Introduction to Computational Thinking and Programming for CFD

Module 13251

Dr. rer. nat. Marten Klein
Numerical Fluid and Gas Dynamics
BTU Cottbus - Senftenberg

Brandenburg
University of Technology
Cottbus - Senftenberg

8 Random numbers

Examples

- How do we convert "Text" $\xrightarrow[\textbf{Encryption}]{}$ "Cipher-text"?
- What are encryptions? ⟶ Mathematical Algorithms
- What are the most important numbers for encryption?
  - PRIME NUMBERS
- But there are other numbers that are as important as prime numbers…
  - RANDOM NUMBERS

General requirements

- Requirements:
  - Should be <u>un</u>predictable.
    (Examples: one-time passwords, unpredictable perturbations)
  - Should be <u>un</u>biased, thus having a uniform distribution.
    (All numbers should be equally likely to occur unless we have physical reasons to change that …)
- Types of random numbers:
  - True random numbers → from a complex physical process, like rolling a dice, radioactive decay, quantum experiments
  - Pseudo random numbers → obtained from a (deterministic) algorithm

Random number generators (RNGs)

- **True RNGs:** They use an unpredictable physical system to generate numbers (like rolling a dice, recording atmospheric noise)

- **Pseudo RNGs:** They use mathematical algorithms for construction, thus they <u>cannot</u> be truly random (all kinds of arithmetically generated number sequences from a computer)

  - *Pseudo means `a kind of but not really the same' or `pretty close'*

  - *We can predict the weather a few days in a row, but we actually cannot fully rely on the weather report*

Overview of some RNGs

	True random number generators TRNG	Pseudo random number generators PRNG
Mechanism	**Physical element involved** (Example: Dice roll)	No physical means, but instead a discrete **mathematical algorithm**
Uniformity	**Yes, if care is taken.**	**Yes, if care is taken.**
Independence (Statistically)	**Independent of the previous action** (Example: Dice roll - The construction mechanism for the numbers is unknown or uncontrollable.)	**Not independent** of the previous action - **periodic** - **deterministic (predictable)** (Why? Because there is an algorithm, hence the construction is known.)
Efficiency	**Usually time consuming** (inefficient)	**Usually very efficient** (millions of calculations done by a computer)

Example: Linear congruential generator (LCG)

- Mathematical formula:

$X_{n+1} = a X_n + c \quad \text{mod } m$

Where $X_0, a, c < m$ all values are positive integers

X_0 is the seed (starting point)

- Example: $X_0 = 1$, $a = 2$, $c = 3$, $m = 5$

$\Rightarrow X_1 = 2*1 + 3 \text{ (mod 5)}$

$\Rightarrow X_1 = 5 \text{ (mod 5)}$

$\Rightarrow X_1 = 0$

$X_2 = (a*X_1 + c) \text{ mod } 5 = (2*0 + 3) \text{ mod } 5 = 3$

$X_3 = (a*X_2 + c) \text{ mod } 5 = (2*3 + 3) \text{ mod } 5 = 9 \text{ (mod 5)} = 4$

$X_4 = 11 \text{ (mod 5)} = 1$ → seed! Now the sequence will cycle again

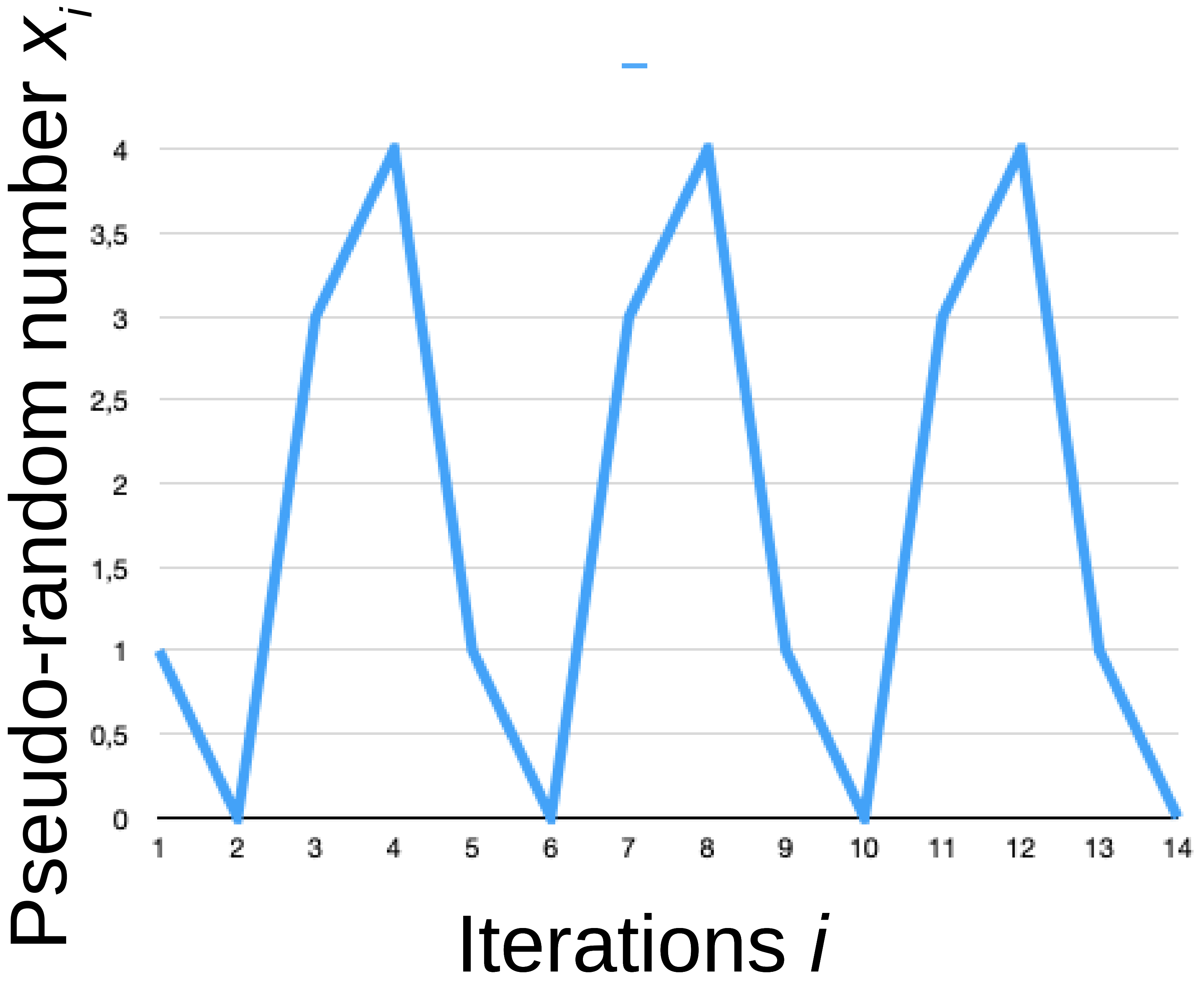
We have generated a seemingly random permutation.
The sequence 1, 0, 3, and 4, however, will retrace itself.

- Why do we only get a sequence of 4 numbers?
- m is 5. We have limited the sequence to the value range {0, 1, 2, 3, 4}
- We need large seed (c) and large modulus (m) for the pseudo random numbers to be \`reasonably good’.

Results of LCG for m = 5

Table 1

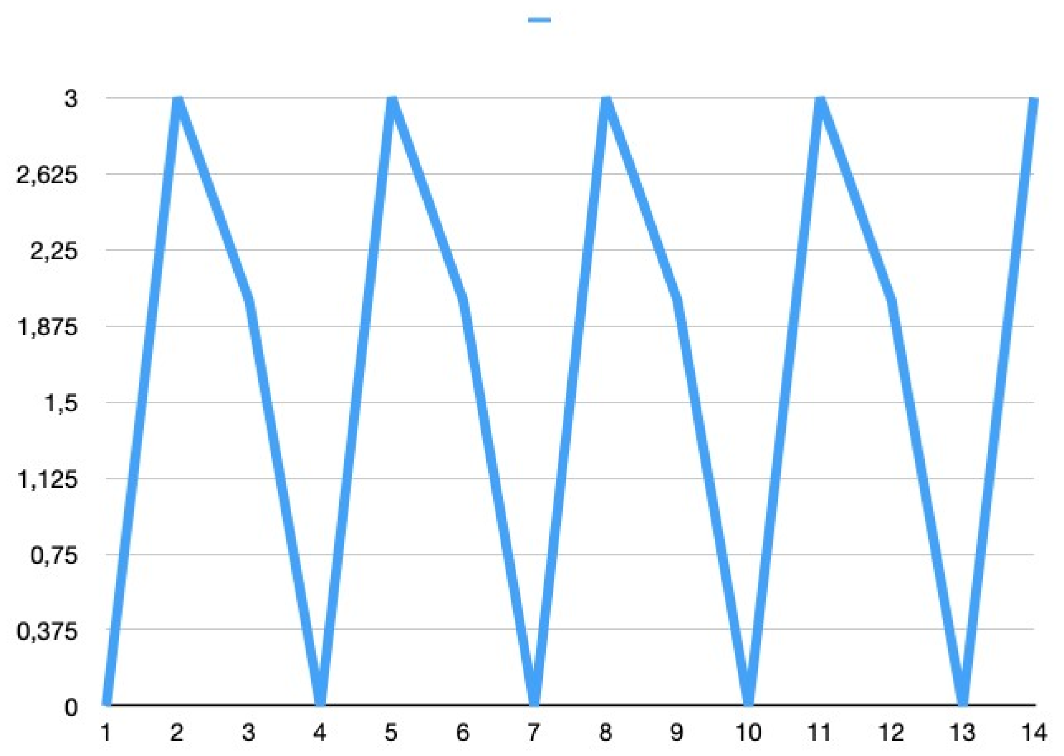
a	2		
c	3		
M	5		
1	1		
2	0		
3	3		
4	4		
5	1		
6	0		
7	3		
8	4		
9	1		
10	0		
11	3		
12	4		
13	1		
14	0		



Results of LCG for various m I

Table 1

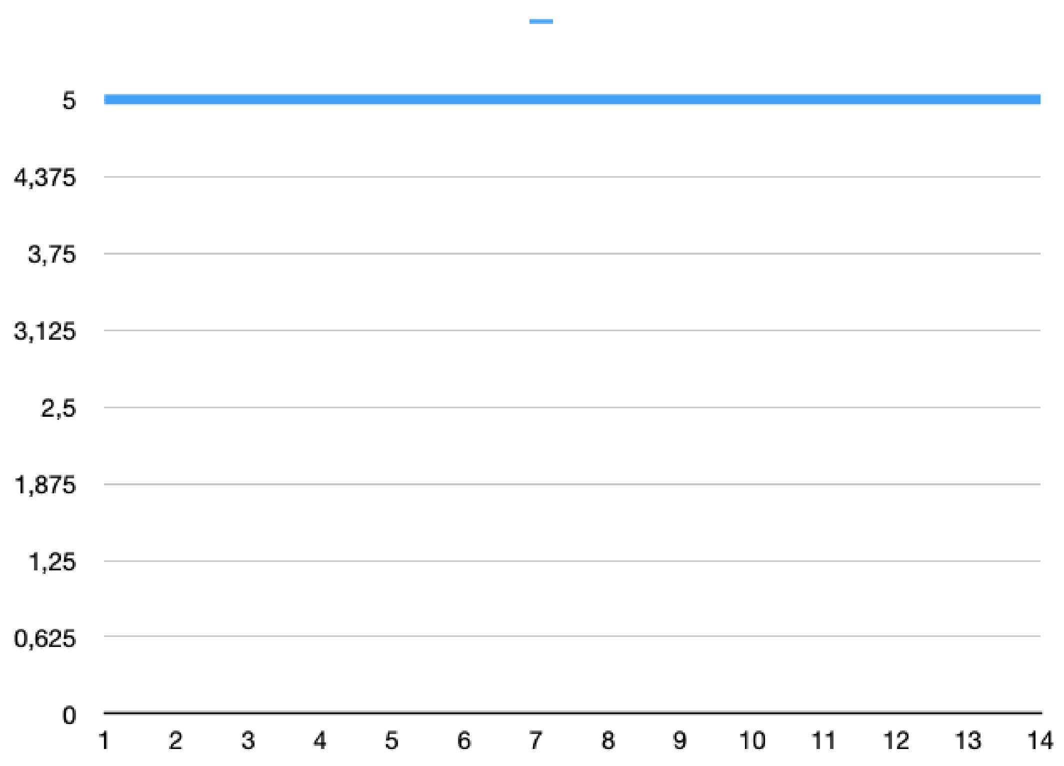
a	2		
c	3		
M	7		
	3		
	2		
1	0		
2	3		
3	2		
4	0		
5	3		
6	2		
7	0		
8	3		
9	2		
10	0		
11	3		
12	2		
13	0		
14	3		



m = 7

Table 1

a	2		
c	3		
M	8		
	3		
	1		
1	5		
2	5		
3	5		
4	5		
5	5		
6	5		
7	5		
8	5		
9	5		
10	5		
11	5		
12	5		
13	5		
14	5		



m = 8

Results of LCG for various m II

Table 1

a	2		
c	3		
M	10		
	3		
	9		
1	1		
2	5		
3	3		
4	9		
5	1		
6	5		
7	3		
8	9		
9	1		
10	5		
11	3		
12	9		
13	1		
14	5		

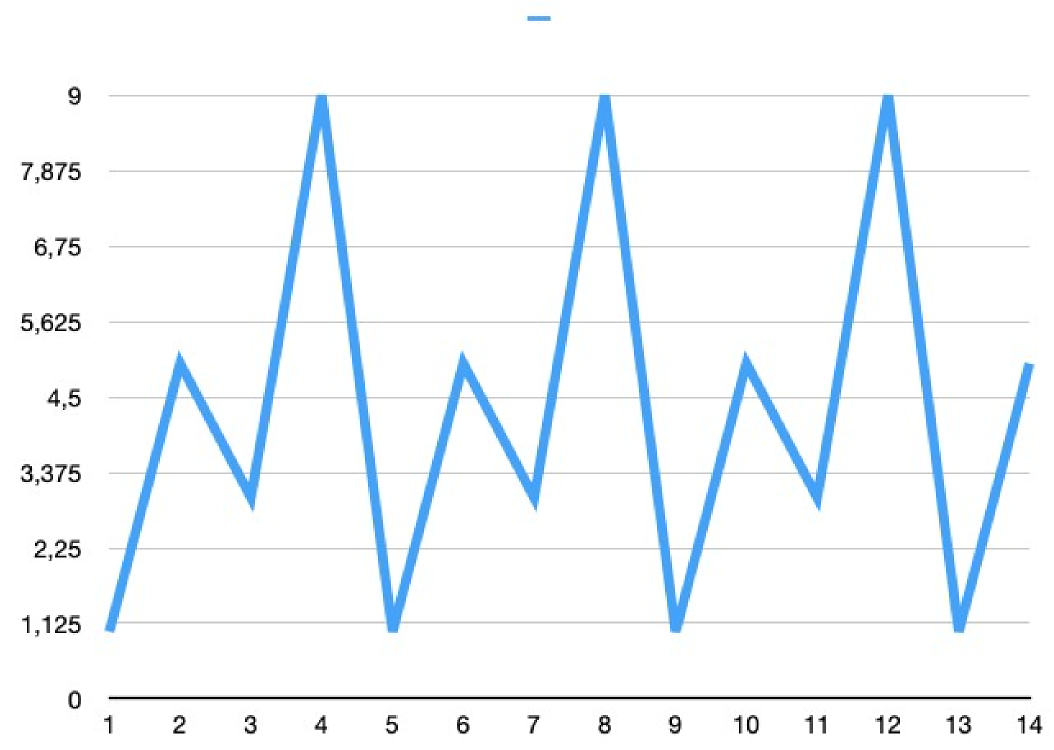
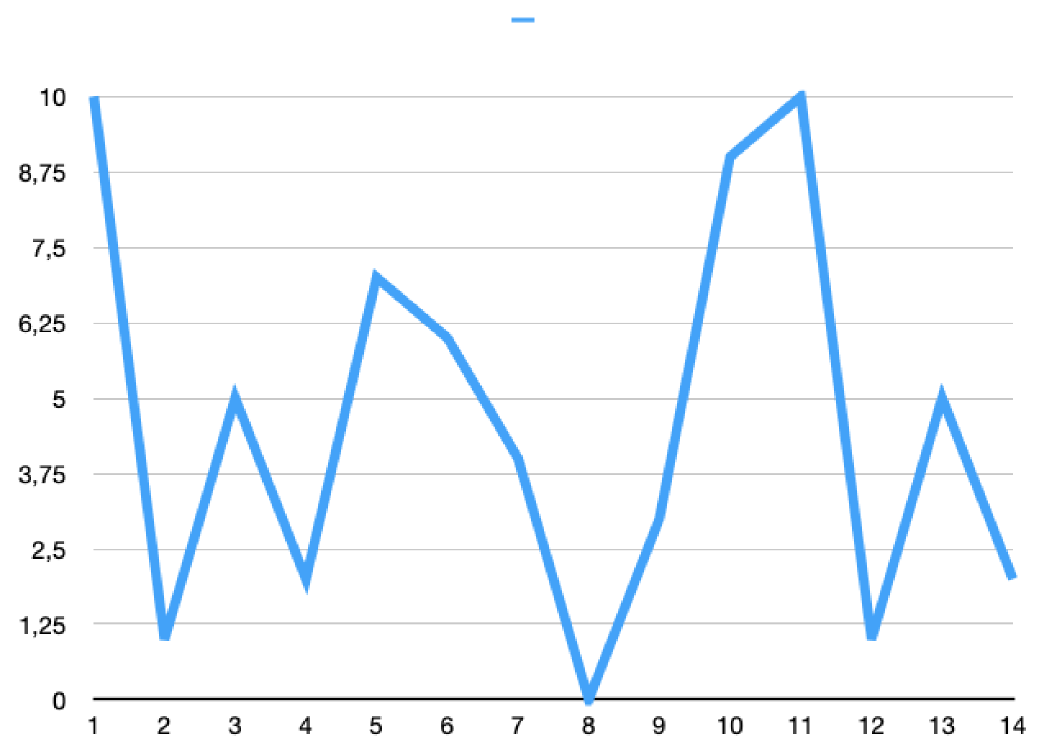


Table 1

a	2		
c	3		
M	11		
	3		
	9		
1	10		
2	1		
3	5		
4	2		
5	7		
6	6		
7	4		
8	0		
9	3		
10	9		
11	10		
12	1		
13	5		
14	2		



m = 10

m = 11

Observations for the LCG

- The pseudo-random **integer range of values** is governed by and increases with the modulus *m*. In some cases it can be smaller, but m is always excluded

$$0 \leq x_i \leq m-1 \qquad \forall i$$

- We obtain **normalized pseudo-random numbers** in the half-open interval [0,1) by floating-point division float(x_i) / float(m)

$$r_i = \frac{x_i}{m} \qquad 0 \leq r_i < 1 \qquad \forall i$$

Summary of the LCG

- Why is m = 11 better than m = 10 and, in particular, than m = 8?
  - 10 factors into 1 * 2 * 5, but 11 is prime
  - 8 is a poor choice since seed 3 and increment 5 yields 5+3, which is commensurate with the present value of m
- Why does the graph change as the modulus m increases?
  - Because we have a wider range of values to choose from.
    The limit has been increased.
- If we further increase the value of m, what will happen? *Try it …*

Applications

- Generation of **perturbed initial conditions** or **random forcing** for CFD applications in order to seed turbulence

- Generation of an **ensemble of flow realizations** by variation of a parameter, boundary, or initial conditions (like in weather forecast, climate prediction, combustion applications)

- Utilization in **stochastic modeling methods** for fluid flow problems
  - stochastic turbulence and mixing models
  - tracer dispersion and tracer diffusion models (random walks)
  - Monte Carlo simulations
  - Ohrenstein–Uhlenbeck processes for stochastic modeling of small-scale noise

Sampling uniform random numbers in Python

- The module **numpy.random** provides means to obtain pseudo random numbers and at least one PRNG (the Mersenne Twister)

- Get an array of N = 10,000 pseudo random numbers $\{r_i\}$ that are sampled from a **uniform distribution** over the interval [0,1], hence, $0 \leq r_i \leq 1$ for all $i = 0,1,2,\ldots,N-1$

```
import  numpy.random  as  rnd
r = rnd.rand(10000)
```

- See the documentation for details: https://numpy.org/doc/stable/reference/random/generated/numpy.random.rand.html

Sampling nonuniform random numbers in Python

- It is possible to sample from **some preimplemented nonuniform distributions** (like Gaussian or exponential distribution etc.)
  See here for details: https://numpy.org/doc/1.16/reference/routines.random.html

- Sampling from **arbitrary distributions**, that is, sampling from an experimentally measured or CFD simulated *probability density function (PDF)* often requires case-specific methods such as:
  - Cumulative density function (CDF) inversion →
  - Monte Carlo methods
  - Rejection sampling

CDF: $c(x) = \int_{-\infty}^{x} p(x')\,\mathrm{d}x'$

- $c(x)$ is monotonic and has values in [0,1]
- $c(x)$ is steepest where $p(x)$ has a maximum
- Procedure:
  1. Calculate a random number r in [0,1]
  2. Invert the CDF to yield random $x = c^{-1}(r)$ obeying the specified PDF

Example: CDF inversion method

- **Independent events** often follow an **exponential distribution**

  (e.g., radioactive decay, but approx. also `eddy events' in stochastic mixing and turbulence models LEM and ODT)

- **Exponential PDF *p*(*t*)** and **CDF *c*(*t*)** for time interval *t* between two successive events; the typical (average) time interval is τ

$$p(t) = \frac{1}{\tau} \mathrm{e}^{-t/\tau} \quad \Rightarrow \quad c(t) = \int_0^t p(t')\,\mathrm{d}t' = 1 - \mathrm{e}^{-t/\tau}$$

Inverse CDF $c^{-1}(r)$ is called for a **uniform random number $0 \le r \le 1$** to yield time increment $0 \le t < \infty$ sampled from exponential distribution

$$r = c(t) \quad \Rightarrow \quad r = 1 - \mathrm{e}^{-t/\tau} \quad \Rightarrow \quad \boxed{t = -\tau \ln(1 - r)}$$

→ *Exercise!*

Keywords

- True and pseudo random numbers
- Algebraic pseudo random number generation