(Pseudo) Random Number Generation

Holger Füßler

Lehrstuhl für Praktische Informatik IV, Universität Mannheim

Course overview

1. Introduction

2. Building block: RNG

3. Building block: Generating random variates I and modeling examples

4. Building block: Generating random variates II and modeling examples

5. Algorithmics: Management of events

6. NS-2: Introduction

7. NS-2: Fixed networks

8. NS-2: Wireless networks

9. Output analysis: single system

10. Output analysis: comparing different configuration

11. Other Simulators

12. Simulation lifecycle, summary

I Why do we need ‘random numbers’?

System → **Mathematical model** → **State variables:** X_t, Y_t, Z_t, …

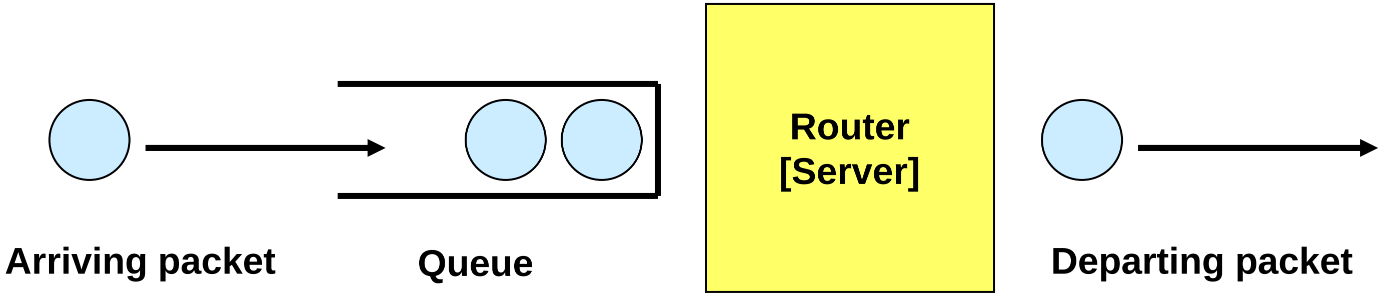
Random effects, e.g.:
- **Link breaks**
- **Node movements**
- **Packet arrivals**
- **Jitter**
- **Random access**
- **…**

→ **Changes of time at discrete ‘events’**

Required building block:
(Pseudo) Random number generation

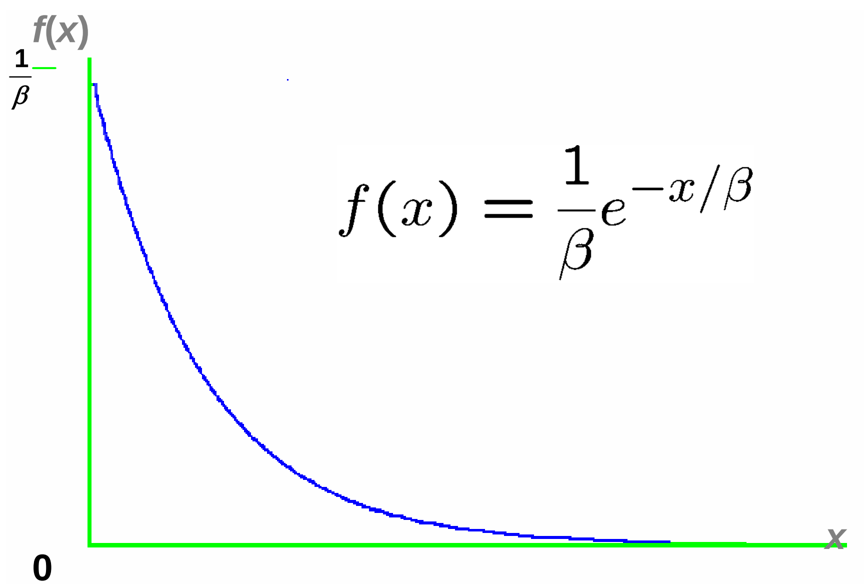
I Example: Why do we need ‘random numbers’?

» **Recap: M/M/1 queue example**



» **Arrival process: exponentially distributed inter-arrival times**

» **Service process: exponentially distributed service times**

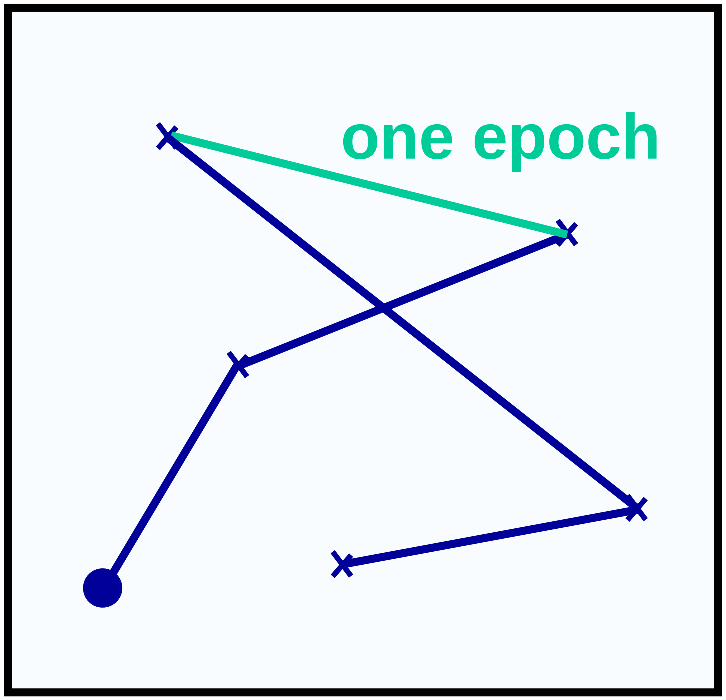


$$f(x) = \frac{1}{\beta} e^{-x/\beta}$$

I What are ‘random numbers’?

- **Independent samples from the uniform distribution over the interval [0,1]**
- **Out of random numbers one can generate arbitrary random variates**

Random waypoint mobility

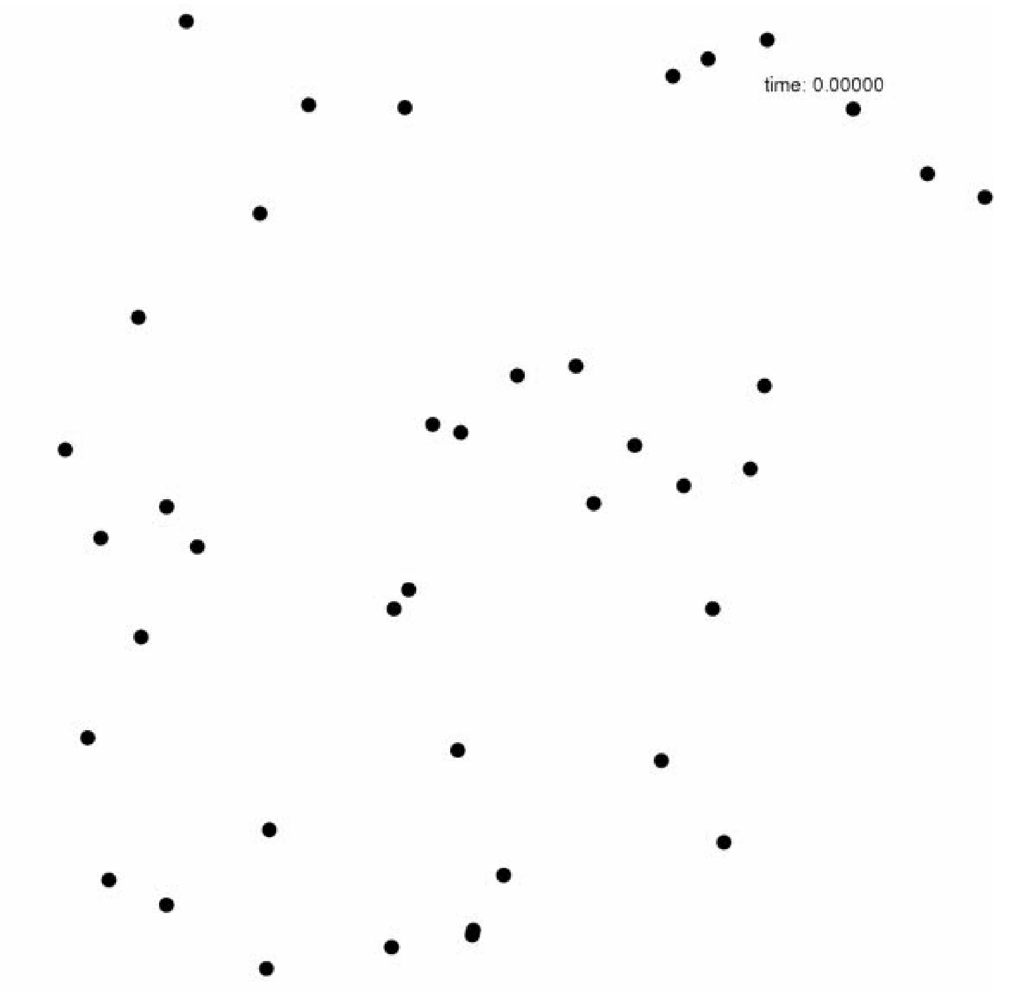


1. Node randomly chooses***** destination point in area
2. Moves with constant **speed** to this point
3. Waits for a certain **pause** time
4. Chooses***** a new destination, moves to this destination

... and so on ...

*****Sampled from uniform distribution

I Example: building a mobility model



I Random number ***generators*** …

Coin flipped on 2004-02-27 16:24:09 GMT
US 5¢ 1913 Liberty Head nickel

Your coin came down tails!

True random numbers from random.org

I Arithmetic (pseudo-) random number generators

- “Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin.” [John von Neumann, 1951]
- “It may seem perverse to use a computer, that most precise and deterministic of all machines conceived by the human mind, to produce ‘random’ numbers.” [Numerical Recipes]
- Why still use arithmetic methods?
  - Reproducability, portability
  - No I/O costs, high speed, low memory
  - Well analyzed
- In the future: random numbers ‘off the shelf’ (from DVD)?

… and now we focus on “pseudo-random numbers” ☺.

I What do *<u>we</u>* need to know about random number generation?

We are not going to design our own new RNG, but:

Case 1: Use an existing simulation tool

- **Examples: NS-2, OMnet++, Glomosim/Qualnet, Opnet, …**
- **What RNG does it use?**
- **Is it known to be a good one?**
- **Appropriate for our simulation task?**
- **Are we really sure?**

→ **Check references, Track RNG, do statistical Tests within simulation**

Case 2: Build your own simulator

- **Since using an existing tool might be 'overkill'**
- **Choose a RNG**
- **Check whether selection was appropriate**

→ **Know what is available, do statistical tests within simulation**

Structure of this lecture

- **Part I: What and why of random numbers**
- **Part II: Various random number generators**
  - **Criteria for random number generators**
  - **RNGs**
    - **Linear congruential generators**
    - **Generalized CGs**
    - **Tausworthe and related generators**
- **Part III: Evaluating and testing RNGs**
- **Summary and side notes**

Remember: if the RNG is not done appropriately, the 'results' are meaningless!

'Disclaimer': Since this set of slides should also be used as a lecture script, we introduce some math results and formulas for completeness.

II Criteria for random number generators

- <u>Uniformity, independence</u>: “Appear” to be distributed uniformly on [0,1] and independent
- <u>Speed and memory</u>: Fast, low memory
- <u>Reproducibility, portability</u>: Be able to reproduce a particular stream of random numbers. Why?
  - a. Makes debugging easier
  - b. Use identical random numbers to simulate alternative system configurations for sharper comparison
- <u>Uncorrelated streams</u>: Have provision in the generator for a large number of separate (nonoverlapping) streams of random numbers; usually such streams are just carefully chosen subsequences of the larger overall sequence

Most RNGs are fast, take very little memory

But beware: There are many RNGs in use that have extremely poor statistical properties

II Linear congruential generators

- **Introduced by Lehmer in 1951**
- **Produce a sequence of integers z_1, z_2, z_3, … as defined by the recursive formula**

$$z_i = (az_{i-1} + c) \mod m$$

m modulus

a multiplier

c increment

z_0 seed

$$u_i = z_i / m \quad \in [0, 1]$$

- **Increment c = 0: "multiplicative congruential generator"**
- **Otherwise: "mixed congruential method"**

II Linear congruential generators: example

$$Z_i = (5Z_{i-1} + 3) \pmod{16}$$

i	Z_i	u_i
0	7	
1	6	0.375
2	1	0.063
3	8	0.500
4	11	0.688
5	10	0.625

Length of period?

$\vdots$	$\vdots$	$\vdots$
14	13	0.813
15	4	0.250
16	7	0.438

II A good and a bad LCG

- **Good (in absolute terms ‘medium quality’): Marse and Roberts implementation (1983)**
  - **a = 630 360 016**
  - **c = 0**
  - **m = 2^{31} -1**

- **Bad: RANDU**
  - **a = 2^{16} + 3=65539**
  - **c = 0**
  - **m = 2^{31}**

II Choice of a, c, m: “wish list”

1. **Choice of modulus:**
   - Modulus m should be large (for a large potential period)
   - Integer division is costly; however, for $m = 2^k$ it is cheap.
2. **Choice of increment:**
   - Preferably, equals zero (less computations)
3. **Choice of multiplier:**
   - Should be chosen in a way that the actually achieved period is large.

But:

» **Some of these requirements are incompatible with each other.**

» **Still many choices left that lead to very bad RNGs.**

II Some theorem …

The linear congruential sequence defined by m, a, c, and Z_0 has period of length m if and only if

i) c is relatively prime to m;

The only positive integer that divides both m and c is 1. Thus, a multiplicative LCG cannot have full period

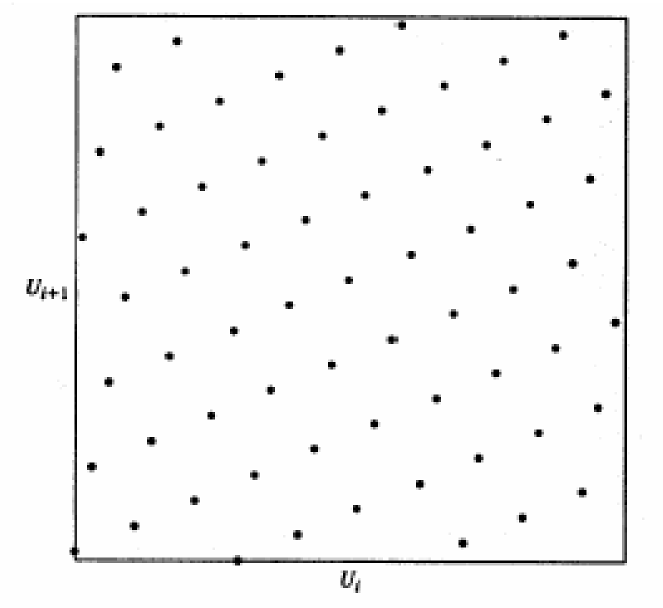
ii) b=a-1 is a multiple of p, for every prime p dividing m

iii) b is a multiple of 4, if m is a multiple of 4

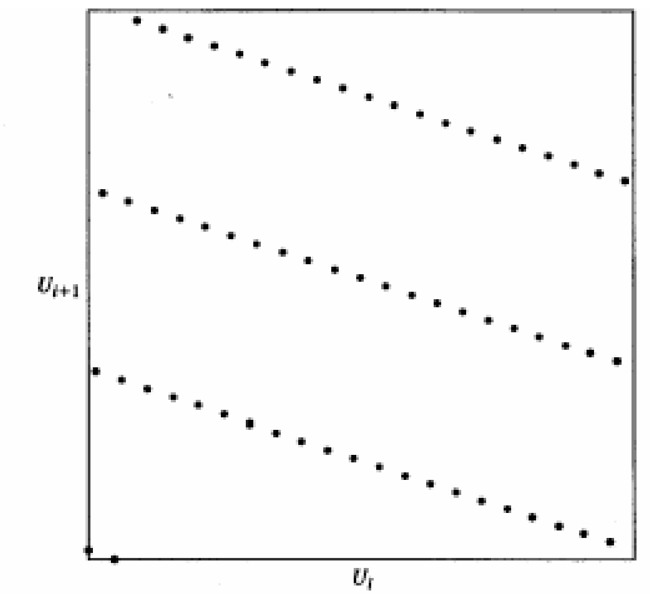
II Fundamental problems of LCGs

“Marsaglia” effect [Marsaglia, 1968, “Random numbers fall mainly in the planes]:

Overlapping d-tuples will all fall in a relatively small number of (d-1)-dimensional hyperplanes.



LCG: m=64, a=37, c=1

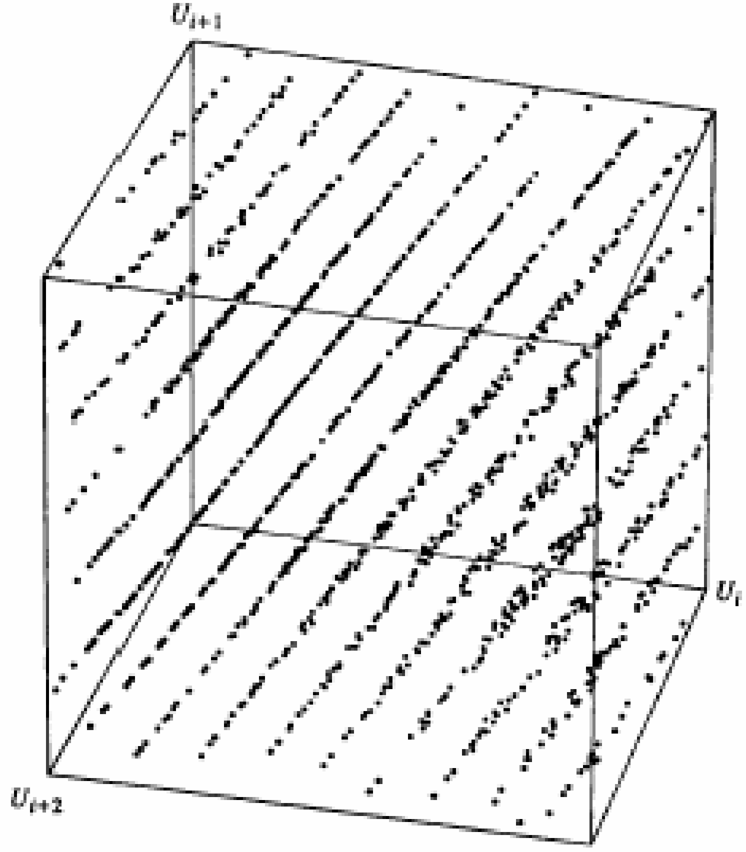


LCG: m=64, a=21, c=1

[LK2000]

II Illustrations from Law/Kelton

$m = 2^{31} = 2{,}147{,}483{,}648$, $a = 2^{16} + 3 = 65{,}539$, $c = 0$ (RANDU):



II Enhanced generators

- **Generalization of LCG:** $Z_i = g(Z_{i-1}, Z_{i-2}, \ldots) \pmod{m}$

- **Multiple recursive generator:**

$$g(Z_{i-1}, Z_{i-2}, \ldots) = a_1 Z_{i-1} + a_2 Z_{i-2} + \cdots + a_q Z_{i-q}$$

- **Composite generators, e.g., combined MRGs:**

Let Z_1 and Z_2 denote two MRGs.

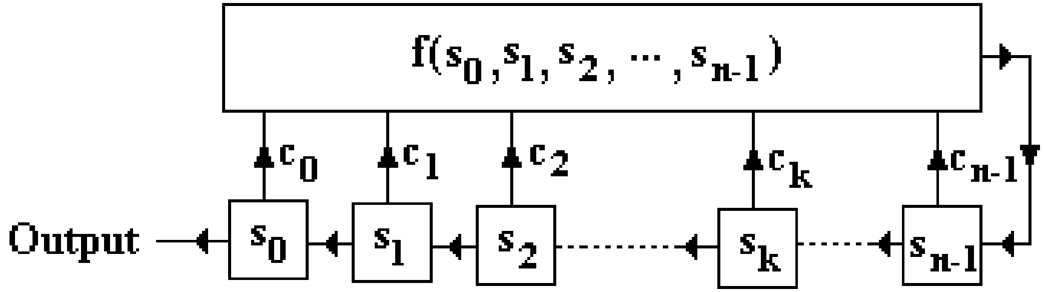
$$Y_i = (\delta_1 Z_{1,i} + \delta_2 Z_{2,i}) \pmod{m_1}$$

II Tausworthe and related generators

- Define a sequence b_1, b_2, … of bits via

  $b_i = (c_1\, b_{i-1} + c_2\, b_{i-2} + \ldots + c_q\, b_{i-q}) \bmod 2$

  where $c_1, \ldots, c_q$ are either 0 or 1.
- Recurrence like with MRGs, but operating on bits
- Can be implemented as feedback shift registers
- Pretty large periods can be achieved



II Current ‘star’: Mersenne Twister

Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random

Number Generator

MAKOTO MATSUMOTO, Keio University and the Max-Planck-Institut für Mathematik, Bonn

TAKUJI NISHIMURA, Keio University

A new algorithm called Mersenne Twister (MT) is proposed for generating uniform pseudorandom numbers. For a particular choice of parameters, the algorithm provides a super astronomical

period of $2^{19937} - 1$

and 623-dimensional equidistribution up to 32-bit accuracy, while using a working area of only 624 words.

http://www.math.keio.ac.jp/~matumoto/emt.html#Colt

C

II PRNGs in Practical Use

- **java 1.4.2 : LCG with 48Bit Seed**
- **glib (part of GTK): Mersenne Twister**
- **GSL (GNU Scientific Library): Almost anything**
- **ns-2: Multiple Recursive Generator (L'Ecuyer)**
- **… (Use the force, read the source ;-))**

II Simple Speed Comparison

```
$ time ./randspeed 1
Initializing rand()
Drawing 100000000 times...
Done!

real    0m2.951s
user    0m2.920s
sys     0m0.000s
```

Standard rand() function

```
$ time ./randspeed 2
Initializing grand() Mersenne Twister
Drawing 100000000 times...
Done!

real    0m1.332s
user    0m1.290s
sys     0m0.010s
```

Mersenne Twister as implemented in glib

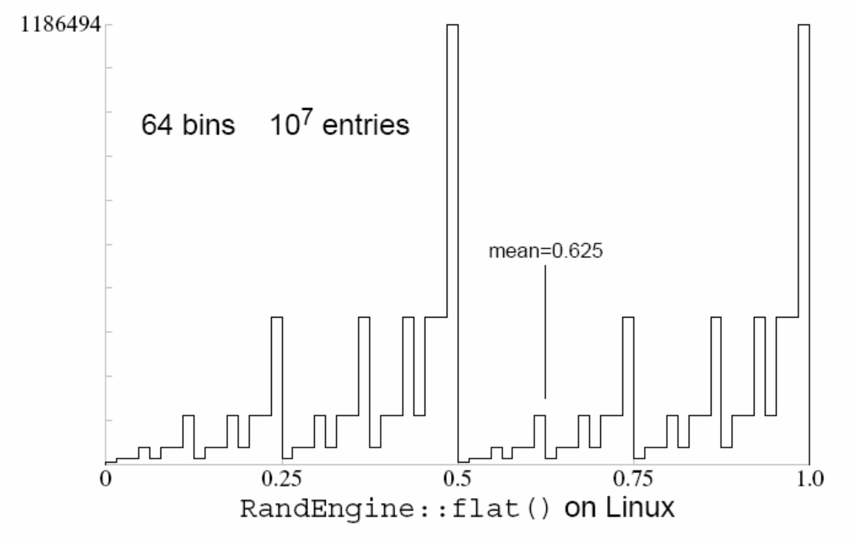
randspeed is a simple program available in the download area

III Criteria revisited

- **Uniformity, independence**
  - **Chi-square tests**
- **Speed, memory**
- **Reproducibility, portability**
- **Uncorrelated streams**

Portability problems:

- **rand function (ANSI C): implementation depends on choice of compiler**
- **How many random bits?**



Assumption: 16bit words, actual 32bit words
Taken from [J. Heinrich]

III Test for uniformity: problem statement

Observed samples

Expected values

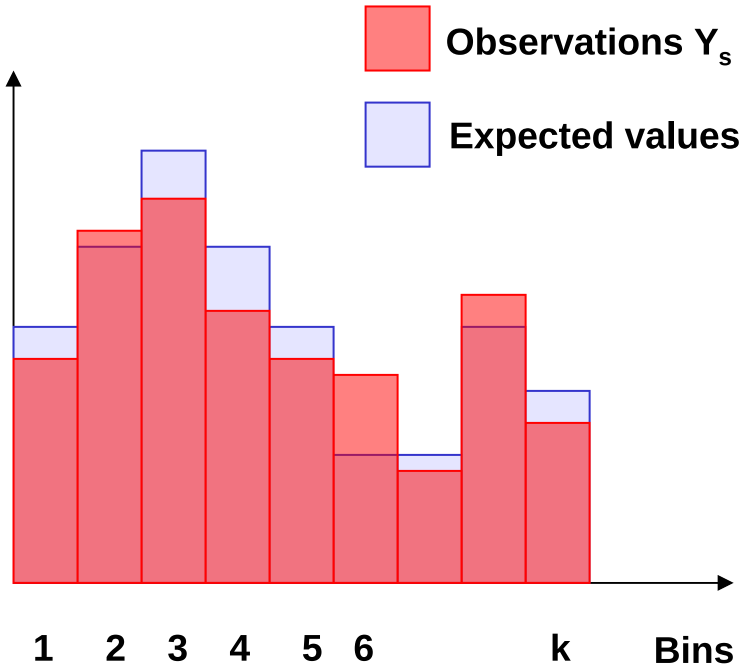
100 000 samples

100 bins

» **Sampled from uniform/non-uniform distribution?**

III Chi-square test: general set-up

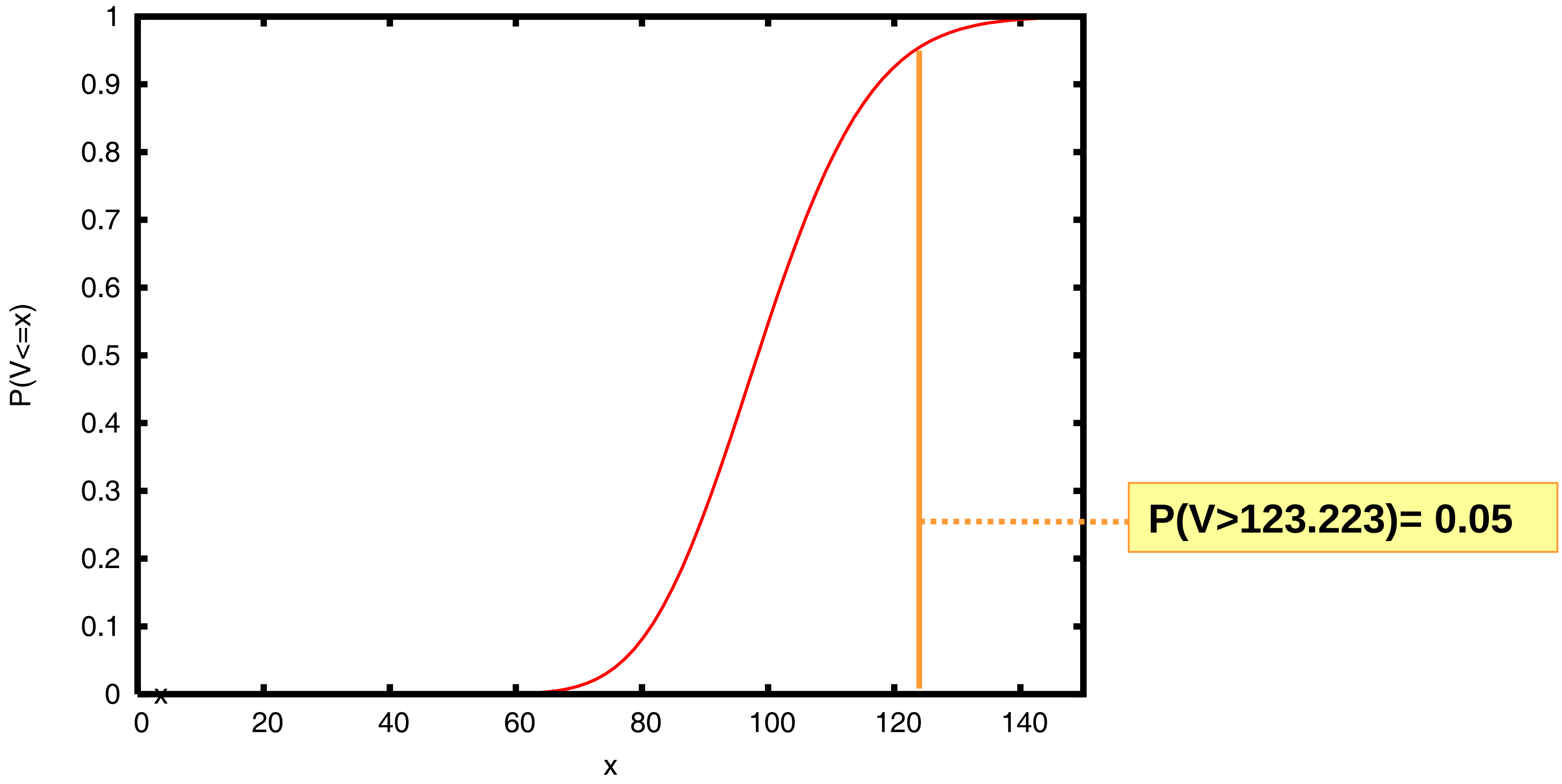
Compare actual observations (n samples) with expected values of the assumed distribution $\{p_s \mid 1 \leq s \leq k\}$ using the following 'mean-squared-error' statistics:



$$V = \sum_{s=1}^{k} \frac{(Y_s - np_s)^2}{np_s}$$

with Y_s being the number of observations that actually fall into category s.

III Chi-square distribution function for k=100



- One can now determine how ‘likely’ the value V actually is under the assumption of the probabilities for the various bins.

III Chi-square test

- **Hypothesis 'Observed sampling is coherent with the distribution assumption'**

- **Accept or reject hypothesis?**

- **Test with level α:**
  - **Compute $\chi_{1-\alpha}$ such that $P(X< \chi_{1-\alpha}) = 1 - \alpha$**
  - **$\chi_{1-\alpha}$ is called 'critical point' for level α**
  - **If $V> \chi_{1-\alpha}$ then reject hypothesis, otherwise accept**

- **Alternative: twosided with level α:**
  - **Compute $\chi_{\alpha 1}$ such that $P(X<\chi_{\alpha 1}) = 1 - \alpha/2$**
  - **Compute $\chi_{\alpha 2}$ such that $P(X<\chi_{\alpha 2}) = \alpha/2$**
  - **If $V> \chi_{\alpha 1}$ or $V < \chi_{\alpha 2}$ reject, otherwise accept.**

III Computation of critical points

- Need to know distribution function for chi-distribution with ‘k-1 degrees of freedom’ (where k is the number of bins)
- Need to know inverse of this distribution function.
- For large k’s, say k>30, one makes use of the critical points $z_{1-\alpha}$ of the normal distribution:

$$\chi_{k-1,1-\alpha} \approx (k-1)\left\{1-\frac{2}{9(k-1)}+z_{1-\alpha}\sqrt{\frac{2}{9(k-1)}}\right\}^3$$

- How to compute critical point of normal distribution?
  - Either table lookup or some ‘standardized’ inversion functions

Example: critical points for k=100

α=**0.025**: $\chi_{1-\alpha}$ **= 128.425**
α=**0.05** : $\chi_{1-\alpha}$ **= 123.223**
α=**0.1** : $\chi_{1-\alpha}$ **= 117.402**
α=**0.25** : $\chi_{1-\alpha}$ **= 108.089**

III Some technical stuff: chi-square distribution

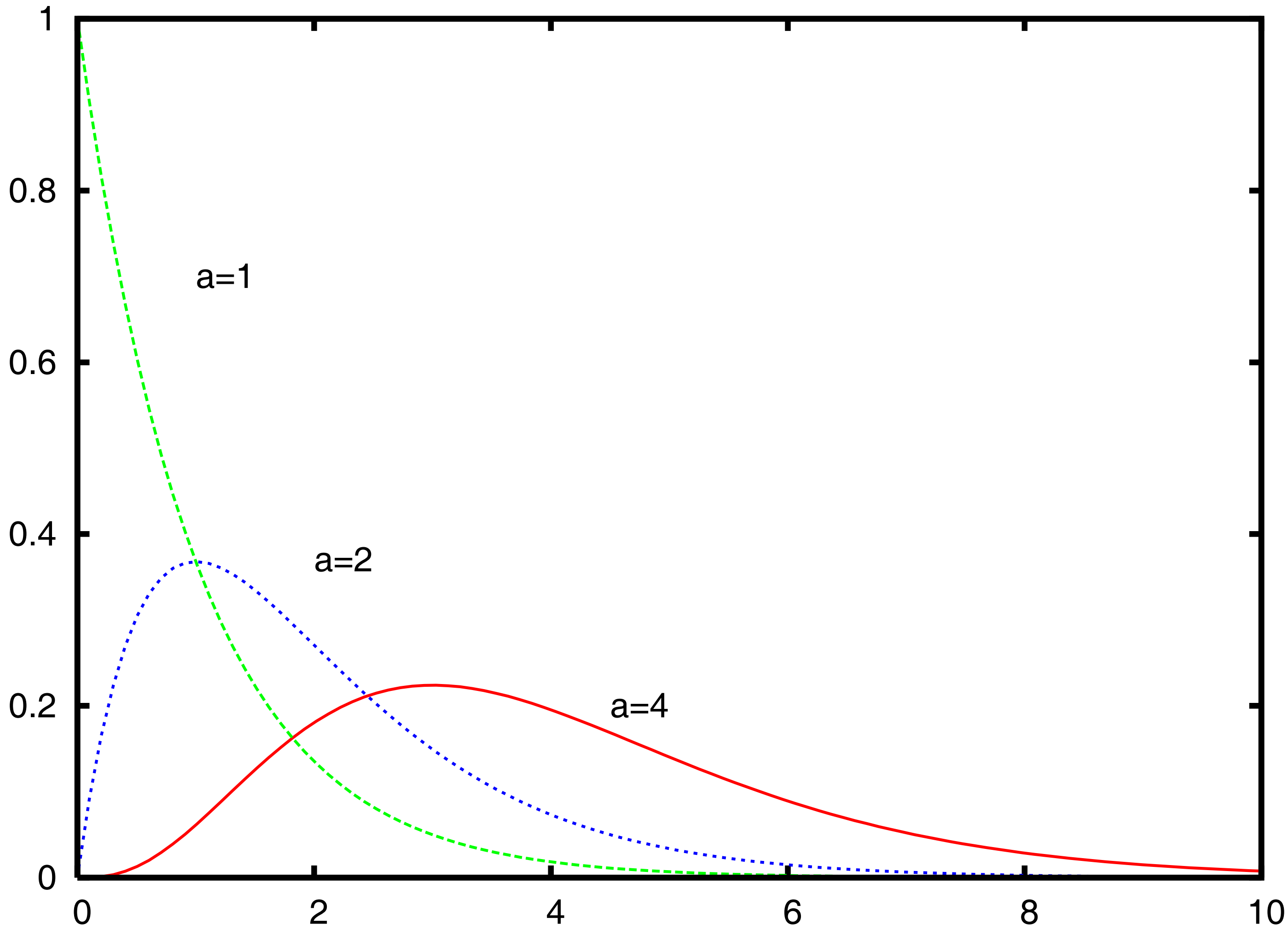
- **What is the distribution of V under the assumption of the distribution?**

- **The distribution is approximately the chi-square distribution with k-1 degrees of freedom, a special type of a gamma distribution with a=(k-1)/2, and b=2. The density function for a gamma distribution is given by**

$$f(x) = \frac{x^{a-1} e^{-x/b}}{b^a \Gamma(a)}$$

for $a, b > 0$, $0 \leq x \leq \infty$, and 0 elsewehre.

III Examples for various ‘degrees of freedom’

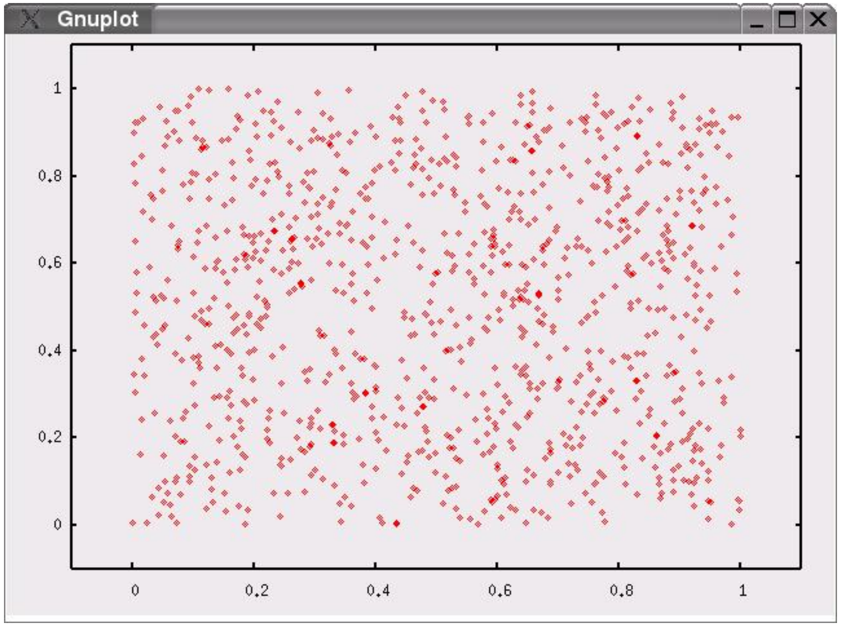


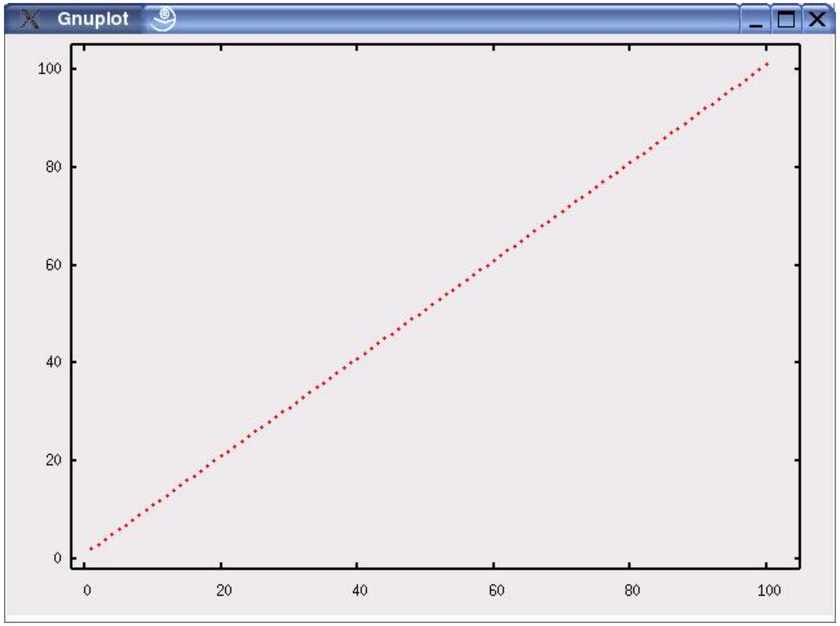
III How many samples do we need for a chi-square test?

- **χ^2-distribution only depends on ‘degrees of freedom’, i.e., number of categories.**
- **χ^2-distribution only approximation, i.e., only valid when the number of observations n is sufficiently large.**
- **Thus, in general, n should be made large.**
- **But: local ‘irregularities’ cannot be detected when n is large.**
  - **1, 2, 3, 4, 5, 6, 7, 8, 9, 10, …., 97, 98, 99, 100, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, …**
  - **Would also pass uniformity test …?**

III Comparing glibc 'rand' and 1,2,3,4, …

- … by looking at non-overlapping 2-tuples of the sequence x_1, x_2, x_3, x_4, …



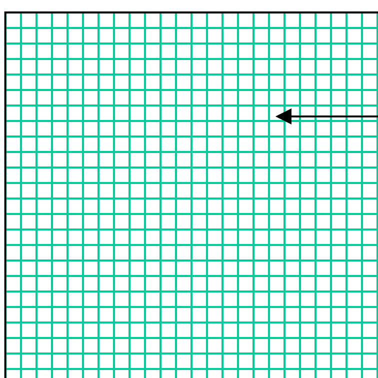


1000 samples

III Chi-square tests for independence of samples

- Serial tests: generalization of χ^2 test to higher dimensions
- Take non-overlapping successive sample to form d-tuples
  - $(x_1, x_2), (x_3, x_4), (x_5, x_6), \ldots$
  - $(x_1, x_2, x_3), (x_4, x_5, x_6), \ldots$

$$V(2) = \frac{k^2}{n} \sum_{j_1=1}^{k} \sum_{j_2=1}^{k} (Y_{j_1,j_2} - n/k^2)^2$$



Count in subinterval j_1, j_2

III The two methods for checking RNGs …

- … we have encountered in this lecture
  - Visual inspection (‘Marsaglia effect’)
  - Chi-square test

Summary, recommendations, side notes

- » **RNGs are a science for itself**
- » **As a simulation person, one acts as a customer of RNGs**
  - – **Probably not as a inventor of RNGs**
- » **But: one is responsible for checking whether the employed RNG is ‘good enough’ for the task under analysis**
- » **Visual tests can analyse ‘Marsaglia effect’**
- » **Statistical tests can easily be deployed to see obvious bugs**
  - – **Chi-square test**
- » **Combined MRGs and the MT are considered to be ‘state-of-the-art’**
- » **RNGs also extremely important for ‘security’**
  - – **RFC 1750 “Randomness Recommendations for Security”**

So what shall we do… ?

- Self-Implement and PRNG? — NO
- Check what PRNG is used by the library? — YES
- Check what properties this PRNG has? — If Needed
  - check the web / documentation
  - check yourself
- Always be aware of Properties! — Definitely
  - simulate n-times exactly the same
  - parallel streams

So what is the purpose of this lecture?

- Assume two nodes sending in a CSMA/CA style wireless network using ns-2 or any other simulator.
- randomized media access:
  - same stream (own PRNG with same seed)
  - → no data transport possible
  - dependant streams
  - → one node gets more share of the bandwidth

References

- **Knuth, D.E., *The Art of Computer Programming*, vol. 2, 3rd edition, Addison Wesley, 1998: chapter 3.**
- **Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., *Numerical Recipes in C*, 2nd edition, Cambridge University Press, 1992: chapter 7.**
- **Hechenleitner, B., Entacher, K., *On shortcomings of the ns-2 random number generator*.**