# Floating Point Issues in Data Analysis

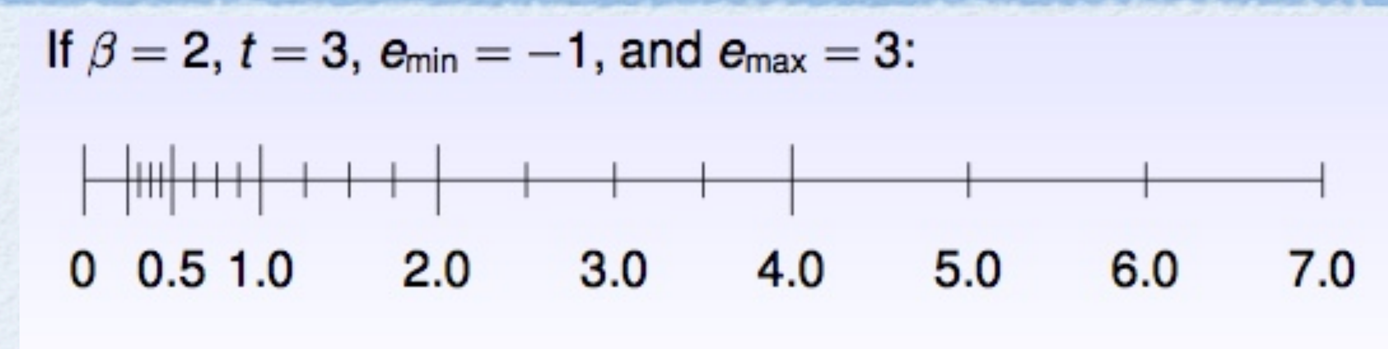
Lorenzo Moneta
CERN, PH-SFT

*CERN/Intel workshop on Numerical Computing*

*February 7-8, 2012, CERN*

# Introduction

- Floating Points

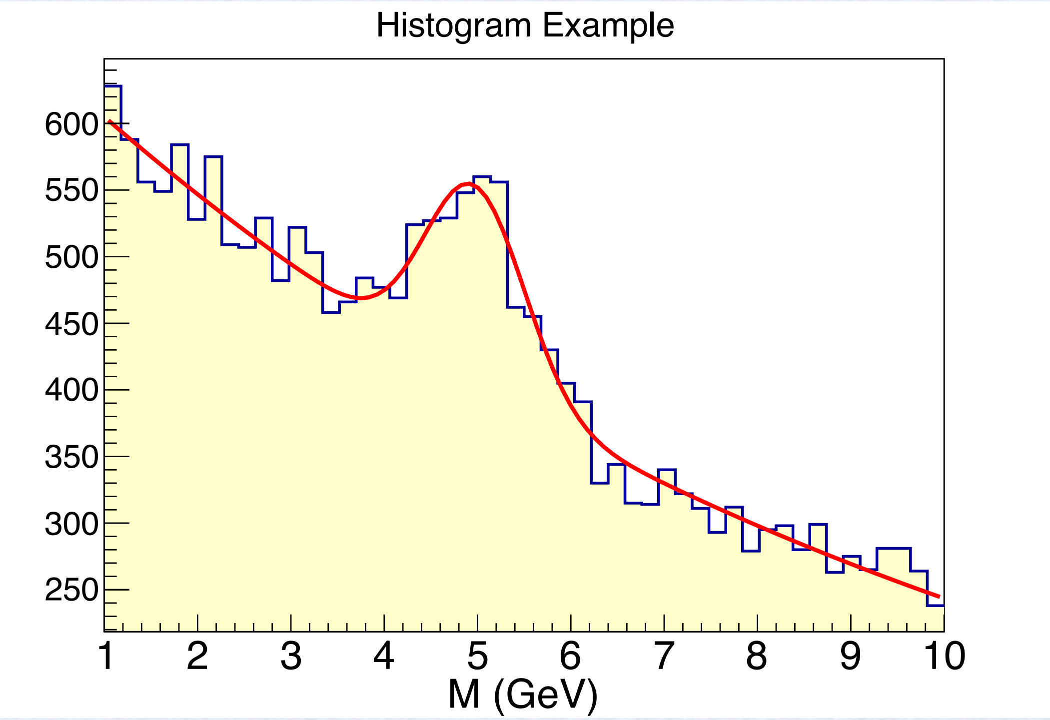



  - $fl(x) = x(1 + \varepsilon)$
  - $fl(x \text{ op } y) = (x \text{ op } y)(1 + \varepsilon)$ op = +,-,/,*  $\epsilon \leq u = \frac{1}{2}\beta^{1-t}$
    - single precision (32 bits), $u = 2^{-24} \approx 6 \times 10^{-8}$  IEEE 754
    - double precision (64 bits): $u = 2^{-53} \approx 1.1 \times 10^{-16}$
  - relative error on result can be much larger
    - e.g. $fl(x-y) \lessapprox \varepsilon\,(|x|+|y|)/(|x-y|)$  large for $x \sim y$
  - $fl(\, fl(x+y) + z\,) \neq fl(\, fl(x+z) + y)$
- 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 μ << σ
  - 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

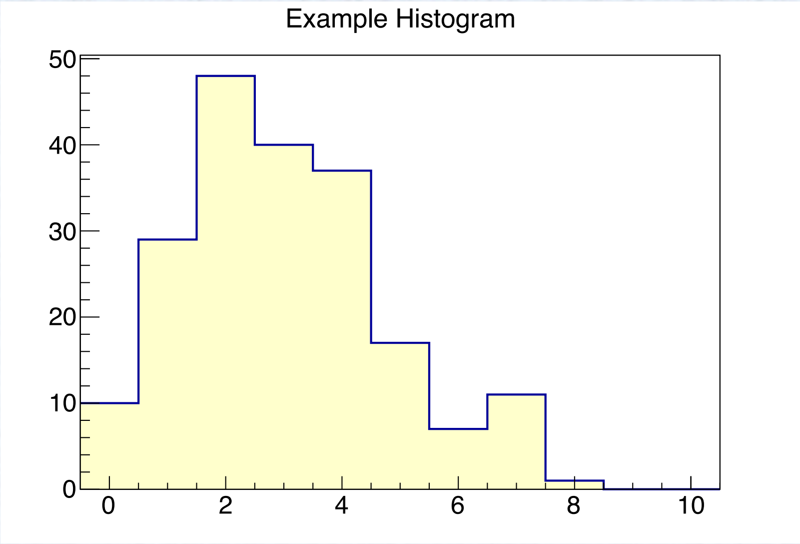
$$M_1 = x_1 \qquad M_k = M_{k-1} + \frac{x_k - M_{k-1}}{k} \qquad \longrightarrow \qquad \hat{\mu} = M_N$$

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

# Example: Histograms

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

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



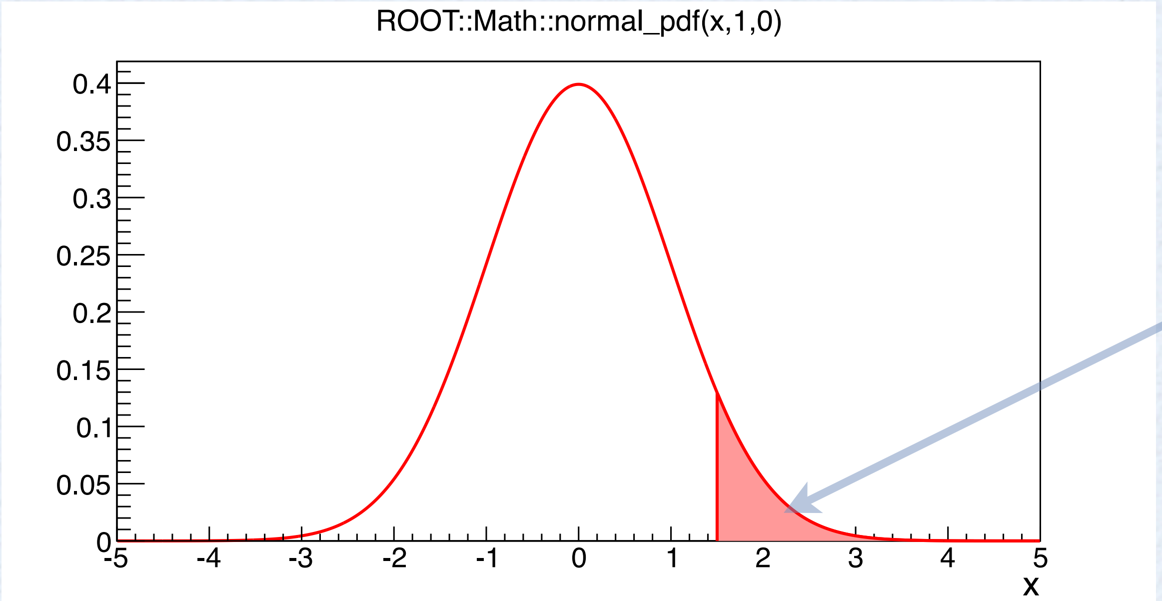

  - single precision often enough
    - save memory for large multi-dim histograms
  - double precision often not really needed (apart from cases with large number of counts/bin)
    - provided also a **TH1I** (integer bin content)
    - if memory is not an issue, better always to use double precision

# 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 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

# Math and Stat Functions

- All Math functions provided in double precision
  - maybe (for some dedicated cases) a faster single-precision function could be needed
- Example: statistical functions:
  - provide cumulative and its complement:




- **normal_cdf(x,σ)** and **normal_cdf_c(x,σ)**
- instead of just using **1.0 - normal_cdf(x,σ)**

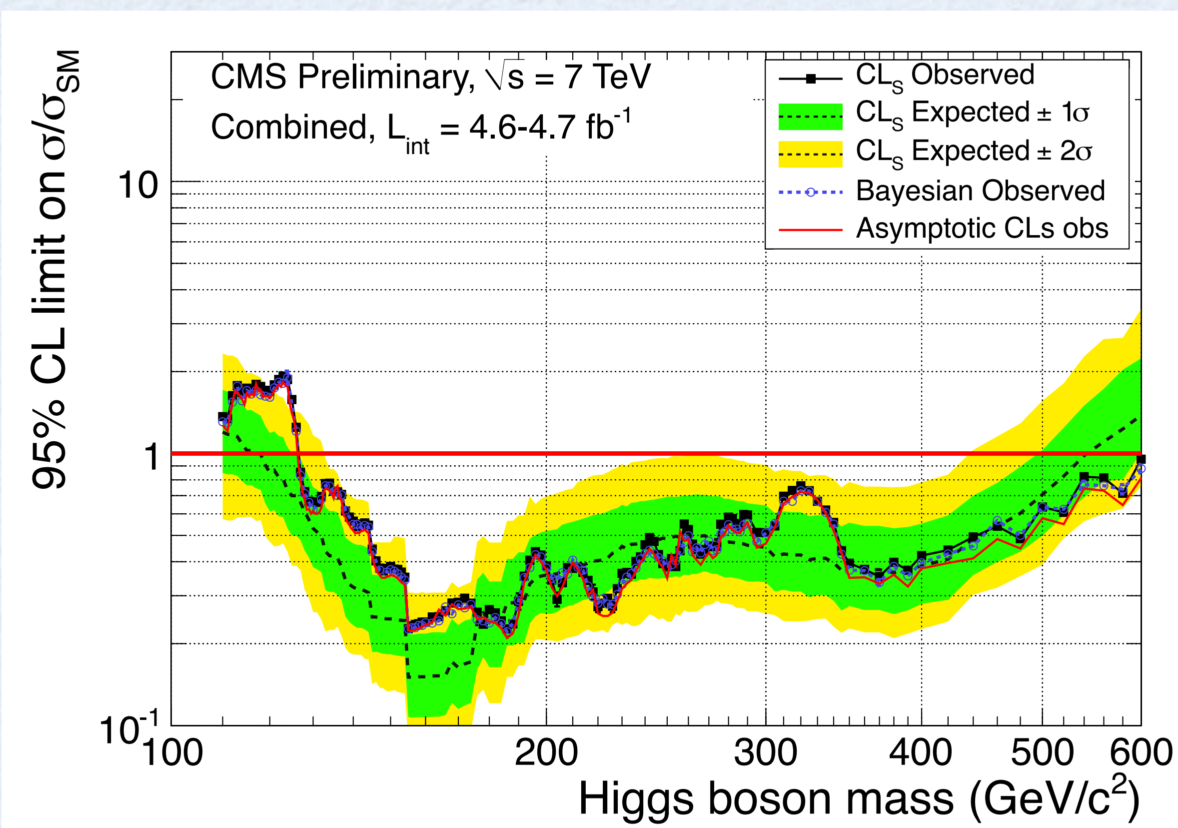
  - Same for the inverse of cumulative (quantile)
    - **normal_quantile(p, σ)** and **normal_quantile_c(p, σ)**
      - significance **$n_\sigma$ = normal_quantile_c(p, σ)**

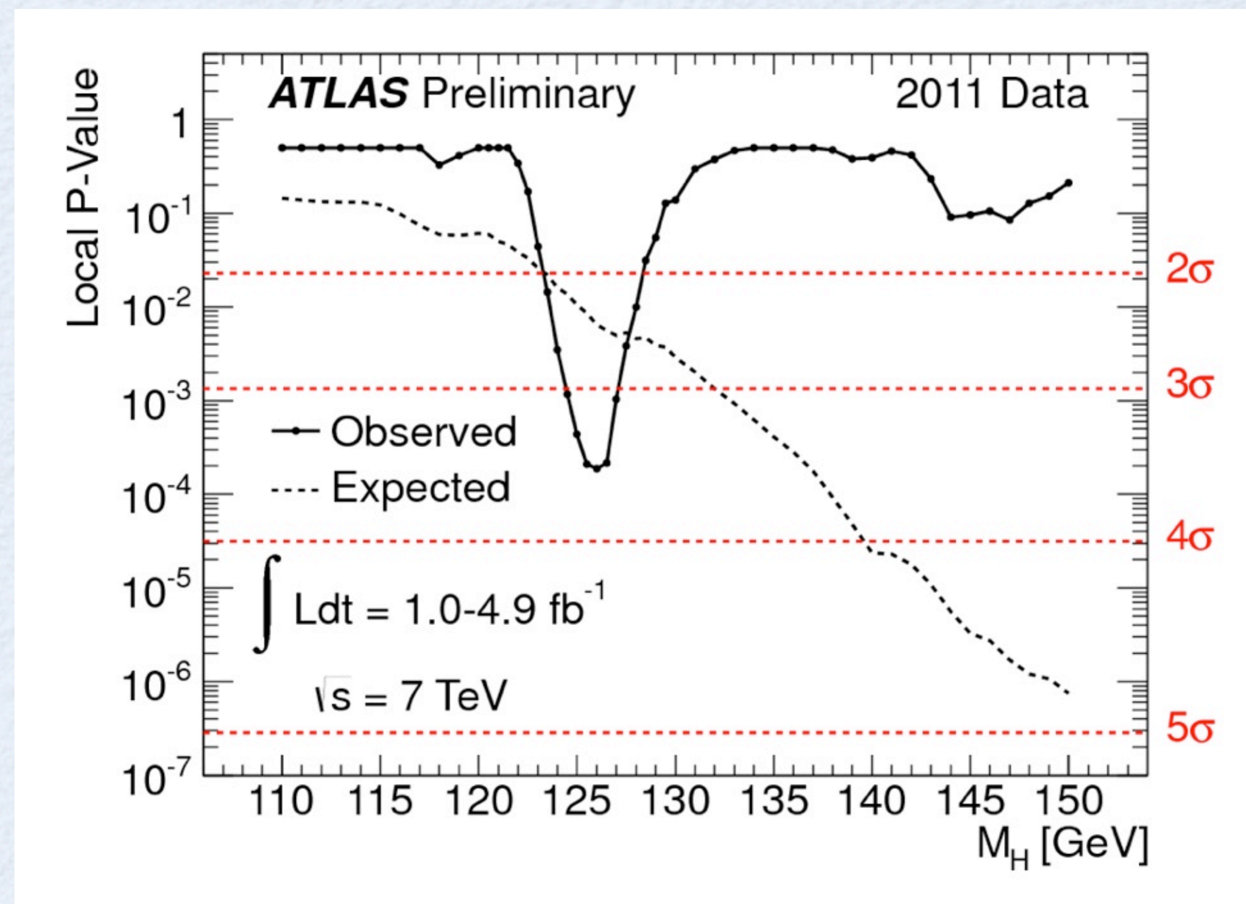
# Function Minimization

- 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 (> 200 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 $\mathbf{F(x)}$ along $\mathbf{d}$
  - compute gradient $\mathbf{g}$ 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 $\mathbf{H}$
    - matrix is positive defined but numerical errors can make it not
  - repeat iteration until expected distance from minimum smaller than required tolerance

# Numerical Errors

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

    ⇒ more iterations ⇒ more CPU time

    but minimizer will converge anyway
  - Minimization could fail, not being able to converge to a minimum
  - Error in inverting the covariance matrix
  - In same case could converge to a different minimum (e.g. a local one)

    ⇒ obtain a wrong result

# Numerical Errors (2)

- What are the cause of numerical errors ?
  - error in F(x) when computing the sum of n elements
    - error : $\sim n\varepsilon$ double precision is needed
    - can be problematic when computing in parallel where sum is done in random order
      - can be solved using compensated summation (Kahan)
  - F(x) can have also errors from:
    - computation of log( P(x) ) in likelihood fits
    - normalization of P(x) due to numerical integration

# Derivative Errors

- MINUIT provides algorithm for computation of derivatives via finite differences
- using analytical derivatives is often prohibitive in case of very complex models
  - automatic 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

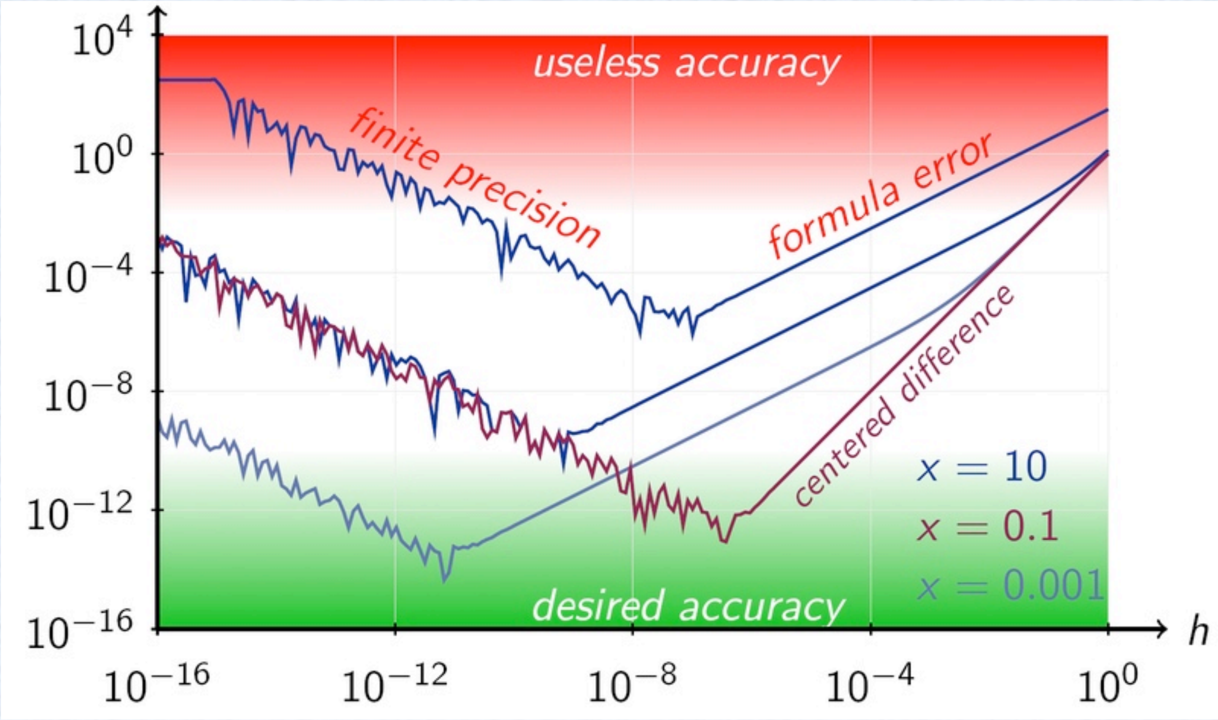
# Computation of Derivatives

- 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 initial user value

# Numerical Integration

- Problematic to use Monte Carlo integration to normalize the PDF when minimizing the likelihood
  - error will be too large and random
- 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

# 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 $\sim \varepsilon 10^{k} \sim 10^{-(16-k)}$ for double precision

# Example: Matrix Inversion

- ROOT provides also fast inversion using Cramer (TMatrix::InvertFast, SMatrix::InvertFast)
  - factor of 2 faster
  - 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