

# Der Zufall im Computer

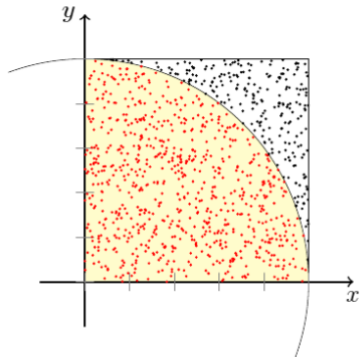
Hannes Nikulski

TU Dresden

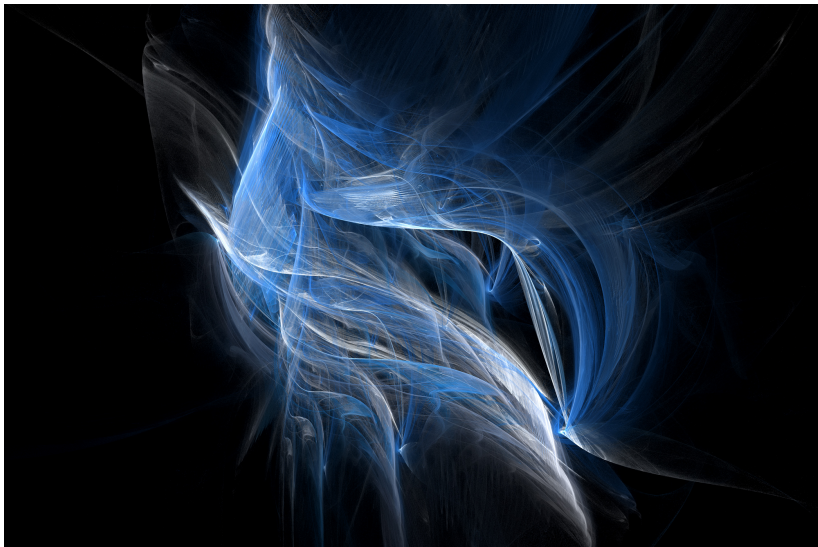
10. Januar 2022

# Zufall?

- Wozu denn?
  - Simulation
  - Numerische Analyse
  - Spiele
  - Kryptografie
  - Kunst
  - ...
- Welcher Zufall?
  - Zahlen in  $(0, 1)$
  - Gleichverteilt
  - Unabhängig





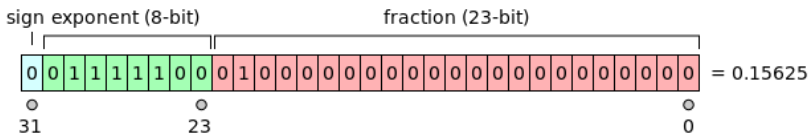


# Echter Zufall

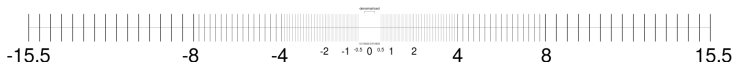
- Algorithmen sind deterministisch
  - Erzeugung von Pseudozufallszahlen
- Echter Zufall durch Messung physikalischer Phänomene
  - Tastatureingaben
  - Mausbewegungen
  - Radioaktiver Zerfall
  - Würfeln

# Zahlen im Computer

## ■ IEEE Darstellung (32-Bit)



## ■ Dichte der Zahlen



## ■ Erzeuge ganze Zahlen mit späterer Normalisierung

# Lineare Kongruenzgeneratoren

Ein Linearer Kongruenzgenerator (LKG) hat die Form

$$x_{i+1} \equiv (ax_i + c) \bmod m$$

- Startwert (Seed)  $0 \leq x_i < m$
- Faktor (Multiplier)  $0 \leq a < m$
- Modul  $m$
- maximale Periode:  $m$
- $u_i := \frac{x_i}{m} \in [0, 1)$



# Periode

- Multiplikativer Kongruenzgeneratoren für ( $c = 0$ )

$$x_{i+1} \equiv ax_i \bmod m$$

- maximale Periode:  $m - 1$
- Wegen  $x_{i+k} \equiv a^k x_i \bmod m$  Wiederholung bei

$$1 \equiv a^k \bmod m$$

# Satz von Euler

## Theorem

Seien  $a, n \in \mathbb{N}$  teilerfremd. Dann gilt

$$1 \equiv a^{\phi(n)} \pmod{n}.$$

## Beweis.

Sei  $k := \phi(n)$ . Dann gilt  $(\mathbb{Z}/n\mathbb{Z})^\times = \{r_1, \dots, r_k\}$  (zu  $n$  teilerfremden Zahlen kleiner als  $n$ ). Die Multiplikation mit  $a$  ist eine Permutation auf  $(\mathbb{Z}/n\mathbb{Z})^\times$ . Es folgt

$$r_1 \cdots r_k \equiv ar_1 \cdots ar_k \equiv a^k r_1 \cdots r_k \pmod{n}$$

Wir erhalten  $1 \equiv a^{\phi(n)} \pmod{n}$ . □

# Maximale Periode

Wir suchen  $a$ , sodass  $k$  in

$$1 \equiv a^k \pmod{m}$$

möglichst groß wird.

- gilt  $k = \phi(m)$ , so heißt  $a$  **Primitivwurzel**
- $a$  heißt **Primitives Element**, wenn es für gegebenes  $m$ , größtes  $k$  liefert
- Oft genutzt werden Mersenne Primzahlen als Modulus

# Implementation in Python

```
class LinearCongruentialGenerator:
    def __init__(self, multiplier: int, modulus: int, seed: int) → None:
        self.a = multiplier
        self.m = modulus
        self.x = seed

    def random(self):
        self.x = (self.a * self.x) % self.m
        return self.x

prng = LinearCongruentialGenerator(15, 29, 17)
prng.random() # >>> 23
```

# Beispiel

$$x_{i+1} \equiv 15x_i \bmod 29, \quad x_0 = 17$$

Output:

[23, 26, 13, 21, 25, 27, 28, 14, 7, 18, 9, 19, ..., 22, 11, 20, 10, 5, 17]

Erwartungswert und Varianz (normalisiert)

$$\mu = 0.5 \quad \sigma^2 = 0.077586$$

# Autokorrelation

- Korrelation zwischen zwei Stichproben  $(x_1, \dots, x_n)$  und  $(y_1, \dots, y_n)$

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

- Autokorrelation mit „Lag 1“

$$y_i = x_{i+1}, \quad y_n = x_1$$

# Autokorrelation

- Im Beispiel

0.4444, 0.111, 0.063

# Autokorrelation

- Im Beispiel

0.4444, 0.111, 0.063

- Zusammenhang der Paare  $(x_i, x_{i+1})$  (Spectral test)

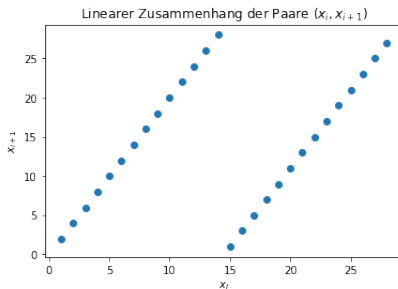


# Autokorrelation

- Im Beispiel

0.4444, 0.111, 0.063

- Zusammenhang der Paare  $(x_i, x_{i+1})$  (Spectral test)

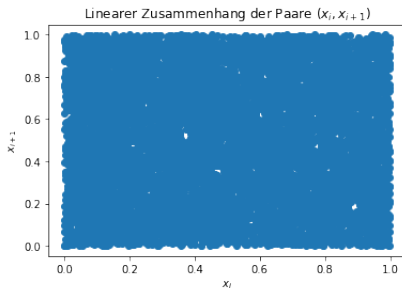


# RANDU

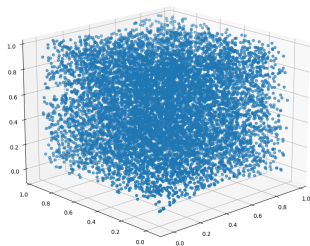
- RANDU ist ein linearer Kongruenzgenerator (genutzt in den 1960-ern bis 1970-ern)

$$x_{i+1} \equiv 65539x_i \bmod 2^{31}$$

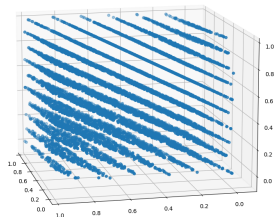
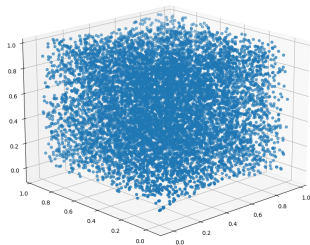
- In 2D



# Spectral test - RANDU in 3D



# Spectral test - RANDU in 3D



# Marsaglia (1968)

- „*RANDOM NUMBERS FALL MAINLY IN THE PLANES*“
  - maximale Hyperebenen-Anzahl

$$(d! \cdot m)^{\frac{1}{d}}$$

- Für RANDU in 3D sind das  $(3! \cdot 2^{31})^{\frac{1}{3}} \approx 2344$
- genauere Schranke in Dimension  $d$

$$|c_1| + \dots + |c_d|,$$

wenn  $c_1x_1 + \dots + c_dx_d = 0, \pm 1, \pm 2, \dots$

THEOREM 1. *If  $c_1, c_2, \dots, c_n$  is any choice of integers such that*

$$c_1 + c_2 k + c_3 k^2 + \dots + c_n k^{n-1} \equiv 0 \text{ modulo } m,$$

*then all of the points  $\pi_1, \pi_2, \dots$  will lie in the set of parallel hyperplanes defined by the equations*

$$c_1 x_1 + c_2 x_2 + \dots + c_n x_n = 0, \pm 1, \pm 2, \dots$$

*There are at most*

$$|c_1| + |c_2| + \dots + |c_n|$$

*of these hyperplanes which intersect the unit  $n$ -cube, and there is always a choice of  $c_1, c_2, \dots, c_n$  such that all of the points fall in fewer than  $(n!m)^{1/n}$  hyperplanes.*

# RANDU - Spektraltest

- $65539 = 2^{16} + 3$  und  $2^{31}$  wegen Berechnungseffizienz gewählt
- Schlechte Wahl, wegen

$$\begin{aligned}x_{i+2} &= (2^{16} + 3)x_{i+1} = (2^{16} + 3)^2 x_i \\&= (2^{32} + 6 \cdot 2^{16} + 9)x_i \\&= [6 \cdot (2^{16} + 3) - 9]x_i \\x_{i+2} &= 6x_{i+1} - 9x_i\end{aligned}$$

- Maximal 16 Ebenen

# Weitere Varianten

- Ausgabewerte mischen (erhöht Periode)

- Bays-Durham Shuffle

- Vektorwertig

$$x_{i+1} \equiv (Ax_i + c) \bmod m$$

- multiple recursive generator

$$x_i \equiv (a_1 x_{i-1} + \dots + a_k x_{i-k}) \bmod m$$

- Nichtlinearer Kongruenzgenerator

$$x_i \equiv f(x_{i-1}, \dots, x_{i-k})$$



# Linear Feedback Shift Registers (LFSR)

- Form:

$$b_i \equiv (a_p b_{i-p} + a_{p-1} b_{i-p+1} + \cdots + a_1 b_{i-1}) \bmod 2$$

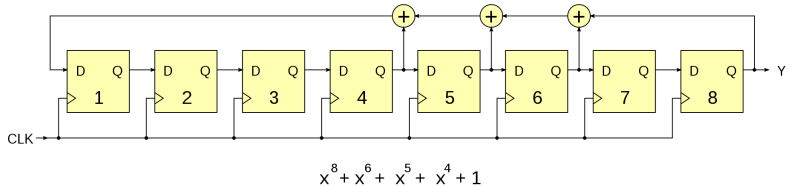
- zugehöriges Polynom über  $\mathbb{Z}_2$

$$f(z) = z^p - (a_1 z^{p-1} + \cdots + a_{p-1} z + a_p)$$

- Addition in  $\mathbb{Z}_2$  entspricht XOR-Operation (meistens sind nur wenige  $a \neq 0$ )

$$b_i = b_{i-p} \oplus b_{i-q}$$

# LFSR



- Operation identisch für  $l$ -Tupel

$$x_i = x_{i-p} \oplus x_{i-q}$$

# Beispiel

- Startsequenz 1, 0, 1, 0 (Periode  $2^4 - 1 = 15$ )

# Beispiel

- Startsequenz 1, 0, 1, 0 (Periode  $2^4 - 1 = 15$ )

..., 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0

# Beispiel

- Startsequenz 1, 0, 1, 0 (Periode  $2^4 - 1 = 15$ )

..., 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0

- bilde Tupel der Länge 4

$$\left( \begin{array}{ccccc} 0011_2 & 1101_2 & 0110_2 & 0100_2 & 0111_2 & 1010_2 \\ 3 & 13 & 6 & 4 & 7 & 10 \end{array} \right)$$

# Beispiel

- Startsequenz 1, 0, 1, 0 (Periode  $2^4 - 1 = 15$ )

..., 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0

- bilde Tupel der Länge 4

$$\begin{pmatrix} 0011_2 & 1101_2 & 0110_2 & 0100_2 & 0111_2 & 1010_2 \\ 3 & 13 & 6 & 4 & 7 & 10 \end{pmatrix}$$

- XOR der Dezimalzahlen mit Startsequenz 6, 4, 7, 10

$$7 \oplus 10 = 13, \quad 4 \oplus 7 = 3, \quad \dots$$

# XORShift

- Einfache Implementierung von LFSR mit, XOR-, Shift-, und Rotationsoperationen
- extrem schnell
- mit nicht linearer Transformation, besteht diese Gruppe fast alle Tests
  - xoshiro256\*\*
  - xoroshiro128\*\*

# XORShift - Beispiel

```
class XORShift:
    def __init__(self, seed:int) → None:
        self.x = np.uint32(seed)

    def random(self):
        self.x ^= self.x << np.uint32(13)
        self.x ^= self.x >> np.uint32(17)
        self.x ^= self.x << np.uint32(5)

        return self.x

prng = XORShift(12541)
prng.random() # >>> 3320704434
```



# Mersenne Twister - (MT 19937)

- Genutzt von: Dyalog APL, IDL, R, Ruby, Free Pascal, PHP, Python, Julia, CMU Common Lisp, Embeddable Common Lisp, Steel Bank Common Lisp
- Periode:  $2^{19937} - 1$
- Initialisierung mit 624 32-Bit Zahlen („Wörter“)
- besteht Spektraltest bis Dimension 623
- Idee:  $x_i = x_{i-p} \oplus Ax_{i-q}$

# Mersenne Twister

## Algorithmus

$$\begin{aligned}h &:= Y_{i-N} - Y_{i-N} \bmod 2^{31} + Y_{i-N+1} \bmod 2^{31} \\Y_i &:= Y_{i-227} \oplus \lfloor h/2 \rfloor \oplus ((h \bmod 2) \cdot 9908B0DF_{\text{hex}}) \\x &:= Y_i \oplus \lfloor Y_i / 2^{11} \rfloor \\y &:= x \oplus ((x \cdot 2^7) \wedge 9D2C5680_{\text{hex}}) \\z &:= y \oplus ((y \cdot 2^{15}) \wedge EFC60000_{\text{hex}}) \\Z_i &:= z \oplus \lfloor z / 2^{18} \rfloor\end{aligned}$$

# Inversionmethode

## Theorem

*Sei  $F$  eine Verteilungsfunktion und  $U \sim \mathcal{U}(0, 1)$  eine gleichverteilte Zufallsvariable. Die Inverse Verteilungsfunktion (Quantil) ist definiert durch*

$$F^{-1}(p) := \inf\{x \in \mathbb{R} : F(x) \geq p\} \quad \forall p \in [0, 1].$$

*Dann ist  $X := F^{-1}(U)$  eine reelle Zufallsvariable mit Verteilungsfunktion  $F$ .*

## Beweis.

$$\mathbb{P}(X \leq x) = \mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x)$$



# Beispiel

- Exponentialverteilung ( $\lambda = 1$ )

$$y = 1 - e^{-x} \implies x = -\ln(1 - y)$$

- Stichprobe mit 100.000 Zahlen

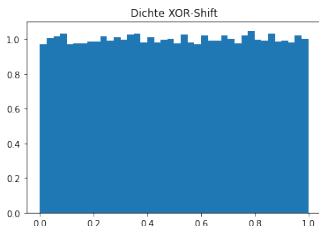


Abbildung: Gleichverteilung von XOR-Shift

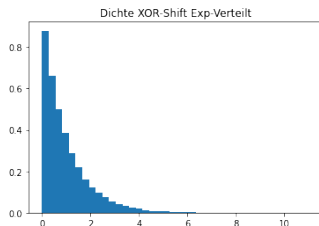


Abbildung: Transformation mit  $y = -\ln(x)$

# Normalverteilung

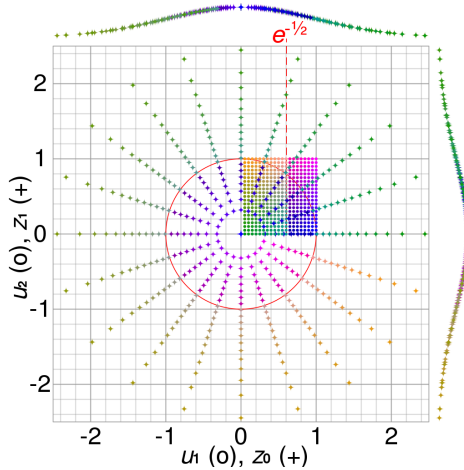
- Box-Muller-Transform

$$Z_1 = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$$

$$Z_2 = \sqrt{-2 \ln U_1} \sin(2\pi U_2)$$

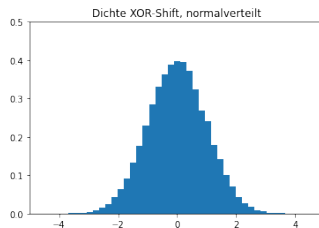
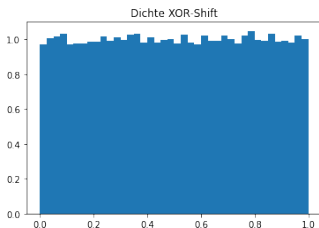
- $U_1, U_2$  gleichverteilt,  $Z_1, Z_2$  normalverteilt
- Polar-Methode vermeidet Berechnung trigonometrischer Funktionen

# Normalverteilung



# Beispiel

## ■ Stichprobe mit 100.000 Zahlen



# Abschluss

- Aktuelle Zufallsgeneratoren
  - ChaCha
  - Xoroshiro128+
- OS abhängig
  - `/dev/random` (UNIX)
  - `CryptGenRandom` (Windows)
- Weitere Tests
  - Diehard tests
  - TESTU01



Dieser Vortrag ist erhältlich unter

<https://github.com/Malmosmo/WL20>