The MIXMAX Random Number Generator in ROOT

L. Moneta
(EP/SFT)

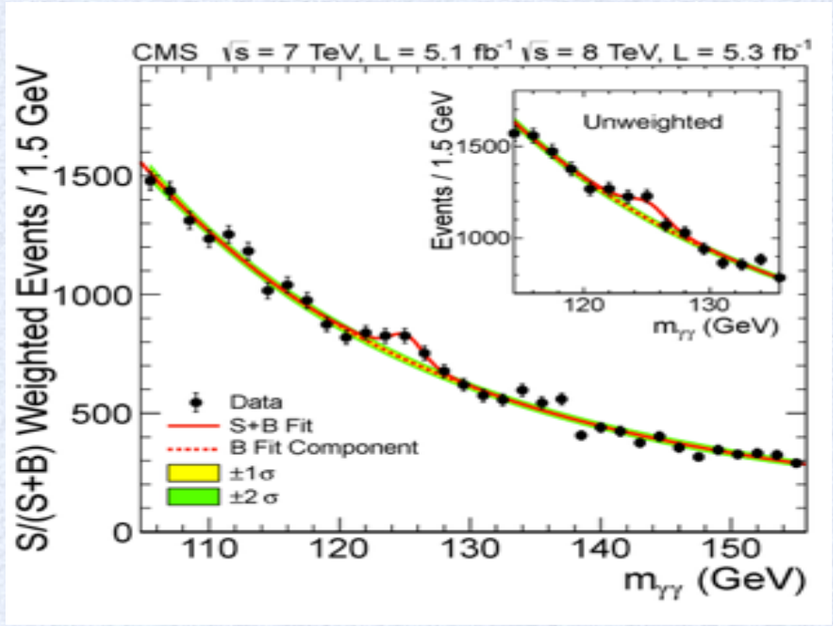
MIXMAX Consortium Meeting, 5 September 2016

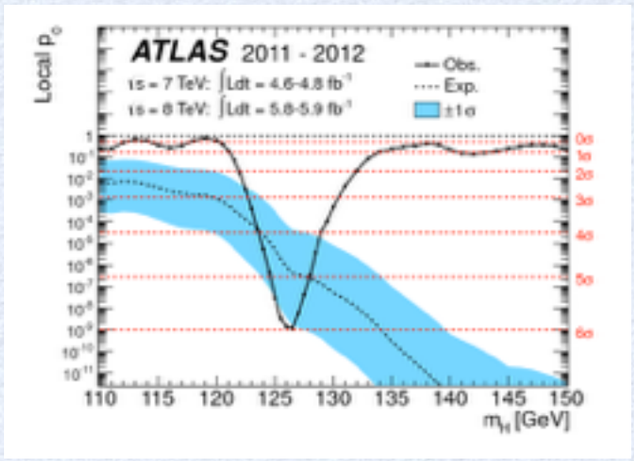
Outline

- Introduction to ROOT
- Random Numbers in ROOT
- Integration of MIXMAX generator in ROOT
- MIXMAX tests on divergence of trajectories
- Summary and future plans

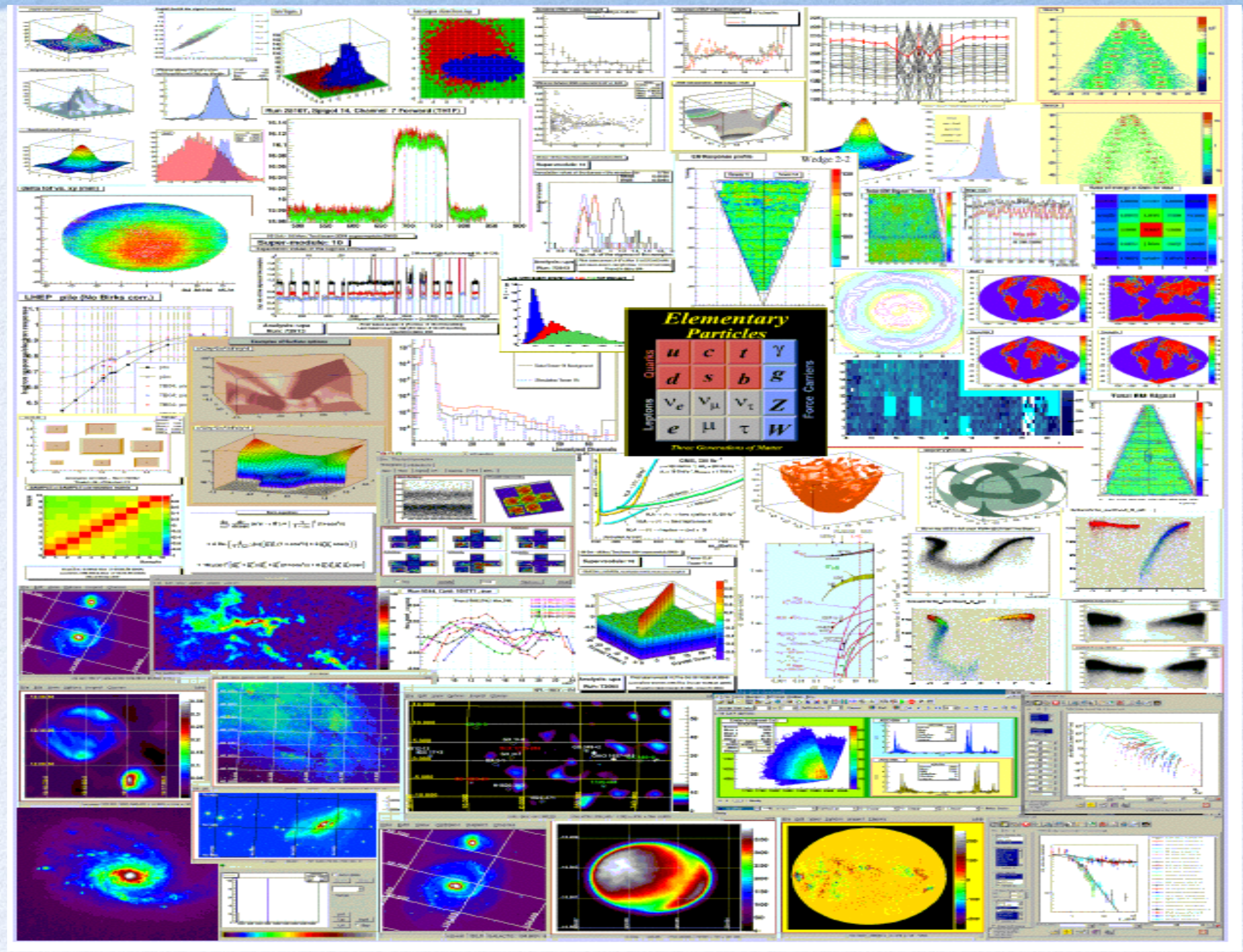
What is ROOT ?

- Framework for large scale data handling
- Provides, among others,
  - an efficient data storage, access and query system
    - possibility to write C++ objects to file
    - used for petabyte/year rates of LHC data
  - statistical data analysis:
    - histogramming
    - fitting and minimization
    - multi-variates analysis algorithms (including machine learning methods)
    - advanced statistical tools (combination of results, discovery significance, etc..)
  - scientific visualization: 2D and 3D graphics, Postscript, PDF, LateX
- Open source project with several thousands of users



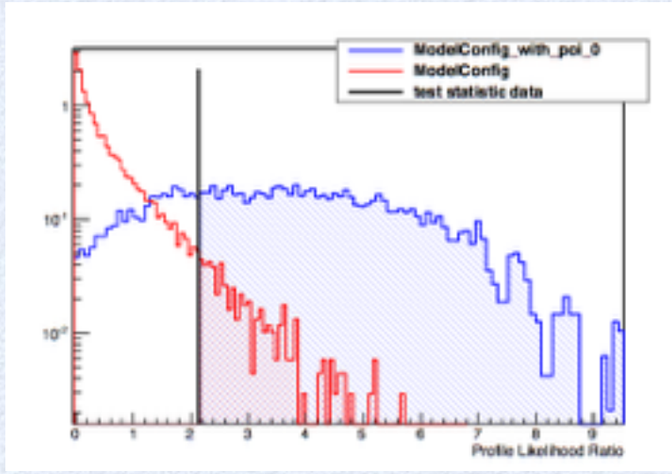


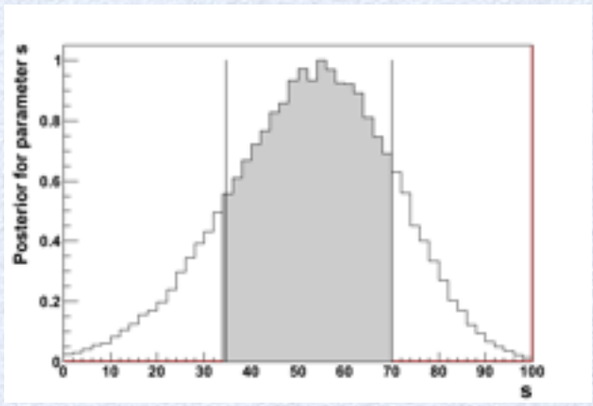
Example: ROOT Graphics



Random Numbers in ROOT

- Random numbers are at the base of Monte Carlo methods for simulation
- ROOT libraries are used in several simulation applications
  - ROOT provides the needed Math libraries, including pseudo-random number generators
- Random numbers are used in statistical analysis
  - Frequentist statistics (e.g hypothesis tests) require generating several random experiments (bootstrapping methods)
- Monte Carlo integration algorithms are also based on random numbers
  - algorithms like VEGAS or Markov-Chain MC are available in ROOT





Requirements on PRNG

- Correctness of generated random number
  - avoid ending-up producing wrong results
- CPU Efficiency of generation
  - time spent in a PRNG should be much smaller than the rest
- Reproducibility of the random sequence is needed for testing and debugging (especially in simulation applications)
- Efficient algorithm for generating random numbers according to given distributions (e.g. Gaussian, Poisson)
- Ability to generate un-correlated random streams for parallel executions

New Random Number Design

- New design for Random number classes has been introduced in ROOT last year
- Clear separation between
  - engine classes (that generates the numbers)
  - generation of numbers according to distributions
- Classes implementing algorithms for generation of pseudo-random numbers
  - e.g. ROOT::Math::MersenneTwisterEngine, ROOT::Math::MixMaxEngine
- Class for implementing most used random number distributions (e.g. Gaus, Poisson, Binomial,…)
  - ROOT::Math::RandomFunctions
- User interface class
  - ROOT::Math::Random

MIXMAX Engine Class

- MIXMAX pseudo-random number generator has been integrated in the ROOT::Math::MixMaxEngine class
  - simple wrapper to the original C implementation (from MIXMAX version 1.0)

```
class MixMaxEngine : public TRandomEngine {

public:

   typedef  TRandomEngine BaseType;

   MixMaxEngine(uint64_t seed=1);

   /// set the generator seed using a 64 bits integer
   void SetSeed64(uint64_t seed);

   /// set the number of iteration to skip
  static void SetSkipNumber(int /*nskip */)

   /// generate a double random number
   inline double operator() ();
…..

private:

   rng_state_t * fRngState;  // mix-max generator state
 };
```

Example of Usage

```
ROOT::Math::RandomMT r(seed);   // RandomMT is a typedef
                                // to Random<MersenneTwisterEngine>

double number = r.Rndm();  // generate number in [0,1]

double gauss_number = r.Gauss(mean,sigma); // generate a normal number

// using MIXMAX
ROOT::Math::RandomMixMax r2(seed);

int n = r2.Poisson(4.2)  // generate a Poisson number with mu=4.2

// using CNRG generator of L'Ecuyer from GSL implementation
ROOT::Math::Random<ROOT::Math::GSLRngCMRG> r3(seed);
double chi_number = r3.ChiSquare(ndf)
```

MIXMAX generator in ROOT

- Since ROOT version 6.05.02 MIXMAX has been included as a new Engine class in the new design
  - ROOT::Math::MixMaxEngine
    - 6.06 contains version 1.0 of MIXMAX with some modifications
    - Master (6.07) contains the version 1.1
- Tests have been performed on the new MIXMAX generators and compared with the other generators available in ROOT

Divergence of nearby trajectories

- Study evolution of the distance of two nearby trajectories
  - test described in Ranlux Luscher paper
- Two trajectories start from two nearby points in the generator state (two generator states differing only by 1-bit)
- Look how the distance of the trajectories evolves (as function of iteration number)
- If the generator is a K-system the distance evolves exponentially
  - since the computer number space is finite, the distance will be constant when half of the maximum is reached
- The rate of divergency is proportional to the log of the maximum eigenvalue
- Various metrics are possible. Decided to use (as in Luscher paper) the Chebyshev distance
  - distance = $\max_i(s1_i,s2_i)$
  - start with several random initial states all with distance=1

Divergence for RanLux

- *from M. Luescher paper:*

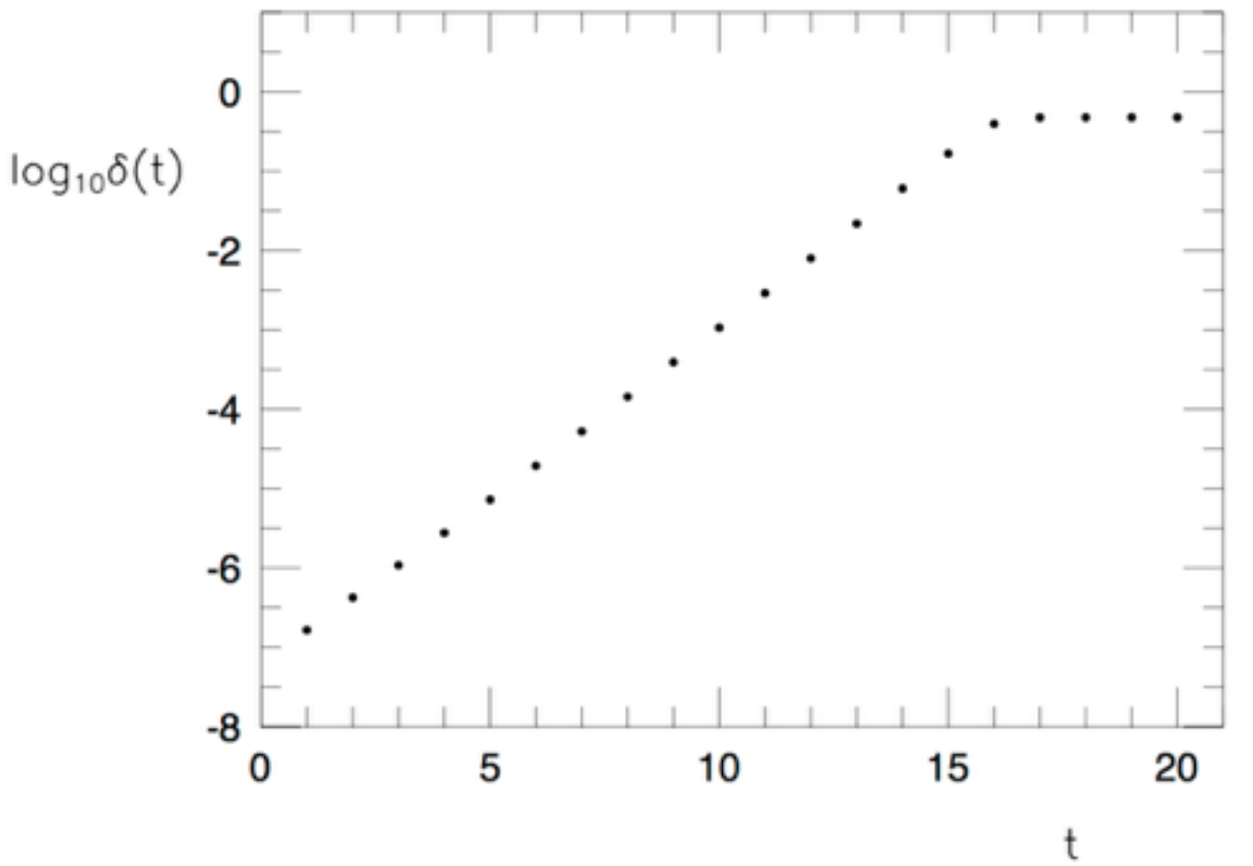
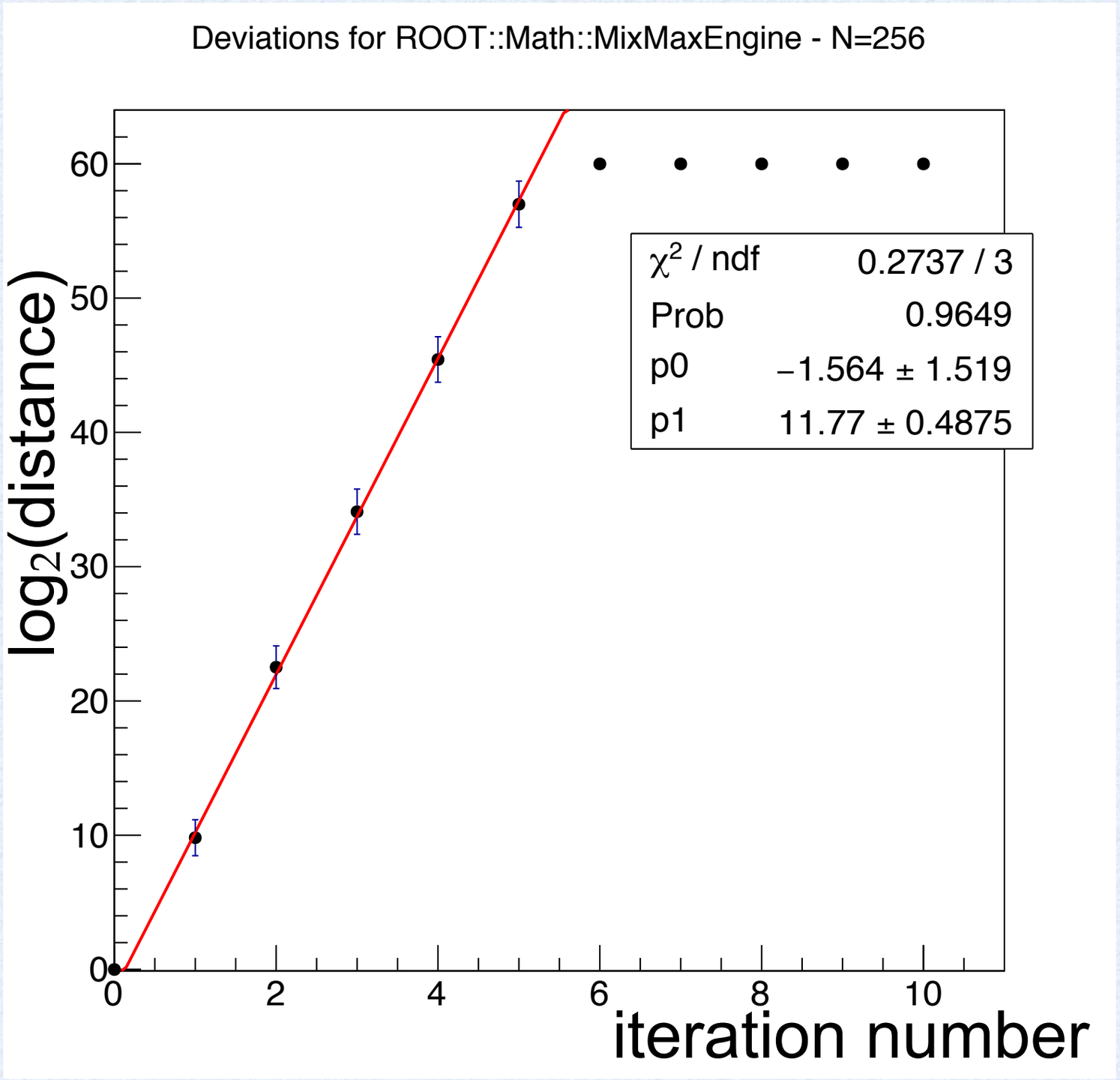


Fig. 1. Average distance $\delta(t)$ between neighbouring trajectories as a function of the evolution time t.

M. Luescher, "A Portable High-Quality Random Number Generator for Lattice Field Theory Simulations", Comput.Phys.Commun. 79 (1994) 100-110

Result for MIXMAX version 1.0

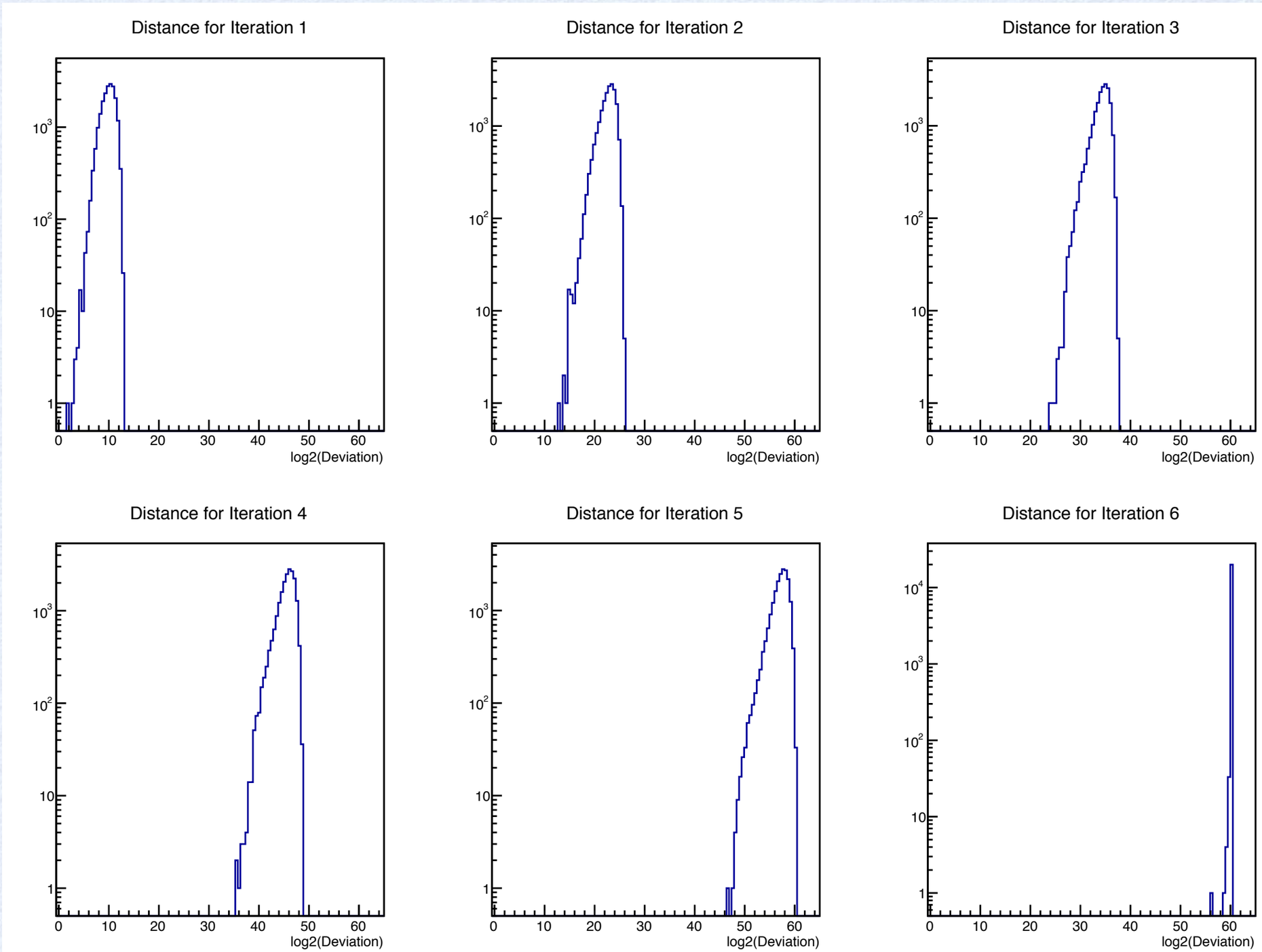


MIXMAX 1.0

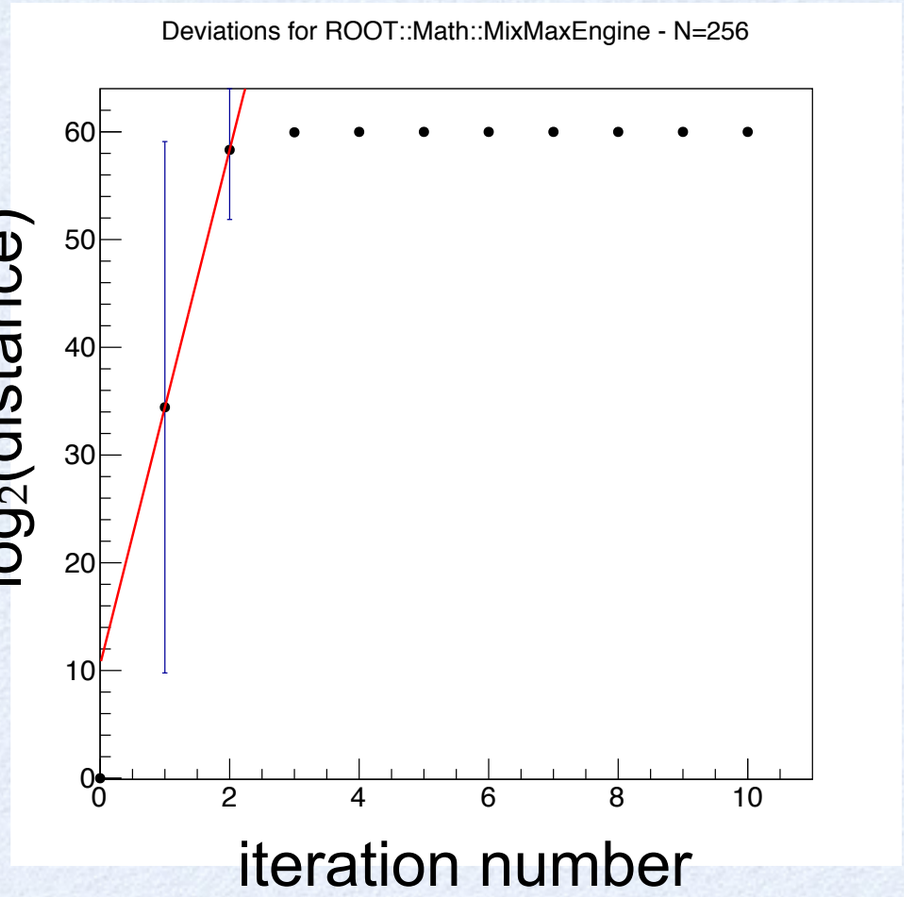
N=256

s = -1

Distance for MIXMAX 1.0



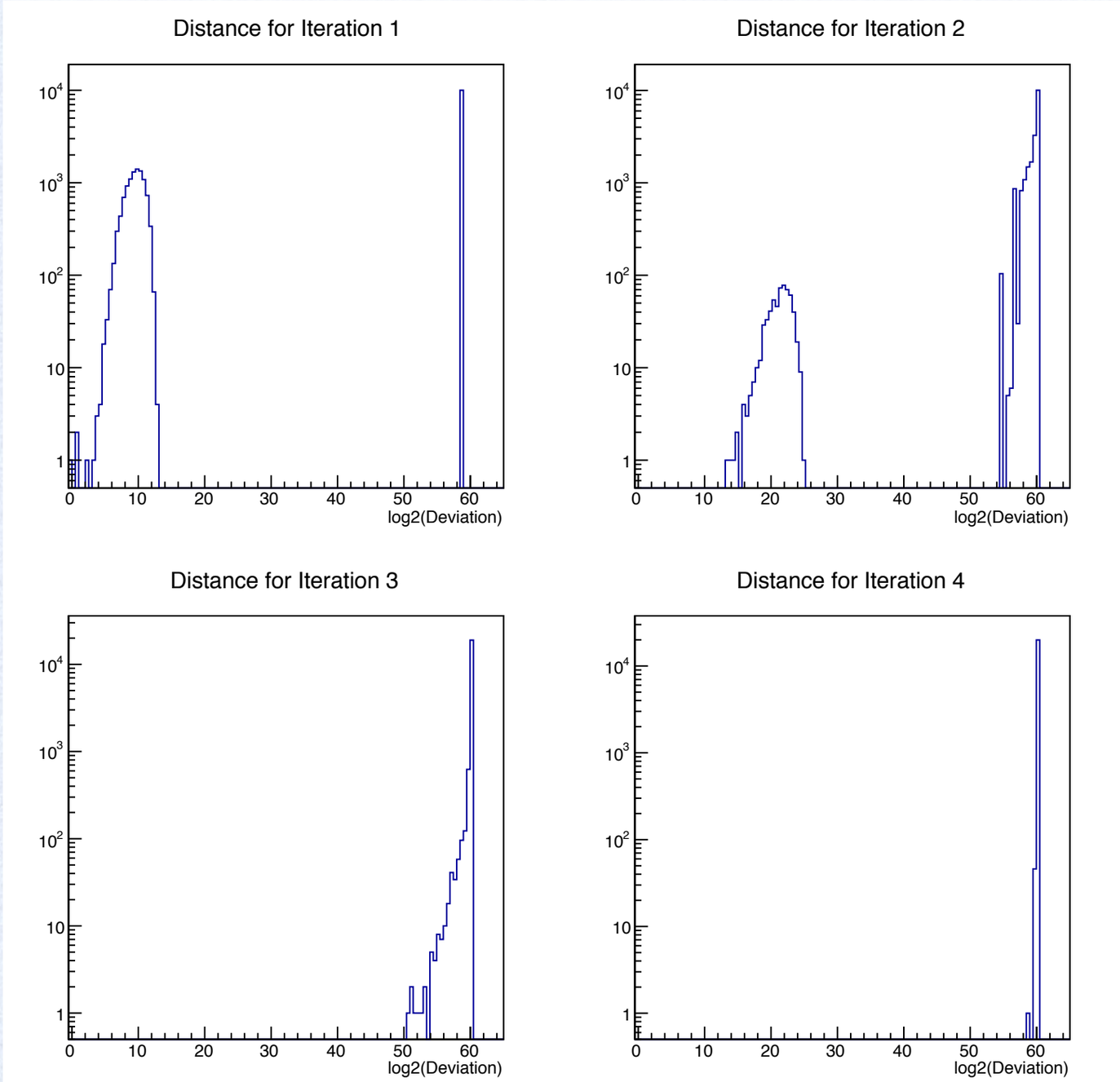
Result for MIXMAX version 1.1



MIXMAX 1.1

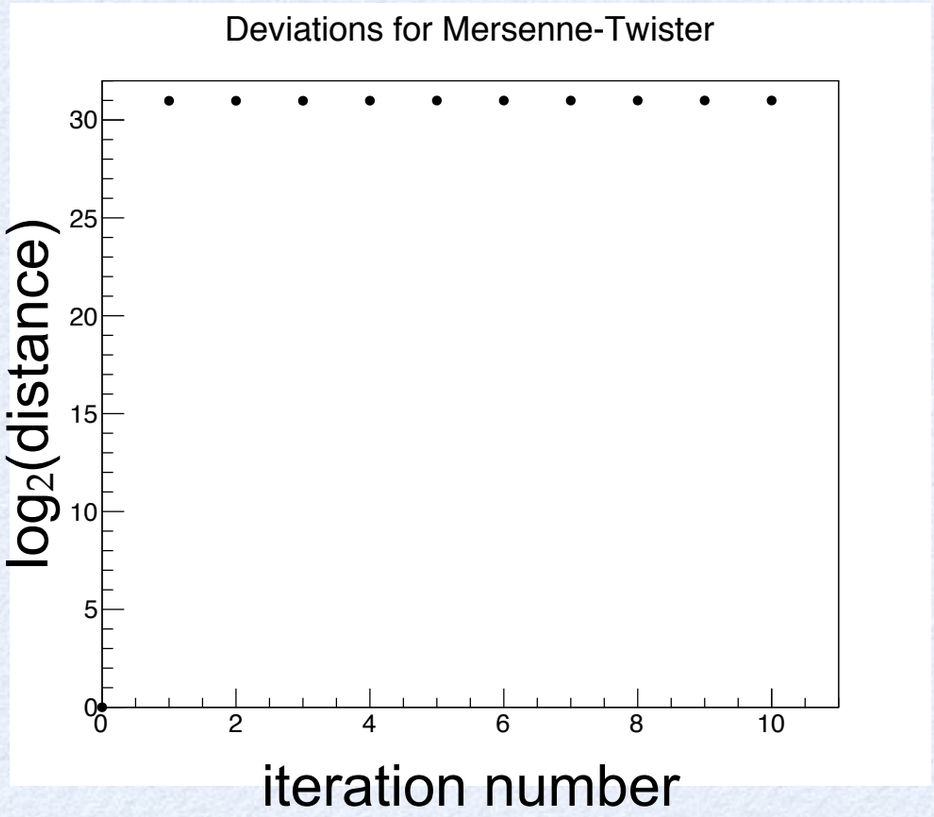
N=256

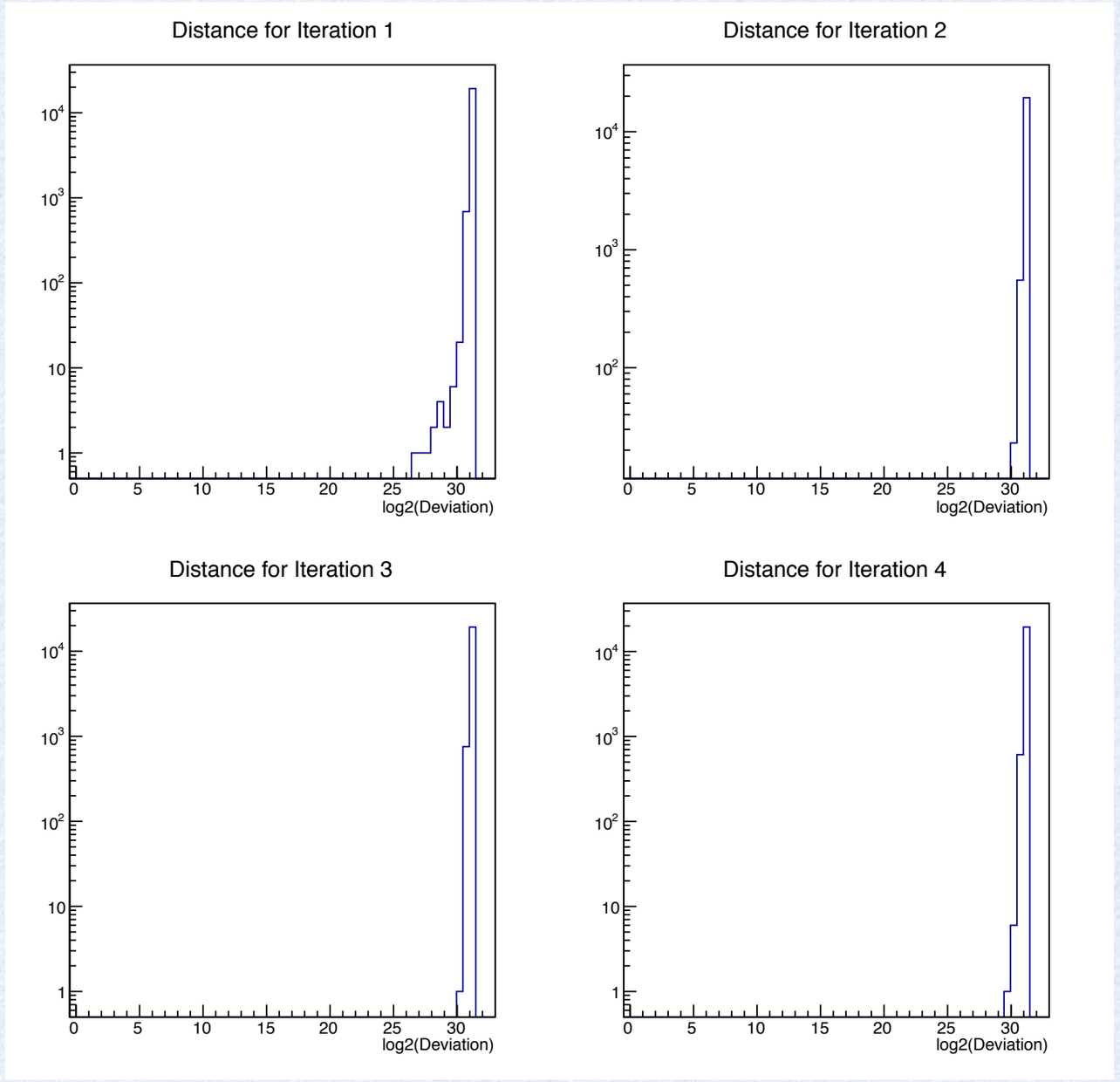
S = 48701323025609906 4



Distance for Mersenne-Twister

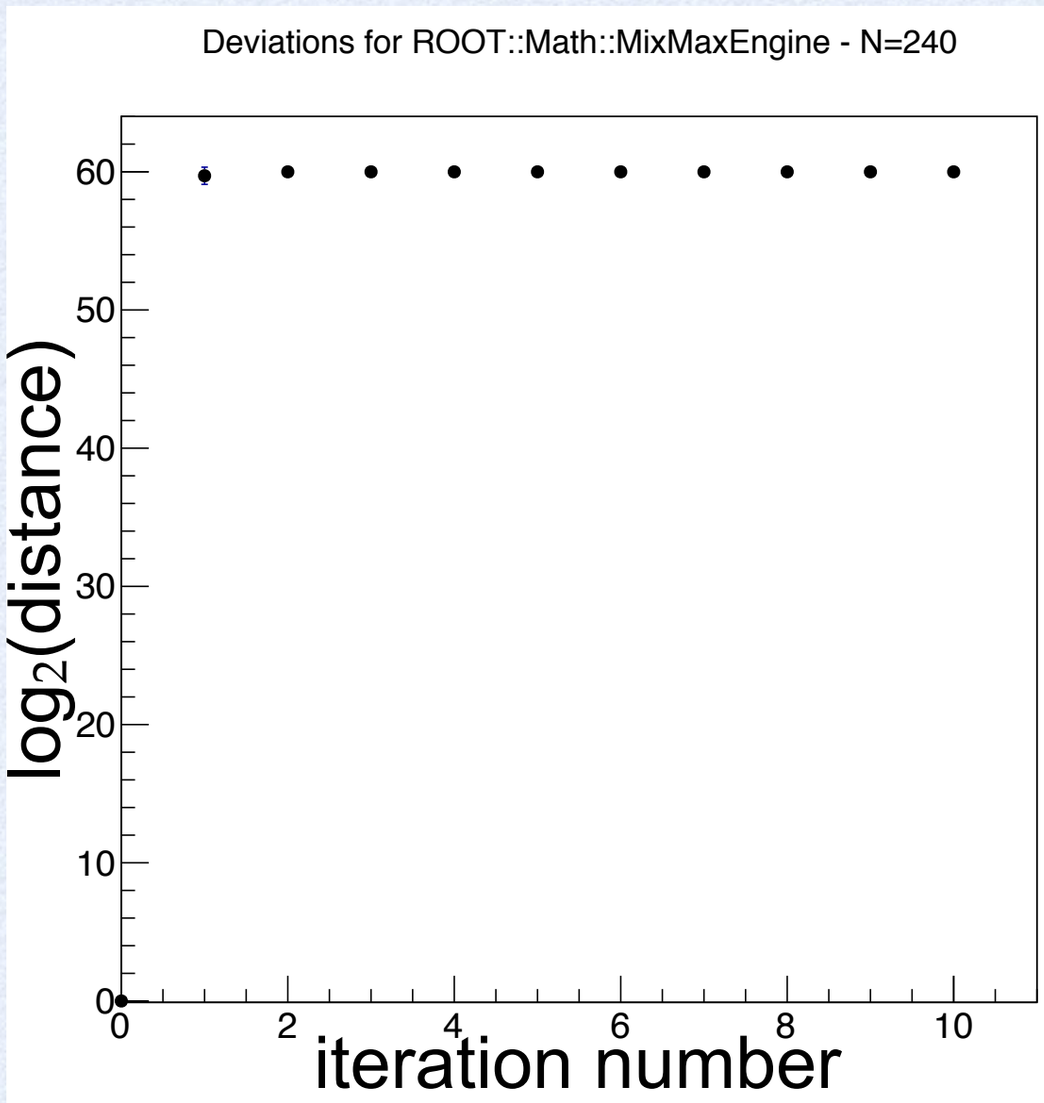
- 32 bits version of MT

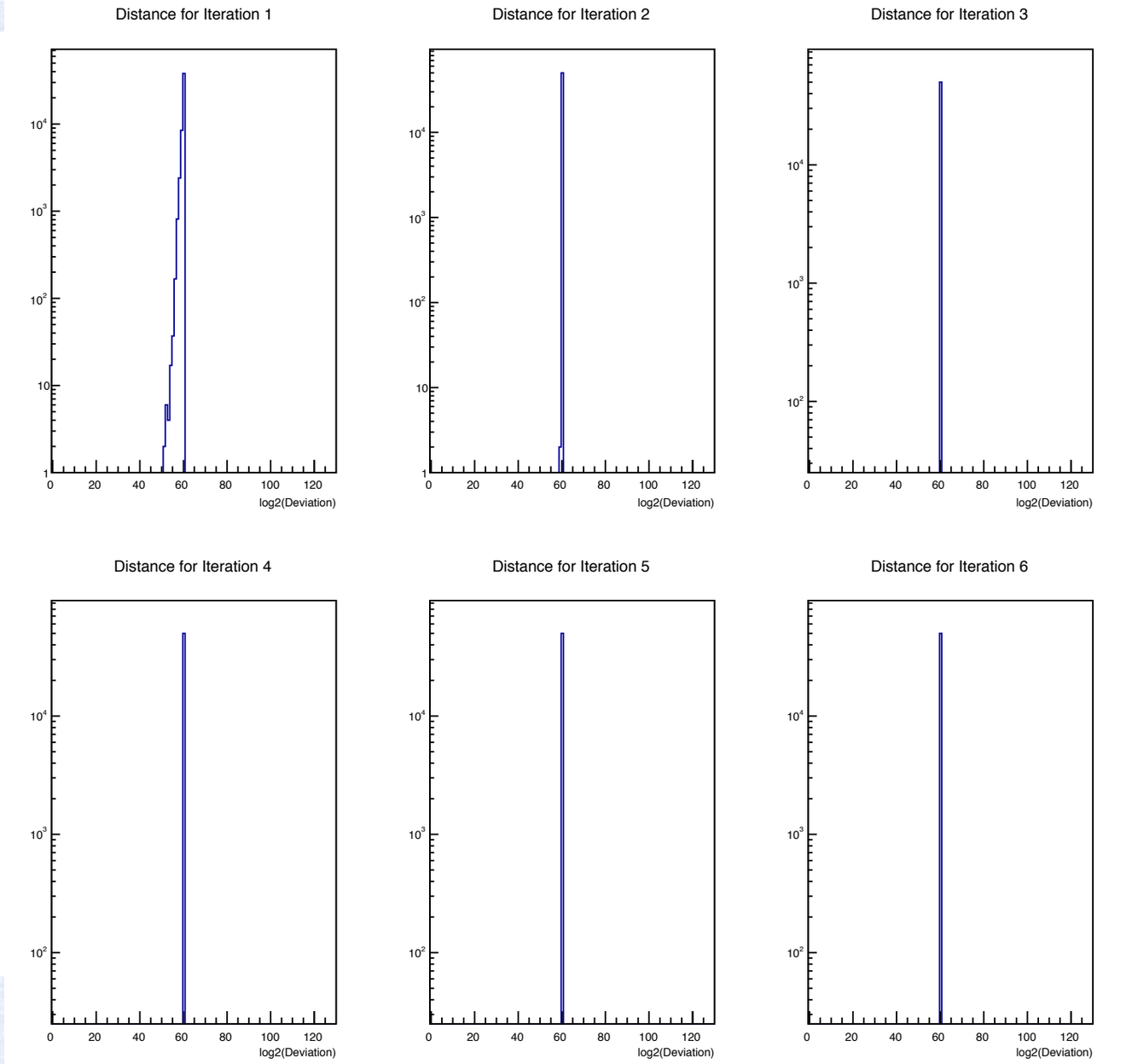




Distance for new MIXMAX (N=240)

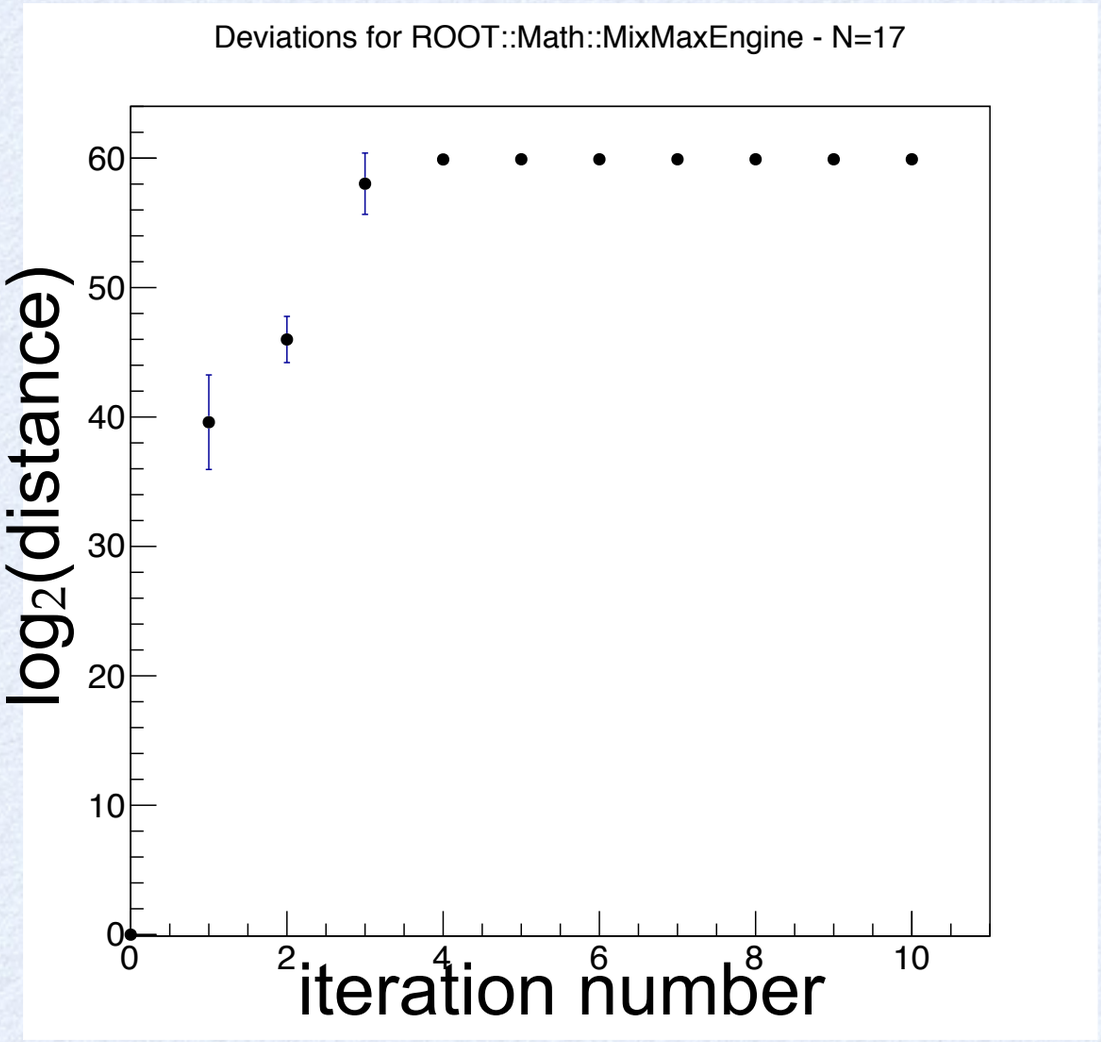
- N=240, s=4870132302560991140, m=$2^{51}+1$

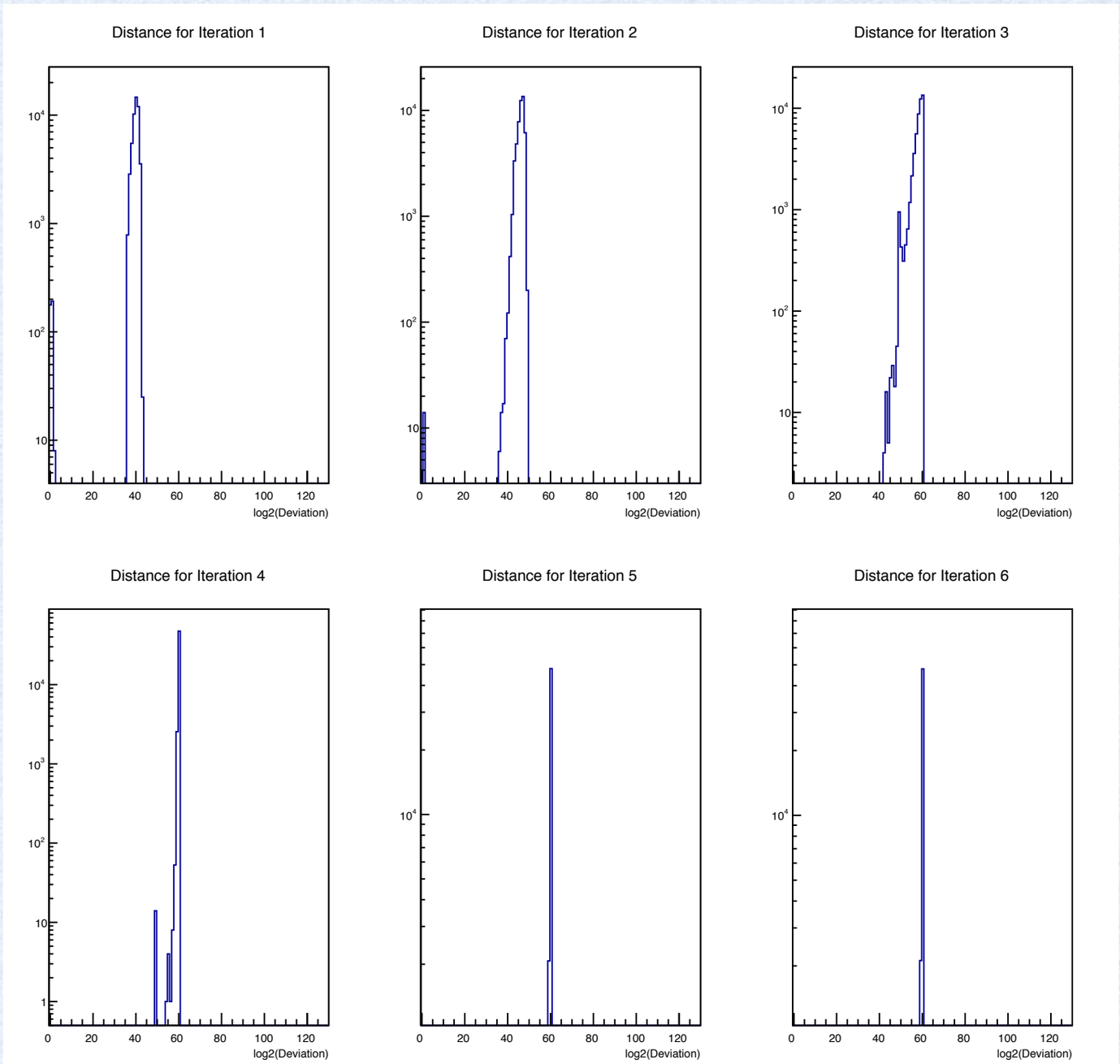




Distance for new MIXMAX (N=17)

- N=17, s=0, m=$2^{36}+1$

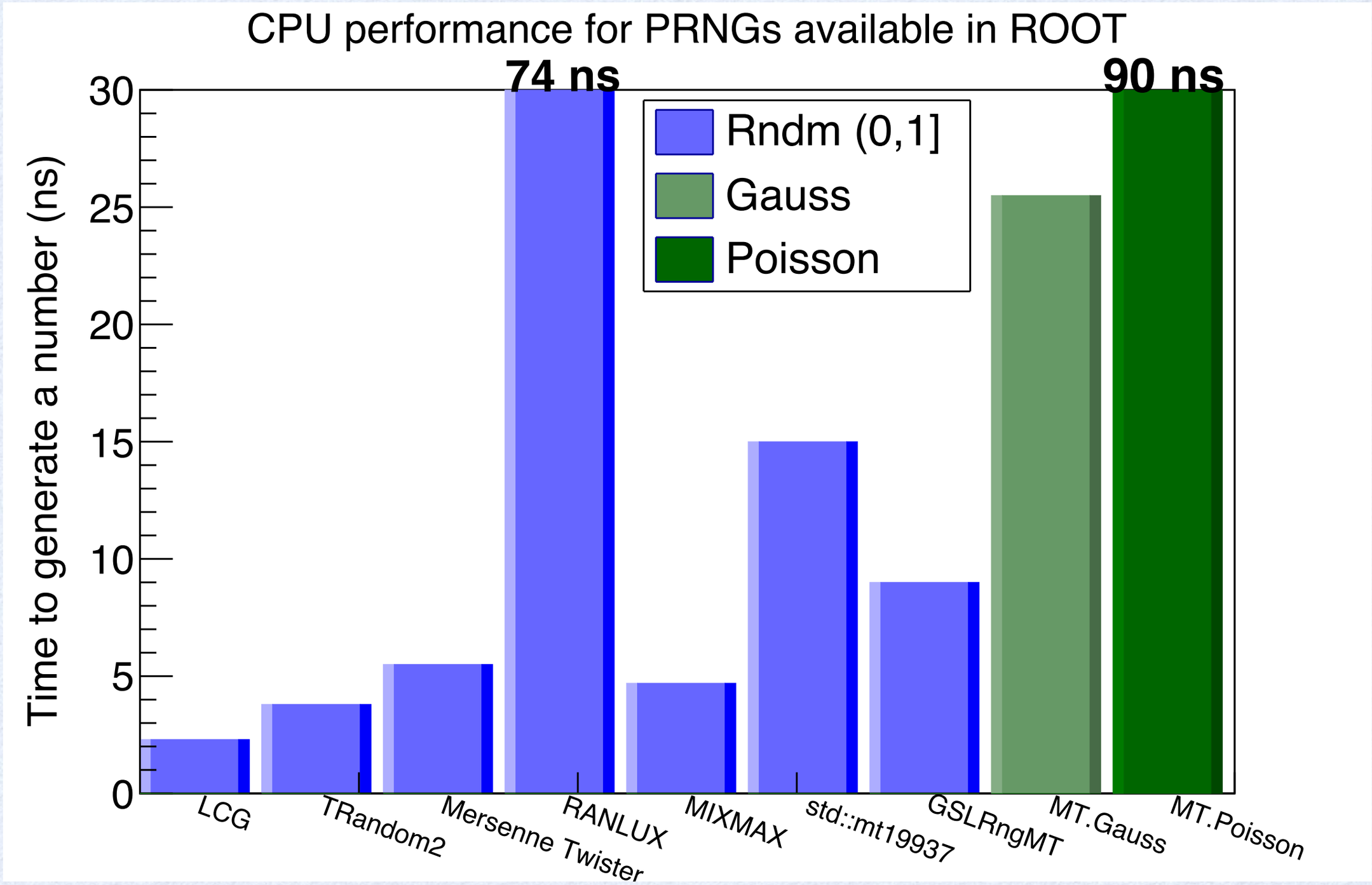




Decimation

- Independence (randomness) is guaranteed by the mixing (K-system)
- When maximum distance is reached the K-system is sufficient asymptotic
  - i.e. states are uncorrelated
  - Idea is then to discard some iterations to reach this state
  - We have modified MIXMAX ROOT version 1.0 to foreseen possibility to discard some iterations
    - Default value of 2 has been used (only one iteration in 3 is maintained) in ROOT version 6.06
  - We would like to have this capability directly inside C++ (e.g. using it as a template parameter)

PRNG CPU Performance

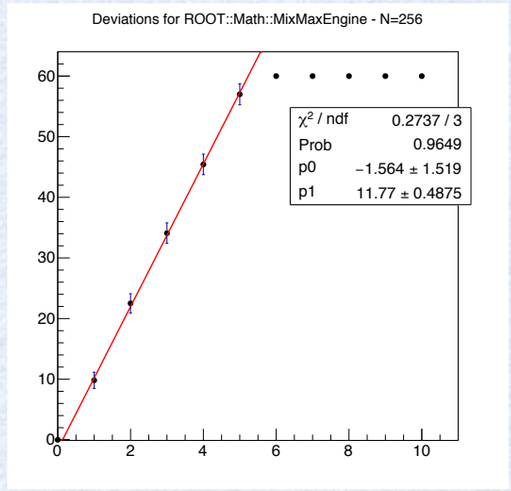


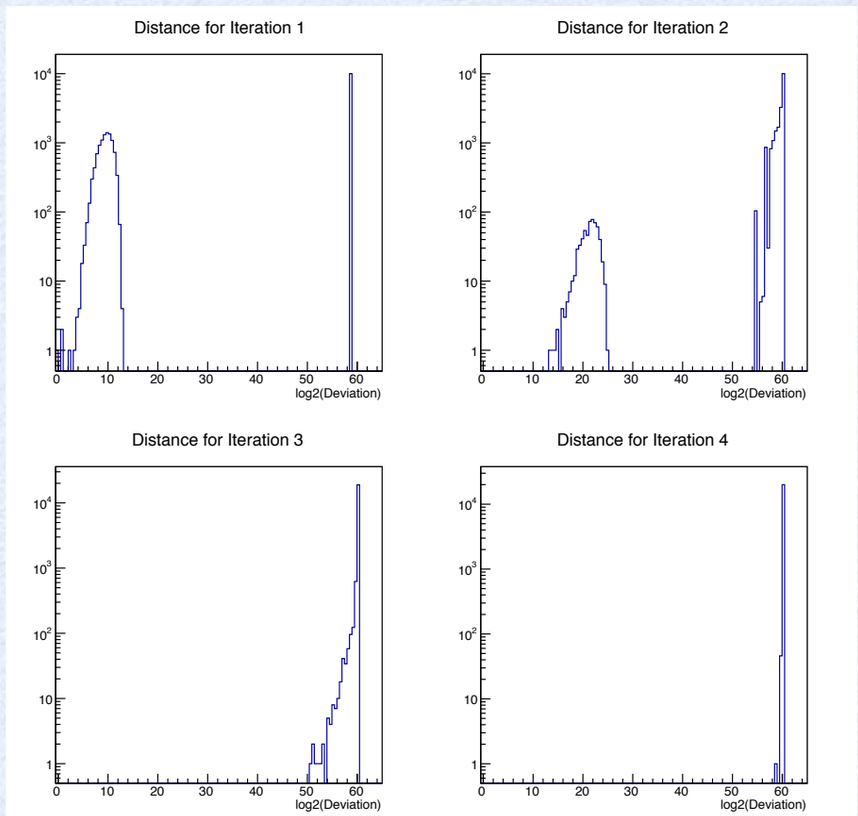
Further Tests of MIXMAX

- ROOT version of MIXMAX 1.0 (with decimation) has been tested by Jiri Hladki
  - used TestU01, PractRand and GJRand
  - no decimation: some tests failing with very small p-values
    - when skipping two iterations: all tests are passing
- MIXMAX version 1.1 (no decimation), N=255 fails also PractRand and GJRand tests
- MIXMAX version 2.0 with N=17,60,96,240 passes all the tests (according to test results communicated by J. Hladki)

Summary

- Confident of version of MIXMAX 1.0 with decimation (with skip number = 2)
- Version with large s (N=256 in MIXMAX 1.1) seems problematic
  - Large spread in distance observed
  - Failing some statistical tests
  - Doubts on using such large number **s** in the matrix





Summary (2)

- New generators (MIXMAX version 2.0) with large s and m values are passing all tests.
  - Do we still need skipping in this case ?
  - Need to gain more confidence on these new generators with large s and m
  - Objection of M. Luscher and F. James that by using large s and m the steps caused by the matrix multiplications are too large and the discrete-continuous approximation breaks.
    - this needs to be understood
- Are generators with large values of N needed ?

Future Plans

- Goal to have a full C++ implementation of MIXMAX
  - nice we have already a preliminary one in version 2.0
  - we need a parametric template generator where user can instantiate some pre-defined parameters
    - size of the state, N with corresponding values of s, m
    - decimation value
- Integrate same C++ version in both ROOT and Geant (CLHEP)
- Version management
  - code maintained in Github ?
    - would be clearer which version one is using

Conclusions

- Modernisation of random classes in ROOT
  - a first version with MIXMAX is available in ROOT 6.06 version
- Hope to reach a consensus on its parameters
  - having a family of generators
    - different N (256 from MIXMAX version 1.0)
    - generators from MIXMAX 2.0 (e.g N=240 and N=17)
    - would like to have also version with moderate s and m values
- Develop a C++ version which can be integrated in ROOT and Geant.
  - making it also the default random number generator in ROOT
  - and we will continue to perform statistical tests of the generators