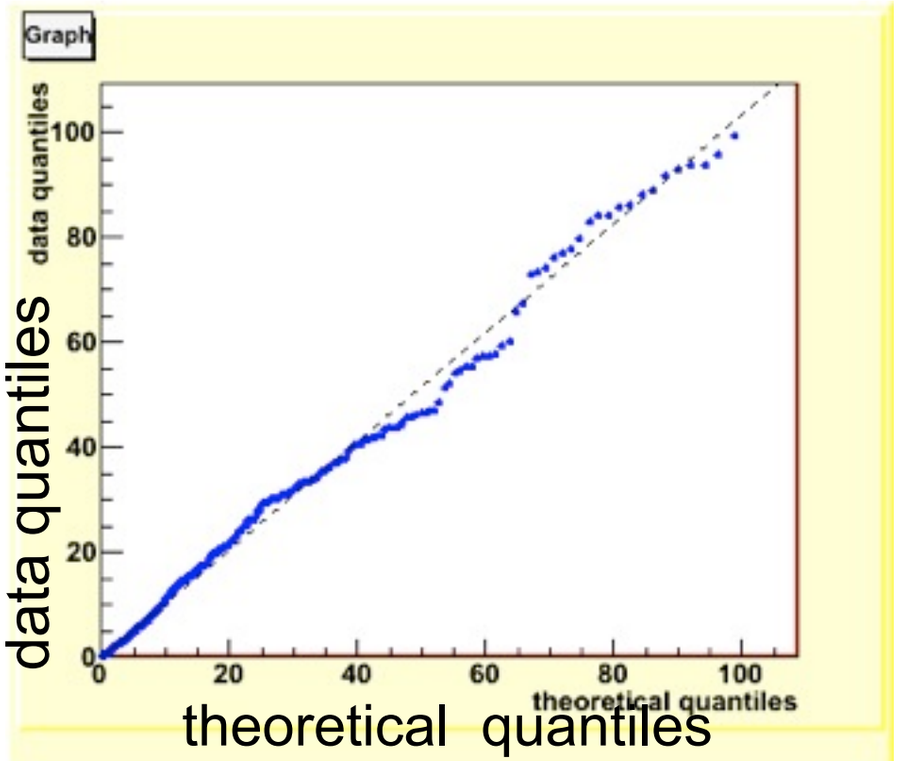
# New GoF Test Class

- New class for goodness of fit tests:
  **`ROOT::Math::GoFTest`** in *libMathCore*
  - 1-sample test
    - test if data are compatible with a reference distribution
    - user provided distributions or standard ones (normal, log-normal,etc..)
  - 2-sample test
    - test if two data sets are compatible
  - working on un-bin data sets
    - we have already the Pearson $\chi^2$ test on the bin data sets (histograms)
  - Kolmogorov-Smirnov test
    - was already existing in ROOT for the 2-sample and bin data
    - add 1 sample test
  - Anderson-Darling test
    - much more sensitive to detect tails variation

# Example of using GoFTest

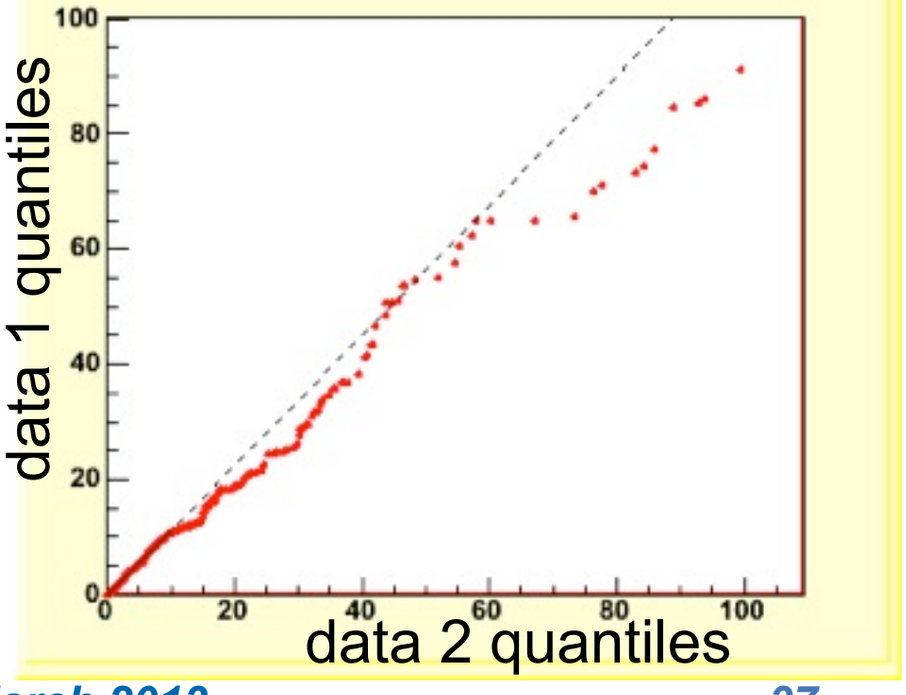
- 1 sample test

```
using namespace ROOT::Math;
// create gof test class on data x[n]={....}
GoFTest  gof(n,x,GoFTest::kLogNormal);
// set a user distribution object
// which must implement operator ()(x)
gof.SetUserDistribution(user_dist);

double pValueAD = gof.AndersonDarlingTest();
double pValueKS = gof.KolmogorovSmirnovTest();
```




- 2 sample test

```
// create GoF test for  data x1[n1] and x2[n2]
GoFTest  gof2(n1,x1,n2,x2);

double pValueAD = gof2.AndersonDarling2SamplesTest();
double pValueKS = gof2.KolmogorovSmirnov2SamplesTest();
```



# Vectorization of Mathematical Functions

- Compare performances in evaluating Math Functions
- Use also VDT Mathematical library (*by D. Piparo*)
  - transcendental mathematical functions which can be auto-vectorized
  - but require a different interface: `std::sin(double x)`
    ⇒ `void vdt::fast_sinv(int n,const double *x, double *r)`

| Speed-up | auto-vect. -Ofast -mavx | Vc scalar (auto-vec.) | Vc SSE | Vc AVX | VDT AVX |
|---|---|---|---|---|---|
| sqrt (x) | **2.4** | 2.4 | 2.3 | **2.4** | **2.4** |
| exp(x) | **1.0** | 1.0 | 2.1 | **4.9** | **4.1** |
| log(x) | **1.0** | 1.0 | 3.8 | **4.9** | **5.4** |
| sin(x) | **1.0** | 1.0 | 0.4 | **1.2** | **1.6** |
| atan(x) | **1.0** | 1.0 | 1.5 | **1.3** | **1.6** |

# Parallelization in Histogram Operations

- Speed-up operation on ROOT histograms like scaling or merging (add) using multi-threads (openMP)
  - good speed-up observed
  - degradation for large number of bins due to cache effects
- Improve also performances by using more efficient serial code in several other histogram functions *(work by I.G. Bucur)*

# Documentation

- Online reference documentation (most up-to date)
  - class description with *THtml* (and also *Doxygen)*
    - see http://root.cern.ch/root/htmldoc/MATH_Index.html
      - see TEfficiency doc as example of a very well documented class
- Math library documentation on Drupal
  - see http://root.cern.ch/drupal/content/mathematical-libraries
  - document most of the recent developments (numerical algorithm, fitting, etc..)
- ROOT User guides: see http://root.cern.ch/root/doc/RootDoc.html
  - not been updated with latest developments
- ROOT Talk Forum (for support, requests and discussions)
  - ✦ a thread is available for only Math and Statistical topics
  - ✦ bugs should be reported to Savannah