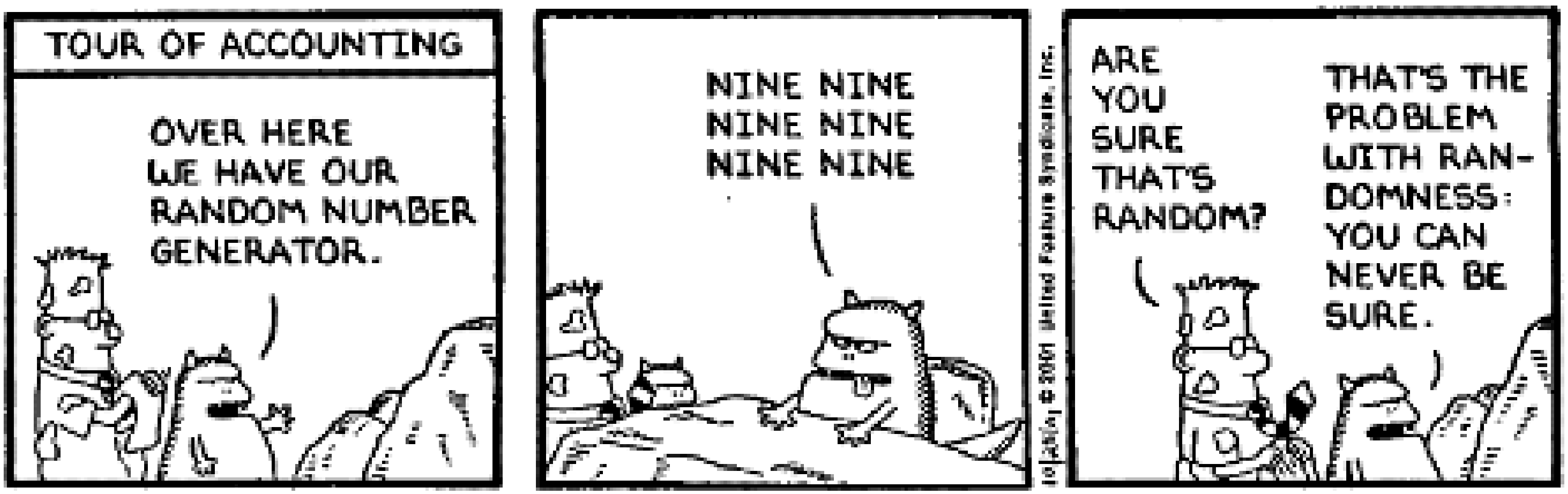
Chair for Network Architectures and Services—Prof. Carle
Department of Computer Science
TU München

Random number generator algorithms and their quality

Some slides/figures taken from:
Oliver Rose
Averill Law, David Kelton
Wikimedia Commons (user Matt Crypto)
Dilbert

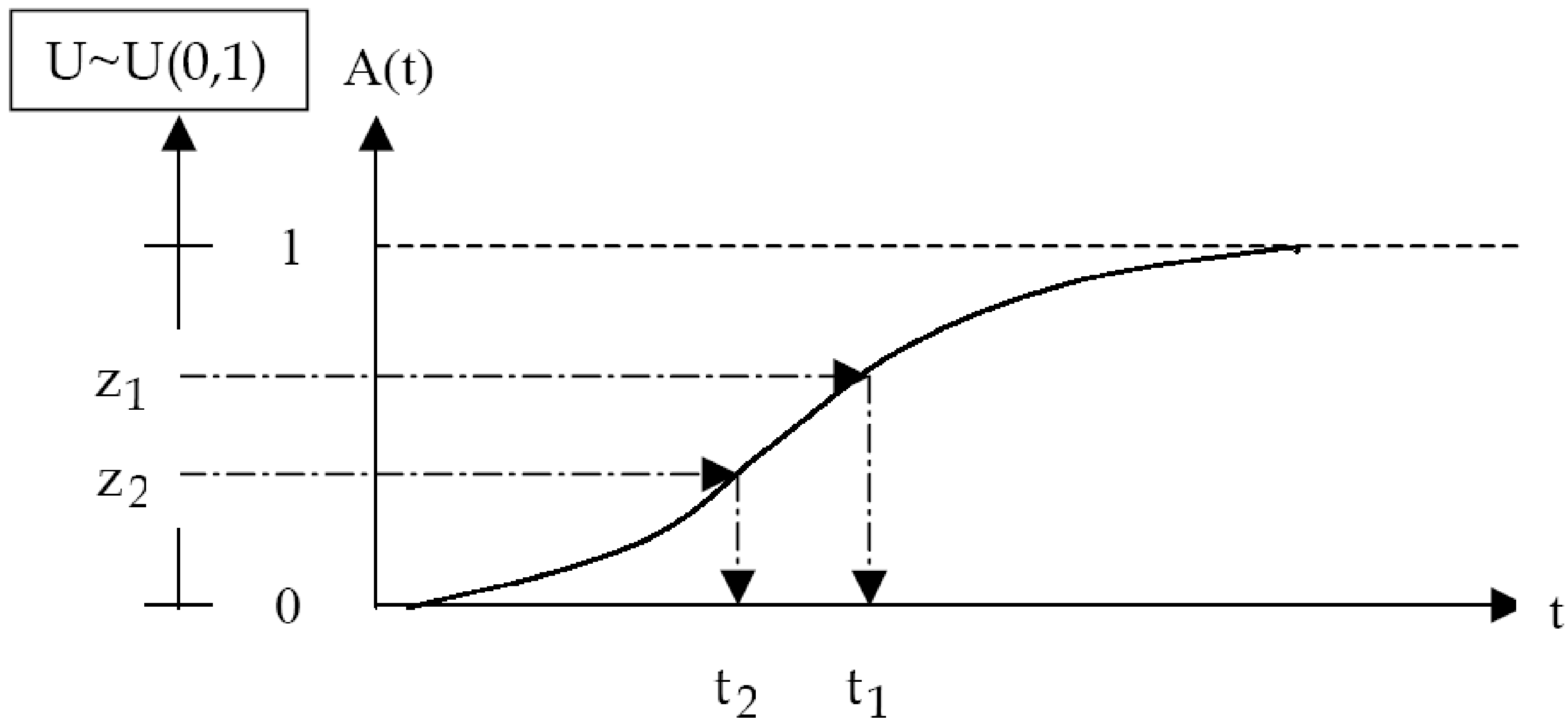
Random number generation



Structure of this lecture

- Generating U(0,1) random numbers
  - Motivation
  - Overview on RNG families
- Linear congruential generators (LCG)
- Statistical properties, statistical (empirical) tests
  - Histogram
  - χ^2 test for uniformity
  - Correlation tests: Runs-up, sequence
- Theoretical aspects, theoretical tests
  - Period length
  - Spectral test
- RNG that are better than LCG

Recall the inversion method



- Generate uniformly distributed numbers ∈ 0.0 … 1.0
- Compute inverse $A^{-1}(t)$ of PDF A(t)
- Generate samples

Generating U(0,1) random numbers is crucial

- For **all** random number generation methods, we need uniformly distributed random numbers from]0,1[
  ⇒ U(0,1) random numbers are required
- Mandatory characteristics
  - Random (…obviously)
  - Uniform (make use of the whole distribution function)
  - Uncorrelated (no dependencies): difficult!
  - Reproducible (for verification of experiments)
    → use *pseudo* random numbers
  - Fast (usually, there is a need for a lot of samples)

RNG in simulation vs. RNG in cryptography

- Also need for random numbers in cryptography
  - Key generation
  - Challenge generation in challenge-response systems
  - …
- Additional requirement:
  - Prediction of future “random” values by sampling previous values must not be possible
  - In simulation: not an issue if there is no real correlation
- Lighter requirement:
  - RNs are not used constantly, only in ~start-up phases
    ⇒ speed is not of much importance
  - In simulation: need lots of numbers
    ⇒ speed is *very* important

Generation of U(0,1) random numbers

- Generation approaches
  - "Real", "natural" random numbers: sampling from radioactive material or white noise from electronic circuits, throwing dice, drawing from an urn, …
    - Problems:
      - If used online: not reproducible
      - Tables: uncomfortable, not enough samples
  - Pseudo random numbers: recursive arithmetic formulas with a given starting value (seed)
    - in hardware: shift register with feedback (based on primitive polynomials as feedback patterns)
    - in software: linear congruential generator (LCG) (Lehmer, 1951), …

Generation of U(0,1) random numbers: overview

Main families:

- Linear congruential generator (LCG): the simplest
- General congruential generators
  - Quadratic congruential generator
  - Multiple recursive generators
- Shift register with feedback (Tausworthe)
  - E.g., Mersenne Twister: state-of-the-art
- Composite generators: output of multiple RNG
  - E.g., use one to shuffle ("twist") the output of the other

RNG: alternatives unsuitable for simulation

- Algorithms from cryptography
  - For example: counter→AES, counter→SHA1, counter→MD5, etc.
  - Usually way too slow
- Calculate transcendent numbers (e.g., π or e), view their digits as random
  - E.g.: digits of 100,000th decimal place of π onwards
  - Problem: Are they really random? There seems to be some structure…
- Physical generators (cf. previous lecture)
  - Not reproducible, no seed
- Tables with pre-computed random numbers
  - We need too many random numbers, the tables would have to be huge…

Linear congruential generators

- Calculate RN from previous RN using some formula
- Sequence of integers $Z_1, Z_2, \ldots$ defined by

$$Z_i = (a \cdot Z_{i-1} + c) \, (\operatorname{mod} m)$$

  with **modulus** m, **multiplier** a,
  increment c, and **seed** Z_0
- c=0: ***multiplicative LCG***
  Example: $$Z_i = 16807 \cdot Z_{i-1} \quad \left(\operatorname{mod} 2^{31} - 1\right)$$
  (Lewis, Goodman, Miller, 1969)
- c>0: ***mixed LCG***

…but they don't create floats, but integers > 1?!

- Obviously,

  $$Z_i = \mathit{something} \bmod m$$

  and

  $$\mathit{something} \bmod m < m$$

- ⇒ Just normalise the result!
  - Divide by m? But then, 1.0 cannot be attained.
  - Better: Divide by m–1.

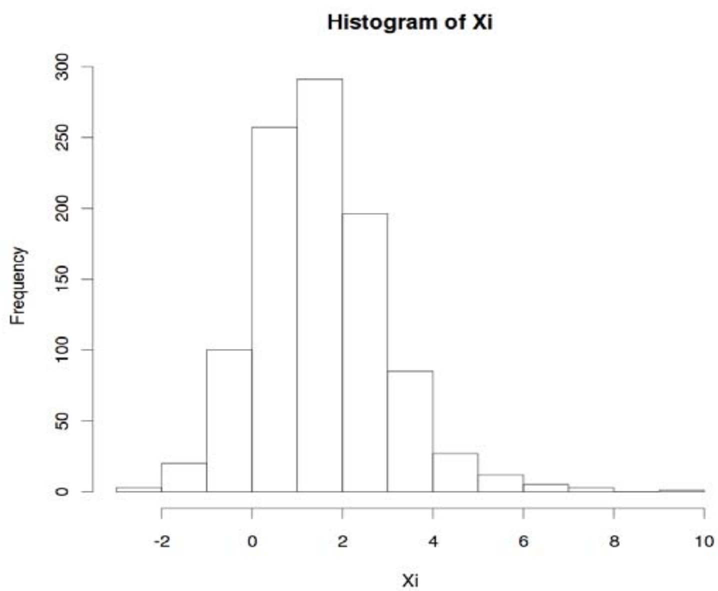
Do they really generate uniformly distributed random numbers?

- Test for uniformity:
  - Create a number of samples from RNG
  - Test if these numbers are uniformly distributed
- A number of statistical tests to do this:
  - χ^2 test (deutsch: Chi-Quadrat-Anpassungstest)
  - Kolmogorov-Smirnov test
  - … and a whole lot of others! For example:
    - Cramér-von Mises test
    - Anderson-Darling test
- Graphical examination (not real tests):
  - Plot histogram / density / PDF
  - Distribution-function-difference plot
  - Quantile-quantile plot (Q-Q plot)
  - Probability-probability plot (P-P plot)

(later in course)

Histogram

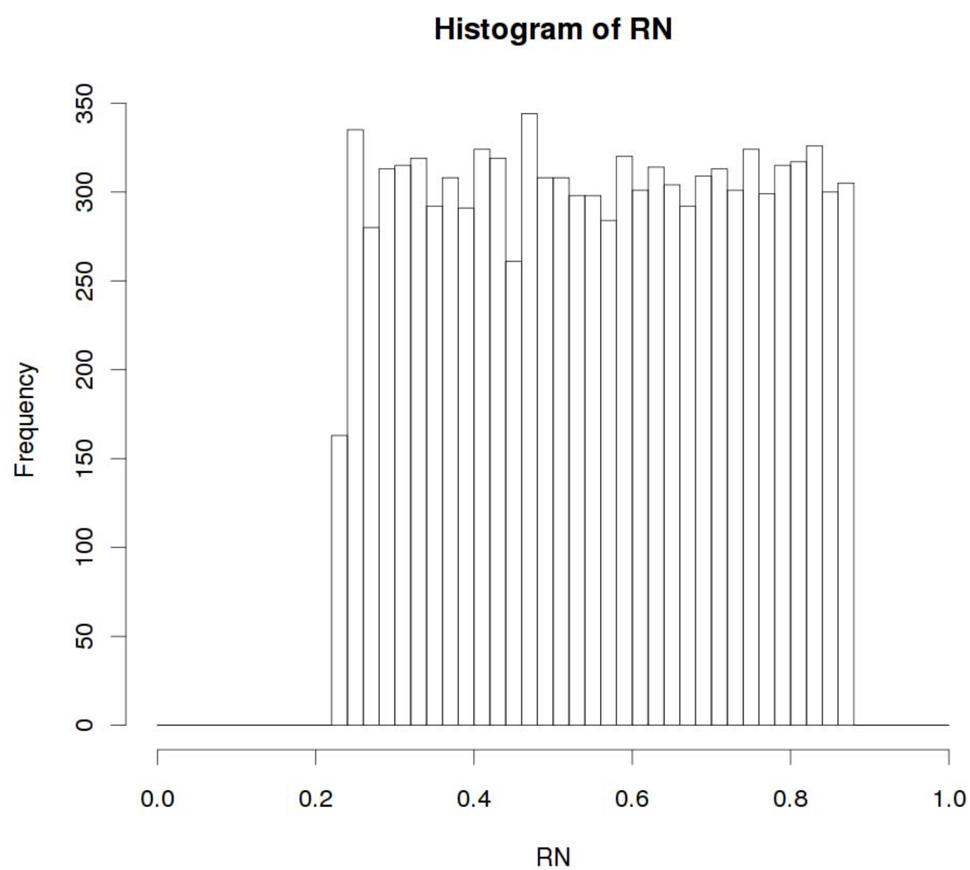
- Given a series of n measurements X_i
- Partition the domain min{X_i} … max{X_i} into m intervals $I_1 \ldots I_m$
- Count how many X_i fall into which interval I_j
- Plot it:



- ~discretised density function
- Recommendation: $m \approx \sqrt{n}$

What the histogram can reveal (1)

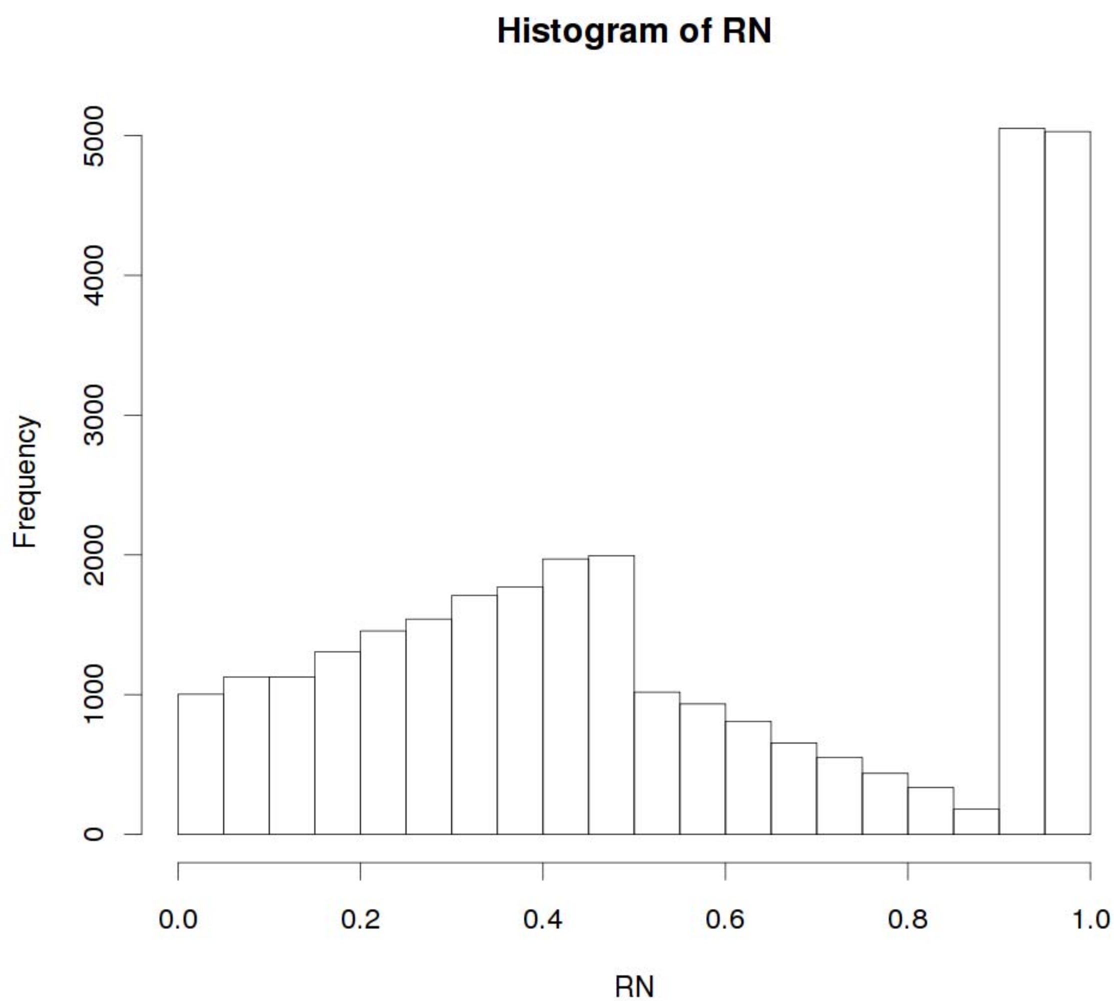
Obviously not U(0,1) random variables:



(…okay, we could have calculated min and max)

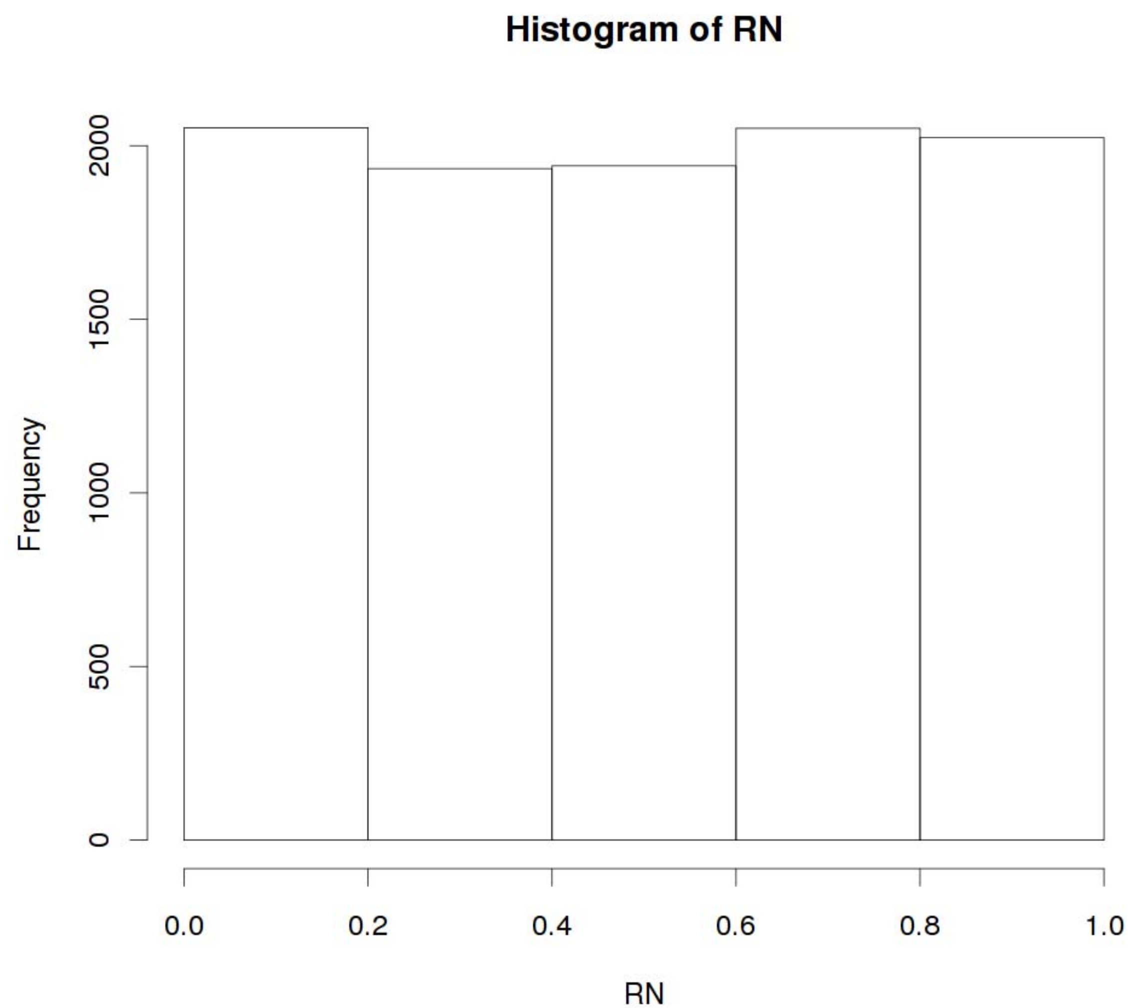
What the histogram can reveal (2)

Obviously not U(0,1) random variables:



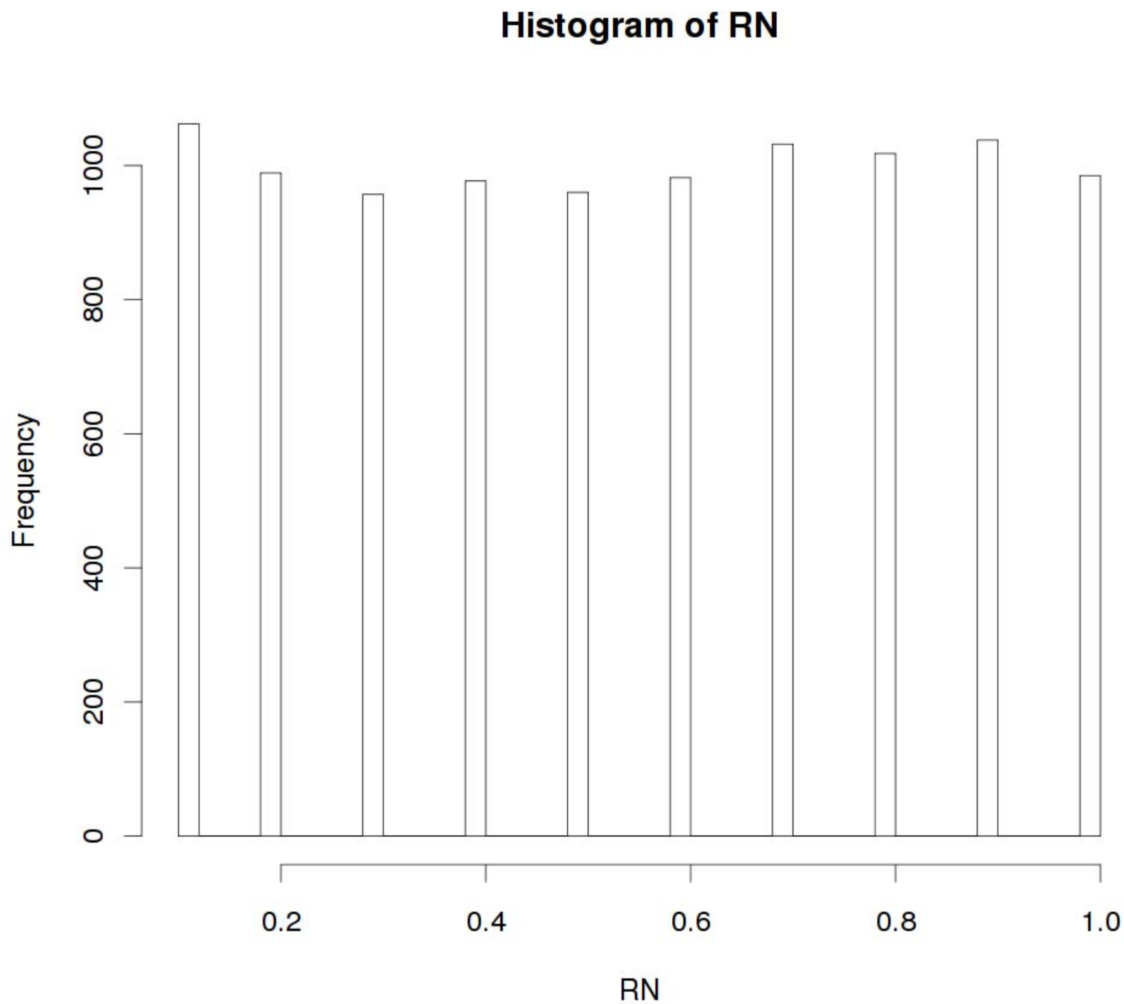
What the histogram can reveal (3a)

Looks like a U(0,1) random variable…:



What the histogram can reveal (3b)

…but obviously not U(0,1) random variables: huge gaps!



Statistical tests

- Scenario: Given a set of measurements, we want to check if they conform to a distribution; here: U(0,1)
- Graphs like presented before are nice indicators, but not really tangible: “How straight is that line?” etc.
- We want clearer things: Numbers or yes/no decisions
- Statistical tests can do the trick, but…
  - Warning #1: Tests only can tell if measurements do *not* fit a particular distribution—i.e., no “yes, it fits” proof!
  - Warning #2: The result is never absolutely certain, there is always an error margin.
  - Warning #3: Usually, the input must be ***‘iid’:***
    - Independent
    - Identically distributed
  - ⇒You never get a ‘proof’, not even with an error margin!

χ^2 test (Pearson, 1900)

- Input:
  - Series of n measurements $X_1 \dots X_n$
  - A distribution function f (the ‘theoretical function’)
- Measurements will be tested against the distribution
  - ~formal comparison of a histogram with the density function of the theoretical function
- Null hypothesis H0:
  The X_i are IID random variables with distribution function f

χ^2 test: How it works

- Divide [0…1] into k equal-size intervals
- Count how many X_i fall into which interval (histogram):

  N_j := number of X_i in j-th interval $[a_{j-1} \ldots a_j[$
- Calculate how many X_i would fall into the j-th interval if they were sampled from the theoretical distribution:

  $$p_j := \int_{a_{j-1}}^{a_j} f(x)dx$$ (f: density of theor. dist.)
- Calculate squared normalised difference between the observed and the expected:

  $$\chi^2 := \sum_{j=1}^{k} \frac{(N_j - np_j)^2}{np_j}$$
- Obviously, if χ^2 is "too large", the differences are too large, and we must reject the null hypothesis
- But what is "too large"?

χ^2 test: Using the χ^2 distribution

- χ^2 distribution
  - A test distribution
  - Parameter: degrees of freedom (short *df*)
  - χ^2(k–1 df) = Γ (½(k–1) , 2) *(gamma distribution)*
  - Mathematically: The sum of n independent squared normal distributions
- Compare the calculated χ^2 against the χ^2 distribution
  - If we use k intervals, then χ^2 is distributed corresponding to the χ^2 distribution with k–1 df
  - Let $\chi^2_{k-1,1-\alpha}$ be the $(1-\alpha)$ quantile of the distribution
  - α is called the confidence level
  - Reject H0 if $\chi^2 > \chi^2_{k-1,1-\alpha}$ (i.e., the X_i do not follow the theoretical distribution function)

χ^2 test and degrees of freedom

- χ^2 test can be used to test against *any* distribution
- Easy in our case: We know the parameters of the theoretical distribution f — it's U(0,1)
- Different in the general case:
  - For example, we may know it's N(μ, σ) (normal distribution) but we know neither μ nor σ
  - Fitting a distribution: Find parameters for f that make f fit the measurements X_i best
  - Topic of a later lecture
- Theoretically:
  Have to estimate m parameters ⇒ Also have to take $\chi^2_{k-m-1,1-\alpha}$ into account
- Practically:
  $m \leq 2$ and large k ⇒ Don't care…

χ^2: which parameters?

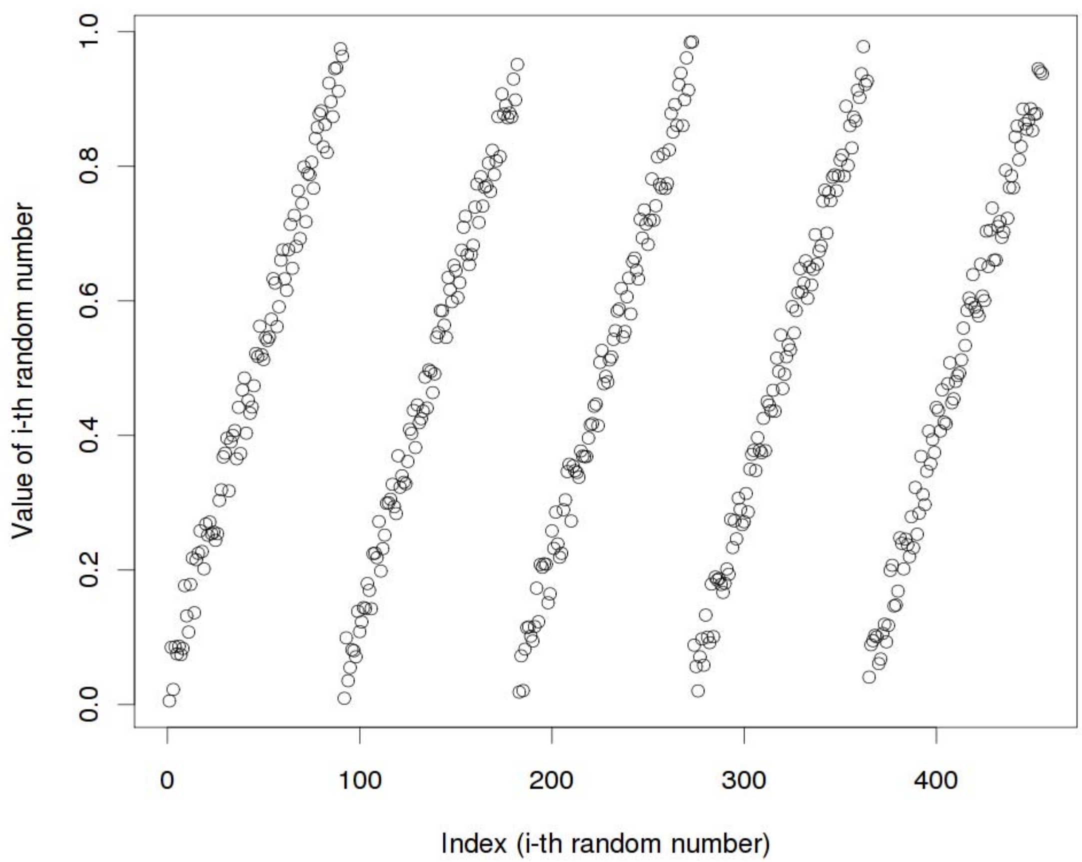
- How many intervals (k)?
  - A difficult problem for the general case
  - Warning: A smaller *or* a greater k may change the outcome of the test!
  - As a general rule, use k>100
  - As a general rule, make the intervals equal-sized
  - As another general rule, make sure that $\forall j: np_j \geq 5$ (i.e., have enough samples that we expect to have at least 5 samples in each interval)
- ⇒ As a general rule, you need a lot of measurements!
- What confidence level?
  - At most α=0.10 (almost too much);
typical values: 0.001, 0.01, 0.05 [, and 0.10]
  - The smaller, the better confidence in the test result

Alternatives to test

- Kolmogorov-Smirnov test (K-S test)
  - Another very popular test
  - Advantages:
    - No grouping into intervals required
    - Valid for any sample size, not only for large *n*
    - More powerful than χ^2 for a number of distributions
  - Disadvantages:
    - Applicability more limited than χ^2
    - Difficult to apply to discrete data
    - If distribution needs to be fitted (unknown parameters), then K-S works only for a number of distributions
- Anderson-Darling test (A-D test)
  - Higher power than K-S for some distributions
- …a lot of other tests

Tests for uniformity: limitations

- Consider this sequence of drawn "random numbers":



- They are in U(0,1) … but do they seem random!?

Recall our requirements for RNG

- RNs have to be *uncorrelated* — how to test this?
- Statistical tests:
  Draw some random numbers and examine them
  - Runs-up test
  - Autocorrelation function *(later in course)*
  - Serial test
  - …
- Theoretical parameters and theoretical tests:
  - Length of period
  - Spectral test
  - Lattice test
  - …

Runs-up test

- *Run up* := the length of a contiguous sequence of monotonically increasing X_i.
- Example sequence:

0.86 >	length: 1
0.11 < 0.23 >	length: 2
0.03 < 0.13 >	length: 2
0.06 < 0.55 < 0.64 < 0.87 >	length: 4
0.10	length: 1

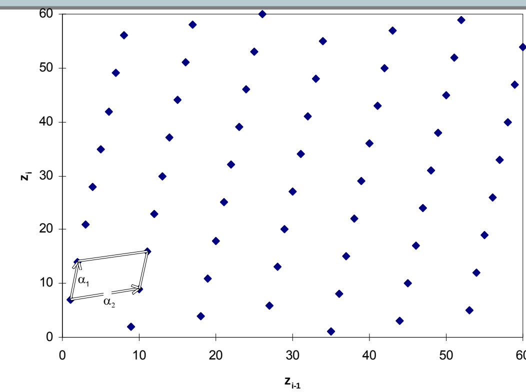
- Calculate r_i (number of runs up of length i)
- Compute a test statistic value R, using the r_i and a bestranging zoo of esoteric constants a_{ij} and b_j
- R will have an approximate χ^2 distribution with 6 df.

Serial test

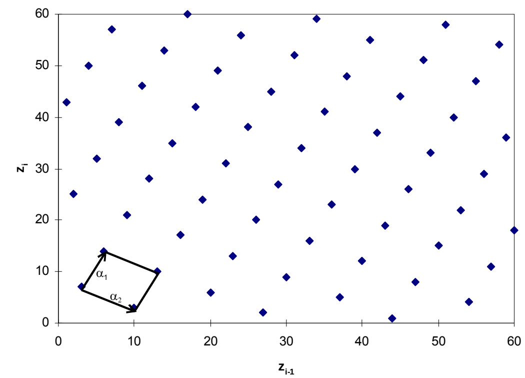
- Find possible correlations between subsequently drawn values
- Visual "tests":
  - 2D plot of X_i and X_{i-1}
  - 3D plot of X_i and X_{i-1} and X_{i-2}
- Generalisation: Serial test

LCG examples (1/5)

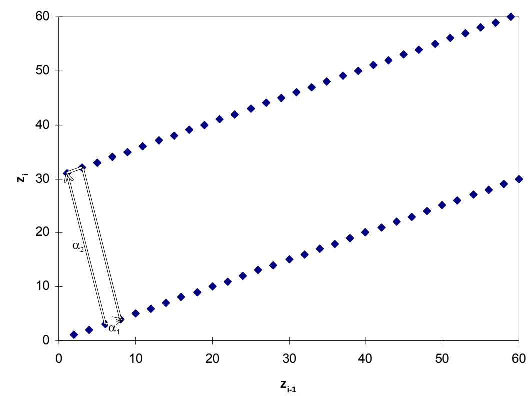
$$Z_i = a \cdot Z_{i-1} \pmod{61}$$



a=7

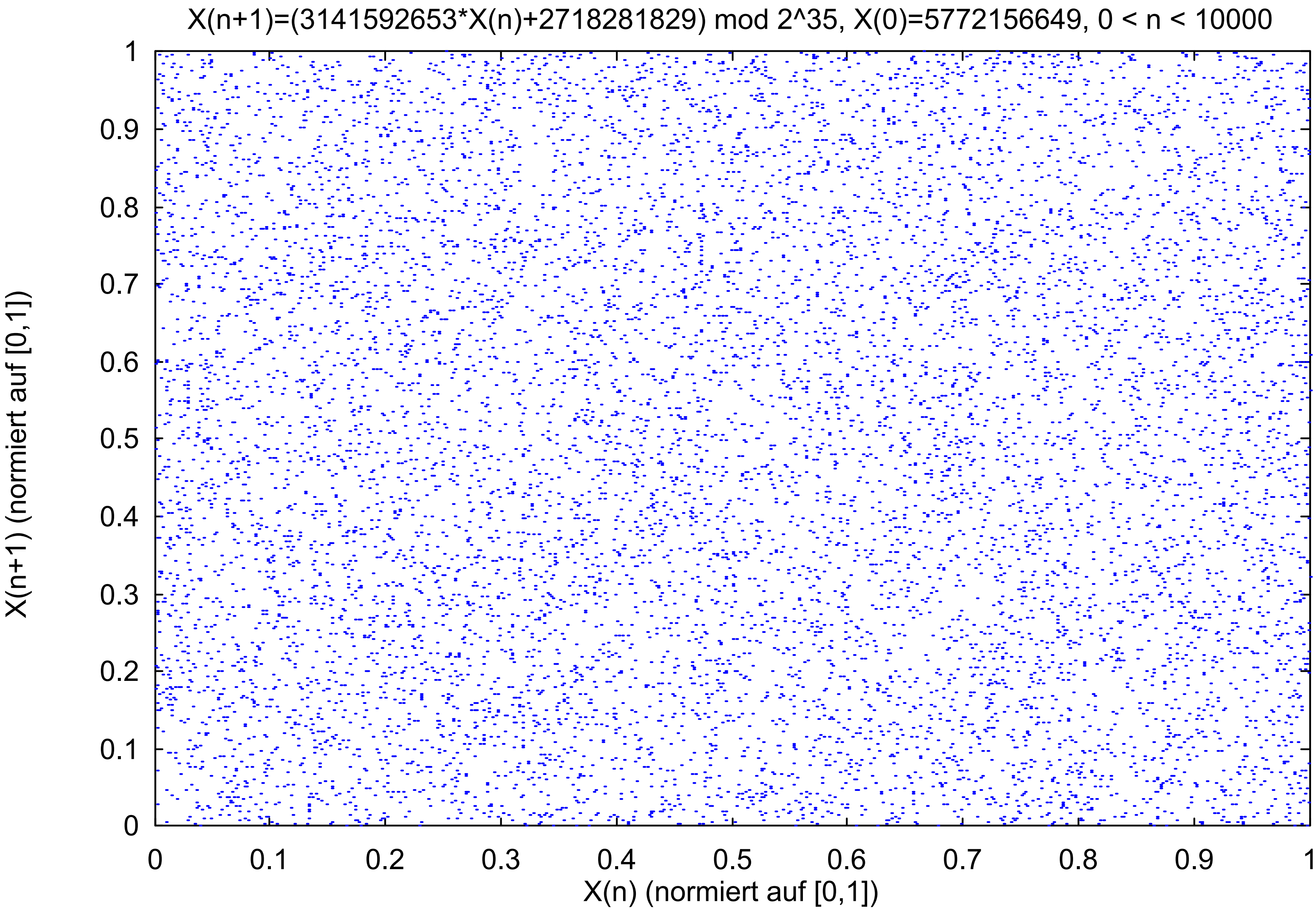


a=43

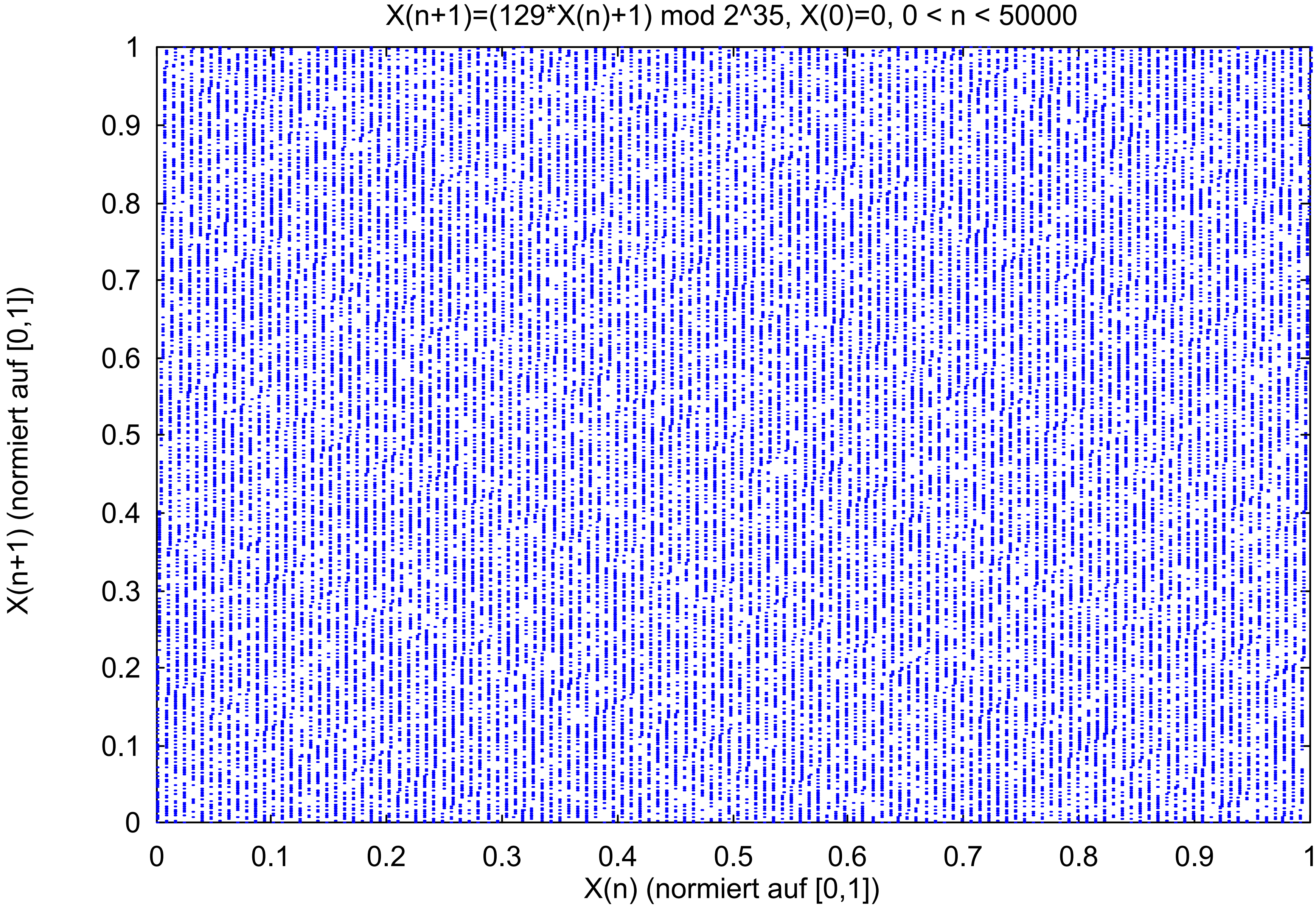


a=31

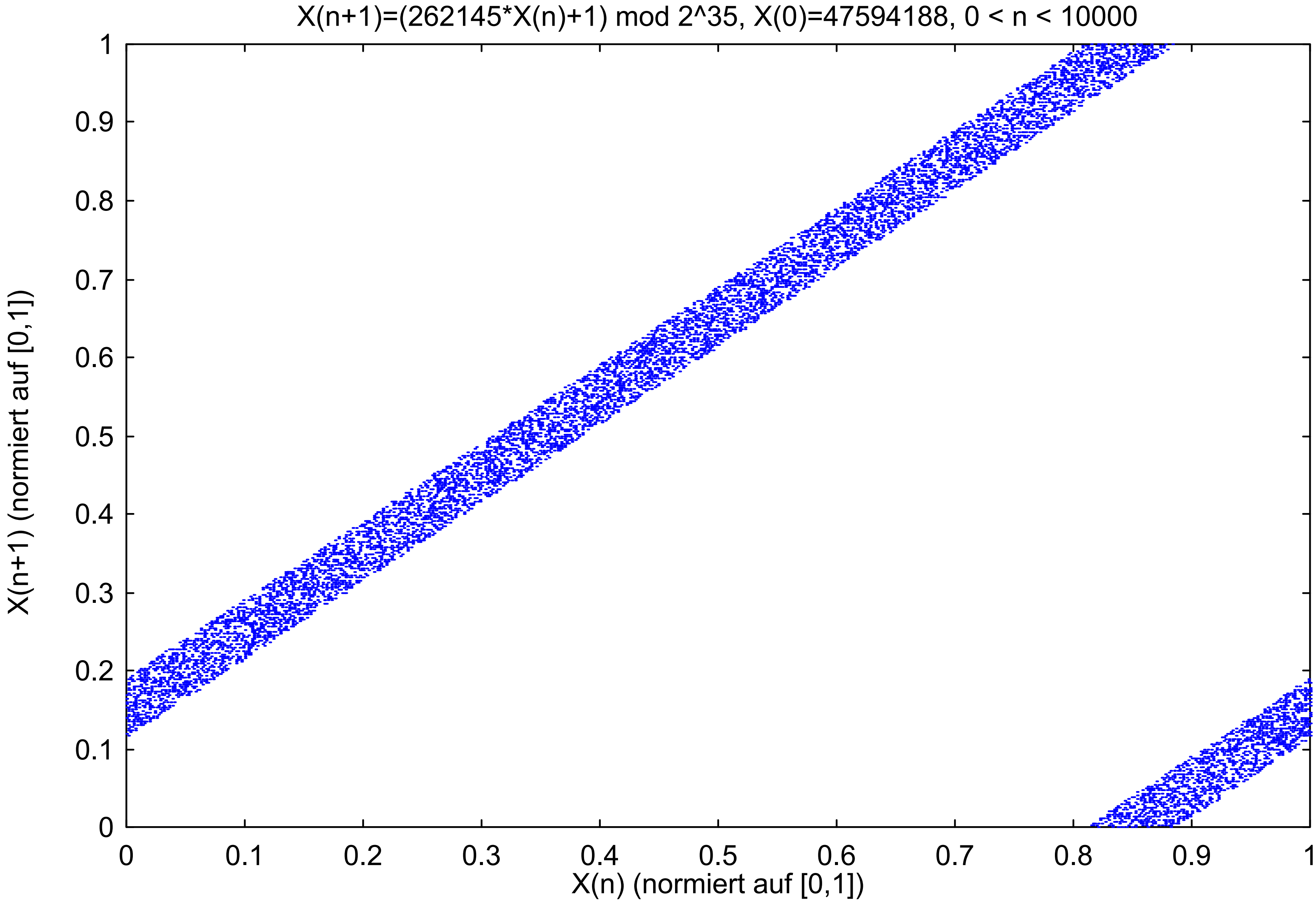
LCG examples (2/5)



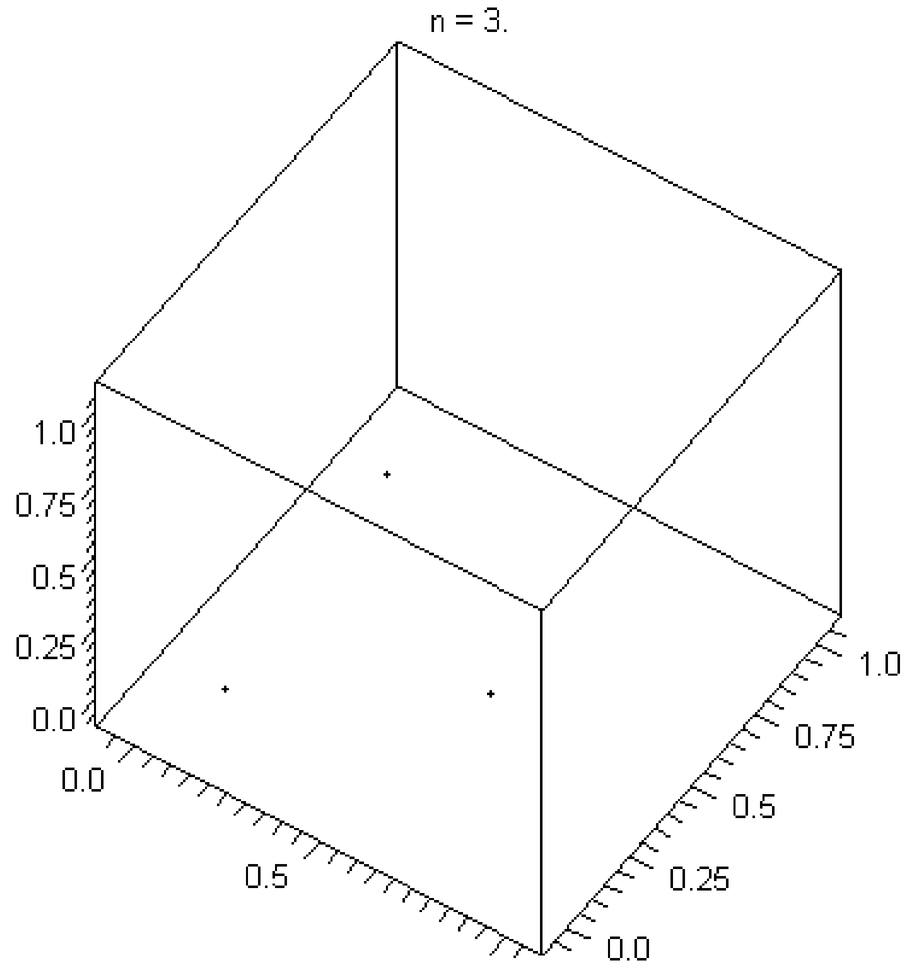
LCG examples (3/5)



LCG examples (4/5)



LCG examples (5/5)



(see Law/Kelton: Simulation Modeling and Analysis, 4th edition, Fig. 7.4, p. 413)

Serial test

"a generalised and formalised version of the plots"

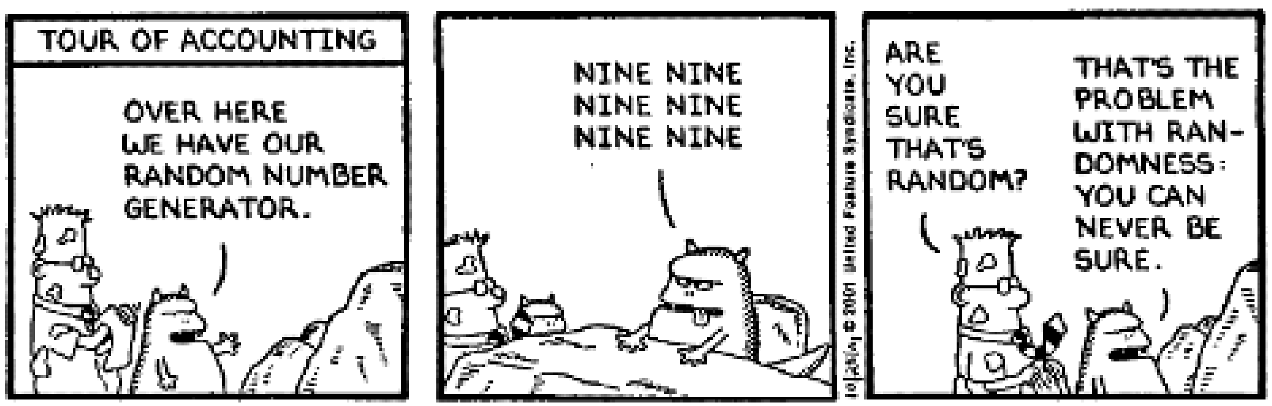
- Consider non-overlapping d-tuples of subsequently drawn random variables X_i :
  $U_1 = (X_1, X_2, \ldots X_d)$ $U_2 = (X_{d+1}, X_{d+2}, \ldots, X_{2d})$ …
- These U_i's are vectors in the d-dimensional space
- If the X_i are truly iid random variables, then the U_i are truly random iid vectors in the space $[0\ldots1]^d$ (the d-dimensional hypercube)
- Test for d-dimensional uniformity (rough outline):
  - Divide [0…1] into k equal-sized intervals
  - Calculate a value $\chi^2(d)$ based on the number of U_i for each possible interval combination
  - $\chi^2(d)$ has approximate distribution $\chi^2(k^d - 1 \text{ df})$
  - Rest: same as χ^2 test above

RANDU

- A LCG with setup:
  $Z_i = 65{,}539 \cdot Z_{i-1} \bmod 2^{31}$
- Advantage: It's fast.
  - mod 2^{31} can be calculated with a simple AND operation
  - 65,539 is a bit more than 2^{16}; thus the multiplication (=expensive operation) can be replaced by a bit shift of 16 bit plus three additions (=cheap operations)
  - Why 65,539? It's a prime number.
- Disadvantage:
  - An infamously bad RNG! Never, ever use it!
  - $d \geq 3$: The tuples are clumped into 15 plains (remember the animated 3D cube? That was RANDU!)
- A lot of simulations in the 1970s used RANDU
  ⇒ sceptical view on simulation results from that time

Theoretical parameters, theoretical tests

- Tests so far: Based on drawing samples from RNG
- No absolute certainty!
  - Usually, only a small subset of entire period is used
  - Remember the χ^2 test
  - Remember Dilbert:



- Theoretical parameters and tests
  - Based directly on the algorithm and its parameters
  - No samples to be drawn
  - Hard mathematical stuff…

Period length

- After some time, the “random” numbers must repeat themselves.
  Why?
  - LCG: Z_i is entirely determined by Z_{i-1}
  - The same Z_{i-1} will always produce the same Z_i
  - There are only finitely many different Z_i
  - How many?
    We take mod $m \Rightarrow$ at most m different values
- Call this the period length

Theorem by Hull and Dobell 1962

- A LCG has full period if and only if the following three conditions hold:
  1. c is relatively prime to m
     (i.e., they do not have a prime factor in common)
  2. If m has a prime factor q,
     then $(a-1)$ must have a prime factor q, too
  3. If m is divisible by 4,
     then $(a-1)$ must be divisible by 4, too
- ⇒Prime numbers play an important role
  - Remember RANDU?
    At least, it used a prime number…
- Multiplicative RNGs (i.e., no increment Z_i+c) cannot have period m.
  (But period $(m-1)$ is possible if m and a are chosen carefully.)

LCG and period length considerations

- On 32 bit machines, $m \leq 2^{31}$ or $m \leq 2^{32}$ due to efficiency reasons ⇒ period length 4.3 billion
- Calculating that many random numbers only takes a couple of seconds on today's hardware
- Theory suggests to use only $\sqrt{period\_length}$ numbers; that's only 65,000 random numbers
- How many random numbers do we need? Example:
  - Simulate behaviour of 1,000 Web hosts
  - Each host consumes on average 1 random number per simulation second
  - Result: We can only simulate for one minute!
- We need much longer period lengths

Spectral test (coarse description)

- ~ The theoretical variant of the serial test
- Observation by Marsaglia (1968):
  "Random numbers fall mainly in planes."
  - Subsequent *overlapping* (!) tuples U_i:
    $U_1=(X_1, X_2, \ldots X_d)$ $U_2=(X_2, X_3, \ldots, X_{d+1})$ …
    fall on a relatively small number of (d–1)-dimensional hyperplanes within the d-dimensional space
  - Note the difference to the serial test! (overlapping)
  - 'Lattice' structure
- Consider hyperplane families that cover all tuples U_i
- Calculate the maximum distance between hyperplanes. Call it δ_d.
- If δ_d is small, then the generator can ~uniformly fill up the d-dimensional space

Spectral test and LCG

- For LCG, it is possible to give a theoretical lower bound δ_d^*:
  $$\delta_d \geq \delta_d^* = 1 / (\gamma_d \, m^{1/d})$$
- γ_d is a constant whose exact value is only known for $d \leq 8$ (dimensions up to 8)
- LCG do not perform very well in the spectral test:
  - All points lie on at most $m^{1/n}$ hyperplanes (Marsaglia's theorem)
  - Serial test: similar
  - There are way better random number generators than linear congruential generators.

Discussion of LCGs

- Advantages:
  - Easy to implement
  - Reproducible
  - Simple and fast
- Disadvantages:
  - Period (length of a cycle) depends on parameters *a*, *c*, and *m*
  - Distribution and correlation properties of generated sequences are not obvious
  - A value can occur only once per period (unrealistic!)
  - By making a bad choice of parameters, you can screw up things massively
  - Bad performance in serial test / spectral test even for good choice of parameters

Beyond LCGs

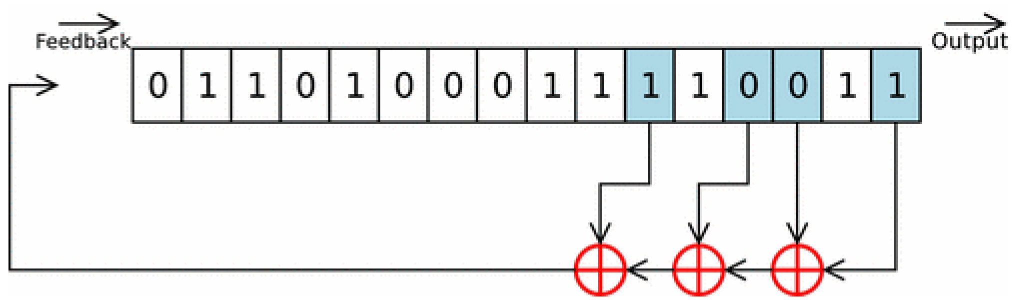
- Why linear?
  - Quadratic congruential generator:
    $$Z_i = (a \cdot (Z_{i-1})^2 + a' \cdot Z_{i-1}) \bmod m$$
  - Period is still at most m
- Why only use one previous X_i?
  - Multiple recursive generator:
    $$Z_i = (a_1 Z_{i-1} + a_2 Z_{i-2} + a_3 Z_{i-3} + \ldots + a_q Z_q) \bmod m$$
  - Period can be m^q-1 if parameters are chosen properly
- Why not change multiplier a and increment c dynamically, according to some other congruential formula?
  - Seems to work alright

Feedback Shift Register Generators (1/2)

- Linear feedback shift register generator (LFSR) introduced by Tausworthe (1965)
- Operate on binary numbers (bits), not on integers
- Mathematically, a multiple recursive generator:

$$b_i = (c_1 b_{i-1} + c_2 b_{i-2} + c_3 b_{i-3} + \ldots + c_q b_q) \bmod 2$$

  - c_i: constants that are either 0 or 1
  - $c_q = 1$ (why?)
  - Observe that + mod 2 is the same as XOR (makes things faster)
- In hardware:



Feedback Shift Register Generators (2/2)

- Usually only two cj coefficients are 1, thus:

  $$b_i = (b_{i-r} + b_{i-q}) \bmod 2$$

- LFSR create random bits, not integers
  - Concatenate ℓ bits to form an ℓ-bit integer:

    $$W_i = b_{(i-1)\ell+1}\, b_{(i-1)\ell+2} \ldots b_{i\ell}$$

- Properties
  - Period length of the $b_i = 2^q - 1$ if parameters chosen accordingly (weird maths involved: characteristic polynomial has to be primitive over Galois field $\mathcal{F}_2$ …)
  - Period length of the generated ints accordingly lower?
    - Depends on whether $\ell \mid 2^q - 1$ or not—probably not the case
    - But there may be some correlation after one period
  - Statistical properties not very good
  - Combining LFSRs improves statistics and period

Generalised feedback shift register (GFSR)

- Lewis and Payne (1973)
- To obtain sequence of ℓ-bit integers Y_1, Y_2, …:
  - Leftmost bit of Y_i is filled with LFSR-generated bit b_i
  - Next bit of Yi is filled with LFSR-generated bit after some “delay” d: b_{i+d}
  - Repeat that with same delay for remaining bits up to length ℓ
- Mathematical properties
  - Period length can be very large if q is very large, e.g., Fushimi (1990): period length = $2^{521}-1 = 6.86 \cdot 10^{156}$
  - If $\ell<q$, then many Y_i's will repeat during one period run (Is that good or bad?)
  - If two bits (as with LFSR), then $Y_i = Y_{i-r} \oplus Y_{i-q}$ XOR

Mersenne Twister

- Before we go into the mathematical details…
  - Very, very long period length: $2^{19,937}-1 > 10^{6,000}$
  - Very good statistical properties: OK in 623 dimensions
  - Quite fast
- State of the art: One of the best we have right now
  - The RNG of choice for simulations
  - Default RNG in Python, Ruby, Matlab, GNU R
  - Admittedly, there are even (slightly) better RNGs, cf. TestU01 paper
- Two warnings:
  - **Not suitable for cryptographic applications:** Draw 624 random numbers and you can predict all others!
  - Can take some time ("warm-up period") until the stream generates good random numbers
    - Usually hidden from programmer through library
    - If in doubt, discard the first 10,000 drawn numbers

Mersenne Twister: details

- Twisted GFSR (TGFSR)
  - Matsumoto, Kurita (1992, 1994)
  - Replace the recurrence of the GFSR by
    $$Y_i = Y_{i-r} \oplus A \cdot Y_{i-q}$$
    where:
    - the Y_i are ℓ x 1 binary vectors
    - A is an ℓ x ℓ binary matrix
  - Period length = $2^{q\ell}-1$ with suitable choices for r, q, A
- Mersenne Twister (MT19937)
  - Matsumoto, Nishimura (1997, 1998)
  - Clever choice of r, q, A and the first Y_i to obtain good statistical properties
  - Period length $2^{19{,}937}-1 = 4.3 \cdot 10^{6001}$ (Mersenne prime: 2^n-1)

Digression: Period lengths revisited

What period lengths do we actually require?

- Estimate #1:
  - A cluster of 1 million hosts
  - each of which draws $1{,}000{,}000 \cdot 2^{32}$ per second (~1,000,000 times as fast as today's desktop PCs)
  - for ten years

  will require…
  - $5.6 \cdot 10^{27}$ random numbers
  - (Make the PCs again 10^6 times faster $\Rightarrow 5.6 \cdot 10^{33}$)
- Estimate #2: What's the estimated number of electrons within the observable universe (a sphere with a radius of ~46.5 billion light years)
  - About 10^{80} (± take or leave a few powers of 10)

Test batteries

- A lot of tests, a lot of different RNGs
- How to compare them?
- Benchmark suites (‘Test batteries’) that bundle many statistical tests:
  - TestU01 (L’Ecuyer)
  - DIEHARD suite (Marsaglia)
  - NIST test suite (National Institute of Standards and Technologies;

    ≙ Physikalisch-Technische Bundesanstalt)

Conclusion: Quality tests for RNG

- **Empirical tests** (based on generated samples)
  - For U(0,1) distribution: χ^2 test
  - For independence: autocorrelation, serial, run-up tests
- **Theoretical tests** (based on generation formula)
  - Basic idea: test for *k*-dimensional uniformity
  - Points of sequence form system of hyperplanes
  - Computation of distance of hyperplanes for several dimensions *k*
  - Rather difficult optimization problem
- **Conclusion**
  - Implement/use only tested random number generators from literature, no “home-brewed” generators!
  - When in doubt, use the Mersenne Twister (but not for cryptography!)

RNG: outlook

- A wide research field, still somewhat active
  - Many more algorithms exist
  - Many more tests for randomness exist
  - More are being developed
- If you are interested in this topic, you might want to have a look at this quite readable paper:
  - L'Ecuyer, Simard
    TestU01: a C library for empirical testing of random number generators
    ACM Transactions on Mathematical Software, Volume 33, No. 4, 2007
  - Daniel Bueb
    Random Number Generators
    Semesterarbeit, EPFL, 2005