Floating Point Issues in Data Analysis

Lorenzo Moneta
CERN, PH-SFT

CERN/Intel workshop on Numerical Computing

May 27-28, 2013, CERN

Outline

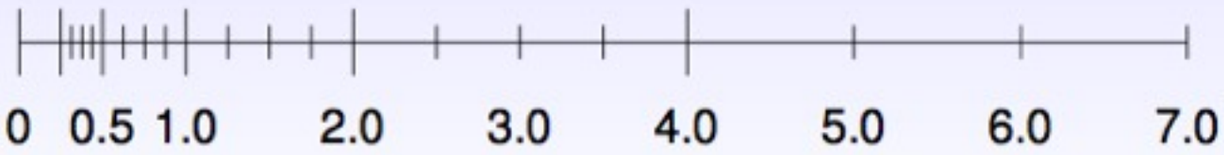
- Introduction
- ROOT Mathematical libraries
- Numerical Errors in Fitting and Minimization
  - numerical derivative error
  - summation error
- Error in Matrix computation (inversion)
- Summary

Introduction

- Floating Points
  - fl(x) = x(1 + ε)
  - fl(x op y) = (x op y) (1+ ε) op = +,-,/,* $\epsilon \leq u = \frac{1}{2}\beta^{1-t}$
    - single precision (32 bits), $u = 2^{-24} \approx 6 \times 10^{-8}$
    - double precision (64 bits): $u = 2^{-53} \approx 1.1 \times 10^{-16}$

IEEE 754

If $\beta = 2$, $t = 3$, $e_{min} = -1$, and $e_{max} = 3$:

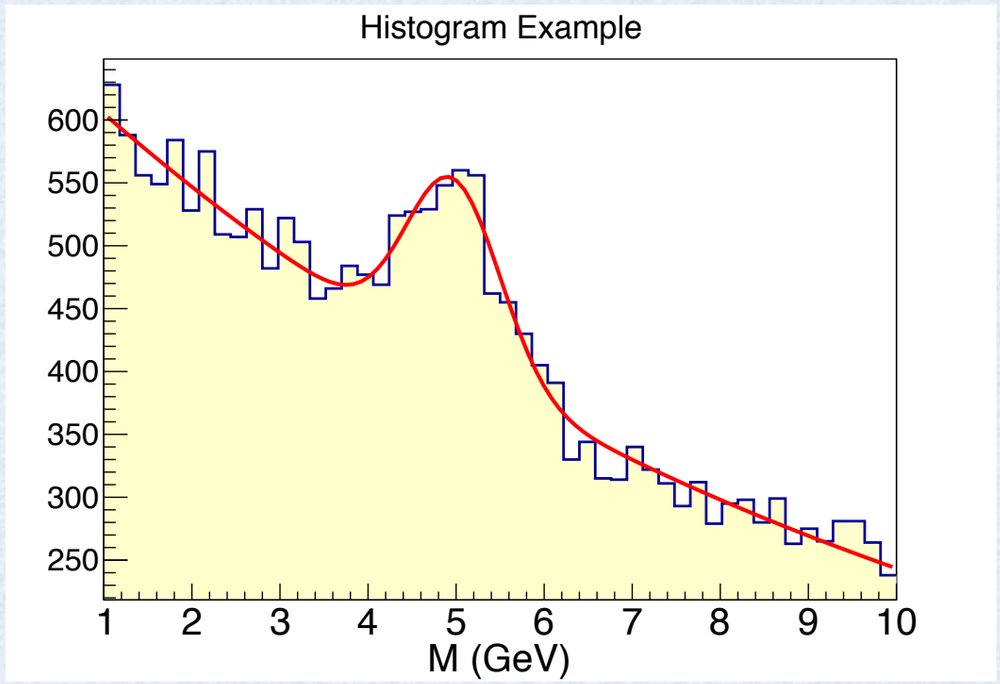


  - relative error on result can be much larger
    - e.g. fl(x-y) ≤ ε (| x | + | y |)/(| x-y |) large for x ~ y
  - fl(fl(x+y) + z) ≠ fl(fl(x+z) + y)
- 32 bits vs 64 bits architectures
  - in 32 bits arch. operations done in double extended precision (t = 64), but stored as double in memory

Scaling

- Importance to try to keep numbers around 1
- Better to apply a linear transformation to the data to have location and scale around 1
  - Non-sense using for observables units not close to 1 (e.g use GeV instead of eV)
  - scale is defined by physical quantities (e.g. detector resolution)
  - use reasonable ranges

do not use here a scale from $1.x10^9$ to $10x10^9$ (eV)



Standard Deviation

- Computing the sample variance is numerically difficult when $\mu >> \sigma$
  - Normally s^2 and μ computed with one pass

$$s^2 = \sum_{i=1}^{N} \frac{(x_i - \mu)^2}{N} = \sum_{i=1}^{N} \frac{x_i^2}{N} - \left(\sum_{i=0}^{N} \frac{x_i}{N} \right)^2$$

  - numerical error when making difference of two positive numbers

- A possible solution is to accumulate

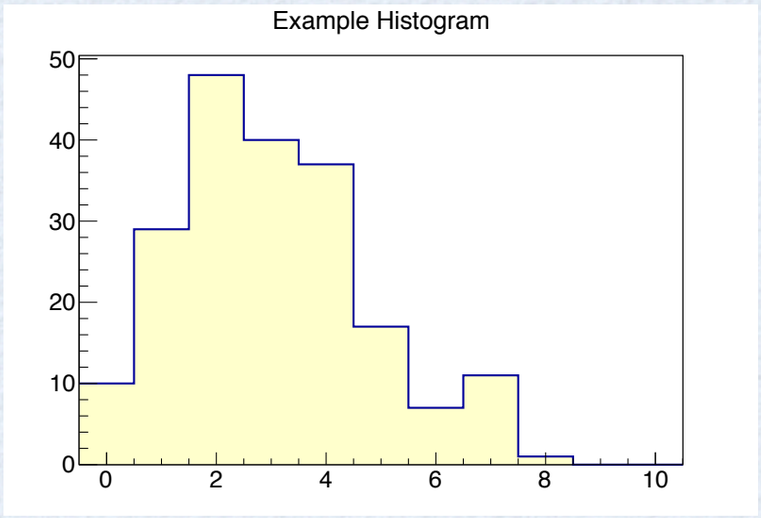
$$M_1 = x_1 \qquad M_k = M_{k-1} + \frac{x_k - M_{k-1}}{k} \quad \longrightarrow \quad \hat{\mu} = M_N$$

$$S_1 = 0 \quad S_k = S_{k-1} + \frac{(k-1)(x_k - M_{k-1})^2}{k} \quad \longrightarrow \quad s^2 = \frac{S_N}{N}$$

Example: Histograms

- Histogram classes in single (**TH1F**) and double precision (**TH1D**)
  - axis always represented in double precision
    - choose correct bin boundaries

$$i_{bin} = \text{int}\left(n_{bins} \frac{x - x_{MIN}}{x_{MAX} - x_{MIN}} \right)$$

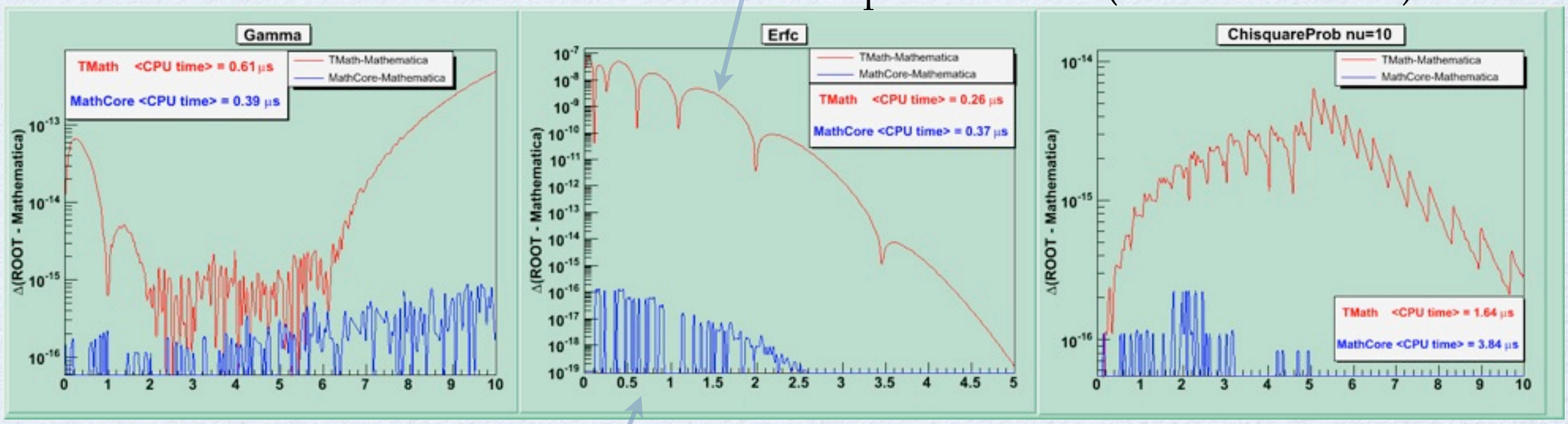


  - single precision often enough for bin content
    - save memory for large multi-dim histograms
  - double precision often not really needed (apart from cases with large number of counts/bin)
    - provided also a **TH1I** (integer bin content)
    - if memory is not an issue, better always to use double precision

Mathematical Functions

- All Math functions (transcendental, special functions and statistical functions) in ROOT are provided in double precision
  - for some dedicated cases a faster single-precision function may be preferable
  - plan to introduce also vectorized versions

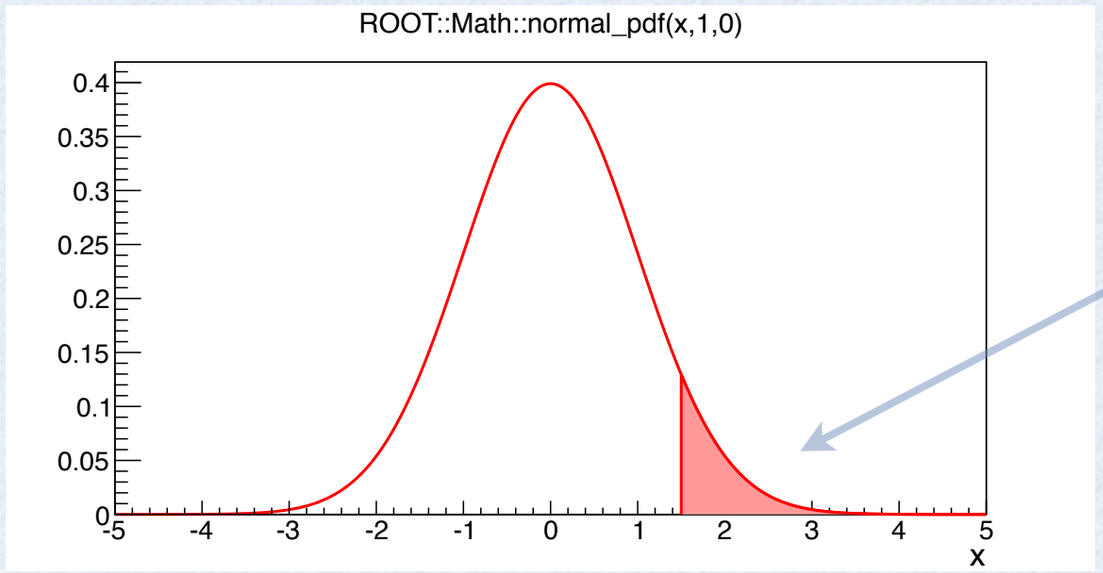
old implementation (ROOT version 4)



new implementation based on Cephes in ROOT 5

Statistical Functions

- Example: statistical functions:
  - provide cumulative and its complement (using a different implementation):



- **normal_cdf(x,σ)** *(left tail)*
- **normal_cdf_c(x,σ)** *(right tail)*

  instead of just using

  1.0 - normal_cdf(x,σ)

- Same for the inverse of cumulative (quantile)
  - **normal_quantile(p, σ)** and **normal_quantile_c(p, σ)**
    - significance $\mathbf{n_\sigma}$ **= normal_quantile_c(p, σ)**

Matrix and Vector Libraries

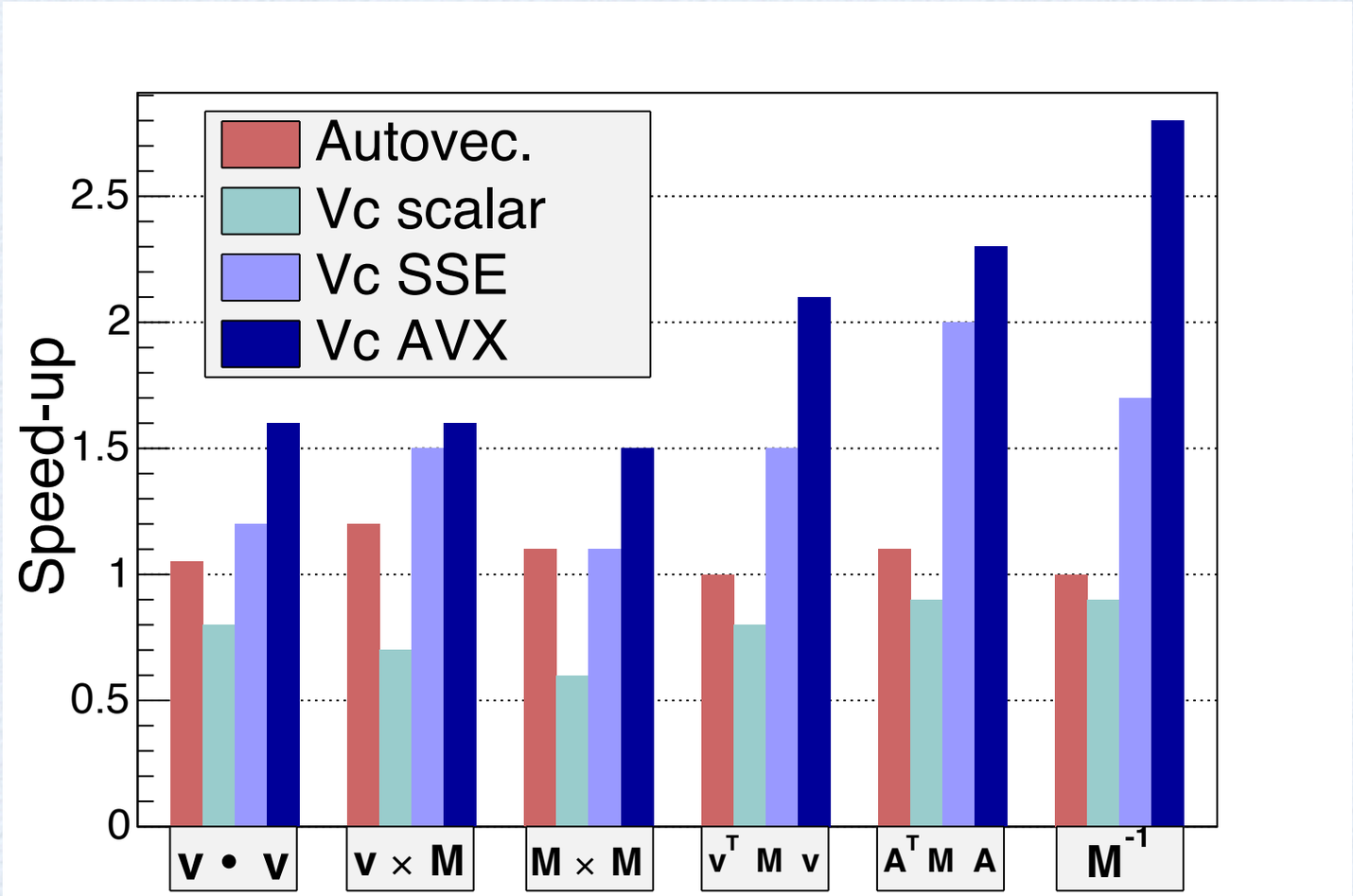
- ROOT Mathematical Libraries provide:
  - Template vector and matrix classes (in any dimension)
    - e.g. **SMatrix< N, double>**
  - Template classes for geometry and physics vectors
    - e.g. **LorentzVector<PxPyPzE4D<double> >**
      - classes can be used in single and double precision
- Often no need of double precision for measured quantities (observables)
- Simple mathematical computations could be done in single precision
  - faster if using vectorization
- Need double precision for transformation (e.g. rotation) or when performing large summation

Vectorization

- Replace the template parameter in SMatrix, SVector and LorentzVector with a vector type
  - allows vectorization on operations performed on list of matrices or physics vectors
- Investigated Vc library:
  - C++ vectors wrapping intrinsics for SIMD operations
    - **`Vc::float_v, Vc::double_v`**
    - same semantics as built-in types
      - one can use **`double_v`** as a **`double`**
  - http://code.compeng.uni-frankfurt.de/projects/vc/

Vectorization using Vc

- Perform operations in SMatrix 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>`

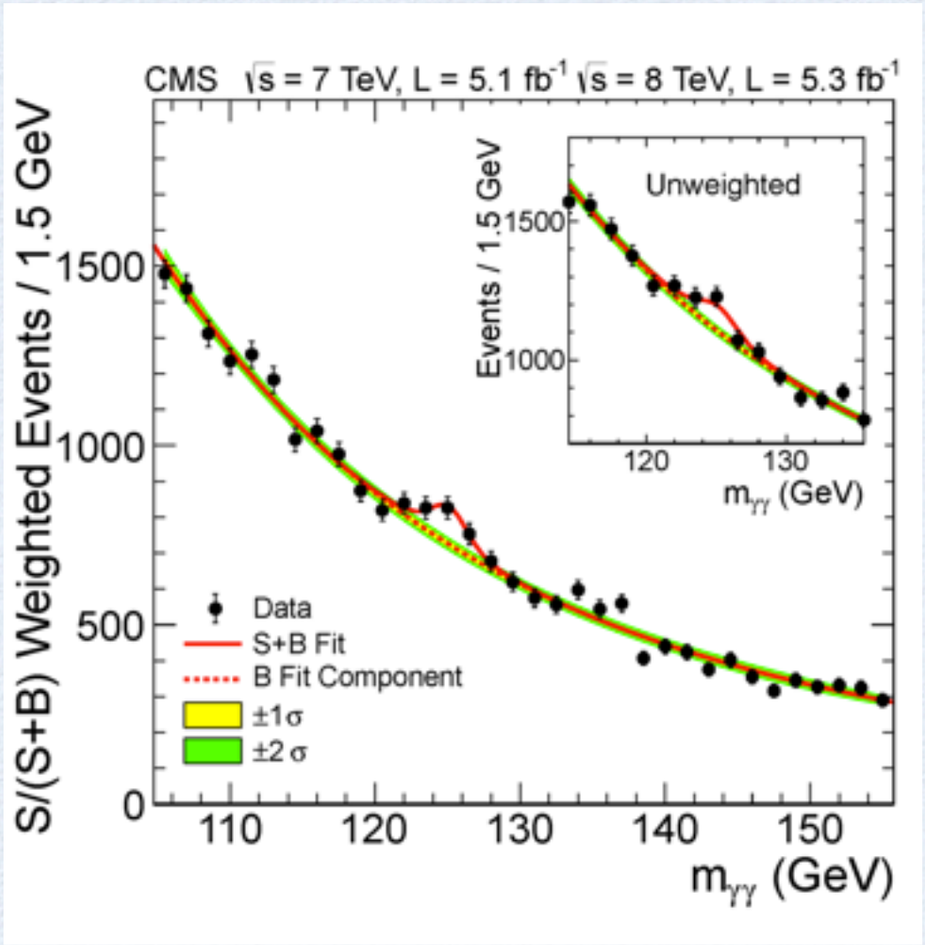


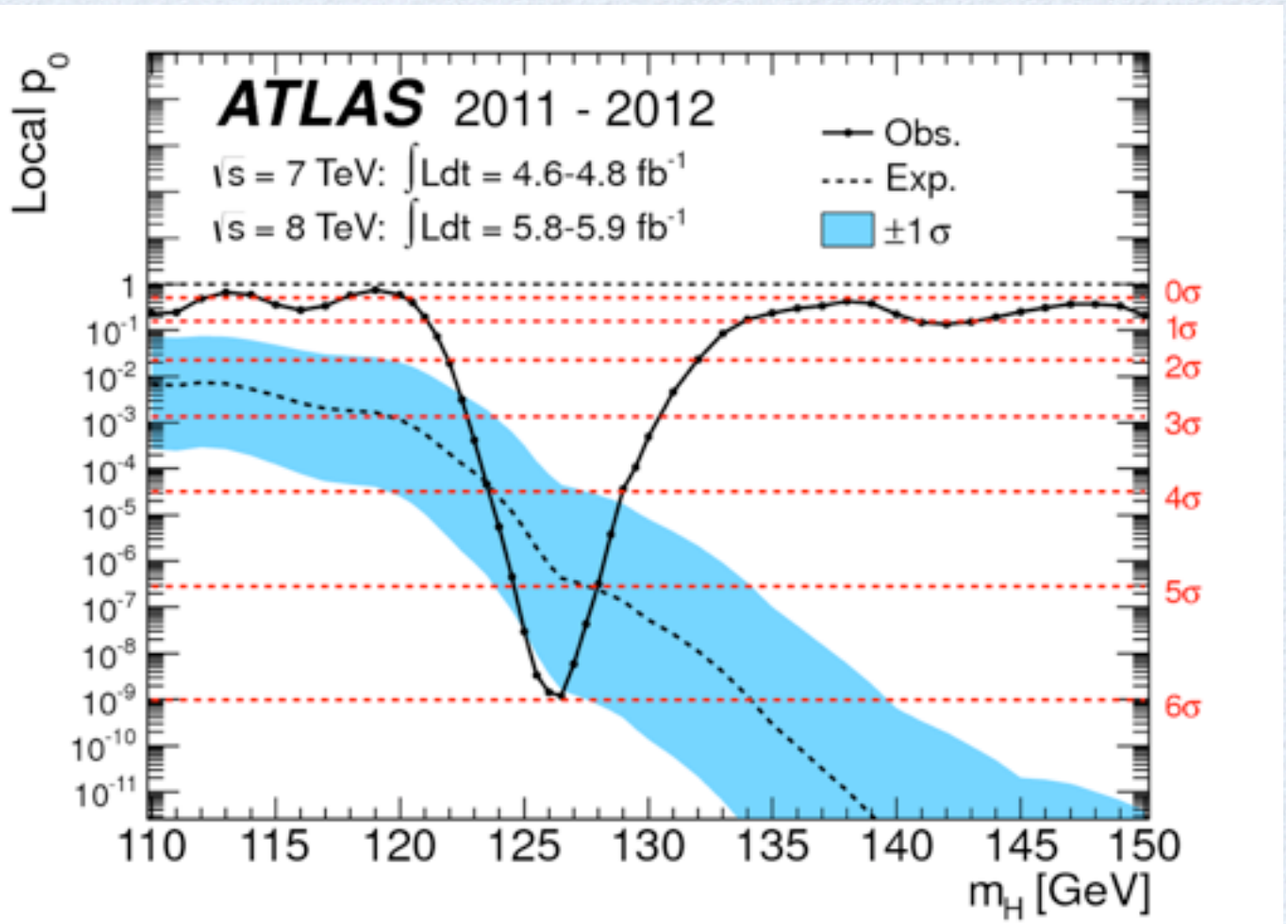
Function Minimization

- One of the most used algorithm in data analysis
- Function minimization is needed in statistical analysis
  - fitting data points (non-linear least square fits)
  - maximum likelihood fits (parameter estimation) and for error analysis (interval estimation)
    - likelihood $L(x|\theta) = \prod_i P(x_i|\theta)$
    - minimum of $-\log L = \sum_i \log P(x_i|\theta)$

Example: Higgs Searches

- Higgs search results require numerous minimization of complex likelihood functions (~ 500 parameters)





MINUIT Algorithm

- Migrad based on Variable Metric algorithm (Davidon)
- Iterate to find function minimum:
  - start from initial estimate of gradient $\mathbf{g_0}$ and Hessian matrix, $\mathbf{H_0}$
  - find Newton direction: $\mathbf{d = H^{-1} g}$
  - computing step by searching for minimum of **F(x)** along **d**
  - compute gradient **g** at the new point
  - update inverse Hessian matrix, $\mathbf{H^{-1}}$ at the new point using an approximate formula (Davidon, Powell, Fletcher)
    - better updating inverse $\mathbf{H^{-1}}$ than Hessian **H**
    - matrix is positive defined but numerical errors can make it not
  - repeat iteration until expected distance from minimum (edm) smaller than required tolerance ($\mathbf{edm = g^T H^{-1} g}$)

Numerical Errors

- What is effect of numerical errors in MINUIT ?
  - Minimization will be less efficient,

    $\Rightarrow$ more iterations $\Rightarrow$ more CPU time

    but minimizer will converge anyway
  - Minimization could fail, not being able to converge to a minimum with the required tolerance

    interest in absolute tolerance: $\Delta L = 0.5 \Rightarrow 1\ \sigma$ error in parameters
  - Error in inverting the covariance matrix
  - In same case could converge to a different minimum (e.g. a local one)

    $\Rightarrow$ obtain a wrong result

Numerical Errors (2)

- What are the cause of numerical errors ?
  - error in objective function when computing the sum of n elements: $$-\log L = \sum_i \log P(x_i|\theta)$$
    - error : ~ nε double precision is needed
  - can have also errors from:
    - computation of log(P(x))
    - normalization of P(x) due to numerical integration
  - Error in computing derivatives of log-likelihood

Derivative Errors

- MINUIT provides algorithm for computation of derivatives via finite differences
- using analytical derivatives is often prohibitive in case of very complex models
  - numerical differentiation is very convenient for users
  - minimization is very sensitive to derivative errors
    - when closer to the function minimum gradient becomes closer to zero
    - difficulty in converging in case of error in derivatives
- Discontinuities in derivatives must also be avoided !

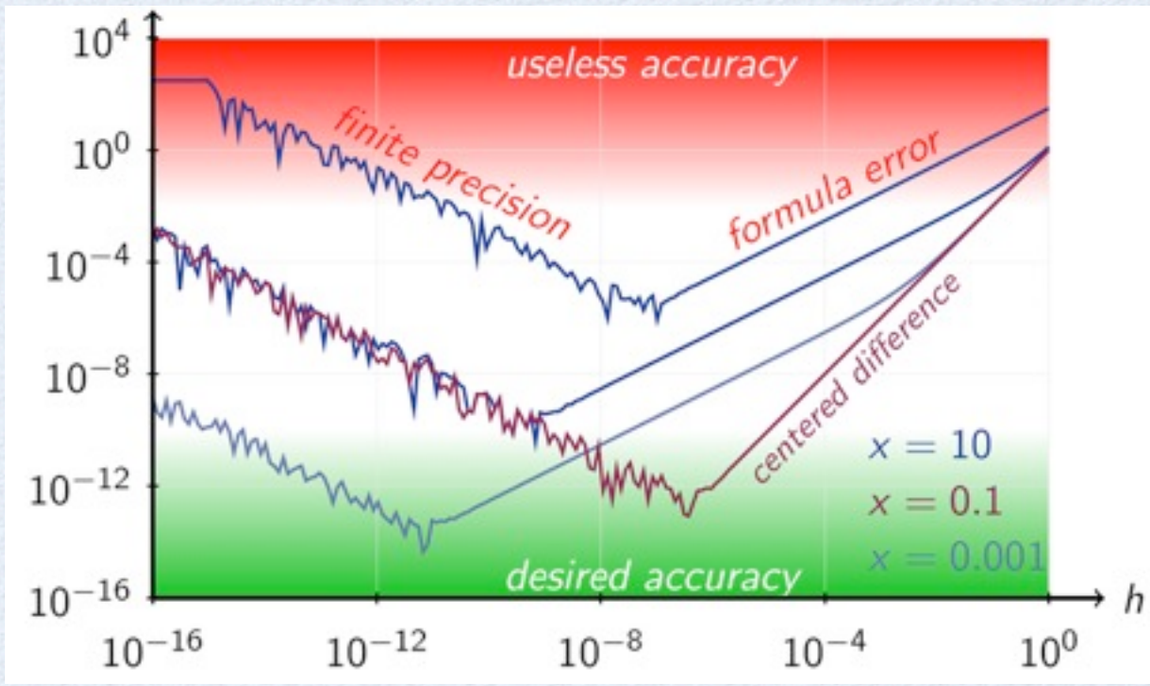
Computation of Derivatives

- Compute derivatives by finite differences

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

$$\epsilon_{TOT} = \frac{|f'''(\mu)|}{6} h^2 + \epsilon_R \frac{|f|}{h}$$

$$h_{OPT} = \left(\frac{3\epsilon_R |f|}{|f'''(\mu)|} \right)^{1/3}$$



Essential to find the right scale or step size
Algorithm in Minuit uses an iterative procedure starting from an initial user estimate

Numerical Integration

- Problematic to use Monte Carlo integration to normalize the PDF when minimizing the likelihood
  - error will be too large and random
  - makes discontinuities in minimization function
- Use adaptive numerical integration:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

  - numerical error under control if sum is not too large
  - important to define the right integration range
    - e.g. when integrating a very sharp peak

Error in Likelihood Evaluation

- In complex fits (e.g Higgs combination) log-likelihood is obtained by adding several channels.

$$-\log L(x|\mu,\theta) = \sum_{c \in channel} \sum_{i} \log P_c(x_{i_c}|\mu,\theta) + \sum_{\theta_k} \log P_k(\theta_k^0|\theta_k)$$

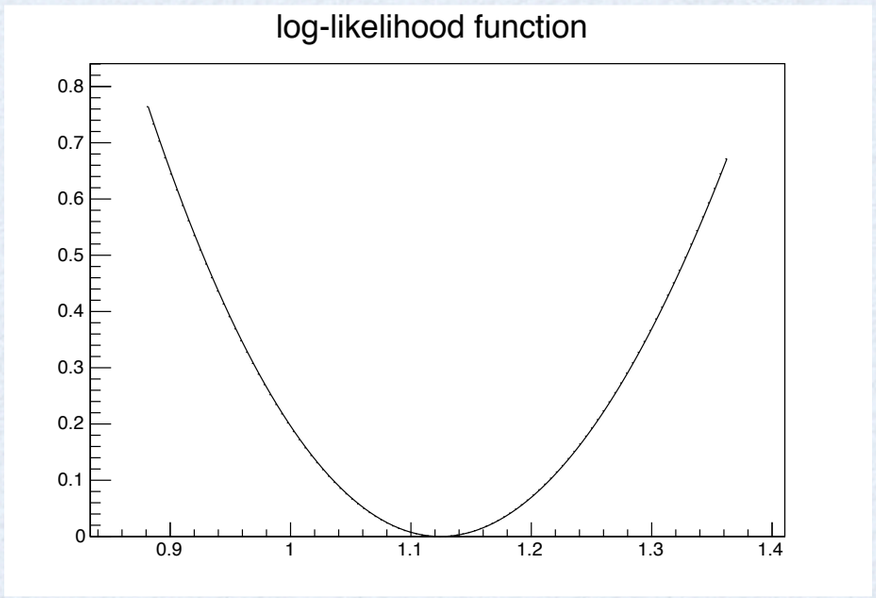
channel P.D.F (model) — parameter constraints

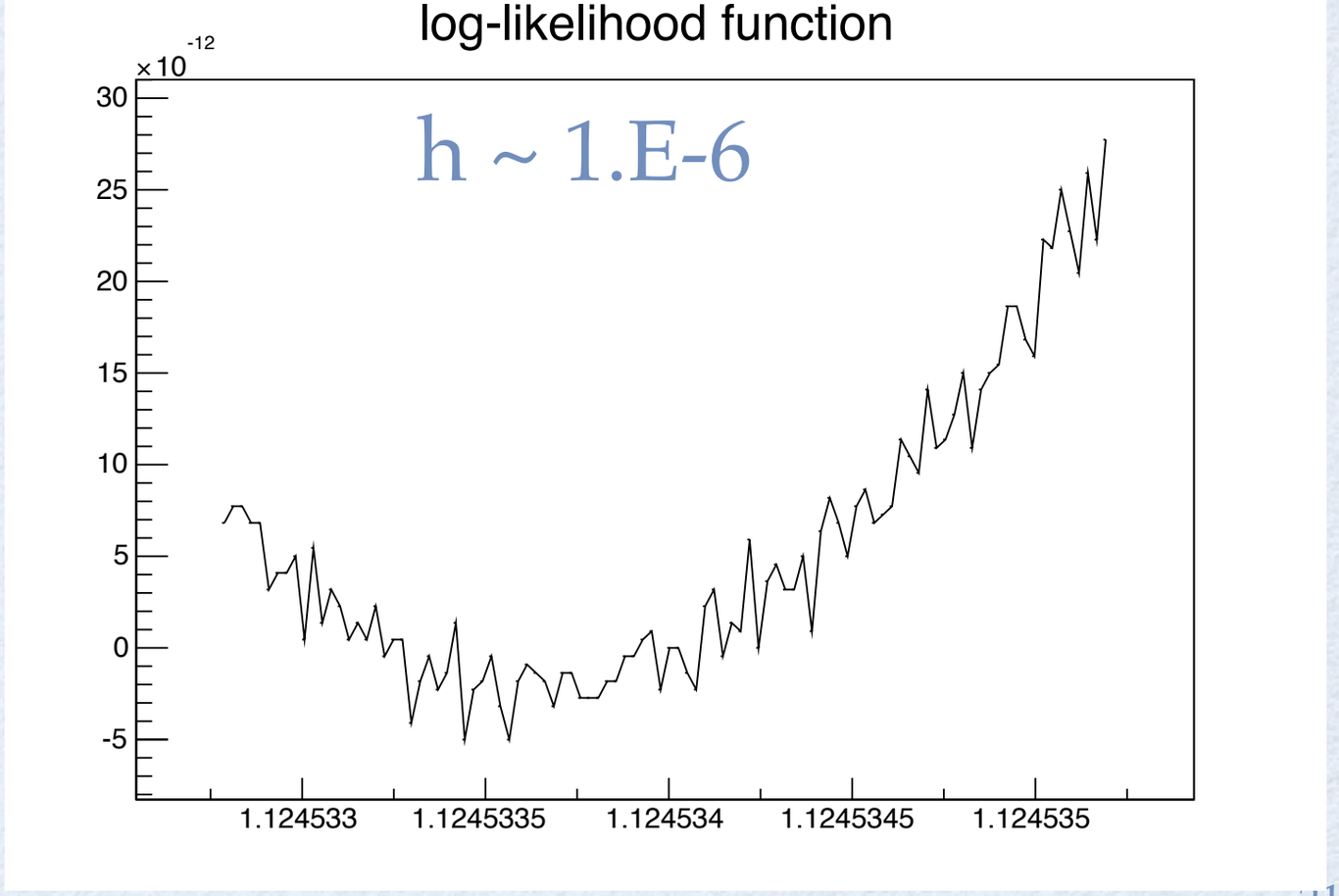
- Each channel can have different scale in log-likelihood and different numerical error
- Possible solution (available now in RooFit):
  - improve error by setting an offset for each log-likelihood component (each channel) so it is equal to 0 for nominal values
  - fits are converging much better after this re-scaling

Numerical Error Estimation

- Rude estimate of numerical error in function evaluation
  - scan the function around x with decreasing step-sizes

h ~ 0.1





Error in Summation

- Summing can be problematic also when computing the likelihood sum in parallel
  - different order of computation will result in different result for numerical error
    - can happen when using multi-threads or multi-processes with some dynamic scheduling
    - one must be careful also with vectorization
      - e.g. using *-fast-math* option in gcc
- Summing errors can be mitigated using compensated summation (Kahan)
  - added in latest Roofit version

Matrix Computation

- Computing inverse of a matrix is very sensitive to numerical errors
  - Linear system: better to solve directly without computing inverse
  - inverse needed for statistical analysis: covariance matrix (parameter errors), unfolding, etc..
- ROOT provides various matrix decomposition algorithms for solving linear systems and finding the inverse
  - LU, Bunch-Kaufmann, Choleski, QR and SVD
  - error depends on condition number
    - $k = \| A \| \| A^{-1} \|$
    - accuracy in solution ~ $\varepsilon 10^{k}$ ~ $10^{-(16-k)}$ for double precision

Example: Small Matrix Inversion

- ROOT provides also fast inversion for small matrices (up to size 5) using Cramer (TMatrix::InvertFast, SMatrix::InvertFast)
  - factor of 2 faster, since code can be written explicitly
  - suffer from numerical problems:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \text{based on} \quad \det(A) = a * d - b * c$$

- Example if A is 5x5 matrix one can get results as
  - AA^{-1} using fast Cramer inv.: error ~ 10^{-6}
  - AA^{-1} with LU decomposition: error ~ 10^{-12}

Summary

- Importance of being aware of floating point traps in performing numerical calculations
  - must not ignore floating point errors, although observables measured at a much less precision
  - learn how numerical errors arise in most used algorithms of data analysis
  - hope you will learn later how you can control better these numerical errors

References

- Wikipedia:
  - http://en.wikipedia.org/wiki/Floating_point
- W. Kahan home page (with code examples)
  - http://www.cs.berkeley.edu/~wkahan/
- N. J .Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM book, 2002
- D. Goldberg, *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, ACM Computing Surveys 23, 5–48
- D. Monniaux, *The pitfall of verifying floating-point computations*, ACM Transactions on Programming Languages and Systems 30, 3 (2008) 12