README

Welcome to the new, computationally efficient implementation of MIXMAX, the random number generator!

MIXMAX is a matrix-recursive random number generator introduced in 1986
by my parents, George Savvidy and Natalia Ter-Arutyunyan-Savvidy.

[1] On the Monte Carlo simulation of physical systems
J.Comput.Phys. 97, 566 (1991),
http://dx.doi.org/10.1016/0021-9991(91)90015-D (published journal version in English)
https://lib-extopc.kek.jp/preprints/PDF/1986/8607/8607219.pdf (preprint version from January 1986)

and, my own paper, studying the period and dynamics of the generator:

[2] The MIXMAX random number generator
Comp. Phys. Commun. 196 (2015), pp 161–165
http://dx.doi.org/10.1016/j.cpc.2015.06.003

a recent review by George Savvily:

[3] Anosov C-systems and random number generators
Theor.Math.Phys. 188 (2016) 1155-1171,
Teor.Mat.Fiz. 188 (2016) 223-243
https://link.springer.com/article/10.1134%2FS004057791608002X
http://arxiv.org/abs/1507.06348

and, a recent paper with the three-parameter MIXMAX family

[4] K. Savvidy and G. Savvidy,
``Spectrum and Entropy of C-systems. MIXMAX random number generator,''
Chaos, Solitons & Fractals, Volume 91, (2016) pp. 33–38
http://dx.doi.org/10.1016/j.chaos.2016.05.003

WHY USE MIXMAX?

If you are doing large sized Monte-Carlo simulations, you should consider using MIXMAX. Examples are particle physics, lattice gauge theory, statistical physics or whenever the quantity of random numbers to simulate each event, trajectory, MCMC or Metropolis update and so on is large. In the case of the Metropolis updates near the critical point, or more generally while simulating arbitrary Markov Chains with long correlation lengths, the dimension of the generator must be larger than the correlation length times the consumption per update. How large are the typical generators? To give an idea, the widely used RANLUX has a state vector of size 24. The Mersenne twister is larger at 623, but not recommended for physics due to very long correlation lengths in the sequence, as explained below in the THEORY section. Unlike generators designed by computer scientists which tend to emphasize equi-distribution and randomness in terms of bits, MIXMAX has strong theoretical guarantees in terms of its floating point output. Multiple streams are guaranteed by theory to be statistically independent. The default dimension used in this implementation is N=240, but we invite the user to increase it as necessary if the Monte-Carlo dimension is larger. At the moment the largest dimension which we provide is N=44851, and for all of the recommended values of N the period is independent of the seed. If you need to make random matrixes of size M x M, you are safest to choose an $N > M^2$. In the case of an arbitrary large value of N, the period will be astronomic but different for different seeds.

WHAT IS MIXMAX?

The random number generator was an outgrowth of research on dynamical systems, namely Yang-Mills classical mechanics. It was noted in the first paper that if a dynamical system possessed the property of Kolmogorov's mixing, then it could be used to generate pseudo-random numbers. A specific system with discrete time was proposed [1], by means of a linear automorphism of a hypercube, $x_{i+1} = A . x_i \bmod 1$ where A is a matrix with integer entries. For mixing, it is required that the determinant is equal to one, and that all eigenvalues are different by absolute value from 1. A specific realization with these properties is the three-parameter (N,m,s) family of MIXMAX matrixes:

$$\begin{matrix}
2 & m+2 & 2m+2 & 3m+2 & \cdots & (N-2)m+2 & 1 \\
1 & 2 & m+2 & 2m+2 & \cdots & (N-3)m+2 & 1 \\
1 & 1 & 2 & m+2 & \cdots & (N-4)m+2 & 1 \\
 & \cdots & & & & & \\
1 & 1 & 1 & 1\ \cdots & 2 & m+2+s & 1 \\
1 & 1 & 1 & 1\ \cdots & 1 & 2 & 1 \\
1 & 1 & 1 & 1\ \cdots & 1 & 1 & 1
\end{matrix}$$

The Matrix contains natural numbers and is defined recursively for all N>1, since the Matrix of size N shares everything except the first column and first row with the Matrix of size N-1. The eigenvalues of the Matrix are widely dispersed and none should lie very close to the unit circle. The largest eigenvalue appears to grow fast with N, and Kolmogorov's entropy is $O(N \log(m))$. Thus, the spectrum of this system is multi-scale, with trajectories exhibiting exponential instabilities on all time-scales.

Some new parameter combinations which give RNGs with excellent statistical properties and performance are offered in this release (v2.1, March 2018). The new default, which passes all statistical tests which we have run, is N=240 and m=3141592653589847 and s=0, and has much larger entropy. The specific value of s and m is chosen such that the generator has a maximal period.

Other parameter combinations also make use of the new ability of setting non-power-of-two calues of m, in particular, we now have generators with a small state in order to lower the memory requirements such as the N=17, $m=2^{52}+2^{44}+1$, s=0 and the N=19, $m=2^{53}+2^{45}+1$, s=0 generators.

The original generator was based on this recursion with real numbers on [0,1), generating directly in floating point. The trajectories which have rational components are periodic and can be simulated on the computer exactly. The present implementation applies to vectors with rational components of the form $x_i = a_i / p$, where p is the Mersenne prime $2^{61}-1$. The real arithmetic modulo 1 is equivalent exactly to the integer arithmetic modulo p: if

$x = a/p$ and $y=b/p$

then

$x+y \bmod 1 = (a+b \bmod p)/p$

INSTALLATION

On Linux, Unix, and Mac OSX systems, unzip the archive and change into the directory:

```
unzip mixmax_release_NNN.zip
cd mixmax_release_NNN
```

The implementation of the generator is in the file mixmax.c and there are various example driver programs included. Type make, and hopefully you will have an executable called mixmax which can be run:

```
make
./mixmax
```

At this point, it will ask you to enter the number of floating point numbers to produce, and then print them one on a line, in 18 decimal digit precision which is the actual usable precision in this implementation of the generator.

Next, I encourage the user to run the test suite by typing

```
make check
```

With the latest optimizations, including an assembler code suggested by Andrzej Görlich from NBI, the speed is comparable to MT19937. It appears that GCC-5 is able to better optimize our code than the earlier versions of gcc or clang.

USAGE

A few example programs for using the generator are provided. It can be as simple as allocating the generator state, seeding it, and getting values out of it (see driver_main.c):

```
#include "mixmax.h"
rng_state_t S;
rng_state_t *X = &S;
seed_spbox(X, 12345);
double z=get_next_float(X);
```

You are permitted to request the double-precision floating point with get_next_float() and integer numbers with get_next() in any order you may need. If you need 32 bit integers, it is ok to simply cast it to uint32_t.

Next, a program (see driver_testU01.c) is provided for testing the generator with the testU01 suite (contains BigCrush), if it is installed on the system. Go ahead and run it on your own system, just to be sure.

Third, there is now a method for initializing and running huge simulations, which absolutely guarantees the non-collision of different streams by a system of hierarchical skipping. You provide the four initialization ID's and the function seed_uniquestream will make a skip by some large number of steps calculated from the four ID's such that the substream derived from it is absolutely guaranteed to not collide with any other stream produced from another four ID's so long as

1) at least one bit in at least one of the IDs is different.
2) less than 10^100 numbers are drawn
(this is good enough : a single CPU will not exceed this in the lifetime of the universe, 10^19 sec,
even if it had a clock cycle of Planck time, 10^44 Hz)

C++

The C++ code is provided as a simple include, in the file mixmax.hpp. Compilation requires the C++11 standard features, so the C++ code will not compile with gcc versions older than 4.7. Example usage is in the file example.cpp which can be compiled with

```
make plusplus
./mixmax.plusplus
```

Roughly speaking, you include the header, initialize the generator and then get random numbers:

```
#include "mixmax.hpp"
mixmax_engine gen{0,0,0,1};
double z=gen.get_next_float();
std::cout << std::setw(18) << std::setprecision(18) <<  z << std::endl;
```

Alternatively, you can use the std::random interface:

```
static std::uniform_real_distribution<double> dist{0,1};
std::cout << dist(gen) << std::endl;
```

WRAPPERS

I am adding a GSL scientific library interface and an example usage in driver_gsl.c. Just do “`make gsl; GSL_RNG_TYPE=MIXMAX ./gsl`“. There exists a ROOT and a CLHEP interface as well. In version 2.0 we have included a new C++ code which contains the C++11 standard interface for random numbers.

PORTABILITY

The generator works on most 64-bit systems, this includes both 64-bit Linux flavors and Intel Mac. It has also been run on Mac OSX systems with PPC architecture. The latest version also runs on 32-bit systems and on Windows. If you require a good quality generator which will work efficiently on embedded 8- or 16-bit systems, let me know and I will see what I can do for you. As well, if you require a vectorized version with substantially higher performance, it can be provided but requires one-off
integration work which I can provide.

It has been recently tested extensively on very wide variety of platforms, as part of the release of ROOT, and so it is safe to say that it successfully compiled on several recent vintages of the GNU compiler gcc and clang, and also on the Intel compiler icc.

The implementation uses bit shifts to implement the modular arithmetic, nevertheless, it produces the same output on big-endian and small-endian machines. This has been specifically tested on ppc. The program outputs uint64_t integers between 1 and 2^61-1 inclusive and therefore the 61 lower bits are usable. The 61 bits of precision is more than the double precision and is good to 18 decimal.

The facilities to allocate and initialize instances of the generator have been provided, and are thread-safe. The generator state has a filehandle associated with it, in order to direct output from different threads.There is now an example program (driver_threads.c, to run type "make multi; ./multi") which initializes several threads and outputs each RNG stream to a different file.

THEORY

Ergodicity is the property of a dynamical system such that space averages are equal to time averages. This is what allows to obtain the value of the observable by Monte-Carlo by averaging over a single trajectory. When we want to speed up, by using multiple CPUs or threads to simulate multiple trajectories, we need the k-mixing property. MIXMAX is a chaotic dynamical system which has the even stronger properties of k-mixing of Kolmogorov and as well is a C-system of Anosov. C-systems of Anosov are the ultimate in chaotic properties - the dynamics is strictly hyperbolic in all of the phase space.

Study of this recursion on a Galois field GF[p] is a complicated subject [7], suffice to say that for maximum period the characteristic polynomial of the matrix should have its eigenvalues as the primitive roots of the root-N-extended field GF[Root[p,N]]. Recursions based on matrices whose characteristic polynomial is a primitive trinomial in GF[2^D] were advocated by Niederreiter [N86] and have made their way into mainstream [MT98] in the form of Mersenne twisters of various D, for example D=19937. This is entirely due to the ease of finding primitive polynomials whenever 2^D-1 is a prime, namely it is sufficient that the polynomial is irreducible. The drawback is that all of the eigenvalues get extremely close to the unit circle, and this gets worse (!) with large D.

When the modulus, p, is not equal to 2, and the determinant of the matrix is equal to one, the standard theory is not applicable. Extension of the theory to this case was done in the paper [2] and the maximum period is equal to q=(p^N-1)/(p-1).

The eigenvalues of the matrix used by MIXMAX are well-separated from unity for all the recommended N. Fortunately, in the current implementation [2] the computational complexity is of order O(N), rather than O(N^2) of the original or the O(N ln(N)) of the improved implementation (ref. [7]). This means that per random number generated, the cost is constant and does not increase with N.

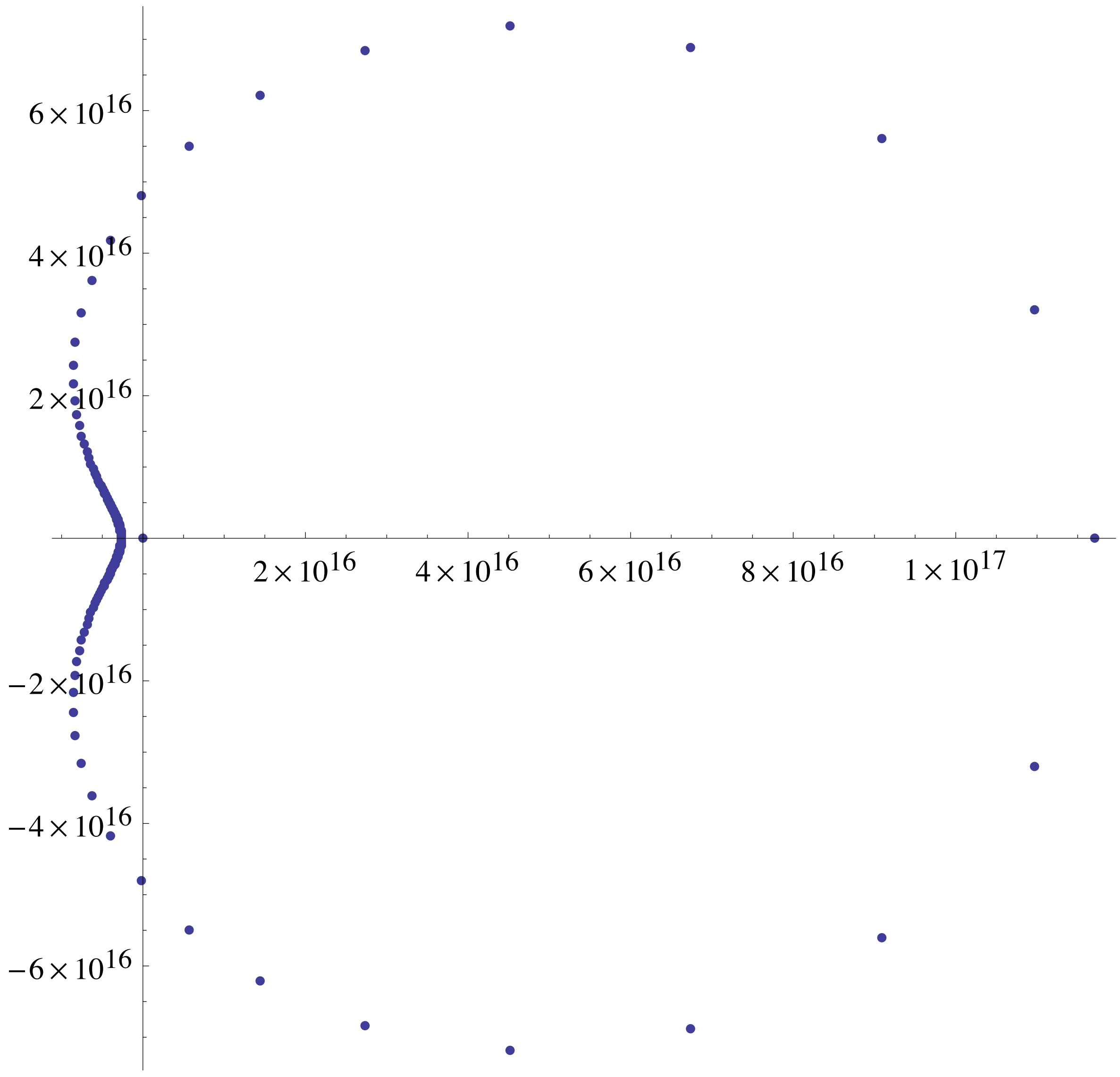

FIG 1: the spectrum of the matrix for N=240

One additional new feature, compared to other multiple recursive generators is the function to skip over some large number of steps k by using the precomputed and stored matrix (A^k mod p).

STATISTICAL TESTS

We have used the very high quality suite of tests, testU01. In the latest release, the generators with all N use the three-parameter family with a nontrivial m, and so pass all tests in BigCrush for all the provided N.

LITERATURE

In addition to the two papers where the generator was introduced, there exists among other the following literature which is most closely related. Niederreiter has proposed using the matrix recursion and a realization on finite fields in 1986 [N86]. Martin Luscher found in [LJ94] the improvement to RCARRY which was sufficient in practice to overcome the high correlations intrinsic to generators based on sparse matrices with many eigenvalues close to the unit circle (such as both RCARRY and MT19937). Luscher's RANLUX accomplishes this by skipping over the sequence which moves the eigenvalues further away from the unit circle at the cost of speed.

[N86] H.Niederreiter,

A pseudorandom vector generator based on finite field arithmetic,
Mathematica Japonica, Vol. 31, pp. 759-774, (1986), see also
"Finite fields, pseudorandom numbers, and quasirandom points,"
in : Finite fields, Coding theory, and Advance in Communications and Computing.
(G.L.Mullen and P.J.S.Shine, eds) pp. 375-394, Marcel Dekker, N.Y. 1993.

[5] Matrix generator of pseudorandom numbers
J.Comput.Phys.97, 573 (1991)
http://dx.doi.org/10.1016/0021-9991(91)90016-E (published journal version in English)
http://ccdb5fs.kek.jp/cgi-bin/img/allpdf?198607220 (preprint version from January 1986)

[LJ94] M. Luscher, Computer Physics Communications 79 (1994) 100
F. James, Computer Physics Communications 79 (1994) 111

[7] "K-system generator of pseudorandom numbers on Galois field,"
G. G. Athanasiu, E. G. Floratos, G. K. Savvidy
arXiv:physics/9703024
International Journal of Modern Physics C, Volume 8, Issue 03, pp. 555-565 (1997).

[MT98] M. Matsumoto and T. Nishimura,
"Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator",
ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 (1998)
DOI:10.1145/272991.272995

[9] Two early works of which I just recently became aware of, one in Japanese and the other in French.
If you happen to have a paper or electronic copy of either of the two, please let me know!

N. Niki,
Finite field arithmetic and multidimensional uniform pseudorandom numbers (Japanese),
Proc. Inst. Statist. Math. 32 (1984) 231–239.

E.-H. A. D. E. Tahmi, Contribution aux generateurs de vecteurs pseudo-aleatoires,
These, Univ. Sci. Techn. Houari Boumedienne, Algiers, 1982.

TIMELINE

January, 1986
the original papers are out

July, 1987
Akopov talks to Fred James at CERN

December, 1991
the original papers are published in JCP

1994
Luscher publishes his method of improving the RCARRY generator by means of skipping

December 2004
I find a way to implement the matrix recursion without explicitly using the matrix,

the computational complexity becomes O(N)

November 24, 2012
the initial version 0.01 is released on hepforge.org

January 2015 - current
This work is supported in part by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie Grant Agreement No 644121.

July 25, 2015
Version 1.0 is released.

Sept 14, 2015
the generator is released as part of the ROOT package from CERN.

November 10th, 2015
released as part of CLHEP package

August 29, 2016
a 2.0beta version is released with a new C++ implementation

September, 2017
2.0 final version is relased, includes a C++ implementation in a simple include.
Software is free to use, LGPL license no longer available, other licenses are available -- please enquire

March, 2018
version 2.1, see the Release Notes, with improved generators that pass the spectral analysis criteria of the new L'Ecuyer etal paper:
https://hal.inria.fr/hal-01634350

In the not too distant future -
a AVX vectorized version

Email me, Konstantin Savvidy, ksavvidis @at@ gmail.com