Friedrich-Schiller-Universität Jena
Physikalisch-Astronomische Fakultät

Design and Implementation of Vectorized Pseudorandom Number Generators and their Application to Simulation in Physics

MASTER'S THESIS

for obtaining the academic degree

Master of Science (M.Sc.) in Physics

submitted by Markus Pawellek

born on May 7th, 1995 in Meiningen
Student Number: 144645

Primary Reviewer: Bernd Brügmann

Primary Supervisor: Joachim Gießen

Jena, November 4, 2019

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Contents

List of Figures

List of Abbreviations

Abbreviation	Definition
iid	independent and identically distributed
RNG	Random Number Generator
TRNG	True Random Number Generator
PRNG	Pseudorandom Number Generator
LCG	Linear Congruential Generator
MCG	Multiplicative Congruential Generator
MT	Mersenne Twister
MT19937	Mersenne Twister with period $2^{19937} - 1$
PCG	Permuted Congruential Generator
CPU	Central Processing Unit
GPU	Graphics Processing Unit
SIMD	Single Instruction, Multiple Data
SSE	Streaming SIMD Extensions
AVX	Advanced Vector Extensions

Symbol Table

Symbol	Definition
$x \in A$	x ist ein Element der Menge A.
$A \subset B$	A ist eine Teilmenge von B.
$A \cap B$	$\{x \mid x \in A$ und $x \in B\}$ für Mengen A, B — Mengenschnitt
$A \cup B$	$\{x \mid x \in A$ oder $x \in B\}$ für Mengen A, B — Mengenvereinigung
$A \setminus B$	$\{x \in A \mid x \notin B\}$ für Mengen A, B — Differenzmenge
$A \times B$	$\{(x, y) \mid x \in A, y \in B\}$ für Mengen A und B — kartesisches Produkt
$\emptyset$	$\{\}$ — leere Menge
$\mathbb{N}$	Menge der natürlichen Zahlen
$\mathbb{N}_0$	$\mathbb{N} \cup \{0\}$
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^n$	Menge der n-dimensionalen Vektoren
$\mathbb{R}^{n \times n}$	Menge der $n \times n$-Matrizen
$f\colon X \to Y$	f ist eine Funktion mit Definitionsbereich X und Wertebereich Y
$\partial\Omega$	Rand einer Teilmenge $\Omega \subset \mathbb{R}^n$
σ	Oberflächenmaß
λ	Lebesgue-Maß
$\int_\Omega f \,\mathrm{d}\lambda$	Lebesgue-Integral von f über der Menge Ω
$\int_{\partial\Omega} f \,\mathrm{d}\sigma$	Oberflächen-Integral von f über der Menge $\partial\Omega$
∂_i	Partielle Ableitung nach der i. Koordinate
∂_t	Partielle Ableitung nach der Zeitkoordinate
∂_i^2	Zweite partielle Ableitung nach i
∇	$\begin{pmatrix}\partial_1 & \partial_2\end{pmatrix}^{\mathrm{T}}$ — Nabla-Operator
Δ	$\partial_1^2 + \partial_2^2$ — Laplace-Operator
$\mathrm{C}^k(\Omega)$	Menge der k-mal stetig differenzierbaren Funktion auf Ω
$\mathrm{L}^2(\Omega)$	Menge der quadrat-integrierbaren Funktionen auf Ω
$\mathrm{H}^1(\Omega)$	Sobolevraum
$f\vert_{\partial\Omega}$	Einschränkung der Funktion f auf $\partial\Omega$
$\langle x, y\rangle$	Euklidisches Skalarprodukt
$[a, b]$	$\{x \in \mathbb{R} \mid a \leq x \leq b\}$
(a, b)	$\{x \in \mathbb{R} \mid a < x < b\}$
$[a, b)$	$\{x \in \mathbb{R} \mid a \leq x < b\}$
$u(\cdot, t)$	Funktion $\tilde{u}$ mit $\tilde{u}(x) = u(x, t)$
A^{T}	Transponierte der Matrix A
id	Identitätsabbildung
$a := b$	a wird durch b definiert
$f \circ g$	Komposition der Funktionen f und g
$\begin{vmatrix} a & b \\ c & d \end{vmatrix}$	Determinante der angegeben Matrix
$\operatorname{span}\{\ldots\}$	Lineare Hülle der angegebenen Menge
$\vert A\vert$	Anzahl der Elemente in der Menge A

1 Introduction

For various mathematical and physical problems, there exists no feasible, deterministic algorithm to solve them. Especially, the simulation of physical systems with many coupled degrees of freedom, such as fluids, seem to be difficult to compute due to their high dimensionality. Instead, a class of randomized algorithms, called Monte Carlo methods, are used to approximate the actual outcome. Monte Carlo methods rely on repeated random sampling to obtain a numerical result. Hence, they are not bound to the curse of dimensionality and are able to evaluate complex equations quickly.

To obtain precise answers with a small relative error, Monte Carlo algorithms have to use a tremendous amount of random numbers. But the usage of truly random numbers generated by physical processes consists at least of two drawbacks. First, the output of the algorithm will be non-deterministic and, as a result, untestable. Second, the generation of truly random numbers is typically based on a slow process and consequently reduces the performance of the entire program. For that reason, Monte Carlo algorithms usually use so-called pseudorandom number generators. PRNGs generate a sequence of numbers based on a deterministic procedure and a truly random initial value as seed. The sequence of numbers is not truly random but fulfills several properties of truly random sequences.

The structure of Monte Carlo methods causes a program to spend most of its time with the construction of random numbers. Even the application of PRNGs does not change that. Today's computer processors provide functionality for the parallel execution of code in different ways, mainly SIMD and MIMD. Hence, to efficiently use the computing power of a CPU for Monte Carlo algorithms PRNGs have to be vectorized and parallelized to exploit such features. Whereas parallelization takes place at a high level, vectorization has to be done by the compiler or manually by the programmer at a much lower level. The implementation of PRNGs constraints automatic vectorization due to internal flow and data dependencies. To lift this restriction, a manual vectorization concerning data dependence and latencies appears to be the right way.

The C++ programming language is a perfect candidate for the development of vectorized PRNGs. It is one of the most used languages in the world and can be applied to small research projects as well as large enterprise programs. The language allows for the high-level abstraction of algorithms and structures. On the other hand, it is capable of accessing low-level routines to exploit special hardware features, like SSE, AVX, and threads. A typical C++ compiler is able to optimize the code with respect to such features automatically. But we as programmers are not bound to this and can manually optimize the code further. Every three years, a new standard is published, such as the new C++20 language specification. The language is evolving by its communities improvements and therefore it keeps to be a modern language. On top of this, other languages, such as Python, usually provide an interface to communicate with the C programming language. Through the design of an efficient implementation in C++, we can easily add support for other languages as well by providing a standard C interface.

Lots of PRNGs have been implemented by different libraries with different APIs. For

example, STL, Boost, Intel MKL, RNGAVXLIB, Lemire, tinyrng,... STL, Boost and ... provide a large set of robust PRNGs which are not vectorized but well documented. Their API makes them likely to be used but shows many flaws. It does not allow to explicitly use the vectorization capabilities of a PRNG, gives you a bad default seeding and makes use of standard distributions difficult and not adjustable. Lemire and RNGAVXLIB provide open-source, vectorized implementations with bad documentation and difficult-to-use code. Intel MKL as well provides vectorized PRNGs but is not available open-source and uses difficult interfaces. There is not any easily-accessible, portable, open-source library which gives a coherent, easy-to-use and consistent interface for vectorized PRNGs.

In this thesis, we develop a new library, called pxart, in the C++ programming language. pxart vectorizes a handful of already known PRNGs which partly do not exist as vectorized versions and provides a new API for their usage to accommodate the disadvantages of the standard random library of the STL. The library itself is header-only, open-source, and can be found on GitHub. It is easily installable on every operating system. Additionally, we compare the performance of our vectorized PRNGs to other already accessible implementations in Boost, Intel MKL, Lemire, RNGAVXLIB and others. The performance is measured by speed, code size, memory size, complexity, and random properties. Meanwhile, we apply the implementations to an example Monte Carlo simulation. For this, a small test framework is implemented which allows us to easily test and evaluate PRNGs with respect to stated measures.

2 Background

To systematically approach the implementation of PRNGs, basic knowledge in the topics of stochastics and finite fields is administrable. Together, these topics will give a deeper understanding of randomness in deterministic computer systems, a formal description of pseudorandom sequences and generators, and the mathematical foundation of Monte Carlo algorithms. Based on them, we are capable of scientifically analyzing PRNGs concerning their randomness properties. Vectorization techniques can be conceptualized by the architecture of modern SIMD-capable multiprocessors and their instruction sets. Especially the knowledge of typical instructions will make the design of a new API and its application to Monte Carlo simulations clear. The following sections will give an overview of the named topics.

2.1 Mathematical Preliminaries

2.1.1 Probability Theory

The observation of random experiments resulted in the construction of probability theory. But probability theory itself does not use a further formalized concept of randomness (Schmidt 2009). In fact, it allows us to observe randomness without defining it (Volchan 2002). Hence, we will postpone an examination of truly random sequences to the next section.

According to Schmidt (2009), Kolmogorov embedded probability theory in the theory of measure and integration. Albeit it heavily relies on measure-theoretical structures, probability theory is one of the most important applications of measure and integration theory. Therefore we will assume basic knowledge in this topic and refer to Schmidt (2009) and Elstrodt (2018) for a more detailed introduction to measure spaces, measurable functions, and integrals. Propositions and theorems will be given without proof.

The underlying structure of probability theory, which connects it to measure theory, is the probability space. It is a measure space with a finite and normalized measure. This gives access to all the usual results of measure theory and furthermore unifies discrete and continuous distributions. (Schmidt 2009, p. 193 ff.)

DEFINITION 2.1: (Probability Space)

A probability space is a measure space $(\Omega, \mathcal{F}, P)$ such that $P(\Omega) = 1$. In this case, we call P the probability measure, $\mathcal{F}$ the set of all events, and Ω the set of all possible outcomes of a random experiment.

Due to the complex definition of a measure space, it is convenient to not have to explicitly specify the probability space when analyzing random experiments. Instead, we use random variables which are essentially measurable functions on a probability space (Schmidt 2009, p. 194). For complicated cases, these will serve as observables for specific properties and will make the analysis much more intuitive.

DEFINITION 2.2: (Random Variable)

Let $(\Omega, \mathcal{F}, P)$ be a probability space and $(\Sigma, \mathcal{A})$ a measurable space. A measurable function $X \colon \Omega \to \Sigma$ is called a random variable.
In this case, we denote with $P_X := P \circ X^{-1}$ the distribution and with $(\Sigma, \mathcal{A}, P_X)$ the probability space of X. Two random variables are identically distributed if they have the same distribution. Additionally, we say that X is a real-valued random variable if $\Sigma = \mathbb{R}$ and $\mathcal{A} = \mathcal{B}(\mathbb{R})$.

From now on, if a random variable is defined then, if not stated otherwise, it is assumed there exists a proper probability space $(\Omega, \mathcal{F}, P)$ and measurable space $(\Sigma, \mathcal{A})$.

Another important concept of stochastics is known as independence. In Schmidt (2009) it is defined for a family of events, a family of sets of events, and a family of random variables. If we think of random variables as observables then their independence means that their outcomes do not influence each other. For our purposes, the general definition of all three forms of independence is distracting. In a computer, it makes no sense to talk about infinite sequences. Therefore the following definition of independence takes only a finite sequence of random variables into account. Furthermore, to make it more understandable, this definition uses a theorem from Schmidt (2009, p. 238) which characterizes the independence of random variables.

DEFINITION 2.3: (Independence)

Let $n \in \mathbb{N}$ and X_i be a random variable for all $i \in \mathbb{N}$ with $i \leq n$. We denote the respective random vector with $X := (X_i)_{i=1}^n$. Then these random variables are independent if the following equation holds.

$$P_X = \bigotimes_{i=1}^{n} P_{X_i}$$

Typical observations of random sequences include the estimation of the expectation value and the variance. Both of these values are needed for analyzing PRNGs and the development of Monte Carlo simulations (Landau and Binder 2014, p. 30 ff.). Due to their deep connection to the integral, both of these moments are defined for real-valued random variables. We give the usual definitions based on Schmidt (2009, p. 274 ff.) in a simplified form.

DEFINITION 2.4: (Expectation Value and Variance)

Let X be a real-valued random variable such that $\int_\Omega |X| \, \mathrm{d}P < \infty$. Then the expectation value $\mathbb{E}X$ and variance $\operatorname{var} X$ of X is defined in the following way.

$$\mathbb{E}X := \int_\Omega X(\omega) \, \mathrm{d}P(\omega) \ , \qquad \operatorname{var} X := \mathbb{E}\left(X - \mathbb{E}X\right)^2$$

To not rely on the underlying probability space directly, we want to be able to compute the expectation value through the respective distribution of the random variable. The theory of measure and integration gives the following proposition, also known as rule of substitution (Schmidt 2009, p. 276).

PROPOSITION 2.1: (Substitution)

Let X be real-valued random variable and $f\colon \mathbb{R} \to \mathbb{R}$ a measurable function such that $\int_\Omega |f| \,\mathrm{d}P_X < \infty$. Then the following equation holds.

$$\mathbb{E}(f \circ X) = \int_{\mathbb{R}} f(x) \,\mathrm{d}P_X(x)$$

In particular, if $\mathbb{E}\,|X| < \infty$ then the above equation can be reformulated as follows.

$$\mathbb{E}X = \int_{\mathbb{R}} x \,\mathrm{d}P_X(x)$$

The distribution of real-valued random variables is univariate and as a result can be described by so-called cumulative distribution functions (CDFs). The CDF intuitively characterizes the distribution and simplifies the analysis. Further, it can be proven that every CDF belongs to a univariate distribution. According to Schmidt (2009, p. 246), this is the theorem of correspondence. Sometimes it is even possible to define a probability density; a function that is the Lebesgue density of the respective distribution (Schmidt 2009, p. 255).

DEFINITION 2.5: (Probability Density and Cumulative Distribution Function)

Let X be a real-valued random variable. Then the respective cumulative distribution function is defined as follows.

$$F_X\colon \mathbb{R} \to [0,1]\ , \qquad F_X(x) \coloneqq P_X((-\infty, x])$$

We call the function $p\colon \mathbb{R} \to [0,\infty)$ a probability density of X if for all $A \in \mathcal{B}(\mathbb{R})$

$$P_X(A) = \int_A p(x) \,\mathrm{d}\lambda(x)\ .$$

As well as CDFs, probability densities can greatly simplify computations which are based on absolute continuous random variables. The following proposition, obtained from Schmidt (2009), shows the simplified computation of an expectation value through a Lebesgue integral.

PROPOSITION 2.2: (Chaining)

Let X be a real-valued random variable with p as its probability density. If $f\colon \mathbb{R} \to \mathbb{R}$ is a measurable function such that $\mathbb{E}\,|f \circ X| < \infty$ then

$$\mathbb{E}\,(f \circ X) = \int_{\mathbb{R}} f(x)p(x) \,\mathrm{d}\lambda(x)\ .$$

A last important theorem to name is the strong law of large numbers (SLLN). According to Graham and Talay (2013, p. 13), the principles of Monte Carlo methods are based on this theorem. Please note, there exist many more variations of this theorem. We will again use a simplified version from Graham and Talay (2013).

THEOREM 2.3: (Strong Law of Large Numbers)

Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of iid real-valued random variables with finite expectation value μ. Then the following equation holds P-almost everywhere.

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} X_i = \mu$$

2.2 Pseudorandom Number Generators

2.2.1 Random Sequences

In the above subsection 2.1 the theory of probability was introduced to make an examination of randomness possible. Randomness is a difficult concept and drives many philosophical discussions. According to Volchan (2002) and Kneusel (2018, pp. 10–11), humans have a bad intuition concerning the outcome of random experiments. But for our purposes, it would suffice to find a formal mathematical definition applicable to RNGs. However, such a formal concept, which is also widely accepted and unique, has not been found yet (Volchan 2002).

The first problem about randomness is the word itself. It is unclear and vague because there is no intentional application. To be more specific, we will observe randomness in form of random sequences of real numbers. But as stated in Volchan (2002) the question if a sequence is random decides at infinity. As long as we are only observing finite sequences, we cannot decide if such a sequence is the outcome of a truly random experiment or the result of a non-random algorithm. Following his explanation, Volchan makes clear that typical characterizations of a random sequence are closely associated with noncomputability. So even if we would be able to algorithmically produce an infinite amount of numbers, the resulting sequence could not be seen as truly random. A modified version of this idea which is easier to understand is given in Kneusel (2018), where a sequence of values $(x_n)_{n\in\mathbb{N}}$ is truly random if there exists no algorithm such that for all $n \in \mathbb{N}$ the value x_{n+1} can be computed as a function of all x_i with $i \in \mathbb{N}$ and $i \leq n$. Put more simply, knowing finitely many elements of a truly random sequence does not enable us to predict the next values within a computer. Furthermore, the question if a sequence is random cannot be decided by an algorithm. Hence, the existing formal concepts for truly random sequences are not applicable to computer systems. Instead, Volchan proposed a more pragmatic principle: “if it acts randomly, it is random” (Volchan 2002) — the use of pseudorandom sequences.

A computer is only capable of using finite sequences of values and for the development of RNGs, it is enough to measure and compare different properties of truly random sequences to

a sequence of real numbers. For this, we rely on probability theory and first define an abstract random sequence drawn from a random experiment. The definition will use realizations of random variables to model the samples of a random experiment. We make sure that these variables are identically and independent distributed (iid). This makes analyzing other sequences simpler and imposes no boundary because every important distribution can be generated out of iid random variables (Kneusel 2018, pp. 81–111).

DEFINITION 2.6: (Random Sequence)

Let I be a countable index set and $(X_n)_{n \in I}$ be a sequence of iid real-valued random variables. Then a realization of $(X_n)_{n \in I}$ is called a random sequence.

Generating a truly random sequence in a deterministic computer system is impossible. An RNG which is able to generate such a sequence is called a true random number generator (TRNG) and is typically implemented as a device drawing random samples from an essentially non-deterministic physical process, like temperature fluctuations (Intel 2018).

2.2.2 Pseudorandom Sequences

The given abstract definition of a random sequence in terms of probability theory helps to assess the randomness properties of a given sequence produced by a computer. Typically, a computer-generated sequence which fulfills various conditions about randomness will be called a pseudorandom sequence. The respective structure and algorithm which produced the sequence is then called a PRNG.

For computer programming and simulations, the usage of a TRNG would introduce severe disadvantages in contrast to a PRNG. Concerning program verification, debugging, and the comparison of similar systems, the reproducibility of results is essential (L'Ecuyer 2015). A truly random sequence produced by physical devices, such as thermal noise diodes or photon trajectory detectors, is not reproducible and can therefore not be conveniently used for mathematical and physical simulations (L'Ecuyer 2015). According to L'Ecuyer (2015), a given simulation should produce the same results on different architectures for every run. This property becomes even more important if parallel generation of random numbers with multiple streams is taken into account. Additionally, considering the performance of random number generation PRNGs tend to be much faster than TRNGs (Intel 2018). Thus, especially for Monte Carlo methods, PRNGs are a key resource for computer-generated random numbers (Bauke and Mertens 2007).

For a detailed discussion about its mathematical properties, design, and implementation, the concept of a PRNG has to be formalized. In this thesis, we use the following slightly modified variation of L'Ecuyer's definition (Barash, Guskova, and Shchur 2017; Bauke and Mertens 2007; L'Ecuyer 1994, 2015). It assumes a finite set of states and a transition function which advances the current state of the PRNG by a recurrence relation. For the output, a finite set of output symbols and a generator function which maps states to output symbols is chosen.

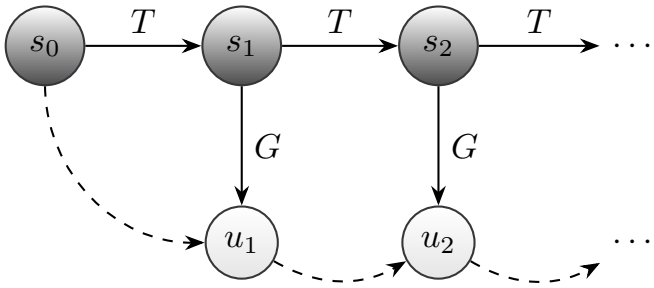


Figure 1: The figure shows a scheme about the generation of a pseudorandom sequence for a given PRNG $\mathcal{G} := (S, T, U, G)$ and seed value $s_0 \in S$. The internal state is advanced by the transition function T through a recurrence relation. To get an output value for the pseudorandom sequence the generator function G is used.

As of Bauke and Mertens (2007), almost all PRNGs produce a sequence of numbers by a recurrence. Hence, the given formalization is widely accepted and builds the basis for further discussions about pseudorandom numbers (Barash, Guskova, and Shchur 2017; Bauke and Mertens 2007; L'Ecuyer 1994, 2015).

DEFINITION 2.7: (Pseudorandom Number Generator)

Let $\mathcal{G} := (S, T, U, G)$ be a tuple consisting of a non-empty, finite set of states S, a transition function $T\colon S \to S$, a non-empty, finite set of output symbols U and an output function $G\colon S \to U$. In this case $\mathcal{G}$ is called a PRNG.

Given a PRNG and a seed value as an initial state, producing a sequence of pseudorandom numbers can be done by periodically applying the transition function on the current state and then extracting the output through the generator function (Barash, Guskova, and Shchur 2017; L'Ecuyer 1994, 2015). Here, we will use this method as the generalization of a pseudorandom sequence. Figure 1 shows this process schematically.

DEFINITION 2.8: (Pseudorandom Sequence of PRNG)

Let $\mathcal{G} := (S, T, U, G)$ be a PRNG and $s_0 \in S$ be the initial state, also called the seed value. The respective sequence of states $(s_n)_{n\in\mathbb{N}}$ in S is given by the following equation for all $n \in \mathbb{N}$.

$$s_{n+1} := T(s_n)$$

The sequence $(u_n)_{n\in\mathbb{N}}$ in U given by the following expression for all $n \in \mathbb{N}$ is then called the respective pseudorandom sequence of $\mathcal{G}$ with seed s_0.

$$u_n := G(s_n)$$

In the definition we have used a recursive formulation. For theoretical discussions and the

initialization of multiple streams of pseudorandom numbers an explicit variation seems to be more adequate. The following lemma will be given without a proof, but it can be shown by mathematical induction.

> **LEMMA 2.4:** (Explicit Formulation of Pseudorandom Sequence)
>
> *Let $\mathcal{G} := (S, T, U, G)$ be a PRNG and $s_0 \in S$ its initial state. Then the respective pseudorandom sequence $(u_n)_{n\in\mathbb{N}}$ is given by the following formula for all $n \in \mathbb{N}$.*
>
> $$u_n = G \circ T^n(s_0)$$

2.2.3 Explanation of the Concept

Using a TRNG in a computer system is like consulting an oracle (Müller-Gronbach, Novak, and Ritter 2012). We are calling a function with no arguments which returns a different value for every call. Let $(u_n)_{n\in\mathbb{N}}$ be the respective pseudorandom sequence of a PRNG $\mathcal{G}$ with a given seed. Then in a computer $\mathcal{G}$ can be interpreted as a function with no parameters which produces the pseudorandom sequence $(u_n)_{n\in\mathbb{N}}$ in the following way. Hereby, we understand $\leftarrow$ as the assignment operator that assigns a value given on the right-hand side to the variable given on the left-hand side.

$$u_1 \leftarrow \mathcal{G}()\,, \qquad u_2 \leftarrow \mathcal{G}()\,, \qquad u_3 \leftarrow \mathcal{G}()\,, \qquad \ldots$$

A PRNG has to artificially model this behavior by an internal state. Every function call must change this state according to the transition function. Consequently, if a PRNG should be used as an oracle in that sense, the set of states and the transition function in its definition are obligatory.

It will be shown that the number of different states a PRNG can reach greatly affects the randomness of a respective pseudorandom sequence. A larger set of states is not a guarantee that the output of a PRNG will look more like a truly random sequence, but at least gives the opportunity to better mask its deterministic nature (O'Neill 2014). Therefore the number of states in general is much bigger than the number of different outputs. Through the usage of output symbols together with a generator function a PRNG can take advantage of a large set of states while returning only a few different values. This idea has two important implications. A generator function which shrinks the set of states to a smaller space of output symbols makes the PRNG less predictable and more secure (O'Neill 2014). The generator function would not be bijective and as a result we as consumers would not be able to draw conclusions about the current state of the PRNG based on its given output. Both properties are highly appreciated because they mimic the behavior of TRNGs. Hence, the set of output symbols and the generator function in the definition of PRNGs is as important as the set of states and the transition function.

In the majority of cases, the transition function T of a PRNG $\mathcal{G}$ should be injective (L'Ecuyer 1994, 2015; O'Neill 2014; Widynski 2019). Because we have a finite set of states

this is equivalent to the proposition that T is a permutation and therefore bijective (Waldmann 2017, pp. 201–202). The property makes sure that every state is reached at a certain point in a sequence without introducing bias in the resulting distribution (O'Neill 2014). The generator function G cannot be a permutation but should not distort the distribution either. Hence, a uniform function which maps to every output value the same number of input values is a perfect candidate (O'Neill 2014).

2.2.4 Randomization

The goal of PRNGs is to imitate the properties of TRNGs as much as possible (L'Ecuyer 1994) and at the same time retaining executability by a computer system and reproducibility for a given seed (L'Ecuyer 2015). These restrictions make a pseudorandom sequence completely predictable and characterizable by its seed. So up until now, we have not introduced any kind of randomness to the definition of a PRNG. But to extend the process of generating a pseudorandom sequence with true randomness, the seed will be chosen to be a truly random number produced by a TRNG. L'Ecuyer (1994) states that receiving such a seed is much less work and more reasonable than acquiring a long sequence of truly random values. A generator with a truly random seed can be seen as an extensor of randomness. Even today, Intel uses hardware-implemented PRNGs repeatedly seeded by a high-quality entropy source in their CPUs to provide a high-performance hardware module for producing random numbers with good statistical quality and protection against attacks (Intel 2018).

DEFINITION 2.9: (Randomized Pseudorandom Sequence)

Let $\mathcal{G} := (S, T, U, G)$ be a PRNG and X be an S-valued random variable with distribution P_X. Then the randomized pseudorandom sequence $(X_n)_{n\in\mathbb{N}}$ of $\mathcal{G}$ with respect to P_X is defined by the following expression for all $n \in \mathbb{N}$.

$$X_n := G \circ T^n \circ X$$

As with abstract random sequences, a truly random seed value is again modeled by a realization of the random variable X. As a result, the randomized pseudorandom sequence becomes a sequence of random variables which all depend on X. For the definition the explicit formulation in Lemma 2.4 was used. Typically, the distribution of seed values P_X is chosen so that it is uniformly distributed in a certain subset of S (Bauke and Mertens 2007; L'Ecuyer 1994, 2015; Matsumoto and Nishimura 1998; O'Neill 2014). This makes sure that no bias will be introduced by the randomization.

2.2.5 Limitations and Mathematical Properties

As was already discussed, PRNGs have certain advantages in comparison with TRNGs. But they are also yielding essential and intrinsic limitations. From the previous paragraph, it

becomes clear that all the samples of a randomized pseudorandom sequence are not stochastically independent. In general, this means the output of a PRNG can consist of certain regular patterns or artifacts (L'Ecuyer 1994; O'Neill 2014). In L'Ecuyer (1994) these artifacts are also called the lattice structure. For applications that are using a large amount of random numbers, such patterns will introduce bias in the evaluated outputs. Hence, we will discuss a few mathematical properties a PRNG should fulfill to reduce the lattice structure as much as possible.

Periodicity Since the set of states in a PRNG is finite, every respective pseudorandom sequence has to be periodic or ultimately periodic (Bauke and Mertens 2007; L'Ecuyer 1994). First, a rigorous definition of this concept should be given.

DEFINITION 2.10: (Periodic and Ultimately Periodic Sequences)

Let U be a non-empty set and $(u_n)_{n\in\mathbb{N}}$ be a sequence in U. Assume there exist $\rho, \tau \in \mathbb{N}$ such that for all $n \in \mathbb{N}_0$ the following holds.

$$u_{\tau+n+\rho} = u_{\tau+n}$$

Then (u_n) is called ultimately periodic. The smallest possible values for ρ and τ, such that the equation holds, are called period and transient respectively. In particular, if τ equals to 1 we call (u_n) periodic with period ρ.

This means an ultimately periodic sequence will be periodic after it reached its transient. Every periodic sequence is therefore ultimately periodic but not vice versa and as another consequence, the given concept is more general than the typical one of a periodic sequence. Please note that the values for ρ and τ are not unique. Let ρ^* be the period and τ^* be the transient. Then the equation given in the above definition holds for all values ρ and τ with respect to $m \in \mathbb{N}$ and $n \in \mathbb{N}_0$ in the following sense.

$$\rho = m\rho^* \,, \qquad \tau = \tau^* + n$$

Choosing the minimal values allows us to talk about a unique transient and a unique period. In the following lemma we show the application of the definition to pseudorandom sequences.

LEMMA 2.5: (Pseudorandom Sequences are Ultimately Periodic)

Let $\mathcal{G} := (S, T, U, G)$ be a PRNG and $s_0 \in S$ its initial state. Then the respective pseudorandom sequence $(u_n)_{n\in\mathbb{N}}$ is ultimately periodic. In this case, for the period ρ and the transient τ the following holds.

$$1 \leq \rho + \tau - 1 \leq \#S$$

In particular, if T is bijective (u_n) will be periodic.

PROOF:
Let $(s_n)_{n\in\mathbb{N}}$ be the respective sequence of states and $N := \#S$ the number of different states. T maps all elements of S to at most N other elements of S. Therefore at least the element s_N has to be mapped to an element s_k for $k \in \mathbb{N}$ with $k \leq N$ which was already reached. Hence, we conclude the following.

$$\exists n, k \in \mathbb{N}, k \leq n \leq N : \quad T(s_n) = s_k$$

We choose n and k appropriately and define the following values.

$$\rho := n - k + 1 \,, \qquad \tau := k$$

Now let $i \in \mathbb{N}_0$ be arbitrary and apply the definition. We get the following chain of equations which show that (u_n) is ultimately periodic.

$$\begin{aligned} u_{\tau+i+\rho} &= u_{n+1+i} = G \circ T^{n+1+i}(s_0) = G \circ T^i \circ T^{n+1}(s_0) \\ &= G \circ T^i(s_k) = G \circ T^i \circ T^k(s_0) = G \circ T^{i+k}(s_0) = u_{k+i} = u_{\tau+i} \end{aligned}$$

The inequality can be shown by directly inserting the values into the definition.

$$1 \leq \rho + \tau - 1 = n \leq N = \#S$$

This proofs the given lemma. □

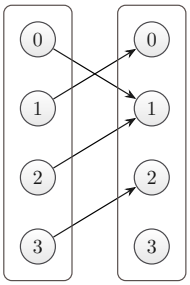
Thus, every pseudorandom sequence will repeat itself after it reached a certain point. The period and the transient are greatly affected by the number of states and the transition function of the PRNG. To get a better insight, we will examine the following idealized examples with different transition functions. Let $\mathcal{G} := (S, T, U, G)$ be a PRNG defined as follows.

$$S := U := \mathbb{Z}_4 \,, \qquad G := \mathrm{id}$$

For a seed $s_0 \in S$ the respective pseudorandom sequence $(u_n)_{n\in\mathbb{N}}$ with period ρ and transient τ will be shown in the following way. Hereby, all elements of the sequence up to the end of the first period are written consecutively and the periodic part is marked by an overline.

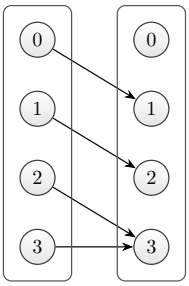
$$(u_n) = u_1 \dots u_{\tau-1}\overline{u_\tau \dots u_{\tau+\rho-1}}$$

To the left of the examples, a scheme of their respective transition function is displayed to make the originating sequences together with their periods and transients more understandable. The boxes are used in place of the set of states S whereas arrows characterize the transition function T.



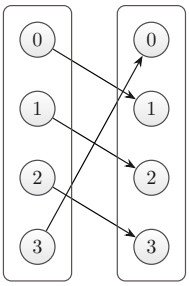
$$T(x) := \begin{cases} 1 & : x \in \{0, 2\} \\ 0 & : x = 1 \\ 2 & : x = 3 \end{cases} \,, \qquad (u_n) = \begin{cases} \overline{10} & : s_0 \in \{0, 2\} \\ \overline{01} & : s_0 = 1 \\ 2\overline{10} & : s_0 = 3 \end{cases}$$

The first example shows a transition function which is not bijective but does not map any element of S to itself. Hence, in all cases we get a period of 2. The transient varies between 1 and 2 and depends on the seed value.



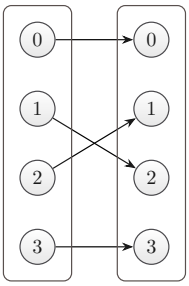
$$T(x) := \begin{cases} x+1 & : x < 3 \\ 3 & : x = 3 \end{cases}, \qquad (u_n) = \begin{cases} 12\overline{3} & : s_0 = 0 \\ 2\overline{3} & : s_0 = 1 \\ \overline{3} & : s_0 \geq 2 \end{cases}$$

In the second example, again a non-bijective transition function is used. This time the value 3 is mapped to itself and as a consequence the period for all possible sequences is 1. As before, the transient varies with respect to the seed value.



$$T(x) := x + 1 \mod 4\,, \qquad (u_n) = \begin{cases} \overline{1230} & : s_0 = 0 \\ \overline{2301} & : s_0 = 1 \\ \overline{3012} & : s_0 = 2 \\ \overline{0123} & : s_0 = 3 \end{cases}$$

In the third example, a bijective transition function is used. The period is maximized and reaches the number of states. In all cases the transient is 1 and as a result all sequences are periodic.



$$T(x) := \begin{cases} x & : x \in \{0,3\} \\ 2 & : x = 1 \\ 1 & : x = 2 \end{cases}, \qquad (u_n) = \begin{cases} \overline{0} & : s_0 = 0 \\ \overline{21} & : s_0 = 1 \\ \overline{12} & : s_0 = 2 \\ \overline{3} & : s_0 = 3 \end{cases}$$

The last example shows again a bijective transition function T. The transient is again always 1 and all possible pseudorandom sequences are purely periodic. But this time, T maps the values 0 and 3 to themselves. Hence, the period becomes dependent on the initial value and differs between the smallest possible value 1 and 2.

The periodic behavior of pseudorandom sequences greatly constrains the possible randomness of a PRNG. Especially for simulations, using a PRNG which is repeating itself while in use introduces unwanted regularities resulting in an incorrect output. As a consequence, developers

of PRNGs try to construct a large period by adjusting the number of states and the transition function. For example, the MT19937 is a PRNG with an extremely large period of $2^{19937} - 1$ if not used with a seed value of zero (Matsumoto and Nishimura 1998). The use of a bijective transition function is not enough to ensure the maximal period. Values that are mapped to themselves result in the smallest possible period even if the transient of the sequence could be large. Especially for linear PRNGs that are mapping 0 to itself, developers tend to exclude such states from the seeding process to always obtain the maximal period (Blackman and Vigna 2019; Marsaglia 2003). As a counter-example, the so-called "Middle Square RNG" which was developed by Von Neumann in the early days of computer science should be named (Kneusel 2018, pp. 12–15; Widynski 2019). This PRNG computed the square of its current state and returned the middle digits as next random number. It was well known to suffer from the "zero mechanism" — once some digits become zero, all following return values would be zero as well (Kneusel 2018, pp. 12–15; Widynski 2019). So besides a large state space and a bijective transition function, the largest possible permutation cycle should be reached when advancing the state of a PRNG.

Equidistribution Pseudorandom sequences should mimic the behavior of truly random sequences. And for that reason, we want them to be uniformly distributed on the set of output values in some sense. This property will make it possible to generate every important distribution of random numbers by applying special transformations based on stochastics. Such distributions can then be used by Monte Carlo simulations to estimate solutions more efficiently. But because we are dealing with actual values instead of random variables, we have to clarify what uniformly distributed means. Consequently, we will again rely on probability theory to elaborate on the details without a deeper understanding of randomness (Eisner and Farkas 2019). To be able to always distinct these two different concepts, we will call a sequence of actual values with the desired properties equidistributed.

DEFINITION 2.11: (Equidistributed Sequence)

Let U be a non-empty, finite set of values and μ be a probability measure on the measurable space $(U, \mathcal{P}(U))$. A sequence $(u_n)_{n\in\mathbb{N}}$ in U is equidistributed with respect to μ if for every measurable function $X\colon U \to \mathbb{R}$ the following is true.

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} X(u_k) = \int_U X \,\mathrm{d}\mu$$

If μ is not specified, we assume it to be the uniform distribution on U.

The idea is that every possible output value should essentially be reached the same amount of times when advancing the state. For pseudorandom sequences generated by a non-bijective transition function the transient part should be ignored as it can be seen as non-recurring "warm-up" time. Therefore equidistribution will be evaluated at infinity in the sense of a limit.

Because we wanted to use probability theory to observe randomness, we had to generalize the idea of counting how often different output values would be reached. Instead we use arbitrary measurable functions as observables to estimate their expectation value with respect to the given sequence and to compare it to their actual expectation value (Eisner and Farkas 2019). Please note that for our needs we have chosen a finite set of elements to simplify the definition of equidistribution. A more general alternative where U has to be a compact metric space with Borel probability measure μ can be found in Eisner and Farkas (2019). Here, measurable functions are interchanged with continuous functions. Because of this, we can further simplify the right-hand side of the definition.

$$\int_U X \,\mathrm{d}\mu = \mathbb{E}X = \sum_{u \in U} f(u)\mu(\{u\})$$

To make sure the generalization is working properly, we proof the following lemma which states that, while observing pseudorandom sequences, the relative frequency in one period of an arbitrary element must be given by its probability.

LEMMA 2.6: (Equidistributed Pseudorandom Sequences)

Let $\mathcal{G} := (S, T, U, G)$ be a PRNG with $s_0 \in S$ as its seed value and $(u_n)_{n\in\mathbb{N}}$ the respective pseudorandom sequence with transient τ and period ρ. Furthermore, let μ be a probability measure on $(U, \mathcal{P}(U))$. Then the following statements are equivalent.

(i) (u_n) is equidistributed with respect to μ.

(ii) For all $u \in U$ the following is true.

$$\frac{1}{\rho} \cdot \#\{n \in \mathbb{N} \mid \tau \leq n < \rho + \tau, u_n = u\} = \mu(\{u\})$$

PROOF:
Because U is a finite set, every measurable function $X : U \to \mathbb{R}$ can be described as a linear combination of characteristic functions with respect to some real coefficients α_u for all $u \in U$ in the following way.

$$X = \sum_{u \in U} \alpha_u \mathbb{1}_{\{u\}}$$

Hence, without loss of generality, it suffices to take only characteristic functions into account. Let $u \in U$ be arbitrary. The right-hand side of the definition will then result in the following.

$$\int_U \mathbb{1}_{\{u\}} \,\mathrm{d}\mu = \mu(\{u\})$$

Applying the characteristic function together with the properties of a periodic sequence to the left-hand side of the definition, looks as follows.

$$\begin{aligned}\lim_{n\to\infty}\frac{1}{n}\sum_{k=1}^{n}\mathbb{1}_{\{u\}}(u_k) &= \lim_{n\to\infty}\frac{1}{n}\sum_{k=1}^{\tau-1}\mathbb{1}_{\{u\}}(u_k) + \lim_{n\to\infty}\frac{1}{n}\sum_{k=\tau}^{\tau+n-1}\mathbb{1}_{\{u\}}(u_k)\\ &= \frac{1}{\rho}\sum_{k=\tau}^{\tau+\rho-1}\mathbb{1}_{\{u\}}(u_k)\\ &= \frac{1}{\rho}\cdot\#\{n\in\mathbb{N}\mid \tau\le n<\rho+\tau, u_n=u\}\end{aligned}$$

This shows the desired equivalence and proofs the lemma. □

Based on this lemma, it directly follows that for equidistributed, pseudorandom sequences with a maximal period the number of different states has to be a multiple of the number of output values.

COROLLARY 2.7: (Equidistributed Pseudorandom Sequence with Maximal Period)

Let $\mathcal{G} := (S, T, U, G)$ be a PRNG with $s_0 \in S$ as its initial state and $(u_n)_{n\in\mathbb{N}}$ the respective pseudorandom sequence. If (u_n) is equidistributed and periodic with maximal period $\#S$ then the following is true.

$$\exists k\in\mathbb{N}: \quad \#S = k\cdot\#U$$

Multidimensional Equidistribution In physical problems, we typically have to deal with partial differential equations in many dimensions. Finding deterministic, numerical solutions through iterated integrals becomes infeasible due to the resulting degrees of freedom. This is called the "curse of dimensionality". With the use of Monte Carlo integration, we can overcome this burden so that for high-dimensional problems we are able to reduce the error of the estimated solutions for every iteration much faster. As a consequence, successive pseudorandom numbers generated by a PRNG should be interpretable as a pseudorandom vector. But due to the shown dependence of successive values in a pseudorandom sequence, again regular patterns and artifacts can arise which can only be observed by some advanced testing techniques for statistical performance. However, a PRNG that is used in more than one dimension should at least provide an equidistribution over all possible multidimensional output values. For a rigorous definition of this concept, we will first clarify how to use a pseudorandom sequence as a sequence of pseudorandom vectors.

DEFINITION 2.12: (Corresponding Vector Sequence)

Let U be a non-empty set of values and $(u_n)_{n\in\mathbb{N}}$ be a sequence in U. Choose $k\in\mathbb{N}$ and $t\in\mathbb{N}_0$ and define the following for all $n\in\mathbb{N}$.

$$v_n := (u_i)_{i\in I_n}\ , \qquad I_n := \{t+(n-1)k+p \mid p\in\mathbb{N}, p\le k\}$$

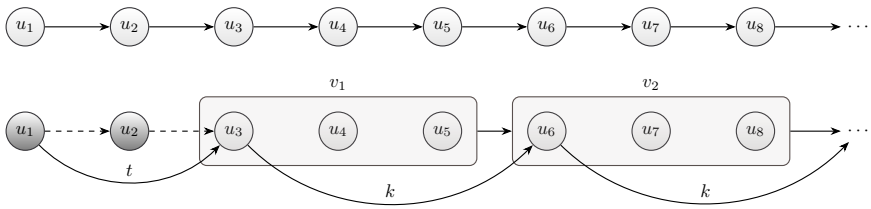


Figure 2: The upper part of the figure shows a schematic view of an arbitrary sequence of values $(u_n)_{n\in\mathbb{N}}$ in an arbitrary non-empty set U. The lower part visualizes the corresponding k-dimensional vector sequence $(v_n)_{n\in\mathbb{N}}$ with translation t, whereby $k = 3$ and $t = 2$. The first two values of (u_n) are skipped due to the translation. Afterwards the elements of (v_n), marked through boxes, emerge from interpreting successive values of (u_n) as their coordinates.

> *We call the sequence $(v_n)_{n\in\mathbb{N}}$ in U^k the corresponding k-dimensional vector sequence with translation t with respect to (u_n).*

Transforming a sequence of values into a sequence of vectors consists of interpreting successive values as coordinates of vectors. Figure 2 shows this process schematically. Corresponding vector sequences inherit the property of being ultimately periodic.

> **LEMMA 2.8:** (Corresponding Vector Sequences are Ultimately Periodic)
>
> *Let U be a non-empty set of values and $(u_n)_{n\in\mathbb{N}}$ be an ultimately periodic sequence in U with period ρ and transient τ. In this case, every corresponding k-dimensional vector sequence $(v_n)_{n\in\mathbb{N}}$ with translation t is ultimately periodic with period ρ' and transient τ' defined as follows.*
>
> $$\rho' := \frac{\rho}{\gcd(\rho, k)}\ , \qquad \tau' := \left\lceil \frac{\max(0, \tau - 1 - t)}{k} \right\rceil + 1$$

PROOF:

Choose $n \in \mathbb{N}_0$ and $i \in \mathbb{N}$ with $i \leq k$ to be arbitrary. We denote with $v_n^{(i)}$ the i. coordinate of the n. vector. By definition the following equality holds.

$$v^{(i)}_{\tau'+n+\rho'} = u_{t+(\tau'+n+\rho'-1)k+i}$$

Observing the index, we separate it into three parts. One for the index, one for the transient one for the period.

$$t + (\tau' + n + \rho' - 1)k + i = \underbrace{(t + \tau' k - k + 1)}_{=:\tilde{\tau}} + \underbrace{(nk + i - 1)}_{=:\tilde{n}} + \underbrace{\rho' k}_{=:\tilde{\rho}}$$

The period part has to be a multiple of the period ρ of (u_n) as can be seen in the following. Hence, $\tilde{\rho}$ has the property of a period.

$$\tilde{\rho} = \rho' k = \frac{\rho k}{\gcd(\rho, k)} = \rho \frac{k}{\gcd(\rho, k)}$$

To apply the periodicity of (u_n), the transient part has to be bigger or equal to the transient τ of (u_n).

$$\tilde{\tau} = t + \tau' k - k + 1 = 1 + t + k \left\lceil \frac{\max(0, \tau - 1 - t)}{k} \right\rceil \geq \tau$$

Inserting the results and applying the periodicity of (u_n), we can conclude that the corresponding vector sequence has to be ultimately periodic as well.

$$v^{(i)}_{\tau'+n+\rho'} = u_{\tilde{\tau}+\tilde{n}+\tilde{\rho}} = u_{\tilde{\tau}+\tilde{n}} = u_{t+(\tau'+n-1)k+i} = v^{(i)}_{\tau'+n}$$

Due to the shown statements, ρ' and τ' are indeed the smallest possible values such that this equation holds and can therefore be denoted as period and transient of (v_n) respectively. □

The given concept shall now be applied to define the equidistribution of a sequence in more than one dimension. As a result, the following property, called multidimensional equidistribution, becomes a generalization of equidistribution and quantifies in how many dimensions a PRNG can be used. We do not follow the typical definitions from L'Ecuyer (1994) and Matsumoto and Nishimura (1998).

DEFINITION 2.13: (Multidimensional Equidistributed Sequence)

Let U be a non-empty, finite set of values, $k \in \mathbb{N}$ and μ be a probability measure on $\left(U^k, \mathcal{P}\left(U^k\right)\right)$. A sequence $(u_n)_{n\in\mathbb{N}}$ in U is k-dimensional equidistributed with respect to μ if for all $t \in \mathbb{N}_0$ the corresponding k-dimensional vector sequence with translation t is equidistributed with respect to μ. If μ is not specified, we assume it to be the uniform distribution on U^k.

In comparison to the one-dimensional equidistribution, the general idea of multidimensional equidistribution is straightforward. For corresponding vector sequences, it reduces to the application of equidistribution. Especially for pseudorandom sequences, we can get a more precise result which will serve as an easily testable criterion for multidimensional equidistribution.

COROLLARY 2.9: (Multidimensional Equidistributed Pseudorandom Sequence)

Let $\mathcal{G} := (S, T, U, G)$ be a PRNG, $s_0 \in S$ its initial state and $(u_n)_{n\in\mathbb{N}}$ be the respective pseudorandom sequence with period ρ. Furthermore, let $k \in \mathbb{N}$ and (u_n) be k-dimensional equidistributed. In this case the following statement is true.

$$\exists a \in \mathbb{N}: \quad \rho = a \cdot \gcd(\rho, k) \cdot \#U^k$$

As a consequence, multidimensional equidistribution is greatly affected by the set of output symbols and the generator function. Furthermore, according to the formula, for k-dimensional equidistribution with $k \geq 2$, the set of output symbols has to be smaller than the set of states. In practice, the seed of a pseudorandom sequence defines the translation of its corresponding vector sequence. For the full period, the definition of multidimensional equidistribution given here is equivalent to the typical definition given in L'Ecuyer (1994). If the corresponding

vector sequence consists of a smaller period then the given concept is stronger than the typical one. We will again show some idealized examples to explain the details of the result and to understand its principles. For this, let $\mathcal{G} := (S, T, U, G)$ again be a PRNG, $s_0 \in S$ its initial state and $(u_n)_{n\in\mathbb{N}}$ the respective pseudorandom sequence with period ρ. The corresponding k-dimensional vector sequence with translation t will be called $(v_n)_{n\in\mathbb{N}}$. Sequences will again be denoted by writing their elements consecutively with their periodic part marked by an overline. G will be shown as table which maps values from the first line to values in the second line.

The first example will use a PRNG with a state size of 8 and a trivial, bijective transition function with full period. The generator function G is chosen so that the resulting pseudorandom sequence is k-dimensional equidistributed for $k = 3$.

$$S := \mathbb{Z}_8 \,, \qquad G := \mathbb{Z}_2 \,, \qquad T(x) := x + 1 \mod 8 \,, \qquad s_0 = 7$$

$$G := \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Choosing $k = 3$ and setting the translation to $t = 0$, the corresponding vector sequence will have a maximal period and will reach every element in U^3. A different translation would only result in a cyclic permutation of the periodic part.

$$(v_n) = \overline{(000)(111)(010)(001)(110)(100)(011)(101)}$$

For $k = 2$ and $t = 0$ the corresponding vector sequence must have the half period. In this case, the sequence is not equidistributed in U^2 because the element (10) is not reached and (01) is reached twice.

$$(v_n) = \overline{(00)(01)(11)(01)}$$

Shifting the sequence by setting $t = 1$, we get its complement. This time, it does not reach (01) but (10) twice instead. Again, the period is 4 and the sequence is not equidistributed.

$$(v_n) = \overline{(00)(11)(10)(10)}$$

Putting both sequences for $t = 0$ and $t = 1$ together results in an two-dimensional equidistributed sequence. Note that the weaker definition of multidimensional equidistribution given in L'Ecuyer (1994) would therefore call the given sequence two-dimensional equidistributed.

In the second example, we have chosen a doubled state size and adjusted the generator function to achieve k-dimensional equidistribution for $k = 2$ and $k = 3$.

$$S := \mathbb{Z}_{16} \,, \qquad G := \mathbb{Z}_2 \,, \qquad T(x) := x + 1 \mod 16 \,, \qquad s_0 = 15$$

$$G := \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

The greatest common divisor of 2 and 16 is again 2. Hence, the corresponding vector sequence has half period. But this time every element is reached. Changing the translation to $t = 1$ would permute the sequence.

$$(v_n) = \overline{(00)(01)(11)(01)(00)(11)(10)(10)}$$

In three dimensions, we get the full period and an equidistribution in U^3. A different translation would again only result in a cyclic permutation of the periodic part.

$$\begin{aligned}(v_n) = &\overline{(000)(111)(010)(011)(101)(000)(011)(101)}\\ &\overline{(001)(110)(100)(001)(110)(100)(111)(010)}\end{aligned}$$

Note that for $k = 4$, according to corollary 2.9, we cannot achieve k-dimensional equidistribution because for all $a \in \mathbb{N}$ we get the following inequality.

$$2^4 = 16 = \rho \neq a \cdot \gcd(\rho, k) \cdot \#U^k = a \cdot 4 \cdot 2^4 = a \cdot 2^6$$

Hence, for the given PRNG we have reached maximal equidistribution by keeping its maximal period.

Linearity According to L'Ecuyer (2015), the transition function of most PRNGs can be viewed as linear recurrence modulo some prime number. Thus, such PRNGs are called linear and exhibit certain advantages and disadvantages in comparison to non-linear PRNGs. The linearity property makes a theoretical and statistical analysis of respective pseudorandom sequences much easier (Bauke and Mertens 2007; Blackman and Vigna 2019; L'Ecuyer 2015). Hence, linear PRNGs are mathematically well-founded and understood. But exactly this makes them vulnerable to certain empirical tests and applications, such as the linear-complexity and the matrix-rank tests, which exploit linearity (L'Ecuyer 2015; Lemire and O'Neill 2019; O'Neill 2014). In general, linear PRNGs suffer from too much regularity in their output. Nevertheless, many well-known and widely used PRNGs are linear. This is due to the fact that while offering more features they tend to be faster and easier to implement than their counterparts (Blackman and Vigna 2019; L'Ecuyer 2015). To name a few examples, the MT19937 (Matsumoto and Nishimura 1998), Xorshift RNGs (Marsaglia 2003; Vigna 2016, 2017), and WELL generators (Panneton, L'ecuyer, and Matsumoto 2006) are all linear PRNGs. We will first introduce a rigorous concept of linearity to understand their underlying theory.

DEFINITION 2.14: (Linear and Scrambled Linear PRNG)

Let $m \in \mathbb{P}$ and $\mathcal{G} := (S, T, U, G)$ be a PRNG. We call $\mathcal{G}$ linear modulo m if S and U are finite-dimensional vector spaces over the finite field $\mathbb{F}_m$ and T and G are linear transformations. In this case, we identify T and G with their respective matrices such that $T \in \mathbb{F}_m^{p \times p}$ and $G \in \mathbb{F}_m^{q \times p}$, whereas $p := \dim S$ and $q := \dim U$. If G cannot be represented by a linear transformation we say $\mathcal{G}$ is a scrambled linear PRNG modulo m.

The state and output space have to be vector spaces over a finite field such that linear transformations are well-defined. The definition makes sure that for linear PRNGs both, the transition and the generator function, are linear transformations. This property is strong and tends to reduce the statistical performance of the PRNG. Therefore in practice, often at least the generator function is chosen to be non-linear (Blackman and Vigna 2019). Examples for scrambled linear PRNGs are given by some generators of the PCG family (O'Neill 2014) and the Xoroshiro128+ (Blackman and Vigna 2019). Due to their non-linear generator function, these PRNGs are more difficult to analyze. As a consequence, the following lemma about the equidistribution and periodicity applies only to linear PRNGs and may under certain circumstances serve as a foundation for a further theoretical analysis of scrambled linear PRNGs (Bauke and Mertens 2007; L'Ecuyer 2015).

LEMMA 2.10: (Period and Equidistribution of a Linear PRNG)

For $m \in \mathbb{P}$, let $\mathcal{G} := (S, T, U, G)$ be a linear PRNG modulo m with $p := \dim S$. Furthermore, let the characteristic polynomial of T be a primitive polynomial over $\mathbb{F}_m$ and let G be a full rank matrix with $q := \operatorname{rank} G$. Then for all seeds $s_0 \in S \setminus \{0\}$ the respective pseudorandom sequences $(u_n)_{n \in \mathbb{N}}$ are periodic with period $m^p - 1$ and for all elements $u \in U$ the following holds.

$$n_u := \#\{n \in \mathbb{N} \mid n \leq m^p - 1, u_n = u\} = \begin{cases} m^{p-q} - 1 & : u = 0 \\ m^{p-q} & : \text{else} \end{cases}$$

In particular, the sequence (u_n) is equidistributed with respect to the following probability measure μ.

$$\mu \colon \mathcal{P}(U) \to [0, 1] \ , \qquad \mu(\{u\}) := \frac{n_u}{m^p - 1}$$

We will give no proof for this lemma. For linear PRNGs, it is not possible to get an equidistribution on the complete output space U. Instead the zero element is always reached once less often, introducing bias in the respective probability distribution μ. But for big values of p, this bias is neglectable as the following limit states.

$$\lim_{p \to \infty} \frac{1}{m^p - 1} = 0$$

Apart from this, by reaching the maximal period, a linear PRNG gives us the equidistribution of its output values for free. Hence, both properties, equidistribution and maximal periodicity, can be mathematically proven by showing that the characteristic polynomial of T is a primitive polynomial over the underlying field. This gives us a general tool for the analyzation of linear PRNGs and explains their widespread use.

Let us again consider an example to further highlight the usage of linear PRNGs. For this, let $\mathcal{G} := (S, T, U, G)$ be a linear PRNG modulo 2, $s_0 \in S$ its initial state, $(u_n)_{n \in \mathbb{N}}$ the respective pseudorandom sequence and $(s_n)_{n \in \mathbb{N}}$ the respective sequence of states. Sequences will again

be denoted by writing their elements consecutively with their periodic part marked by an overline. In this example, column vectors will be written as row vectors without spaces or commas.

$$S := \mathbb{F}_2^3\,, \qquad U := \mathbb{F}_2^2\,, \qquad s_0 := (001)$$

$$T := \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}\,, \qquad G := \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In this example, G has full rank and the characteristic polynomial of T is given by the following expression which is a primitive polynomial over $\mathbb{F}_2$.

$$p_T(x) = \det(T - xI) = x^3 + x^2 + 1$$

As a result, we will get the maximal period of 7 elements for the state and output sequence.

$$(s_n) = \overline{(001)(011)(111)(110)(101)(010)(100)}$$

$$(u_n) = \overline{(11)(11)(10)(01)(10)(00)(01)}$$

In one period, (u_n) reaches the elements (01), (10) and (11) exactly two times whereas (00) only appears one time.

Predictability and Security

2.2.6 Implementation-Specific Performance

Seekability

2.2.7 Analyzation

visualization, proof, experiments, benchmarks (runtime), test suites, code analyzation

statistical quality and performance vs implementation quality and performance

Visualizations: randograms 2d and 3d, histograms, simulation plots and images

Period and Uniformity, Empirical Testing, predictability and Security, Speed, Memory Size, Code Size, Output Range, Seekability, multiple streams, k-dimensional equidistribution, theoretical support, repeatability, portability, ease of implementation

2.2.8 Examples

DEFINITION 2.15: (Linear Congruential Generator)

Let $m \in \mathbb{N}$ with $m \geq 2$ and $a, c \in \mathbb{Z}_m$. We define the PRNG $\mathrm{LCG}(m, a, c) :=$

(S, T, U, G)

$$S := U := \mathbb{Z}_m \ , \qquad G := \mathrm{id}_{\mathbb{Z}_m}$$

$$T \colon S \to S \ , \qquad T(x) := ax + c$$

Multiplication and addition are understood in the sense of $\mathbb{Z}_m$. We call $\mathrm{LCG}(m, a, c)$ the linear congruential generator with modulus m, multiplier a and increment c.

DEFINITION 2.16: (Linear Feedback Shift Registers)

DEFINITION 2.17: (Mersenne Twister)

Let $w, n, m \in \mathbb{N}$ and $r \in \mathbb{N}_0$ with $m \leq n$ and $r < w$. Further, let $a, b, c \in \mathbb{Z}_2^w$ and $u, s, t, l \in \mathbb{Z}_w$. Then the Mersenne Twister $\mathrm{MT}(w, n, m, r, a, b, c, u, s, t, l) := (S, T, U, G)$ is defined as a PRNG in the following way.

$$S := \mathbb{Z}_n \times \mathbb{Z}_2^{w \times n} \ , \qquad U := \mathbb{Z}_2^w$$

$$T \colon S \to S$$

$$\forall i \in \mathbb{Z}_{n-1} : T(i, x) := (i + 1, x)$$

$$T(n - 1, x) := (0, y)$$

$$\begin{aligned} \forall i \in \mathbb{Z}_{n-m} : y_i &:= x_{m+i} \oplus \left(x_i^u \middle| x_{i+1}^l\right) A \\ \forall i \in \mathbb{Z}_{m-1} + (n - m) : y_i &:= y_{i-(n-m)} \oplus \left(x_i^u \middle| x_{i+1}^l\right) A \\ y_{n-1} &:= y_{m-1} \oplus \left(x_{n-1}^u \middle| y_0^l\right) A \end{aligned}$$

$$xA := \begin{cases} x \gg 1 & : x_0 = 0 \\ (x \gg 1) \oplus a & : x_0 = 1 \end{cases}$$

$$f_1(x) := x \oplus (x \gg u)$$

$$f_2(x) := x \oplus ((x \ll s) \odot b)$$

$$f_3(x) := x \oplus ((x \ll t) \odot c)$$

$$f_4(x) := x \oplus (x \gg l)$$

$$G \colon S \to U \ , \qquad G(i, x) := f_4 \circ f_3 \circ f_2 \circ f_1(x_i)$$

DEFINITION 2.18: (Permuted Congruential Generator)

Given $\mathcal{G} \coloneqq \mathrm{LCG}(b, a, c)$ *with transition function* T*. Let* $t \in \mathbb{Z}_b$ *and* $f_c \colon \mathbb{Z}_{2^{b-t}} \to \mathbb{Z}_{2^{b-t}}$ *be a permutation for all* $c \in \mathbb{Z}_{2^t}$*.*

$$S \coloneqq \mathbb{Z}_{2^b} \ , \qquad U \coloneqq \mathbb{Z}_{2^{b-t}}$$

$$G \coloneqq \pi_2 \circ f_* \circ \mathrm{split}_t$$

$$f_*(a, b) \coloneqq (a, f_a(b))$$

$$\mathrm{PCG}(\mathcal{G}, t, \{f_c \mid c \in \mathbb{Z}_{2^t}\}) \coloneqq (S, T, U, G)$$

DEFINITION 2.19: (Xoroshiro128+)

Let $a, b, c \in \mathbb{Z}_{64}$*.*

$$S \coloneqq \mathbb{Z}_{2^{64}}^2 \ , \qquad U \coloneqq \mathbb{Z}_{2^{64}}$$

$$T(x, y) \coloneqq (x \circlearrowleft a \oplus f(x, y) \oplus (f(x, y) \leftarrow b), f(x, y) \circlearrowleft c)$$

$$f(x, y) \coloneqq x \oplus y$$

$$G(x, y) \coloneqq x + y$$

DEFINITION 2.20: (Middle Square Weyl Sequence RNG)

Let $s \in \mathbb{Z}_{2^{64}}$ *be an odd constant. The middle square Weyl sequence RNG* $\mathrm{MSWS}(s) \coloneqq (S, T, U, G)$ *is defined as a PRNG in the following way.*

$$S \coloneqq \mathbb{Z}_{2^{64}}^2 \ , \qquad U \coloneqq \mathbb{Z}_{2^{32}}$$

$$T \colon S \to S \ , \qquad T(w, x) = (w + s, f(x^2 + w + s))$$

$$f \colon \mathbb{Z}_{2^{64}} \to \mathbb{Z}_{2^{64}} \ , \qquad f(x) \coloneqq (x \gg 32)\mathrm{or}(x \ll 32)$$

$$G \colon S \to U \ , \qquad G(w, x) \coloneqq x \mod 2^{32}$$

2.3 Simulation in Physics and Mathematics

2.3.1 Mathematical and Physical Preliminaries

2.3.2 Baseline Model Problems

2.4 SIMD-Capable Processors

2.4.1 Architecture of Modern Central Processing Units

2.4.2 SIMD Instruction Sets and Efficiency

2.4.3 SSE, AVX, AVX512

2.5 Summary

Kneusel (2018) and Volchan (2002) (Volchan 2002, p. 1; Kneusel 2018, p. 2)

3 Previous Work

3.1 The C++ API and Further Progressions

3.2 Techniques for Vectorization and Parallelization

3.3 Summary

The topic of PRNGs consists of several smaller parts. From a mathematical point of view, one has to talk about their definition and construction as well as methods on how to test their randomness. There have been a lot of publications concerning these issues. Hence, I am not able to give you a detailed overview. Instead, I will focus on the most relevant PRNGs and test suites, as well as some modern examples.

The creation of new PRNGs is sometimes understood to be black magic and can be hard since basically, one has to build a deterministic algorithm with a nearly non-deterministic output. In Kneusel (2018) one can find numerous different families of PRNGs. The most well-known ones are Linear Congruential Generators, Mersenne Twisters and Xorshift with its Variants. Whereas LCGs tend to be fast but weak generators in O'Neill (2014), one can find a further developed promising family of algorithms, called PCGs. Widynski (2019) describes another RNG based on the so-called middle square Weyl sequence. All of these generators have certain advantages and disadvantages in different areas such as security, games, and simulations.

After building a PRNG, one has to check if the generated sequence of random numbers fulfills certain properties. In general, these properties will somehow measure the randomness of our RNG. Typically, there are a lot of tests bundled inside a test suite such as TestU01 and Dieharder.

4 Design of the API

What do we want from the interface of our RNG? It should make testing with given frameworks like TestU01, dieharder, ent and PractRand easy. Benchmarking should be possible as well. Therefore we need a good API and a good application interface. Most of the time we want to generate uniform distributed real or integer numbers. We need two helper functions. So we see that the concept of a distribution makes things complicated. We cannot specialize distributions for certain RNGs. We cannot use lambda expressions as distributions. Therefore we want to use only helper functions as distributions and not member functions. So we do not have to specify a specialization and instead use the given standard but we are able to do it. Therefore functors and old-distributions are distributions as well and hence we are compatible to the standard.

Additionally, we have to be more specific about the concept of a random number engine. The output of a random number engine of the current concept is magical unsigned integer which should be uniformly distributed in the interval [min,max]. But these magic numbers can result in certain problems if used the wrong way, see Melissa O'Neill Seeding Surprises. Therefore the general idea is to always use the helper functions as new distributions which define min and max explicitly and make sure you really get those values. This is also a good idea for the standard. And it is compatible with the current standard.

Now think of vector registers and multiprocessors. The random number engine should provide ways to fill a range with random numbers such that it can perform generation more efficiently. Think about the execution policies in C++17. They should be provided as well.

5 Testing Framework

6 Implementation of Vectorized PRNGs

6.1 Linear Congruential Generatiors

6.2 Mersenne Twister

6.3 Permuted Congruential Generators

6.4 Xoroshiro

6.5 Middle Square Weyl Generator

6.6 Summary

7 Application to Simulations

8 Evaluation and Results

godbolt google benchmark intel vtune amplifier testu01 dieharder

9 Conclusions

References

Barash, L. Yu., Maria S. Guskova, and Lev. N. Shchur (2017). "Employing AVX Vectorization to Improve the Performance of Random Number Generators". In: *Programming and Computer Software* 43.3, pp. 145–160. DOI: 10.1134/S0361768817030033.

Bauke, Heiko and Stephan Mertens (2007). "Random Numbers for Large-Scale Distributed Monte Carlo Simulations". In: *Physical Review E* 75.6, p. 066701. DOI: 10.1103/PhysRevE.75.066701.

Blackman, David and Sebastiano Vigna (August 1, 2019). "Scrambled Linear Pseudorandom Number Generators". In: *arXiv.org*. URL: https://arxiv.org/abs/1805.01407v2 (visited on 10/26/2019).

Eisner, Tanja and Balint Farkas (January 9, 2019). "Ergodic Theorems". Course Notes to 22. Internetseminar of Virtual Lectures about Classical and Modern Ergodic Theory. URL: https://ergodic.mathematik.uni-leipzig.de/uploads/default/original/1X/74a3d34dfcdce08c3423f3ec62aa43db88abf189.pdf (visited on 10/27/2019).

Elstrodt, Jürgen (2018). *Maß- und Integrationstheorie*. Achte Auflage. Springer Spektrum. ISBN: 978-3-662-57938-1. DOI: 10.1007/978-3-662-57939-8.

Graham, Carl and Denis Talay (2013). *Stochastic Simulation and Monte Carlo Methods. Mathematical Foundations of Stochastic Simulation*. Springer. ISBN: 978-3-642-39362-4. DOI: 10.1007/978-3-642-39363-1.

Intel (2018). *Intel Digital Random Number Generator (DRNG) Software Implementation Guide*. Revision 2.1. URL: https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide (visited on 09/17/2019).

Kneusel, Ronald T. (2018). *Random Numbers and Computers*. Springer. ISBN: 978-3-319-77697-2. DOI: 10.1007/978-3-319-77697-2.

Landau, David P. and Kurt Binder (2014). *A Guide to Monte Carlo Simulations in Statistical Physics*. Fourth Edition. Cambridge University Press – University of Cambridge. ISBN: 978-1-107-07402-6. DOI: 10.1017/CBO9781139696463.

L'Ecuyer, Pierre (December 1994). "Uniform Random Number Generation". In: *Annals of Operations Research* 53, pp. 77–120. DOI: 10.1007/BF02136827.

— (2015). "Random Number Generation with Multiple Streams for Sequential and Parallel Computing". In: *2015 Winter Simulation Conference (WSC)*. IEEE, pp. 31–44. DOI: 10.1109/WSC.2015.7408151.

Lemire, Daniel and Melissa E. O'Neill (2019). "Xorshift1024*, Xorshift1024+, Xorshift128+ and Xoroshiro128+ Fail Statistical Tests for Linearity". In: *Journal of Computational and Applied Mathematics* 350, 139–142. ISSN: 0377-0427. DOI: 10.1016/j.cam.2018.10.019.

Marsaglia, George et al. (2003). "Xorshift RNGs". In: *Journal of Statistical Software* 8.14, pp. 1–6.

Matsumoto, Makoto and Takuji Nishimura (1998). "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator". In: *ACM Transactions on*

Modeling and Computer Simulation (TOMACS) 8.1, pp. 3–30. DOI: 10.1145/272991.272995.

Müller-Gronbach, Thomas, Erich Novak, and Klaus Ritter (2012). *Monte Carlo-Algorithmen*. Springer. ISBN: 978-3-540-89141-3. DOI: 10.1007/978-3-540-89141-3.

O'Neill, Melissa E. (2014). *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. Tech. rep. HMC-CS-2014-0905. Claremont, CA: Harvey Mudd College. URL: https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf (visited on 08/28/2019).

Panneton, François, Pierre L'ecuyer, and Makoto Matsumoto (2006). "Improved Long-Period Generators Based on Linear Recurrences Modulo 2". In: *ACM Transactions on Mathematical Software (TOMS)* 32.1, pp. 1–16. DOI: 10.1145/1132973.1132974.

Schmidt, Klaus D. (2009). *Maß und Wahrscheinlichkeit*. Springer. ISBN: 978-3-540-89729-3. DOI: 10.1007/978-3-540-89730-9.

Vigna, Sebastiano (2016). "An Experimental Exploration of Marsaglia's Xorshift Generators, Scrambled". In: *ACM Transactions on Mathematical Software (TOMS)* 42.4, p. 30. DOI: 10.1145/2845077.

— (2017). "Further Scramblings of Marsaglia's Xorshift Generators". In: *Journal of Computational and Applied Mathematics* 315, pp. 175–181. DOI: 10.1016/j.cam.2016.11.006.

Volchan, Sérgio B. (2002). "What is a Random Sequence?" In: *The American Mathematical Monthly* 109.1, pp. 46–63. DOI: 10.2307/2695767.

Waldmann, Stefan (2017). *Lineare Algebra 1. Die Grundlagen für Studierende der Mathematik und Physik*. Springer Spektrum. ISBN: 978-3-662-49914-6. DOI: 10.1007/978-3-662-49914-6.

Widynski, Bernard (July 31, 2019). "Middle Square Weyl Sequence RNG". In: *arXiv.org*. URL: https://arxiv.org/abs/1704.00358v4 (visited on 08/28/2019).

Statutory Declaration

I declare that I have developed and written the enclosed Master's thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Master's thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.
On the part of the author, there are no objections to the provision of this Master's thesis for public use.

Jena, November 4, 2019

Markus Pawellek