Numerical Simulation and Scientific Computing I

Lecture 7:
Random Numbers & Monte Carlo Integration

Clemens Etl, Paul Manstetten,
and Josef Weinbub

Institute for Microelectronics
TU Wien

nssc@iue.tuwien.ac.at

WS 2023

Quiz

- Q1: Represent the following matrix in the CRS format

$$\begin{bmatrix} 1 & 4.6 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 3.7 & 0 \\ 0 & 6 & 2.7 & 0 & 0 \\ 0 & 4.6 & 0 & 4.8 & 9.4 \\ 0 & 0 & 5.6 & 0 & 1 \end{bmatrix}$$

Quiz

- Q1: Represent the following matrix in the CRS format

$$\begin{bmatrix} 1 & 4.6 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 3.7 & 0 \\ 0 & 6 & 2.7 & 0 & 0 \\ 0 & 4.6 & 0 & 4.8 & 9.4 \\ 0 & 0 & 5.6 & 0 & 1 \end{bmatrix}$$

- 1st step: Put the values in row-major order

$$V = [1 \quad 4.6 \quad 8.5 \quad 3.7 \quad 6 \quad 2.7 \quad 4.6 \quad 4.8 \quad 9.4 \quad 5.6 \quad 1]^T$$

Quiz

- Q1: Represent the following matrix in the CRS format

$$\begin{bmatrix} 1 & 4.6 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 3.7 & 0 \\ 0 & 6 & 2.7 & 0 & 0 \\ 0 & 4.6 & 0 & 4.8 & 9.4 \\ 0 & 0 & 5.6 & 0 & 1 \end{bmatrix}$$

- 2^{nd} step: Assign the corresponding column indices

$$V = [1 \quad 4.6 \quad 8.5 \quad 3.7 \quad 6 \quad 2.7 \quad 4.6 \quad 4.8 \quad 9.4 \quad 5.6 \quad 1]^T$$

$$JA = [0 \quad 1 \quad 2 \quad 3 \quad 1 \quad 2 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4]^T$$

Quiz

- Q1: Represent the following matrix in the CRS format

$$\begin{bmatrix} 1 & 4.6 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 3.7 & 0 \\ 0 & 6 & 2.7 & 0 & 0 \\ 0 & 4.6 & 0 & 4.8 & 9.4 \\ 0 & 0 & 5.6 & 0 & 1 \end{bmatrix}$$

- 3rd step: Identify the indices in V where a new row starts

$$V = [1 \quad 4.6 \quad 8.5 \quad 3.7 \quad 6 \quad 2.7 \quad 4.6 \quad 4.8 \quad 9.4 \quad 5.6 \quad 1]^T$$

$$JA = [0 \quad 1 \quad 2 \quad 3 \quad 1 \quad 2 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4]^T$$

$$IA = [0 \quad 2 \quad 4 \quad 6 \quad 9 \quad 11]^T$$

Quiz – Poll 1

- Q2: Under which conditions would you use the CG method instead of GMRES?
- A) The matrix is indefinite
- B) The problem benefits from preconditioning
- C) The problem is ill-posed
- D) All eigenvalues are positive
- E) The LU factorization is known

Quiz – Poll 1

- Q2: Under which conditions would you use the CG method instead of GMRES?
- A) The matrix is indefinite
- B) The problem benefits from preconditioning
- C) The problem is ill-posed
- **D) All eigenvalues are positive**
- E) The LU factorization is known

Quiz – Poll 2

- Q3: What information from a matrix A could be useful to help choosing an adequate solver?
- A) Maximum norm
- B) Set of eigenvalues
- C) QR decomposition
- D) Transpose

Quiz – Poll 2

- Q3: What information from a matrix A could be useful to help choosing an adequate solver?

- A) Maximum norm

- **B) Set of eigenvalues**

- C) QR decomposition

- D) Transpose

Outline

- Introduction to Randomness
- Random Number Generators (RNGs)
  - Linear Congruential Generators
  - Mersenne Twister
  - Hardware
- C++11 `<random>`
- Monte Carlo Integration

Take-home message

- Be careful with your choice of RNG
- Seeding is a consequence of pseudo-RNG
- Watch out for pitfalls on parallel systems

Main References

- Random Numbers and Computers
  - Author: Ronald T. Kneusel
  - eBook available: https://catalogplus.tuwien.ac.at:443/UTW:UTW:TN_springer_s978-3-319-77697-2_453445

Additional References

- Handbook of Monte Carlo Methods
  - Authors: D. P. Kroese, T. Taimre, Z. I. Botec
  - eBook available: https://catalogplus.tuwien.ac.at:443/UTW:UTW:TN_scopus2-s2.0-84949783693

- Monte Carlo Strategies in Scientific Computing
  - Author: Jun S. Liu
  - https://catalogplus.tuwien.ac.at:443/UTW:UTW:UTW_alma2146615720003336

Randomness

- How to formally define randomness?

- Philosophical conundrum: does randomness even exist?
  - Good thing that it is not our job to figure it out!

- Too hard - operational definition of a random sequence [Kneusel]:
  - a sequence of numbers n, within some bounded range, where it is not possible to predict n_{k+1} from any combination of preceding values $n_i, i = 0,1,\dots,k$.

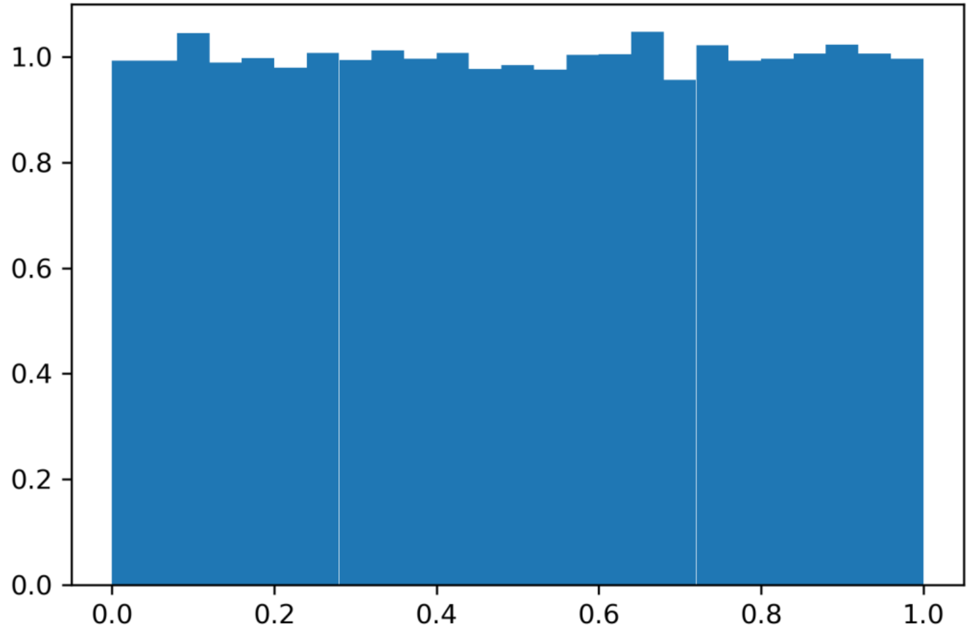
“True” Random vs. Pseudorandom

- “True” Randomness
  - From here on, we will assume that these exist
  - Found in the physical world
  - Examples?

- Pseudorandomness
  - [Kneusel]: A pseudorandom sequence is a deterministically generated sequence of numbers that is indistinguishable from a true random sequence of numbers
  - Perfect indistinguishability is impossible – but how much is enough?
  - Determinism: introduces the concept of a seed
    - Seed: starting point of the deterministic sequence

Uniform Random Numbers

- Even if n_{k+1} is truly non-predictable, it can still have a distribution – shows up in a histogram
  - Drawing from a parent distribution
- Uniform (real) Distribution: $n_k \in [0,1)$
  - Unless otherwise stated, it is often the default



Linear Congruential Generators

- Nomenclature
  - a: multiplier
  - c: increment
  - m: modulus
  - x_0: seed

$$x_{n+1} = (ax_n + c) \bmod m$$

- Notes
  - Generates integers $\in [0, m)$. Floats: $f_n = \frac{x_n}{m}$
  - Periodicity is a big problem – can be lower than m
  - If $x_n = x_m$ then $x_{n+1} = x_{m+1}$
  - The method is defined by the choice of a, c and m
- Examples
  - GCC: $a = 1103515245$, $c = 12345$, $m = 2^{31} - 1$
  - C++11 minstd_rand: $a = 48271$, $c = 0$, $m = 2^{31} - 1$

LCGs – Poll 3

- What is the issue of using this generator with $m = 2^{31} - 1$ to generate a double?

$$x_{n+1} = (ax_n + c) \bmod m$$

- A) The periodicity is not high enough to cover the 53-bit significand
- B) It is impossible to seed this generator with a double
- C) You cannot generate a double from an integer
- D) The equation $f_n = \frac{x_n}{m}$ doesn't hold anymore

LCGs – Poll 3

- What is the issue of using this generator with $m = 2^{31} - 1$ to generate a double?

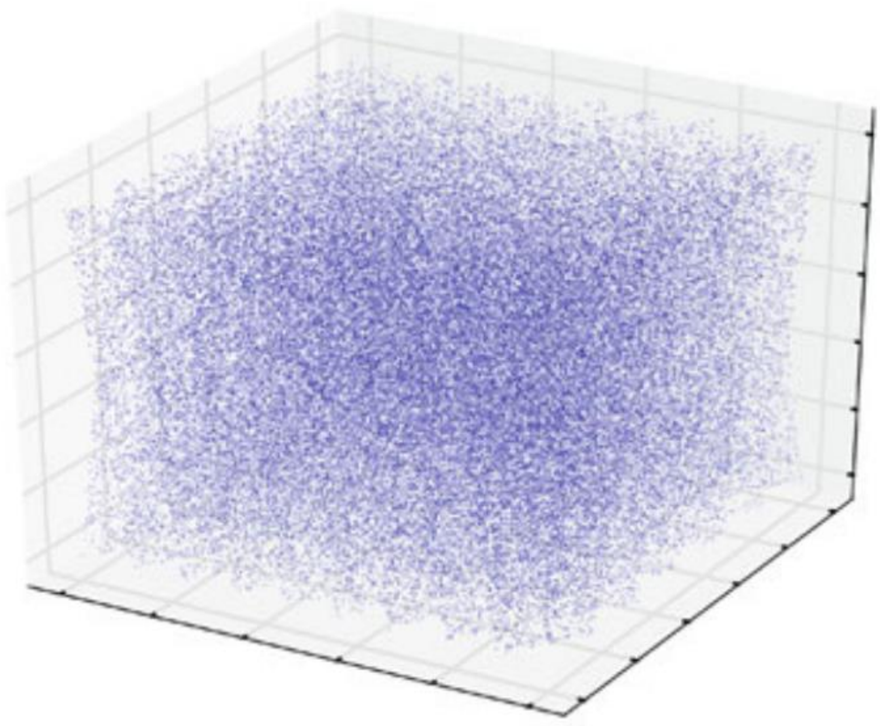
$$x_{n+1} = (ax_n + c) \bmod m$$

- **A) The periodicity is not high enough to cover the 53-bit significand**
- B) It is impossible to seed this generator with a double
- C) You cannot generate a double from an integer
- D) The equation $f_n = \frac{x_n}{m}$ doesn't hold anymore

A Lesson from History: RANDU

- How bad can it be if we choose poorly?
  - RANDU: $a = 65593$, $c = 0$, $m = 2^{31}$

$$x_{n+2} = 6x_{n+1} - 9x_n$$

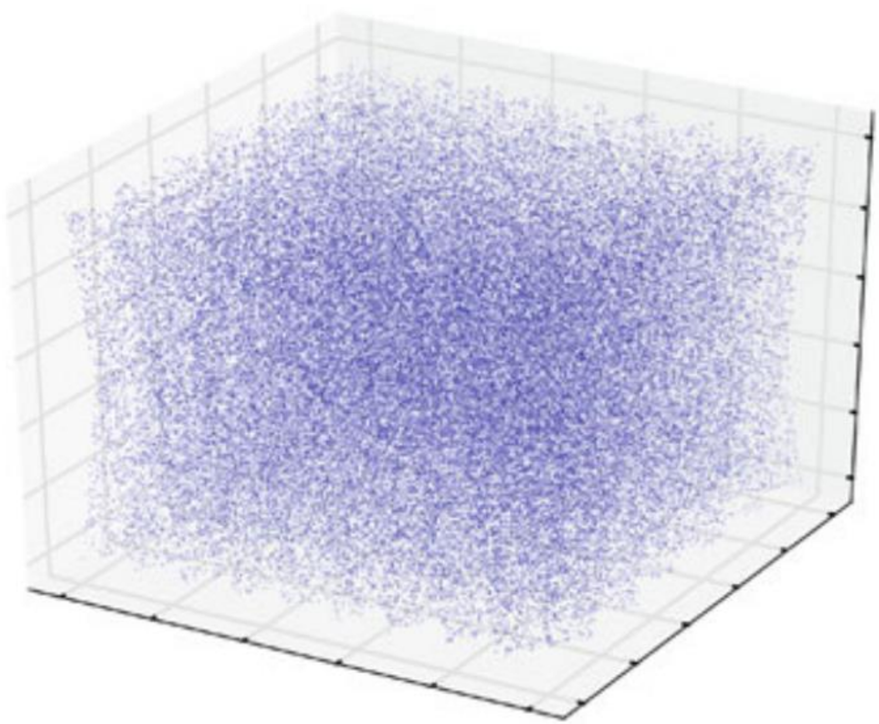


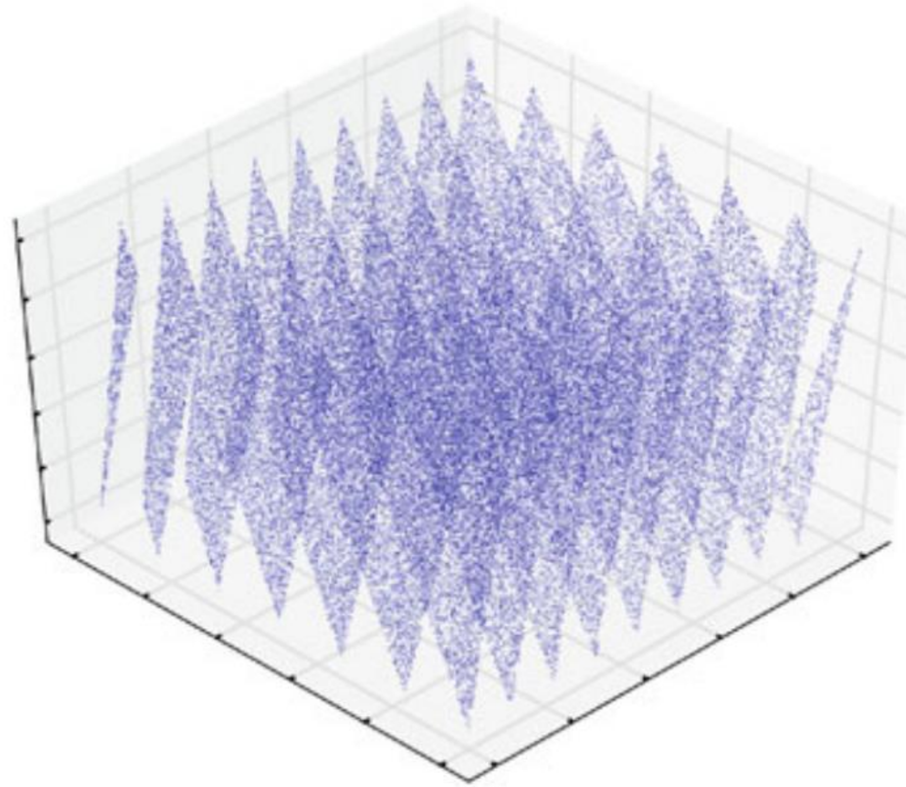
Source: Kneusel: Random Numbers and Computers, 1st ed., 2018, Springer

A Lesson from History: RANDU

- How bad can it be if we choose poorly?
  - RANDU: $a = 65593$, $c = 0$, $m = 2^{31}$

$$x_{n+2} = 6x_{n+1} - 9x_n$$

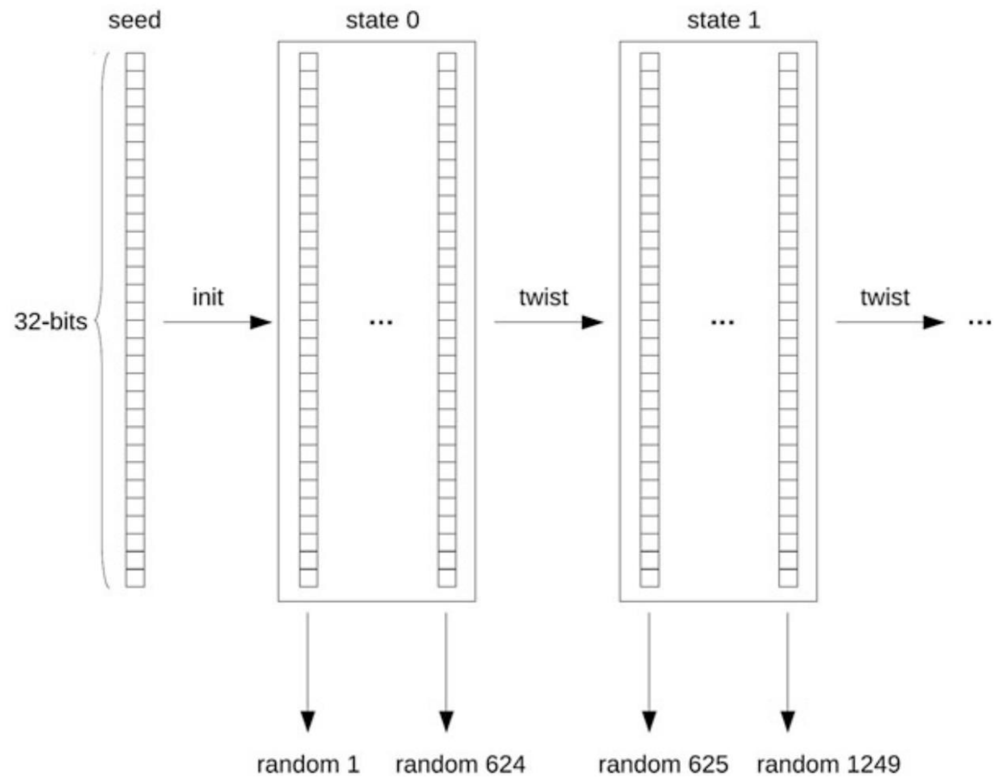




Source: Kneusel: Random Numbers and Computers, 1st ed., 2018, Springer

Mersenne Twister

- MT is the current *de facto standard* for non-cryptographic applications



- Larger memory and construction requirements!

Source: Kneusel: Random Numbers and Computers, 1st ed., 2018, Springer

Choosing your RNG

- Performance is only part of the equation

- MINSTD is only 32 bits
  - a, c and m are linked
  - Don't mess around with constants!

- The performance penalty of MT can be mild (up to 50%)

- Other RNG may be available!
  - May involve other performance tradeoffs…

- C rand()
  - NOOOOOOO (RAND_MAX ≥ 32767)

https://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful

Generator	Time (s)
MINSTD	9.164 ± 0.006
xorshift32	11.097 ± 0.004
xorshift128+	11.853 ± 0.028
CMWC	12.671 ± 0.003
Middle Weyl	12.849 ± 0.028
xorshift1024*	12.985 ± 0.004
Mersenne Twister	14.353 ± 0.007
KISS64	17.252 ± 0.015
Philox	18.097 ± 0.048
Threefry	32.442 ± 0.010

Source: Kneusel: Random Numbers and Computers, 1st ed., 2018, Springer

Choosing your RNG – Poll 4

- How does MT compare wrt. MINSTD?
  - Higher performance: less time to generate 1 RN

- A) Higher performance, higher periodicity, higher memory requirements
- B) Lower performance, higher periodicity, higher memory requirements
- C) Lower performance, lower periodicity, higher memory requirements
- D) Lower performance, higher periodicity, lower memory requirements
- E) Higher performance, lower periodicity, lower memory requirements

Choosing your RNG – Poll 4

- How does MT compare wrt. MINSTD?
  - Higher performance: less time to generate 1 RN

- A) Higher performance, higher periodicity, higher memory requirements
- **B) Lower performance, higher periodicity, higher memory requirements**
- C) Lower performance, lower periodicity, higher memory requirements
- D) Lower performance, higher periodicity, lower memory requirements
- E) Higher performance, lower periodicity, lower memory requirements

Choosing your RNG – Poll 5

- For a certain random sequence, in the past we drew the number 189 immediately after the number 25. Now in the same sequence, we just drew the number 25 again.

- A) MT: the next number will be 189. LCG: the next number will be 189
- B) MT: the next number will be random. LCG: the next number will be 189
- C) MT: the next number will be 189. LCG: the next number will be random
- D) MT: the next number will be random. LCG: the next number will be random

Choosing your RNG – Poll 5

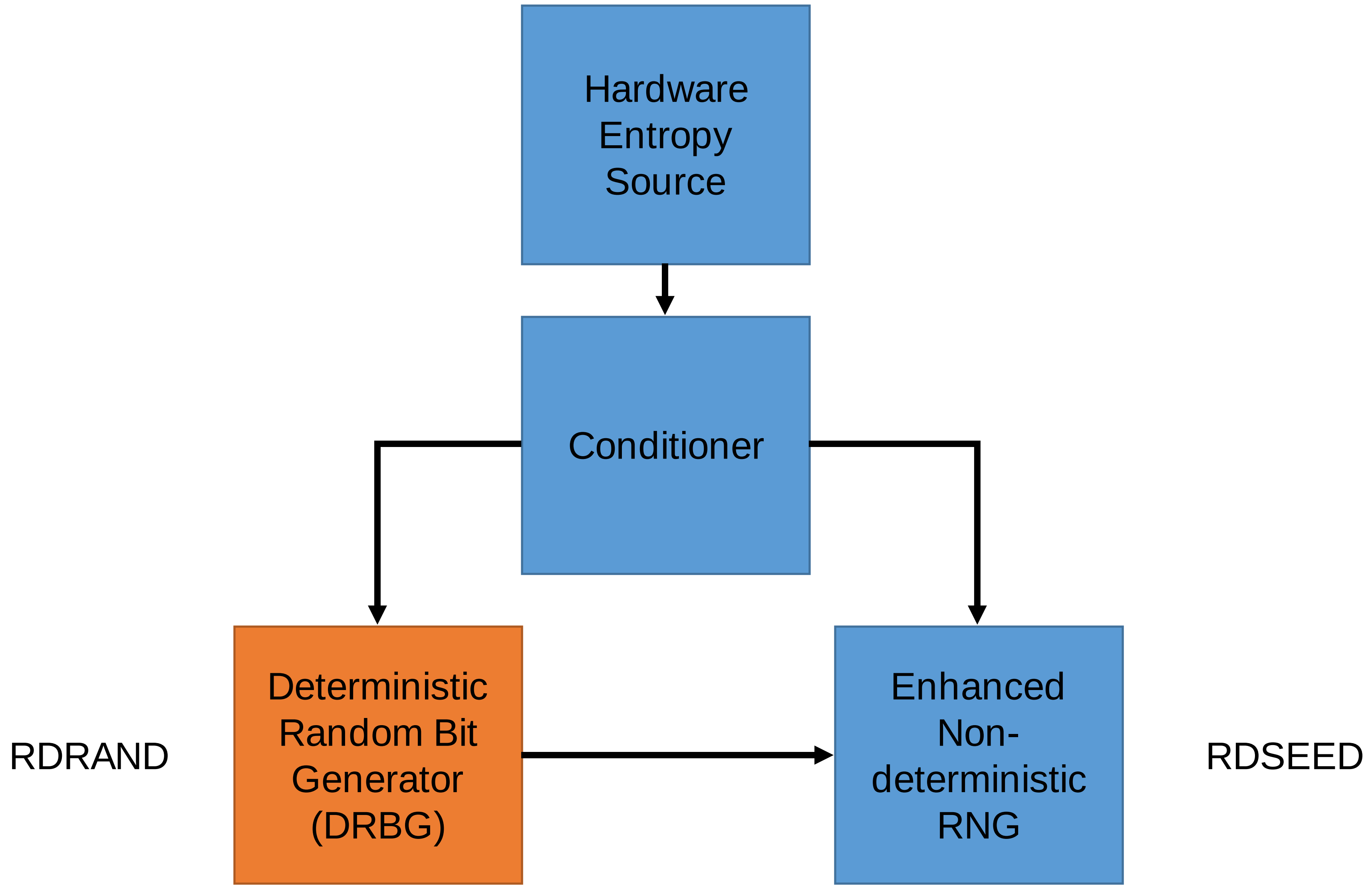
- For a certain random sequence, in the past we drew the number 189 immediately after the number 25. Now in the same sequence, we just drew the number 25 again.

- A) MT: the next number will be 189. LCG: the next number will be 189
- **B) MT: the next number will be random. LCG: the next number will be 189**
- C) MT: the next number will be 189. LCG: the next number will be random
- D) MT: the next number will be random. LCG: the next number will be random

Quiz Q4 – Poll 6

- Is it possible to generate true random numbers from a digital computer (e.g. x86 architecture)?
  - Yes
  - No

Intel DRNG



https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide

Entropy Pool

- Entropy pool: where you store the true randomness
  - E.g.: hardware, /dev/random

- Depletion:
  - Every time you use the pool to generate a RN, you gain information about it
  - The next query might not be random!
  - The pool usually “knows” it is depleted -> needs more randomness (time) to replenish

RDRAND - Example

```
#include <iostream>

int main() {
    unsigned long long int random;

    for (int i = 0; i < 10; ++i) {
        if (__builtin_ia32_rdrand64_step(&random)) break;
    }

    std::cout << random << std::endl;
}
```

- GCC specific: compilation with g++ -mrdrnd
- Available in Intel Ivy Bridge and AMD Zen onwards
- Downside: lack of seeding means losing reproducibility

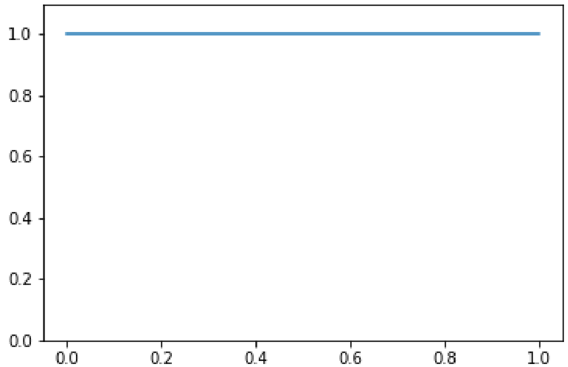
Generating Other Distributions

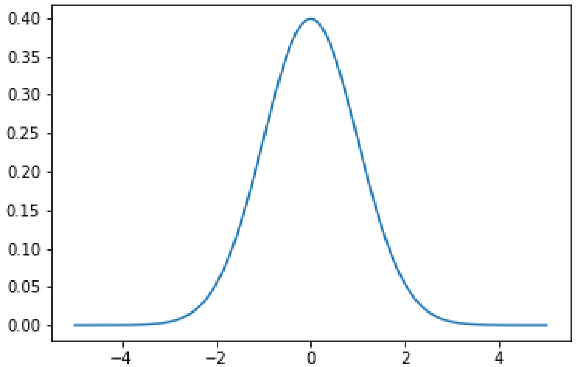
- The basic algorithms generate a uniform $u \in [0,1)$

- What if we need other distributions?
  - Common example: normal (Gaussian) distribution $N(\mu, \sigma)$
  - $N(\mu,\sigma) = \sigma * N(0,1) + \mu$

- We would like a map $u \rightarrow z \in N(0,1)$

- Very common trick: map $u_1, u_2 \in [0,1) \rightarrow z_1, z_2 \in N(0,1)$





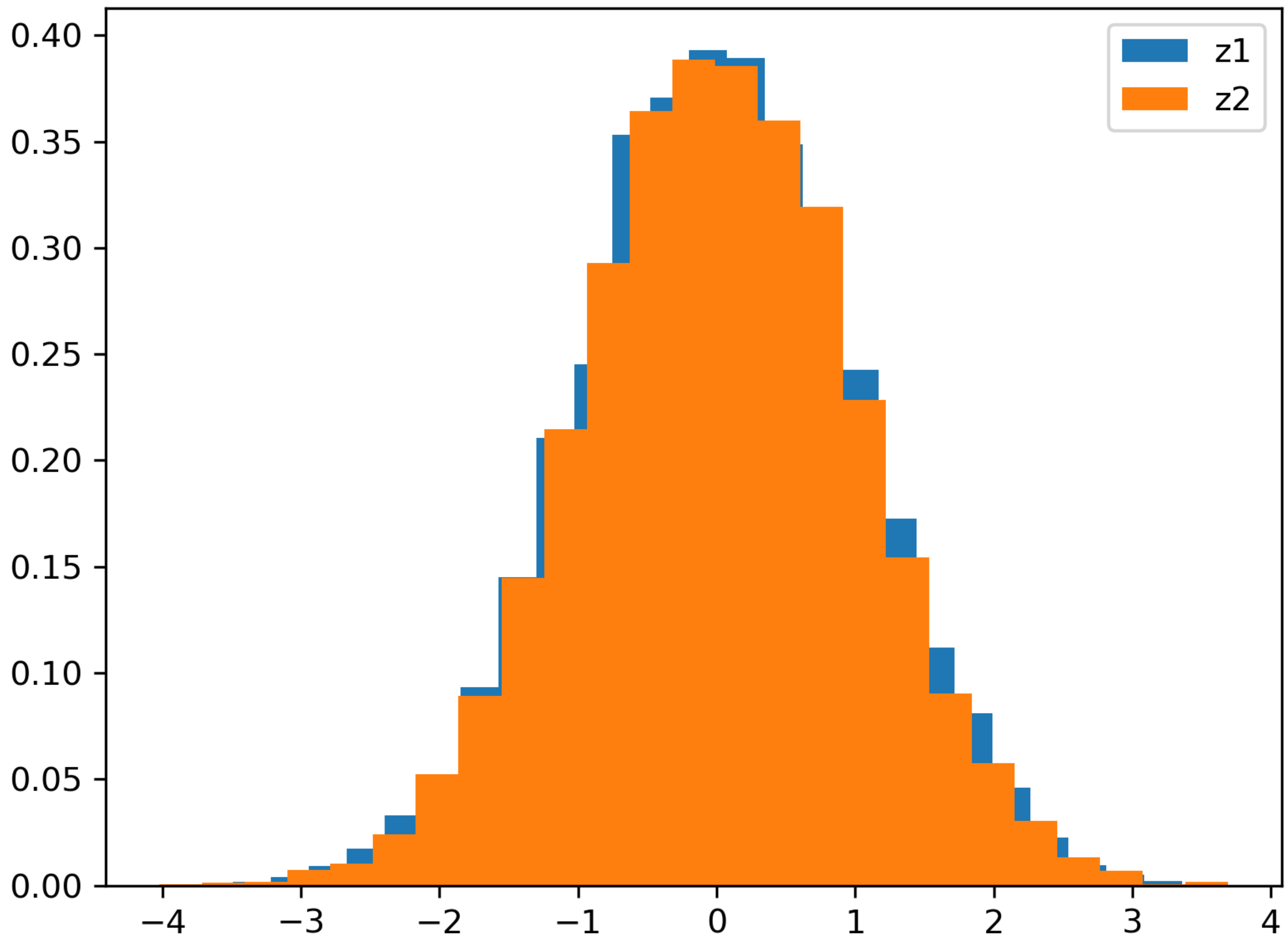
Normal Distribution - Box-Muller Algorithm

- Draw 2 uniform samples u_1 and u_2

$$z_1 = \sqrt{-2\ln(u_1)}\cos(2\pi u_2)$$
$$z_2 = \sqrt{-2\ln(u_1)}\sin(2\pi u_2)$$

```python
import matplotlib.pyplot as plt
import numpy as np

u1 = np.random.random(10000)
u2 = np.random.random(10000)

z1 = np.sqrt(-2*np.log(u1))*np.cos(2*np.pi*u2)
z2 = np.sqrt(-2*np.log(u1))*np.sin(2*np.pi*u2)

fig, ax = plt.subplots()
ax.hist(z1, 25, density=1, label="z1")
ax.hist(z2, 25, density=1, label="z2")
ax.legend()

plt.savefig('boxmuller.png', dpi=300)
```

Normal Distribution - Box-Muller Algorithm



Random Numbers in C++11

- C++11 introduced the `<random>` header
  - https://www.youtube.com/watch?v=6DPkyvkMkk8

- Random number engines and engine adaptors
  - E.g. `std::mersenne_twister_engine`
  - These folks mess around with the constants and the internal state of the RNGs
  - Don't mess around with the given constants

- Predefined RNGs
  - E.g. `std::mt19937_64` is an instantiation of the engine with the correct constants for a 64-bit random number
  - A collection of popular RNGs including both 32 and 64-bit Mersenne Twister

Random Numbers in C++11

- Random Number Distributions
  - E.g. `std::uniform_int_distribution`
  - It can be imagined as a view of the RNG
  - Given a certain generator, extract a new random number according to the distribution
  - Many common choices, including Gaussian, Poisson…

- Non-deterministic Random Numbers
  - `std::random_device` generates a non-deterministic uniform 32-bit int
  - It may use hardware-specific RNG.
  - Does your implementation actually use RDRAND? ¯_(ツ)_/¯
  - Common application: generate a good seed for your RNG
  - It can be used for general purpose, but might become slower

Random Numbers in C++11 - Example

```
#include <iostream>
#include <random>

int main()
{
    std::random_device rd;
    unsigned int rd_seed = rd();
    std::mt19937_64 gen(rd_seed);
    std::normal_distribution<double> d(0,1);


    for(int i=0; i<5; ++i) {
        std::cout << d(gen) << std::endl;
    }
}
```

Random Numbers in a Parallel System – Poll 7

- What happens when different RNGs are initialized in a parallel system with the same seed?
- A) Each stream generates different, independent RNs
- B) The result of each stream is random, depending on initialization
- C) All threads read from the same RNG, leading to data races
- D) All streams generate the same sequence of RNs

Random Numbers in a Parallel System – Poll 7

- What happens when different RNGs are initialized in a parallel system with the same seed?

- A) Each stream generates different, independent RNs

- B) The result of each stream is random, depending on initialization

- C) All threads read from the same RNG, leading to data races

- **D) All streams generate the same sequence of RNs**

Random Numbers in a Parallel System

- Random number server
  - The program only has a single RNG and all threads and processes draw numbers from it
  - Commonly used for GPUs: the CPU generates a list of random numbers which is then loaded to the GPU

- Per-stream generator
  - Assign a different RNG per thread or process
  - Pay attention to the seeds!
  - One approach: 1 master generator with 1 seed -> 1 seed per process/thread

Sidenote: CSRNG

- Working definition [Kneusel] of Cryptographically Safe RNG:
  - A CSPRNG is a random number generator that passes the next-bit test and one where an attacker's knowledge of the state of the generator at time t makes knowing the state at any previous time impossible.

- Irrelevant for simulations

- But a requirement for cryptography!

- MT is NOT cryptographically safe

Rules of Thumb

- If you do not want to think, use the Mersenne Twister
  - Consider the 64-bit version when generating doubles…
- Each thread or process needs to have its own RNG
  - With different seed!
- Avoid LCG if possible
  - Every time you use C-style `rand()`, a transistor cries
  - In general, their period is too short for large scale simulations
- Hardware RNGs might make sense in many applications!
  - Watch out for entropy pool depletion
- Correct seeding can be hard – but allows for reproducibility!

Monte Carlo

- What is/Where is Monte Carlo?

- Term dates to the Manhattan Project

- Very loosely defined: any type of algorithm which uses random sampling to achieve a numerical result
  - The problem itself might be deterministic!
  - Las Vegas methods (subclass): uses randomness but achieves a deterministic result

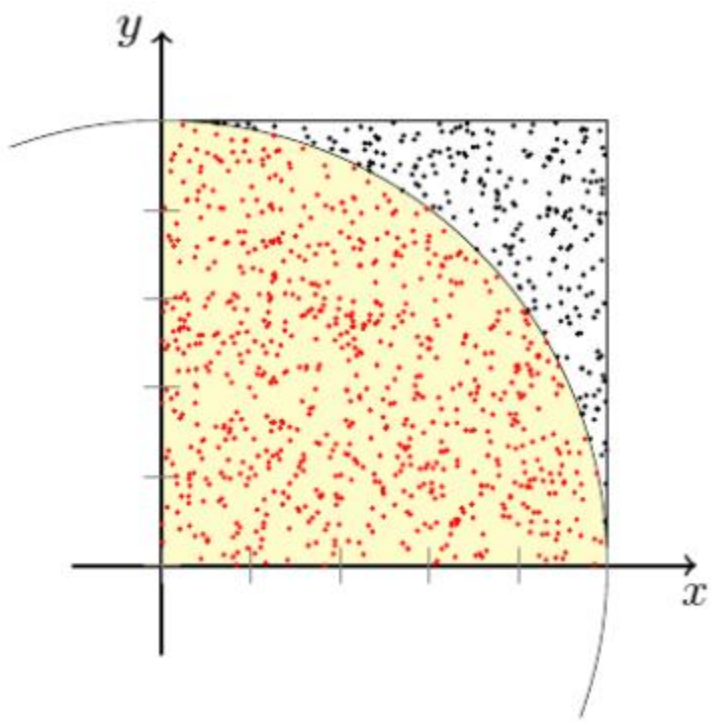
- “Monte Carlo” has in practice many definitions for each field

Quiz – Question 5

- How would you estimate the value of π using random numbers?

Monte Carlo Integration – one way

- Draw a circle inside the unit square
  - $g(x) = \sqrt{1 - x^2}$
- Generate pairs of random numbers
  - $x, y \in [0,1)$
- Count the number of hits inside
  - $y < g(x)$
- Calculate the ratio:
  - $\frac{\pi}{4} \approx \frac{\#_{inside}}{\#_{total}}$



Monte Carlo Integration – another way

$$I = \int_D g(\boldsymbol{x}) d\boldsymbol{x}$$

$$\hat{I}_m = \frac{1}{m} \{ g\left(\boldsymbol{x}^{(1)}\right) + \cdots + g\left(\boldsymbol{x}^{(m)}\right) \}$$

- Where $\boldsymbol{x}^{(n)}$ are drawn uniformly from D
- For the circle, $D: x \in [0,1)$ and $g(x) = \sqrt{1 - x^2}$

Take-home message – version 2

- Be careful with your choice of RNG
  - The Mersenne Twister is the standard for a reason
  - Hardware RNGs are becoming more useful
  - Older LCG are to be avoided whenever possible

- Seeding is a consequence of pseudo-RNG
  - Not always a bad thing – reproducibility!

- Watch out for pitfalls on parallel systems
  - Usually, you must make sure that each process/thread is generating a different random number sequence

Quiz

- Q1: Consider a 32-bit LCG. How many random numbers can you generate before the sequence repeats?
- Q2: What are the consequences of entropy pool depletion? How would you handle it?
- Q3: What are the requirements to initialize correctly one RNG per thread?
- Q4: Which STL containers are usually implemented using hashes? What is the advantage of doing so?
- Q5: How are elements accessed on a std::list?

Next stop

- Lectures swapped: OpenMP -> 14.12.2023
- Algorithm Analysis
- Data Structures