Computational Physics 2023

Sommersemester, 3th April, 2023 – 14th Juli, 2023

1)Introduction

2)Numbers and errors

3)Differentiation and integration

4)Ordinary differential equations

5)Molecular dynamics simulations

6)Partial differential equations

7)Iteration processes

8)Matrixdiagonalisation & Eigenvalue problems

9)Minimization

10) **Random numbers**

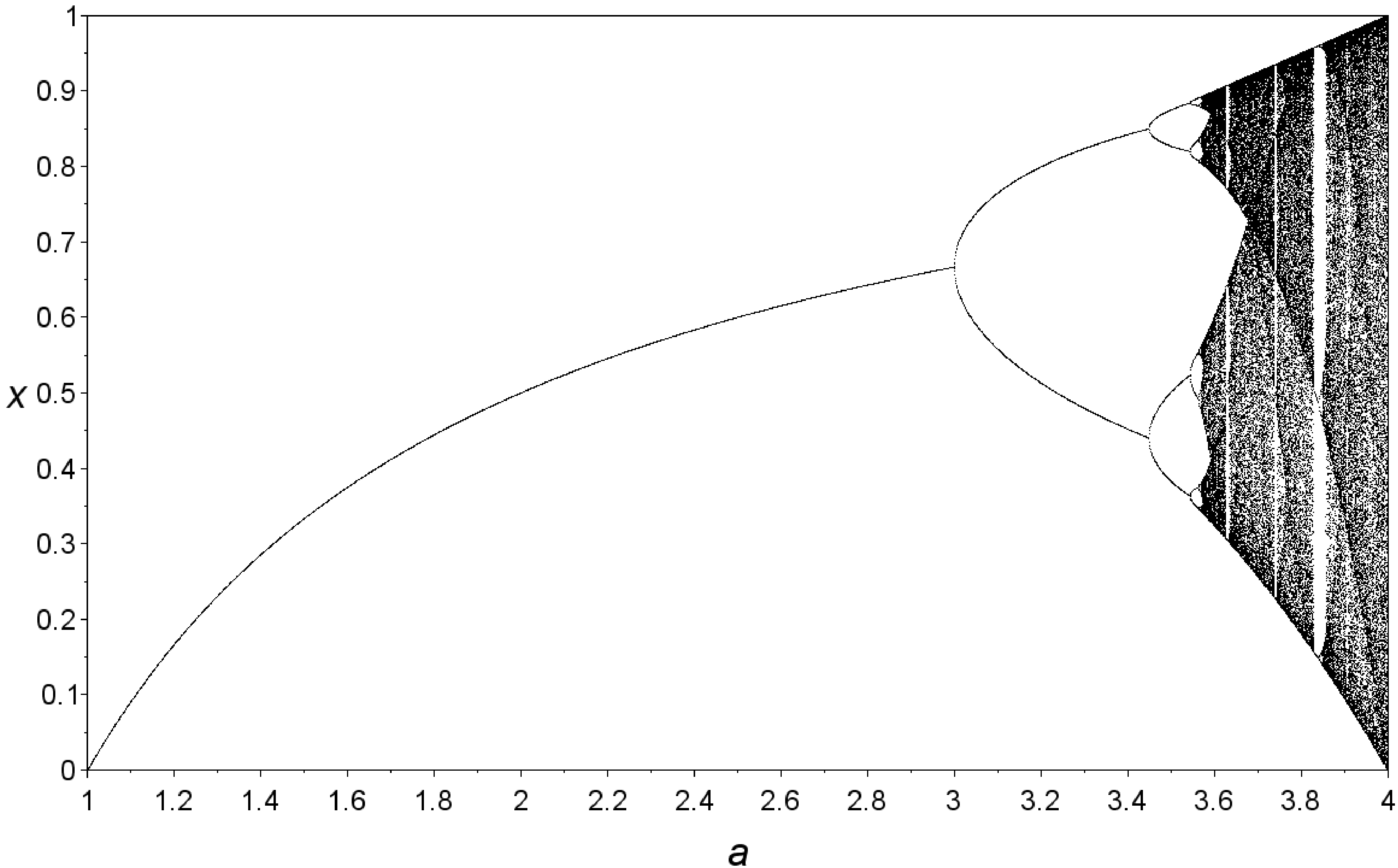
11) Monte Carlo (MC) Simulations

12)Perkolation

13)Stochastic Dynamics

technische universität
dortmund

Logistic Map



Pseudo Random Generators produce periodic sequences

technische universität
dortmund

Linear Congruential Generator

https://en.wikipedia.org/wiki/Linear_congruential_generator

Linear congruential generator - Wikipedia — Mozilla Firefox

Linear congruential gene × | lecture_4.pdf ×

https://en.wikipedia.org/wiki/Linear_congruential_generator

Most Visited | Fedora Project | Alex Christoforou - ... | Red Hat | Free Content | synaptotagmin - G... | 11043.full.pdf | Mail - herre.rissela... | Frontiers | Decline I... | Other Bookmarks

Parameters in common use [edit]

The following table lists the parameters of LCGs in common use, including built-in *rand()* functions in runtime libraries of various compilers. This table is to show popularity, not examples to emulate; *many of these parameters are poor.* Tables of good parameters are available.[10][2]

Source	modulus *m*	multiplier *a*	increment *c*	output bits of seed in *rand()* or *Random(L)*
ZX81	$2^{16} + 1$	75	74	
Numerical Recipes	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30..16 in *rand()*, 30..0 in *lrand()*
glibc (used by GCC)[17]	2^{31}	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++ [18] C90, C99, C11: Suggestion in the ISO/IEC 9899,[19] C17	2^{31}	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63..32 of *(seed × L)*
Turbo Pascal	2^{32}	134775813 (8088405_{16})	1	
Microsoft Visual/Quick C/C++	2^{32}	214013 ($343FD_{16}$)	2531011 ($269EC3_{16}$)	bits 30..16
Microsoft Visual Basic (6 and earlier)[20]	2^{24}	1140671485 ($43FD43FD_{16}$)	12820163 ($C39EC3_{16}$)	
RtlUniform from Native API[21]	$2^{31} - 1$	2147483629 ($7FFFFFED_{16}$)	2147483587 ($7FFFFFC3_{16}$)	
Apple CarbonLib, C++11's `minstd_rand0` [22]	$2^{31} - 1$	16807	0	see MINSTD
C++11's `minstd_rand` [22]	$2^{31} - 1$	48271	0	see MINSTD
MMIX by Donald Knuth	2^{64}	6364136223846793005	1442695040888963407	
Newlib, Musl	2^{64}	6364136223846793005	1	bits 63..32
VMS's **MTH$RANDOM**,[23] old versions of glibc	2^{32}	69069 ($10DCD_{16}$)	1	
Java's java.util.Random, POSIX [ln]rand48, glibc [ln]rand48[_r]	2^{48}	25214903917 ($5DEECE66D_{16}$)	11	bits 47..16
`random0` [24][25][26][27][28]	134456 = $2^3 7^5$	8121	28411	$\frac{X_n}{134456}$
POSIX[29] [jm]rand48, glibc [mj]rand48[_r]	2^{48}	25214903917 ($5DEECE66D_{16}$)	11	bits 47..15
POSIX [de]rand48, glibc [de]rand48[_r]	2^{48}	25214903917 ($5DEECE66D_{16}$)	11	bits 47..0
cc65[30]	2^{23}	65793 (10101_{16})	4282663 (415927_{16})	bits 22..8

Marsaglia effect

https://en.wikipedia.org/wiki/George_Marsagli
a

George Marsaglia

From Wikipedia, the free encyclopedia

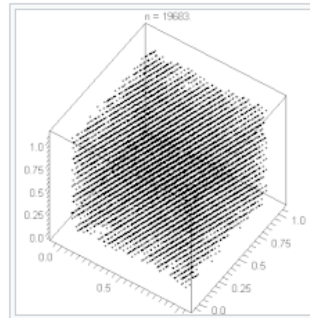
George Marsaglia (March 12, 1924 – February 15, 2011)[1] was an American mathematician and computer scientist. He is best known for creating the diehard tests, a suite of software for measuring statistical randomness.

George Marsaglia	
Born	March 12, 1924 Denver, Colorado
Died	February 15, 2011 (aged 86) Tallahassee, Florida
Nationality	American
Alma mater	Ohio State University
Scientific career	
Fields	Mathematics
Institutions	Florida State University Washington State University
Doctoral advisor	Henry Mann

Contents [hide]

Research on random numbers [edit]

George Marsaglia established the lattice structure of linear congruential generators in the paper "Random numbers fall mainly in the planes",[2] later termed the Marsaglia's theorem.[3] This phenomenon means that *n*-tuples with coordinates obtained from consecutive use of the generator will lie on a small number of equally spaced hyperplanes in *n*-dimensional space.[4] He also developed the diehard tests, a series of tests to determine whether or not a sequence of numbers have the statistical properties that could be expected from a random sequence. In 1995 he published a CD-ROM of random numbers, which included the diehard tests.[5]



Visual demonstration of Marsaglia's theorem

His diehard paper came with the quotation "Nothing is random, only uncertain" attributed to *Gail Gasram*, though this name is simply the reverse of *Marsaglia G*, and so likely to be a pseudonym.

He also developed some of the most commonly used methods for generating random numbers and using them to produce random samples from various distributions. Some of the most widely used being the multiply-with-carry, subtract-with-borrow, xorshift, KISS and Mother methods for random numbers, and the ziggurat algorithm for generating normally or other unimodally distributed random variables.

Life [edit]

He was Professor Emeritus of Pure and Applied Mathematics and Computer Science at Washington State University and Professor Emeritus of Statistics at Florida State University.

In the 1995 CD-ROM release of diehard, Marsaglia included several papers that outline the process by which the random number files were created. In several places he mentions that, along with deterministic and physical devices:

Lavarand

https://en.wikipedia.org/wiki/Lavarand

Xorshift

https://de.wikipedia.org/wiki/Xorshift

Xorshift – Wikipedia — Mozilla Firefox

```c
    s1 ^= s1 << 31; // a
    s1 ^= s1 >> 11; // b
    s1 ^= s0 ^ (s0 >> 30); // c
    s[p] = s1;
    return s1 * (uint64_t)1181783497276652981;
}
```

Xorshift+ [Bearbeiten | Quelltext bearbeiten]

Statt der Multiplikation kann man auch die in der Regel schnellere Addition als nichtlineare Transformation einsetzen. Diese Idee wurde zuerst von Saito und Matsumoto (von denen auch der Mersenne Twister stammt) vorgeschlagen, und zwar im XSadd-Generator, der zwei aufeinanderfolgende Ausgaben eines zugrundeliegenden Xorshift addiert.[10]

XSadd hat Schwächen in den niederwertigen Ausgabebits und fällt bei einigen BigCrush-Tests durch, wenn man die Ausgabewörter invertiert, also die niederwertigsten Bits an die höchste Stelle setzt und umgekehrt. Als Abhilfe hat Vigna[11] die Xorshift+ Familie konstruiert, die mit 64-Bit-Wörtern arbeiten: nachfolgender Xorshift128+ nutzt 128 Zustandsbits und hat eine Periodenlänge von $2^{128}-1$. Er besteht BigCrush, auch bei Invertierung.

```c
#include <stdint.h>

uint64_t s[2]; // nicht komplett mit 0 initialisieren

uint64_t xorshift128plus(void) {
    uint64_t x = s[0];
    uint64_t const y = s[1];
    s[0] = y;
    x ^= x << 23; // a
    s[1] = x ^ y ^ (x >> 17) ^ (y >> 26); // b, c
    return s[1] + y;
}
```

Es ist einer der schnellsten Generatoren, die BigCrush bestehen.[12] Ein Nachteil der Addition von aufeinanderfolgenden Ausgabewörtern ist, dass der Generator so nur noch in einer Dimension gleichverteilt ist, obwohl dies für den zugrundeliegenden Xorshift in 2 Dimensionen gilt[11]

Xoroshiro und Xoshiro [Bearbeiten | Quelltext bearbeiten]

Diese von Sebastiano Vigna und David Blackman entwickelten Generatoren basieren auf der gleichen Theorie wie Xorshift.[13] Sie enthalten jedoch auch die Bitrotation als elementare Operation. Nachfolgende Versionen haben eine Periodenlänge von $2^{128}-1$.[14]

```c
#include <stdint.h>

inline uint64_t rotl (uint64_t a, int w) {
   return a << w | a >> (64-w);
}
```