NEW TESTS OF RANDOM NUMBERS FOR SIMULATIONS IN PHYSICAL SYSTEMS

Thesis for the degree of Licentiate in Technology

Ilpo Vattulainen

1994

Preface

It has been said that doing research is the most wonderful thing a man can do with clothes on. I agree. Although the sweat always flows in the middle of disturbing problems, all troubles disappear (for a moment) when the solution is found, or when a pint finds its owner.

This Thesis has been done at TFT (Research Institute for Theoretical Physics, University of Helsinki, Finland) under the supervision of Doc. Tapio Ala-Nissilä. Although there are several persons to whom I am indebted during the course of this work, for obvious reasons it is Tapio who I want to thank first. He certainly has made a real effort to convince me that the idea of "unbearable lightness of being a grad student" is not valid. I still think it is. Anyway, although he hasn't fully succeeded in molding me, in many other ways he has taught me a lot. For that reason, among others, working in his group has certainly been worthwhile. Kari Kankaala has been an important character in this work as well, not only due to his good attitude for life, but also because some pseudorandom number generators like him. Let them make him rich some day. Working with Tuomo Hjelt and Sami Majaniemi has also been valuable many times, mainly because of Sami's unusual and intriguing character. The comments of Dr. Frederick James, Prof. Kimmo Kaski, and Prof. Timo Mäkeläinen have improved this Thesis quite a lot. In addition, I want to thank Kimmo for helping me to achieve my current "position" at TFT about one and a half years ago, where there is a fine atmosphere among the personnel which I haven't experienced before. Thank you all.

During this work I visited the research groups of See Chen Ying at Brown University (Providence, USA) and Martin Grant at McGill University (Montreal, Canada), whose hospitality was the most welcome surprise. For the same reason, I want to thank Mikko and Merja Karttunen in Montreal and Minna Hovinen and Arto Salokatve in

Providence. Moreover, for several reasons that would take too much space to explain, I want to thank Mikko in particular. In practical matters, it is a pleasure to thank several organizations, whose support has made this work and my graduate studies possible. Center for Scientific Computing, Tampere University of Technology, and University of Helsinki have provided ample computing resources, and Jenny and Antti Wihuri Foundation, Vaisala Foundation, Neste Foundation, Academy of Finland, and TFT have given me invaluable financial support.

Finally, let me mention in passing how unpleasant working in nightshift may sometimes be. Fortunately, the Finnish Nightradio has often given consolation to my soul during many nights of lonely silence. Additionally, several amazing groups such as the 10 000 Maniacs, The The, and 22 Pistepirkko (among others) have turned my attention to many irrelevant ideas, for which reason, thanks!

Kirsi, *go raibh maith agat!*

At night in November in Helsinki, vattu.

Abstract

The aim of this Thesis is to present five new tests for random numbers, which are widely used *e.g.* in computer simulations in physics applications. The first two tests, the cluster test and the autocorrelation test, are based on analogies to the two-dimensional Ising model. The next two, the random walk test and the n-block test, are based on studies of random walks, and the condition number test presented last uses some results of Gaussian distributed random matrices. Studies with several commonly used pseudorandom number generators reveal that the cluster test is particularly powerful in finding periodic correlations on bit level, and that the autocorrelation test, the random walk test, and the n-block test are very effective in detecting short-ranged correlations. The results of the condition number test are mostly inconclusive, however. By means of the tests presented in this work, two important results are found. First, we show quantitatively that the reason for erroneous results in some recent high precision Monte Carlo simulations for some commonly used pseudorandom number generators are the so called triple correlations in the sequences. Then, we show that the properties of such a sequence may be considerably improved, if only a certain portion of it is used.

TAMPEREEN TEKNILLINEN KORKEAKOULU
Sähkötekniikan osasto
VATTULAINEN, ILPO: *New tests of random numbers for simulations in physical systems*
Lisensiaattityö, 85s. + liitteet 3s.
Fysiikka
Tarkastajat: Prof. Kimmo Kaski ja Dr. Frederick James
Rahoittajat: Jenny ja Antti Wihurin säätiö, Nesteen säätiö, Suomalainen Tiedeakatemia, Suomen Akatemia ja Teoreettisen fysiikan tutkimuslaitos (TFT/HY)
Marraskuu 1994
PACS-numerot: 02.70.Lq, 75.40Mg, 05.40+j, 05.50.+q.
Avainsanat: Monte Carlo methods, random numbers

Satunnaislukuja käytetään lukuisissa sovelluksissa erityisesti teoreettisen fysiikan alalla. Näiden sovellusten oikean toiminnan kannalta on satunnaislukujen hyvyydellä keskeinen merkitys. Perinteisesti tätä aihetta on tutkittu kehittämällä erilaisia testejä, jotka etsivät satunnaislukujonoista erilaisia korrelaatioiden ilmentymiä.

Tässä lisensiaattityössä on kehitetty viisi uutta satunnaisuuden testiä, jotka perustuvat analogioihin fysikaalisten mallien kanssa. Testeistä kaksi ensimmäistä, klusteritesti ja autokorrelaatiotesti, perustuvat kaksiulotteisen Ising-mallin eksaktiin ratkaisuun. Seuraavat satunnaiskävelytesti ja n-blokkitesti perustuvat satunnaiskävelyihin, ja viimeisenä esiteltävä häiriöalttiuslukutesti hyödyntää joitakin normaalijakautuneiden satunnaismatriisien tunnettuja ominaisuuksia. Suoritettujen tutkimusten perusteella klusteritesti on erittäin tehokas löytämään bittitason korrelaatioita, ja autokorrelaatiotesti, satunnaiskävelytesti ja n-blokkitesti ovat erittäin tehokkaita lyhyen kantaman korrelaatioiden etsimiseen. Häiriöalttiuslukutestin tulokset eivät rohkaise sen laajamittaiseen käyttöön. Kehitettyjen testien tehokkuutta kuvaa edelleen se, että niiden avulla on pystytty osoittamaan syy siihen, miksi eräät yleisesti käytetyt pseudosatunnaislukugeneraattorit ovat antaneet virheellisiä tuloksia joissakin Monte Carlo

-simulaatioissa. Testien perusteella nämä virheelliset tulokset johtuvat pseudosatunnaislukujonojen sisältämistä niin kutsutuista kolmoiskorrelaatioista. Lisäksi on voitu osoittaa, että näiden korrelaatioiden haitallista vaikutusta voidaan merkittävästi vähentää käyttämällä satunnaislukujonosta vain tietty osa.

Contents

List of Tables

List of Figures

Prologue

"Do you believe in ghosts?"
"No," I say.
"Why not?"
"Because they are *un*-sci-en-*ti*-fic."

"*Zen and the Art of Motorcycle Maintenance*"
R. M. Pirsig

A *good* random number generator *is* a ghost. You may consider any random number source, physical or deterministic. You may test it extensively and find amazingly good test results. You may even have faith in its good properties. But then, suppose you have managed to surround it in a corner; if you try to look at it more closely, it has already disappeared, leaving behind a vague outline in the mist around you — just like a mirage in the desert.

Chapter 1

Introduction

In modern computational science, long sequences of random numbers are required in various fields such as statistical mechanics, particle physics, and applied mathematics. Methods utilizing random numbers include Monte Carlo simulation techniques [10], stochastic optimization [1], and cryptography [165], all of which usually require fast and reliable random number sources. In practice, the random numbers needed for these methods are produced by deterministic rules, implemented as pseudorandom number generators which usually rely on simple arithmetic operations. Obviously, these pseudorandom number sequences can be "random" only in some limited sense, and therefore their main purpose is only to *imitate* random behavior as well as possible. Assuming that physical stochastic processes such as nuclear decay and thermal noise allow us to generate "truly" random number number sequences (in the sense that they do not contain correlations), using this approach might be a more reliable method than use of pseudorandom number sequences. However, due to practical reasons physical sources are usually not used.

Since the very idea of using deterministic algorithms in generation of random variables is in conflict with any idea of randomness, an obvious question arises: how can these sequences be used in applications such as Monte Carlo simulations, whose performance is fully based on the assumption of truly random numbers? In an illustrative sense, the justification for their use may be considered in terms of the accuracy: when the number of independent samples N is small, the precision of the Monte Carlo method is poor (the error being proportional to $1/\sqrt{N}$ [97]). Therefore, *subtle* deviations from randomness in pseudorandom number sequences may not appear unless very many samples are taken. Thus, for such computational applications in which high precision is not a crucial requirement, there are numerous fairly good pseudorandom number generators which will work just fine. Such generators are like compasses for the sailors in the 15^{th} century: in those days, when long distance voyages were not a standard routine, a compass often lead the ship close to the desired place, where other means such as local knowledge could be utilized to find the precise location.

However, the technological development of computers has lead to a situation, where carrying out ever demanding computational tasks is possible. In the case of Monte Carlo simulations, this means that in addition to studying more challenging problems, more accurate simulations (with larger N) can be carried out. When such high precision simulations are being done, however, there must be better sources of randomness than just "fairly" good pseudorandom number generators; the compass must be replaced with a satellite navigation system. In other words, improvement of the accuracy leads to a situation where the quality of pseudorandom number sequences should improve as well. Otherwise, ambiguous results may appear. For example, in the mid 1980's high precision calculations of the critical temperature in the three-dimensional Ising model [8] received a lot of attention, and in the cases where dubious results were found, the quality of some pseudorandom number generators was questioned [6, 76, 77, 145]. This raises another question: how can we determine that a pseudorandom number sequence is "random enough" for some particular application? Naturally, if an exact analytic answer is known in a special case, for example, we may check the quality of a pseudorandom number sequence *in situ* by using it in this application. Otherwise, the quality of pseudorandom number sequences must somehow be tested indirectly. Basically, there are two such indirect means. *Theoretical tests* ([96] pp. 75-110) are based on studying some properties such as the period length and uniformity of a pseudorandom number generator algorithm. However, since almost without exception theoretical tests study the properties of algorithms over their entire cycle, *empirical tests* ([96] pp. 59-73) are also needed. In addition to studying properties of finite subsequences instead of the whole period, empirical tests may also give us further insight on how the pseudorandom number generator behaves. Furthermore, since the implementation of the algorithm (written for a computer) may be incorrect, empirical tests provide the means to detect possible errors.

In the course of time, many theoretical [33, 96, 106, 142, 177] and empirical [12, 21, 60, 64, 68, 71, 89, 92, 105, 116, 127, 137, 163, 189, 200, 201] tests for pseudorandom number generators have been suggested. However, since all pseudorandom number generators are based on a deterministic algorithm, it is always possible to construct a test for every generator where it will fail. Therefore, although the success of pseudorandom number generators in extensive testing improves confidence in their properties, it is never a sufficient condition for their use in all applications. Recently, this phenomenon was observed in some high precision Monte Carlo simulations, in which several commonly used pseudorandom number generators gave incorrect results [28, 49, 65, 66, 166] when special simulation algorithms were employed. Still, these generators have performed well in several earlier tests [26, 72, 94, 121, 190]. Since it is important to make sure that a pseudorandom number generator is good enough for a chosen application, it is important to test it with such tests, which mimic the properties of the application where the generator will be used. In other words, efficient *application specific tests* of randomness are clearly needed.

The aim of this work is to present five new tests for pseudorandom number generators. These tests have been developed from the point of view of a physicist, in the sense that they are based on direct analogies to some physical systems such as the Ising model [8], which has been utilized in the first two tests. *The cluster test* [86] is based on comparing the cluster size distribution of a random lattice with the Ising model at an infinite temperature. Then, in *the autocorrelation test* [191] we calculate the integrated autocorrelation time of some quantities of the Ising model, when the Wolff updating method [197] is being used. In addition, two other tests related to random walks will also be proposed. In *the random walk test* [191], we consider the distribution of the final position of a random walk on a plane which is divided into four equal blocks. *The n-block test* [191] is based on the idea of renormalizing a sequence of uniformly distributed random numbers, and it is essentially a random walk test in one dimension. Finally, *the condition number test* [192] utilizes some exact results on Gaussian distributed random matrices.

The outline of this Thesis is as follows. In Chapter 2, we first consider the concept of randomness, which because of its extraordinary character has no unique definition. We proceed by considering generation methods of randomness and their desired properties mainly from a practical point of view, and therefore the emphasis of this discussion is on pseudorandom number sequences. Then, a brief historical perspective of random number generation leads us to Chapter 3, where pseudorandom number generators and their use are studied in more detail. Test methods for randomness are discussed in Chapter 4, where more detailed descriptions of the tests developed in this work are also presented. Following this background, the results of these tests are given in Chapter 5, where we first demonstrate that the cluster test is particularly powerful in finding periodic bit level correlations, being the most efficient of three bit level tests whose efficiency we have studied in this work. The other two bit level tests are included in this work for the purpose of comparison only. Moreover, we show that the autocorrelation test, the random walk test, and the n-block test are very effective in detecting short-range correlations, whereas the results of the condition number test are mostly inconclusive. Finally, the summary and discussion are given in Chapter 6.

Chapter 2

Concept of randomness

Randomness may be regarded as a notion opposite to being deterministic, which means that if the history of some process is known then its future may be predicted. Therefore, in random phenomena no memory effects should be present, and hence consequtive events should be *independent* of each other [208]. Despite this clear description, there are fundamental problems in the actual definition of randomness. Although this problem concerns mostly mathematicians, it has also relevance in several applications such as cryptography and reliability of modern Monte Carlo simulations. Therefore, in the beginning of this Chapter we will briefly discuss the current situation concerning the definition of randomness. An extensive discussion for infinite sequences is given by Knuth ([96] pp. 142-161). For more recent reviews, see *e.g.* van Lambalgen [100] and Compagner [30].

Due to the essential requirement of random behavior, *i.e.* the independence of consequtive events, random numbers should be produced by measuring a stochastic process such as radioactive decay or flipping a fair coin. When computer simulations are concerned, however, this is not very practical. Therefore, for practical purposes random numbers are generated by using completely deterministic rules, so called pseudorandom number generators. In order to compare these two approaches, we will consider the pros and cons of both methods and give reasons for preferring deterministic methods in computational applications. Furthermore, the desired properties of random number sources will also be considered. For good reviews on random number generation see *e.g.* Jansson ([84] pp. 22-68), Anderson [2], and L'Ecuyer [106] and for desired properties of random number sequences see James [82] and L'Ecuyer [106], for example. Also, due to its long and interesting history, the main steps in the development of generation methods of random numbers will also be considered. An interested reader is referred to Hull and Dobell [78].

2.1 Definitions of randomness

Despite its simple meaning to a layman, for mathematicians the definition of *randomness* has caused a lot of worry and despair. Still, in spite of all the work done, the definition of randomness is not unique but the discussion on this subject is still in progress. In the following, we will consider randomness mainly in terms of two formal concepts: complexity and unpredictability. Furthermore, since the formal approaches are not very useful for practical purposes, we will also consider two more practical approaches to define randomness.

For a layman, randomness is usually related to such practical ideas as irregularity and unpredictability. Regularity may be easily illustrated visually, like considering the distribution of (motivated) soldiers in a marching order; physicists may consider the spatial distribution of spins in the two-dimensional Ising model in its ground state at zero temperature. As an opposite notion, the distribution of trees in a forest in its natural state might seem irregular from the point of view of a layman. The notion of unpredictability is also clear. For example, if all the winning numbers of lottery are collected for several years and are then statistically analysed, one should not be able to predict following winning numbers better than by flipping a fair coin.

For mathematicians, on the other hand, defining randomness is not as simple. A major advancement for its understanding occurred in 1919, when von Mises introduced the notion of *Kollektiv*, standing for a single infinite sequence of random events [30]. In his work he also attempted to set a foundation for probability theory. For a recent review on von Mises' work, see Ref. [100]. Later in the 1960's, Kolmogorov [30], Martin-Löf [123], and Chaitin [22] described random sequences in terms of complexity. In the course of time, this discussion has led to the identification of randomness with polynomial-time unpredictability [107], a required property *e.g.* in cryptography [107]. Later, Compagner [29, 30] has proposed defining randomness in the case of binary sequences in terms of being uncorrelated.

The essence of the work of Kolmogorov, Martin-Löf [123], and Chaitin [22, 23, 24, 25] is based on the idea that the information embodied in a random piece of data cannot be reduced to a more compact form. For example, consider two sequences $\{x_i\}$ and $\{y_i\}$ of zeros and ones, $i = 1, \ldots, 30$:

$$\begin{aligned}\{x_i\} &= 100100100100100100100100100100\\ \{y_i\} &= 101101101111010101101110001000\end{aligned}$$

The sequence $\{x_i\}$ can be written in a more compact form as "repeat 100 ten times", whereas for the sequence $\{y_i\}$ such "compression" is not possible. Hence, for the random bit sequence $\{y_i\}$ the shortest way to write it is to give each element explicitly. The idea of *complexity* is based on this compactivity: it is defined as the length of the shortest program on a Turing machine (a universal but formal binary computer [188]) that produces the binary sequence [30]. As a result, the definition of randomness may be written as follows [30]: *"A binary sequence is random if its complexity is not smaller than its length."* Naturally, this approach is not restricted to binary sequences only.

In the course of time, the idea of complexity has lead to another formal way of defining randomness. The key notion in this approach is *unpredictability*, which due to its importance in many practical applications such as cryptology has recently received a lot of attention [12, 15, 16, 87, 98, 163]. This approach is based on the ideas of computational complexity [144] (for a review see *e.g.* L'Ecuyer [106]), where we consider the time it takes to guess the next element x_{n+1} of the sequence $\{x_i\}$, $i = 1, \ldots, n$, when the entire past is known. Randomness in the sense of unpredictability is then satisfied, if no polynomial time algorithm (in size of the sequence) can guess the next element significantly better than by flipping a fair coin [107]. As an example, consider the following three sequences of bits:

$$\begin{aligned}\{x_i\} &= 100100100100100100100100100100\\ \{y_i\} &= 101101101111010101101110001000\\ \{z_i\} &= 010011010111000100001111011001\end{aligned}$$

In the case of the sequence $\{x_i\}$, it takes just a moment to say that the 31^{st} element is most probably 1. For the sequence $\{y_i\}$ we are not able to predict the "correct" value, since this sequence has been produced by asking 30 different people to give arbitrarily one of two values: one or zero. The sequence $\{z_i\}$ also appears "random" at first sight, but further investigations may reveal that it has been generated by using a well-defined and simple formula [46]. Therefore, we may assume that $\{y_i\}$ is the only sequence which would satisfy randomness from the point of view of unpredictability.

Then, let us consider previous formal definitions from a practical point of view. Let us assume that we have a random number sequence, which must be tested against the hypothesis that it obeys (at least) one of these definitions; *i.e.* it is "random" or it is not. In the case of the notion of complexity, we note that since the number of possible programs increases exponentially with its length [199], and each program of progressively greater length must be tried in order to find the shortest one, and any one of them may run for an arbitrarily long time, testing this approach is clearly impractical. For unpredictability some tests have also been developed [12, 163], but besides being very tedious tasks to perform, to our knowledge no such tests have even been carried out. Therefore, when testing randomness is needed, more practical approaches to define randomness must be considered.

One such approach has been studied by Compagner and coworkers [29, 30, 31]. They considered finite binary sequences and proposed testing the values of all possible correlation coefficients of an ensemble of a given sequence. They then suggested that the essential requirement for randomness is *uncorrelatedness*; *i.e.* the disappearance of all the correlation coefficients [29]. Unfortunately, although this approach gives a well-defined way to test randomness, it is obvious that even this definition appears rather formidable for practical purposes.

Another more practical definition of randomness has been provided by Lehmer [2], whose definition essentially describes randomness from the empirical point of view: a random sequence is "*a vague notion embodying the idea of a sequence in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests, traditional with statisticians and depending somewhat on the uses to which the sequence is to be put.*" The underlying reason for the use of this definition is simple: since no unique practical recipe for testing a finite sequence of numbers has been given [100], various authors have developed different tests [12, 21, 33, 60, 64, 68, 71, 89, 92, 105, 116, 127, 137, 163, 177, 189, 200, 201] which probe some properties of the sequences. Then, if a random number sequence passes several well chosen tests, nothing of its "randomness" is proven but the confidence towards its properties increases. Moreover, if the chosen tests mimic the properties of an application in which the random number sequence will be used, this criterion may be good enough for practical purposes.

As a brief conclusion of this Section we may note that regardless of the application in which random numbers are used, their quality must be tested by some means. In this sense, although some test can be constructed for all the aforementioned definitions, the approach of Lehmer [2] to find support for random behavior by conducting several *practical* and well chosen tests is the most suitable one. Therefore, from now on, we will consider randomness from the point of view of Lehmer's definition. Instead of discussing tests, however, we will now briefly consider two different types of randomness which such test methods study. In Chapter 4, we will return to testing random number sequences and consider that subject in more detail.

2.2 Global and local randomness

Let us briefly consider two possible types of randomness in a random number sequence: global and local randomness. Since the classification between the two is more or less obscure, in this work we try to avoid confusion by defining them as follows. Consider a random number sequence $\{x_i\}$, $i = 1, 2, \ldots, N_T$ with $N_T \gg 1$. By means of some test for randomness, we study its properties over $n \leq N_T$ elements in this sequence. *Global* properties are studied when $n/N_T \sim \mathcal{O}(1)$. On the other hand, when $n/N_T \ll 1$, *local* properties are considered. This definition follows the ideas of Kendall and Babington-Smith, who were the first to introduce the concept of local properties of randomness in the 1930's [88, 89]. Moreover, since the test is not explicitly specified, this definition is general in the sense that it is not restricted to the Lehmer's definition for randomness [2] only.

There is one point which must be further specified: global randomness does not guarantee realization of local randomness. To illustrate differences between local and global properties, let us consider uniformity in a sequence $\{1, 2, , \ldots, 100\}$ of 100 integers. In global sense, this sequence is uniformly distributed since every number between one and 100 occurs exactly once, but when uniformity of the first ten numbers are studied, it certainly is not (in the range $1, 2, \ldots, 100$). Another interesting example concerns truly random sequences, by which we mean sequences where no correlations are present. In such (finite) sequences local randomness is not necessarily realized [96] whereas global randomness is.

2.3 Generation methods of randomness

Since random numbers have applications in many fields, a variety of different generation methods have been developed, each with their own set of advantages and disadvantages. Depending on the application, three types of methods are usually used: truly random, pseudorandom, and quasirandom sequences.

True randomness corresponds to ideally random behavior, meaning that in a truly random sequence of numbers no correlations are present, as already mentioned in the previous Section. Usually, one attempts to generate truly random numbers by measuring some (physical) stochastic process such as radioactive decay [79] or thermal noise [156]. For that reason, these sources are sometimes also called physical random number generators. Although this method allows generation of truly random numbers in principle, in practice, however, it is troubled with several problems such as bias due to human preference for certain digits [90], human errors in measurements [74, 89], or ignorance of the correlation time in the system measured [195]. Moreover, although there are methods to improve the properties of the output of physical random number generators *e.g.* by extracting most random bits of the output [27, 139], this method is still too slow for use in most computational applications. Hence, in practice truly random numbers are generated only for specific purposes such as lottery [185] and testing in algorithm development [44, 190]. Also, tables (see *e.g.* Refs. [91, 153, 168, 186] and [84] pp. 23-26) and some high capacity storage devices [156] have been made for further use of truly random numbers.

On modern computers, instead of using truly random numbers, several alternative methods have been developed for generating sequences which are *not* random but try to *imitate* random behavior for simulation purposes. These so called *pseudorandom* number sequences are produced by deterministic algorithms, implemented as pseudorandom number generators which usually rely on simple arithmetic operations. Despite the obvious conceptual conflict concerning deterministic algorithms in production of random number sequences, in many computational applications in which pseudorandom numbers are used, real problems seldom occur *provided that* some care is taken to ensure that the pseudorandom number generator is "good"; *i.e.* it has passed several well chosen tests. When such "good" pseudorandom number generators are used, many computational applications in which their output is utilized are fairly robust for subtle (but inevitable) deviations from randomness. The problems usually appear only when high precision simulations and some special algorithms [28, 49, 65, 66, 135, 166, 206] are used. However, a word of warning must be given here. In the course of time, hundreds (or even thousands) of pseudorandom number generators have been suggested, but the theoretical and empirical properties are *well* known only for few. As a matter of fact, generators known to be bad are certainly still used in several computing centers around the world. As James [82] has pointed out, there are "*many internal reports devoted to the revelation that the local 'official random number generator' is not random enough*". Hence, since pseudorandom number sequences are not truly random, most results based on the use of these sequences must be taken with a sceptical attitude.

Roughly speaking, *quasirandom* number sequences are defined as sequences of points, whose purpose is not to even imitate true randomness, but to estimate a given problem with as small an error as possible. A good example is numerical Monte Carlo integration [97]. There, one can estimate the integral for some particular function f by taking a sample of N points over the integration domain. Then, the average of f at those points, multiplied by the volume of the integration domain, gives an estimator for the integral. Now, if the points are taken from a truly random or pseudorandom sequence of numbers, the error will be proportional to $1/\sqrt{N}$ [97]. But one can do better if the sample points are spread throughout the integration domain "more evenly" [106] than in the case of truly random sequences. Such quasirandom number sequences are tailored to satisfy equidistribution criteria better than truly random and pseudorandom number sequences, resulting in an error proportional to $1/N$ [95]. However, despite the obvious importance of quasirandom sequences in some applications such as interpolation problems and the numerical solution of integral equations [140], in this work we will not consider them any further. For a review of quasirandom number methods see, *e.g.*, Niederreiter [140]. One practical implementation is given in Ref. [17].

For the purpose of completeness, let us also consider two other possible sources of randomness: transcendental numbers and chaos. For a long time decimals of *transcendental numbers* such as π and e have been considered random, and for that reason they have been calculated in various occasions [70, 155, 168]. In several studies, no excessive deviations from randomness have been found[1] [70, 130, 147], but they still share the same problem with arithmetic algorithms: they are deterministic. In this sense these numbers are also pseudorandom numbers. Moreover, since the cost for the computation of the decimals of π, for example, gets progressively higher when more decimals are needed, this method is clearly impractical. *Chaos*, on the other hand, can be effectively utilized as a source of randomness. This idea is meaningful, since chaos denotes a state of disorder and irregularity [164], which are common features of randomness as well. In practice, this connection has been made use of by developing pseudorandom number generators [112, 151] which are based on the theory of deterministic chaos [164]. In spite of its general interest, in this work we will not consider this idea any further.

In the remaining part of this work, only the terms of truly random and pseudorandom number sequences will be used. Moreover, when it is not essential to specify which sequence we are considering, we will speak of random number sequences in general.

As we have noticed, all three methods discussed here have their own place in the sea of applications. Truly random number sequences are needed as long as people want to gamble. Pseudorandom number sequences are useful in simulations where properties like speed and repeatability are essential. These and other desired properties of random number sequences in computational applications are discussed in more detail in the following Section. Finally, in applications in which uniformity is the most essential requirement, quasirandom number sequences are very useful. Therefore, let us quote Knuth [96]: "*We are forced to conclude that no sequence of 'random' numbers can be adequate for every application.*"

2.4 Desired properties of random number sequences

There are several properties which are required for or at least desired from random number generators used in modern computational applications [82, 106]. Since a computational approach of physical problems is our main interest, we will consider these properties in some detail. Furthermore, for the purpose of completeness, we will also

[1] It has been argued ([84] p. 38) that the decimals of e mimic truly random behavior too well, since in some tests the empirical distribution has been found to follow the theoretical one too closely. However, due to a small number of decimals studied no conclusions can be drawn.

consider realization of these properties in the case of physical sources of random numbers.

Naturally, the most important property is "*good randomness*". In the sense of Lehmer's definition, a pseudorandom number sequence should then pass a well chosen set of tests before extensive use in some particular application. As we mentioned above, physical random number generators do not necessarily produce truly random sequences, and therefore their output must be tested as well. The final criterion of goodness of a random number sequence, however, is determined by the application: if the random number sequence does not give the correct answer within error limits, it is not random enough. Problems arise, if no such check against analytic results is possible. Then, a simple check of randomness is to calculate one typical problem with few different random number generators, and compare their results.

Another often desired property of random number sequences is *a uniform distribution*. Although this is not essential, it is of considerable importance since most nonuniform distributions can be formed by using uniformly distributed random numbers between zero and one [18]. In this work, only uniform random number generators will be considered. For an introductory survey of computer generation of nonuniform distributions, see Ripley [157]. A library of FORTRAN routines for this purpose is given in Ref. [20].

For physical random number sequences, *a long period* is not a problem due to their aperiodic nature. For pseudorandom number sequences, however, this may be a problem since almost all pseudorandom number sequences are finite and reproducible. Aperiodic algorithms have also been suggested [46, 202], but they are troubled either with poor results in empirical tests [46] or with computational difficulties [202]. In the case of periodic pseudorandom number sequences, there are occurrencies in which the existing long-range correlations may be avoided by using only a small portion of the whole period [125]. Furthermore, in some applications such as parallel simulations, the output of the generator is split into numerous disjoint subsequences, which should be independent of each other. These facts emphasize the need for extremely long periods, so that only a small portion of the entire cycle needs be used in a single simulation. The longer the period, the better.

Also, depending on the application, the efficiency of random number generation may be important. Most pseudorandom number generators currently used in modern computational applications (in supercomputer environments) produce about 10^6 or more random numbers per second, and therefore in most cases the question of efficiency may be neglected. Its importance arises mainly in high precision Monte Carlo simulations in which huge amounts of pseudorandom numbers are used. In the case of physical sources of random numbers this question is not insignificant either. For example, in 1983 one such method [79] was able to generate about 600 physical random numbers in a second, when the storing time to a magnetic tape was included. Clearly this is too slow for use in real time. Of course, the time can be shortened if the machine is used on-line, but then the results of the calculation would no longer be repeatable. In computer simulations, however, *repeatability* is a crucial requirement mostly because of testing purposes: sometimes it may be necessary to repeat the simulation with exactly the same random number sequence. For pseudorandom number generators this requirement is usually fulfilled.

Finally, from the point of view of computer simulations, portability and parallelizability are often required. In general, *portability* means that when a random number sequence has been generated on some particular machine, there must be means to generate exactly the same sequence on other machines also. For pseudorandom number generators this is usually not a problem, if high level programming languages (like FORTRAN or C) and good programming techniques are used. In the case of physical random numbers, however, this property is fulfilled only if they are first stored and then transferred. When huge numbers of random numbers are needed, this certainly becomes impractical. *Parallelizability* has become important with the development of parallel computers. In practice, parallelizability means that pseudorandom number sequences generated on different processors are independent of each other. For certain classes of pseudorandom number generators this is possible [40], although problems with independence have also appeared [135]. In the case of physical sources of random numbers, consideration of parallelizability does not make much sense.

2.5 Brief history of random number generation

Random number generation and testing is a very widely studied subject. Prior to 1979 hundreds of articles had been published already (for an extensive biography, see Refs. [138, 162, 170, 171]), and ever since their number has been continuously increasing. A rapid increase of articles related to this subject occurred in the 1950's, hand in hand with the development of computers which needed practical and reliable sources of random numbers. However, before describing any further the development of arithmetic methods used in computers, we will consider the main steps leading up to the computer era.

Means for generating "randomness" have been known for a long time. For example, many games rely on the random nature of well shuffled cards and dice, when the possibility of cheating is excluded, of course. Roulette is another famous example, but unfortunately empirical results of Pearson [149] are against its random nature: in 1900, he analysed a series of runs of colour (red or black) in the throws of the roulette ball in Monte Carlo, and concluded that the odds are at least 10^{30} to 14.5 against such a series! Other methods have also been tried. In the 1920's, Tippett [186] collected about 40 000 random digits of census reports. Amazingly, although these digits were tested with several tests, no evidence for nonrandomness were found [38, 89, 137, 201]. Later, Kendall and Babington-Smith [89] attempted to construct a random sequence by selecting digits from the London Telephone Directory. However, it was found that the series was significantly biased, and therefore the London Telephone Directory was "*useless as a source of random digits*" [89]. On the other hand, Kermack and McKendrick [92] also tested certain telephone numbers which were found favourable to randomness. This result was commented by Kendall and Babington-Smith that "*Kermack and McKendrick are apparently dealing with a five-figure Scottish exchange*" [89].

Collecting data of census reports or throwing a dice is a slow process, however. Therefore, mechanized machines were built to produce numbers more efficiently. The first such work was done by Kendall and Babington-Smith [89, 90], who generated a table of 100 000 random digits by using a rapidly spinning disk divided into ten equal sections. This work is a remarkable token of patience, since all the 100 000 digits were collected by Babington-Smith only, with a rate of 1500 digits per hour on the average. Moreover, in four tests only about 5% of the digits were considered suspicious. Probably the most famous example of physical random number generation is the book of one million random digits and 100 000 normal deviates, published by the RAND Corporation in 1955 (see reviews in Refs. [136, 175]). These digits were generated by a kind of automatic "electronic roulette", and they have passed several tests after minor adjustments [84]. Later, special machines for purposes of lottery have been introduced. A classical example is ERNIE (Electronic Random Number Indicator Equipment) [185], which was used by the British General Post Office to pick the winners in the Premium Savings Bond Lottery. Naturally, the output of ERNIE was tested with several tests, which revealed no excessive deviations from randomness [185]. We are not aware if this machine is still in operation.

Recently, to our knowledge, only few studies of physical random number generators have been published. In the 1980's, Inoue *et al.* [79] described a method for generating random digits by measuring decay from radioactive nuclei. Richter [156] generated a vast amount of random digits by means of measuring thermal noise from a semiconductor device. These digits have been permanently stored on a computer disk, from which they can be transferred when needed.

Because of practical reasons, however, physical methods were quickly replaced by arithmetic methods as soon as the development of computers took place. Hence, the history of random number generation during the computational era is closely related to the development of computers.

Generally speaking, the Monte Carlo method denotes any method in which random numbers are used [81]. Hence it is very clear that the history of these two fields of science, pseudorandom number generation and Monte Carlo methods, are closely related to each other. As a matter of fact, since the development of the first electronic computer, ENIAC [62], many of the people who took part in its development and use, have had a strong influence in these fields of science. For example, N. Metropolis originally suggested [129] an obvious name for the Monte Carlo simulation method [131, 132], and with J. von Neumann studied randomness of the decimals of π and e [130] and developed the first algorithm for generating pseudorandom numbers: the so called midsquare method [78, 96]. In this method, an arbitrary n-digit integer is squared, creating a $2n$-digit product. A new integer is formed by extracting the middle n digits from the product. Although the properties of random numbers generated with the midsquare method are bad [96] (consider 50 with $n = 2$, for example), it was still used in the 1960's [54].

At the same time, another arithmetic method was suggested by Lehmer: the so called multiplicative linear congruential generator [109]. This method was already used on ENIAC, and due to its simplicity and well understood theoretical properties it is still widely used. Ever since, development of computers and Monte Carlo algorithms have set continuously increasing demands for the properties of pseudorandom number generators. As a result, several new classes of pseudorandom number generators have been suggested during the last few decades. These generators and their properties are the subject of next Chapter.

Chapter 3

Pseudorandom number generators

In this Chapter, we will consider the properties and use of pseudorandom number generators. First, most commonly used pseudorandom number generator algorithms will be presented, the emphasis being on their most important factors such as their structure and known properties. In this context, we will not pay attention to the particular values of the parameters in these algorithms but consider their properties in general. In addition, we shall also consider few other promising methods, which may prove useful in the near future. Reviews of current state of generation methods can be found *e.g.* in Anderson [2], James [82] (see also corrections in Refs. [83, 112]), and L'Ecuyer [104, 106].

In the next Section, we shall describe in more detail the generators (with specified values for the parameters), which have been chosen for the tests developed in this work (cf. Chapter 4). In order to better serve the physics community, we have tried to choose those algorithms which are widely used or otherwise seem promising, and which have been previously tested. Thus, we are able to summarize the current understanding of their properties, and since developing new, more accurate tests is one of our main objectives, we are then able to compare our new tests with tests developed by other authors.

Finally, even the best pseudorandom number generator algorithm can be defeated by an incorrect computer implementation, improper use, or bad initialization. These and other practical questions shall be studied in the last Section of this Chapter, where some tips for avoiding them shall also be given.

3.1 Classification of pseudorandom number generators

Most commonly used pseudorandom number generator algorithms are the *linear congruential method*, the *lagged Fibonacci method*, the *shift register method*, and *combination methods*. Other methods such as the *add-with-carry* and *subtract-with-borrow* generators and improvements of previous methods have also been proposed. In the following, main features of these methods will be given.

Linear congruential generators

Among the simplest algorithms are the *linear congruential generators* which use the integer recursion

$$X_{i+1} = (a\ X_i\ +\ c\ ) \bmod m, \tag{1}$$

in which the integers a, c and m are constants. It generates a sequence $X_1, X_2, \ldots$ of random integers between 0 and $m-1$ (or in the case $c = 0$, between 1 and $m-1$). Each X_i is then scaled into the interval [0,1). If the multiplier a is a primitive root modulo m (and $X_0 \neq 0$ in the case $c = 0$) and m is prime, the period of this generator is $m-1$. For other cases, the period length is given in Ref. [2], but then the low order bits are not random. Linear congruential generators can be classified into *mixed* ($c > 0$) and *multiplicitive* ($c = 0$) types, and are usually denoted by LCG(a, c, m) and MLCG(a, m), respectively.

Since the introduction of this algorithm by Lehmer [109], its properties have been studied in detail. Marsaglia [114] pointed out about 25 years ago that the random numbers in d dimensions lie on a relatively small number of parallel hyperplanes. This lattice structure was further studied by Ripley [158]. Boyar (see Ref. [15] and references therein) proved that LCG generators are efficiently (in polynomial time) predictable when the constants a, c, and m are unknown. Despite these deficiencies, in simulations LCG generators are widely used, and therefore a vast amount of theoretical work [33, 51, 52, 53, 126] has been done to weed out bad choices of these constants.

Fibonacci method

To increase the period of the linear congruential algorithm, it is natural to generalize it to the form

$$X_i = (a_1\ X_{i-1}\ + \cdots +\ a_r\ X_{i-r}) \bmod m, \tag{2}$$

in which $r > 1$ and $a_r \neq 0$. The period is the smallest positive integer λ for which

$$(X_0,\ \ldots,\ X_{r-1}) = (X_\lambda,\ \ldots,\ X_{\lambda+r-1}). \tag{3}$$

Since there are m^r possible r-tuples, the maximum period is $m^r - 1$. The use of $r = 2$, $a_1 = a_2 = 1$ leads to *the Fibonacci generator*

$$X_i = (X_{i-1}\ +\ X_{i-2}) \bmod m. \tag{4}$$

Since no multiplications are involved, this implementation has the advantage of being fast. Due to its poor properties, however, the Fibonacci generator is rarely used ([96] p. 26).

Lagged Fibonacci generators

A natural extension to the Fibonacci method is the *lagged Fibonacci* generator, which requires an initial set of elements $X_1, X_2, \ldots, X_r$ and then uses the integer recursion

$$X_i = (X_{i-r}\ \otimes\ X_{i-s}) \bmod m, \tag{5}$$

in which r and s are two integer lags satisfying $r > s$ and $\otimes$ is one of the binary operations $\{+, -, \times, \oplus\}$, $\oplus$ being an exclusive-or operation. The corresponding generators are designated by $\mathrm{LF}(r, s, m, \otimes)$. Usually the binary operation is addition or subtraction modulo 2^w, w being the word length (in bits). Then, the maximal period with suitable choices of r and s is $(2^r - 1)2^{w-1} \approx 2^{r+w-1}$ [116]. When multiplication (with odd integers) or exclusive-or are used, the period lengths are $(2^r - 1)2^{w-3} \approx 2^{r+w-3}$ and $2^r - 1$, respectively [116].

Excluding knowledge of the period length, theoretical properties of lagged Fibonacci generators (in terms of r and s) are not deeply understood, which makes their recommendation quite difficult. An exception are the lagged Fibonacci generators based on the exclusive-or operation, but they will be considered in the context of the generalized feedback shift register generators. For a relative goodness of the operations $\{+,-,\times,\oplus\}$, some empirical results are fortunately known. First, according to Marsaglia [116], exclusive-or should never be used. Results of Coddington's high precision Monte Carlo simulations [28] then suggest that multiplication is the best choice of the operations $\{+,-,\times,\oplus\}$, although it gives a shorter maximal period than addition and subtraction. For further details of LF generators see, *e.g.*, Refs. [116, 119].

Tausworthe generators

An alternative generator type is the *shift register generator*. Feedback shift register generators are also sometimes called Tausworthe generators [176], which are based on the theory of primitive trinomials of the form $x^p + x^q + 1$ [63]. Given such a primitive trinomial and p binary digits $x_0, x_1, x_2, \ldots, x_{p-1}$, a binary shift register sequence can be generated by the following recurrence relation:

$$x_i = x_{i-p} \ \oplus \ x_{i-q}, \tag{6}$$

in which $\oplus$ is the exclusive-or operator, which is equivalent to addition modulo 2. b-bit words can be formed from bits taken from this binary sequence as

$$W_j = x_{jb} \ x_{1+jb} \cdots x_{(b-1)+jb}, \tag{7}$$

in which $j = 0, 1, 2, \ldots$. The resulting binary words are then treated as random numbers. Such a sequence of random integers will have the maximum possible period of $2^p - 1$, if $x^p + x^q + 1$ is a primitive trinomial and if this trinomial divides $x^n - 1$ for $n = 2^p - 1$, but for no smaller n. These conditions can be met by choosing p to be a Mersenne prime, *i.e.* a prime number p for which $2^p - 1$ is also a prime. A list of Mersenne primes can be found *e.g.* in Refs. [75, 99, 203, 204]. Also, Tezuka [180] has shown that Tausworthe sequences form structures similar to lattice structure of linear congruential sequences [114]. Finally, let us just mention that results of some empirical tests suggest that generators based on small values of p should not be used [116], and the value of q should be small or close to $p/2$ [187].

Generalized feedback shift register generators

Generalization of Tausworthe sequences has been suggested by Lewis and Payne [111]. They formed b-bit words by introducing a delay between the words. The corresponding generator is called the *generalized feedback shift register* generator, denoted by GFSR$(p, q, \oplus)$. In a GFSR generator with two lags p and q the words W_i satisfy the recurrence relation

$$W_i = W_{i-p} \oplus W_{i-q}, \qquad p > q, \tag{8}$$

which clearly shows that the GFSR generator is a special case of lagged Fibonacci methods. Hence, with properly chosen lags [75, 99, 203, 204] maximal period length of $2^p - 1$ can be achieved for GFSR generators.

An important aspect of the GFSR algorithm concerns its initialization, in which p initial seeds are required: in the least fortunate case, if the j^{th} bit is zero in each of the first p integers of the sequence, it will remain zero throughout. Theoretically this question has been studied in Refs. [55, 56, 57, 59, 177, 178]. Based on theoretical studies [3, 5, 58, 63, 111, 141, 177, 179], GFSR generators have good properties in general, although the lattice structure observed for Tausworthe sequences is also a problem for GFSR generators [180]. Golomb ([63] pp. 78-79) has theoretically shown that the decimation of a maximum-length GFSR sequence by powers of two[1] results in equivalent sequences. Moreover, based on Ref. [31] the correlation length ξ of GFSR generators equals the lag parameter p. This results from the so called three-point (triple) correlations by which we mean (trivial) correlations of the form $x_i \oplus x_{i-p} \oplus x_{i-q} = 0$, in which x_i's denote single bits in the words W_i. Such correlations dominate the properties of GFSR generators (over their full period), as Ziff has shown in an unpublished work [206].

Decimation of GFSR sequences and primitive pentanomials

As mentioned above, the values of lags p and q determine the properties of GFSR generators (with two lags). In order to improve their properties there are two possibilities: one may increase the values of lags p and q, or one may increase their number.

[1] By such decimation we mean that only every k^{th} number (k being a power of two) of the sequence produced by Eq. (8) is used.

In an unpublished work [206], Ziff considered the latter possibility, developing GFSR generators with four lags:

$$W_i = W_{i-p} \oplus W_{i-q_1} \oplus W_{i-q_2} \oplus W_{i-q_3}, \tag{9}$$

in which $p > \max(q_1, q_2, q_3)$. Such a generator will be denoted by GFSR(p,q_1,q_2,q_3). The theory underlying the choice of lags p, q_1, q_2, and q_3 is given in Ref. [206], and is based on the decimation of Eq. (8) with some value of k which is not a power of two (such as $k = 3, 5, 7$). For that reason, this method is also called *k-decimation* of GFSR($p, q, \oplus$). Moreover, although the parameter p in Eqs. (8) and (9) is the same, the values of lags q_1, q_2, and q_3 are determined from the value of k.

The period of the k-decimated sequence is also $2^p - 1$. In addition to theoretical studies of Ziff [206], his empirical studies [206] indicate that this method really improves the properties of GFSR generators.

Despite the decimation of GFSR generators with two lags, these decimated sequences are still based on primitive trinomials. Another way to increase the number of lags is to develop generators based on *primitive pentanomials* of the form $x^p + x^{q_1} + x^{q_2} + x^{q_3} + 1$. Some values for the lags $p > q_1 > q_2 > q_3 > 0$ are proposed in Ref. [99], which allows construction of an another set of generators like Eq. (9). To our knowledge no studies for such generators have been carried out.

Combination methods

Given the inevitable dependencies that will exist in a pseudorandom number sequence, it seems natural that one should try to *shuffle* a sequence [80] or to *combine* separate sequences. An example of such approach is given by MacLaren and Marsaglia [113] who were apparently the first to suggest the idea of combining two generators together to produce a single sequence of random numbers. The essential idea is that if X_1, $X_2, \ldots$ and Y_1, Y_2, $\ldots$ are two random number sequences, then the sequence Z_1, Z_2, $\ldots$ defined by $Z_i = X_i \otimes Y_i$ will not only be "more uniform than either of the two sequences but will also be "more independent" [116]. The symbol $\otimes$ mentioned above is one of the binary operations $\{+, -, \times, \oplus\}$. Algorithms using this idea are often called *mixed* or *combination* generators.

Despite strong empirical support for combination methods [4, 35, 72, 103, 105, 116, 118, 120, 122, 183, 196], their theoretical understanding is still limited. Usually, the period of the combination is much longer than that of its single components [103, 120, 116], but only few theoretical studies suggest that combination really improves properties such as uniformity and "independence" [19, 116, 36]. However, as L'Ecuyer has pointed out [106], "statistical defects" are a common problem of many fast and simple generators such as LCG and GFSR generators, which when combined could yield an efficient generator with much better properties. Further theoretical studies of the combination method can be found in Refs. [108, 182, 183].

Other methods

Finally, let us consider few other generation methods of randomness, which may prove useful in the near future. The generators proposed by Marsaglia and Zaman [121] will be considered in some detail, whereas some other methods shall only be mentioned. Moreover, since these methods will be considered only here, theoretical and empirical test results for some particular generators will also be given.

Recently, Marsaglia and Zaman [121] have proposed the so called *add-with-carry* (AWC) and *subtract-with-borrow* (SWB) generators. These generators are basically lagged Fibonacci generators, with an extra addition of the carry bit (AWC) or subtraction of the borrow bit (SWB). Their main advantage is a very long period, the smallest for the generators suggested in Ref. [121] being approximately 10^{171}. Fairly soon, however, defects in these generators were found. Tezuka *et al.* [184] proved these generators to be equivalent to LCG's with large moduli, and therefore they have an unfavourable lattice structure [32, 184]. In addition, some results of SWB generators in empirical tests [105, 190] and high precision Monte Carlo simulations [49] do not support their general use in their current form. Recently, some light to this problem has been given by Lüscher [112], who has suggested a way to improve the properties of one particular SWB generator called RCARRY [82] by neglecting some of the generated random numbers. An actual implementation of this improved generator has been given by James [83].

Recently, Marsaglia has continued his previous studies and proposed a so called *multiply-with-carry* (MWC) generator [117]. Although knowledge of its properties is still incomplete, Marsaglia states that [117] "*all bits of the integers produced by this new method, whether leading or trailing, have passed extensive tests of randomness.*" The so called *inversive congruential generators* have also received considerable attention. For a review of their properties see Ref. [47]. For cryptologic purposes some nonlinear generators have also been proposed, the most known being the so called *BBS generator* [13]. This work has been extended *e.g.* in Refs. [9, 133, 154]. Finally, Matsumoto and Kurita [124] have proposed a variant of GFSR generators, known as *twisted GFSR*.

3.2 Tested pseudorandom number generators

In this Section, we shall describe in more detail the generators which have been chosen for the tests. We will focus on the main details, properties, and drawbacks of these generators, including some discussion on open problems. In the end, we will present a short summary of their relative goodness.

GGL

GGL is a uniform random number generator based on the linear congruential method [146]. The form of the generator is MLCG$(16807, 2^{31}-1)$ or

$$X_{i+1} = (16807\ X_i) \bmod (2^{31}-1), \tag{10}$$

and it generates pseudorandom numbers between 1 and $2^{31}-2$ (initial seed value $X_0 = 0$ is forbidden). This generator has been particularly popular [146]. It has seen extensive use in the IBM computers [209], and is also available in some commercial software packages such as subroutine RNUN in the IMSL library [210] and subroutine RAND in the MATLAB software [211].

MLCG$(16807, 2^{31}-1)$ generators are quite fast and have been argued to have "highly satisfactory" properties [110]. The main disadvantage of MLCG$(16807, 2^{31}-1)$ is its poor lattice structure in low dimensions ($d = 2, 3$) [2, 103], which explains very poor results in some recently developed empirical tests [105]. In other empirical tests these generators have performed well [60, 94, 102, 110, 189], and results of several bit level tests also support its good properties [4, 190]. Another drawback of MLCG$(16807, 2^{31}-1)$ is its period $2^{31}-2$ ($\approx 2 \times 10^9$ steps) [94], which can be exhausted fast on a modern high speed computer.

RAND

RAND uses the linear congruential method with a period of 2^{32} [207] to return successive pseudorandom numbers in the range from 0 to $2^{31}-1$. The generator is $\mathrm{LCG}(69069, 1, 2^{32})$ or

$$X_{i+1} = (69069\ X_i\ +\ 1) \bmod 2^{32}, \tag{11}$$

and our implementation of this algorithm is equivalent to the implementation by Convex Corp. on the Convex C3840 computer system [207], in which the sign bit is always set equal to zero.

The multiplier 69069 has been used in many generators, probably because it was strongly recommended in 1972 by Marsaglia [115], and is part of the famous SUPER-DUPER combination generator [2]. Known properties of $\mathrm{LCG}(69069, 1, 2^{32})$ do not support its use, however. Although its test results have been fairly good [102, 116], in higher dimensions ($d \geq 6$) its lattice structure is poor [2], and only its most significant bits have passed bit level tests [190]. The last property is due to the modulus, which is a power of two: the least significant bit has a period of two, the second least significant a period of four, and so on [40]. In addition, because of the poor bit level properties of $\mathrm{LCG}(69069, 1, 2^{32})$ both most and least significant bits of SUPER-DUPER are also correlated [4].

RAN3

RAN3 follows the algorithm of a lagged Fibonacci generator LF(55,24,m,$-$) or

$$X_i = (X_{i-55} - X_{i-24}) \bmod m. \tag{12}$$

This algorithm was originally Knuth's suggestion [96] for a portable routine but with an add operation instead of a subtraction. This was translated to a FORTRAN implementation by Press *et al.* [152] who chose $m = 10^9$ for RAN3. The period length of RAN3 is $2^{55}-1$ [96], and it requires an initializing sequence of 55 numbers. Based on results of some empirical tests [190], RAN3 has fairly good properties, although both its most and least significant bits are correlated [190]. Furthermore, in recent simulations of three-dimensional self-avoiding random walks a LF(55,24,m,+) gave incorrect results [66].

R*p*

In this work, generalized feedback shift register generators GFSR(p,q,$\oplus$) [111] with two lags p and q will be denoted by Rp. The value of q shall be given explicitly when necessary. These generators follow an algorithm given by Eq. (8), and suggested values for lags are given *e.g.* in Refs. [75, 99, 203, 204].

One particular example of GFSR generators is R250 [94], which generates 31-bit integers through a recurrence of the form GFSR(250,103,$\oplus$) or

$$X_i = X_{i-250} \ \oplus \ X_{i-103}. \tag{13}$$

Our implementation of this algorithm is done by Helin [73], and it needs $p = 250$ words of memory to store the 250 latest random numbers. A new term of the sequence can be generated by a simple bitwise exclusive-or ($\oplus$) operation. The period of R250 is $2^{250} - 1$ [94], and based on several empirical tests its properties are good [26, 72, 94, 190].

In recent high precision simulations, however, several GFSR generators including R250 have produced incorrect results when special simulation algorithms have been employed [28, 49, 65, 66, 166, 206]. It was suggested [49] that the most significant bits of R250 are correlated, but based on a recent study [190] at least the individual bits[2] of R250 pass many empirical tests on bit level. Grassberger has proposed triple correlations with a correlation length of "$\approx 10^2 - 10^3$" [65] or "≈ 400" [66], but even his studies have not been able to determine the correlation length precisely. Therefore, although we have good reason to assume that the underlying reason for poor performance of the GFSR generators in these high precision simulations is due to the three-point correlations with a correlation length $\xi = p$ (cf. Section 3.1), efficient test methods for confirming this are still lacking.

In this work, initialization of GFSR generators was performed with 32-bit integers produced by GGL. Other initialization methods including the one in Ref. [57] were also checked, but the results (given in Chapter 5) were unaffected.

[2]By individual bits we mean bits from a particular sequence i in a sequence of b-bit random words (i being one of $1, 2, \ldots, b$).

ZIFFp and PENTAp

In order to study the effect of k-decimation of GFSR$(p, q, \oplus)$ generators, we have implemented [173] the algorithm GFSR$(p, q_1, q_2, q_3, \oplus)$ [206] or

$$X_i = X_{i-p} \oplus X_{i-q_1} \oplus X_{i-q_2} \oplus X_{i-q_3}, \tag{14}$$

whose period is $2^p - 1$, $p > \max(q_1, q_2, q_3)$. Such a generator will de denoted by ZIFFp, and values of lags q_1, q_2, and q_3 [206] shall be given explicitly when necessary. One particular generator of this kind has been given in Ref. [205]. Excluding the results of Ziff [206], no test results of these generators have been available.

In addition, generators based on pritimive pentanomials have also been constructed. These generators also follow Eq. (14) but with a different choice of lags [99]. Such generators will be denoted by PENTAp, where values of other lags will again be given when necessary. The period of this generator is also $2^p - 1$.

The initialization of these generators was performed bit by bit by using GGL: all $b \times p$ bits in the initial seed vector were initialized by using the most significant bits of integers produced by GGL.

RANMAR

RANMAR is a combination of two different generators [82, 120]. The first is a lagged Fibonacci generator

$$X_i = \begin{cases} X_{i-97} - X_{i-33}, & \text{if } X_{i-97} \geq X_{i-33}; \\ X_{i-97} - X_{i-33} + 1, & \text{otherwise}, \end{cases} \tag{15}$$

in which only 24 most significant bits are used for single precision reals. The second part of the generator is a simple arithmetic sequence for the prime modulus $2^{24} - 3 = 16777213$. This sequence is defined as

$$Y_i = \begin{cases} Y_i - e, & \text{if } Y_i \geq e; \\ Y_i - e + f, & \text{otherwise}, \end{cases} \tag{16}$$

in which $e = 7654321/16777216$ and $f = 16777213/16777216$. The final random number Z_i is then produced by combining X_i and Y_i as

$$Z_i = \begin{cases} X_i - Y_i, & \text{if } X_i \geq Y_i; \\ X_i - Y_i + 1, & \text{otherwise}. \end{cases} \tag{17}$$

The total period of RANMAR is about 2^{144} [120]. A scalar version of the algorithm has been tested on bit level with good results [120]. We used the implementation by James [82] which is available in the Computer Physics Communications (CPC) software library, and has been recommended for a universal generator. This version has passed several empirical tests [72, 190], showing no apparent drawbacks.

We may conclude this Section by saying that, based on current knowledge of the tested pseudorandom number generators, RANMAR seems to have best properties in general. Despite its poor lattice structure in low dimensions, also GGL seems to have good properties, and it has been recommended as a "minimal standard" generator [146]. Rp generators have succeeded in several tests, but due to recent incorrect results in some high precision Monte Carlo simulations they cannot be recommended for general use. ZIFFp and PENTAp generators may bring some light to this problem. Finally, RAN3 and RAND seem to be the worst of the tested generators, showing correlations both on bit level and in other tests.

3.3 Pitfalls in the use of pseudorandom number generators

For several classes of pseudorandom number generators, theoretical knowledge is already so extensive that some generators within these classes can be recommended, if their known limitations are taken into account. For example, due to their apparent lattice structure linear congruential generators should not be used in lattice simulations without great care. However, good theoretical properties lose their significance, if incorrect practical procedures such as incorrect implementation of the algorithms are performed. In this Section, we will consider few such cases.

Starting from the pseudorandom number generator algorithm, the first mistake one can do is to *implement* it incorrectly for the computer. Practically this means that the output of the implementation (pseudorandom number generator) is not identical with the output of the underlying algorithm; *i.e.* the implementation is not exact. In addition to human error, reasons for such behavior may include many machine dependent features such as finite precision of real numbers, limited word size of the computer, and numerical accuracy of mathematical functions, but all these can be circumvented if care is taken. Furthermore, it would be desirable that the implemented routine were as fast as possible and performed correctly in many different environments, *i.e.* it would be *portable*. For further details see Refs. [61, 167] and references therein.

One example of an incorrect FORTRAN implementation is given in the Table below. The generator on the left corresponds to a correct 32-bit implementation of GGL,

```
REAL FUNCTION GGL(DS)              REAL FUNCTION GGL(DS)
DOUBLE PRECISION DS, D1, D2        REAL*4 DS, D1, D2
DATA D1/2147483648.D0/             DATA D1/2147483648./
DATA D2/2147483647.D0/             DATA D2/2147483647./
DS = DMOD(16807.D0*DS,D2)          DS = AMOD(16807.*DS,D2)
GGL = DS/D1                        GGL = DS/D1
RETURN                             RETURN
END                                END
```

which was introduced in Section 3.2. In this implementation, `DOUBLE PRECISION` reals are used to carry the state of the generator, and therefore the modulo operation is also performed in `DOUBLE PRECISION`. In the second implementation on the right, the generator has been speeded at the expense of accuracy by using `REAL*4` definitions, which improves the speed by a factor of 1.30 on DEC 3000 AXP. A "slight" drawback is a change in the period length, which decreases by a factor of about 3.4×10^7: in the incorrect implementation it is only 64.

This example emphasizes the sensitivity of the implementation procedure. Therefore, great care is needed both in the implementation of pseudorandom number generator algorithms and in the use of implemented algorithms on other machines. In the latter case, realization of portability must be checked.

A subject which is also closely related to machine dependent features concerns *optimization* during the compiling process. Although most compilers are usually reliable, full optimization should not be used blindly, however. A quick check of the output of the generator with several optimization levels is very wise. Moreover, since parallel computing has its own problems in pseudorandom number generation (related to independence of disjoint subsequences), great care in these matters is needed.

All pseudorandom number generators must be initialized before their use. For linear congruential generators *initialization* is not a problem, since the only restriction is set by the limits of their operation (possible values of output). For example, to initialize

GGL, zero should not be used as then the sequence will remain zero throughout. For generators with a larger recurrence length, more care must be taken to ensure that the seed values in the initial seed vector are independent of each other. GFSR generators have been argued to be particularly sensitive to initialization, and therefore special initialization methods have been proposed, a nice example being in Ref. [57]. In most cases, however, simpler methods are used. One practical method is to use a simple generator such as a good LCG to generate the initial seed vector. Marsaglia [116] has recommended constructing the seeds bit by bit using the least significant bit of LF(3,1,32707,−). Computer-specific features such as system date and time could also be used in initialization, but despite its usefulness this procedure has no theoretical support. Finally, using tables of truly random numbers might be a good idea, if very long seed vectors need not be used.

The effects of poor initialization have been studied by Altman [4] and Vattulainen *et al.* [190]. Altman observed that lagged Fibonacci generators are very sensitive to bit level correlations in the initializing sequence. Bit level properties of GFSR generators were studied by Vattulainen *et al.*, who found that when a GFSR generator was initialized with a correlated sequence, the correlations did not vanish by "warming up" the pseudorandom number generator but seemed to persist instead.

Unfortunately, in most applications in which pseudorandom number generators are used, their "random" behavior is taken for granted. Otherwise, it is hard to understand why most of these studies do not report the pseudorandom number generator algorithm used in the calculations. From the point of view of comparing results of separate studies, however, this information would be very valuable. Therefore, in any publication where results of Monte Carlo experiments are given, mentioning the used pseudorandom number generator is highly recommended.

Finally, let us mention the biggest mistake in use of pseudorandom number generators: a "random" choice of the generator. Before any generator is given considerable attention for possible use, one should have both theoretical and empirical support for their properties. Good theoretical properties form the starting point for empirical studies, which must also be performed to confirm that the implementation of this particular algorithm is correct, and to give more confidence for its properties. Unless these two criteria are satisfied, that particular generator should be avoided.

Chapter 4

Testing randomness

As was pointed out in Section 2.1, there must be some means to determine the "goodness" of random numbers. Traditionally this problem has been approached by probing properties of random number sequences by means of tests for randomness. Unfortunately, this practical approach is troubled with several problems, the most important being that no single (practical) test can verify realization of randomness in a random number sequence. For that reason numerous tests probing different manifestations of nonrandomness have been suggested, each having its own characteristic features. When combined together, such a more complete test program may give a better insight on how good the properties of the tested random number sequence really are. In this Chapter, we will first consider the main categories of test methods for randomness: empirical and theoretical tests, in addition to their own subclasses. Good reviews have been given *e.g.* by Knuth ([96] pp. 38-110) and L'Ecuyer [105].

Then, in the framework of this classification scheme we will present the main steps in the development of tests until now. Since the number of tests suggested by other authors is already numerous, we are forced not to pay much attention to the details but to give their main features instead. The only exceptions are the chi-square test, the Kolmogorov-Smirnov test, the d-tuple test, and the rank test, which will later be made use of.

The main topic of this Chapter concerns the new test methods, which have been developed in this work. Detailed descriptions of these tests will be given in Section 4.4, followed by presentation of few transformation methods for normally (Gaussian) distributed random variables which are needed in one of the new tests.

4.1 Classification of test methods

Recent practice in the classification of tests for randomness seems confusing, since a well-established and unique division between various test methods seems to be lacking. Instead of that, several schemes have been proposed. The classification between empirical and theoretical tests is a welcomed exception, since they are easily distinguished from each other. These classes are studied in more detail below. The notions of their subclasses, however, seem to vary from author to the other. The term "standard tests" is often used to mean the tests proposed in the book by Knuth ([96] pp. 59-73), followed by notions of *e.g.* some "more stringent tests" [116] and "universal tests"[127, 163]. Notions of "visual tests" [31, 111, 190] and "tests on bit level" [4, 116, 119, 190] are also widely used. Moreover, Vattulainen *et al.* have also added this confusion by introducing the term "physical testing" [190], denoting tests which are based on direct analogies to physical systems such as the Ising model [8] in statistical mechanics. In this work, we will try to avoid such confusion by neglecting the concepts "standard", "more stringent", "universal", and "physical". Instead, we will use an important notion: *application specific testing*. By this we mean tests, which mimic the most important properties of the application in which the random number sequence will be used. The advantage of conducting such tests is that they will yield the most relevant information of the properties of the random number sequence from the point of view of this particular application. In addition, the concepts of visual and bit level tests will be used throughout this work. The classification scheme following this approach is schematically shown in Fig. 1, which we will consider in more detail in the following.

The first step in our classification system is the division between studies of random bits and random words, random words consisting of several bits each. Since both random bits and random words may be interpreted as integers, they will be called random numbers in general. Physical sources and some algorithms such as the Tausworthe method generate random bits, whereas most pseudorandom number generators produce random words. The main reason for this classification are some applications such as combinatorics [116] and cryptography [127], in which good properties of individual random bits are required. In most applications, however, random words are usually used.

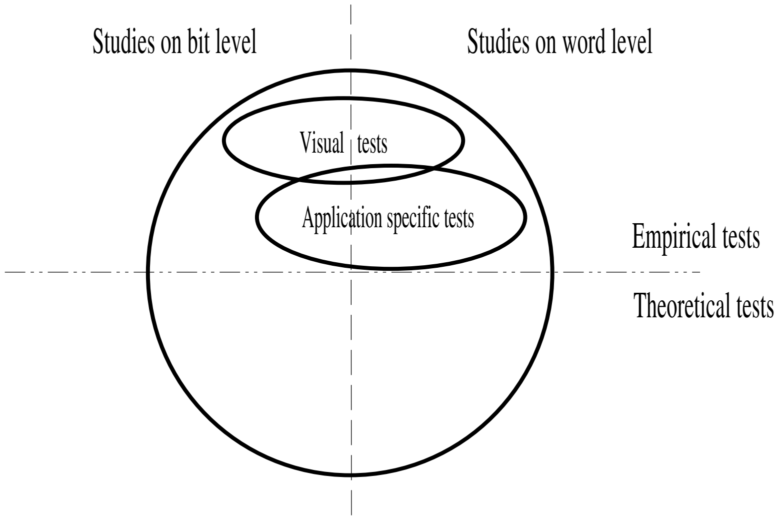

Figure 1: The schematic illustration of the classification scheme of tests for randomness discussed in this work. The set of all the tests is denoted by a circle, whose separate parts denote different subclasses.

The most traditional distinction between test methods for randomness has been done between empirical and theoretical tests, which can usually be designed to study properties of random bits as well as random words. In general, *empirical tests* are designed to study any manifestations of nonrandomness such as regular patterns in visual images or deviations from the desired distribution. Since they probe the output of the generation method regardless of its nature (physical or algorithmic) and not the generation method itself, empirical tests are complementary to *theoretical tests*, which study characteristics of the arithmetic method itself by using number theoretic methods. Common characteristics for theoretical tests are *e.g.* the period length, distribution, and autocorrelation properties. Another difference between these two test methods is the type of randomness they study. Theoretical tests are global in the sense that almost without exception they study pseudorandom number generator algorithms over their entire cycle. Empirical tests, on the other hand, can be designed to study both local and global properties (cf. Section 2.2) of random number sequences. Furthermore, while empirical tests may study any random number sequence regardless of its source, theoretical tests are further restricted in the sense that for each class of pseudorandom number generator algorithms (such as LCG and GFSR generators) their own specific set of theoretical tests must be constructed.

In principle, there is no compelling reason for classifying the test methods any further. For our purposes, however, two relevant concepts will be further discussed: application specific and visual tests. Since neither of them concerns theoretical tests in particular, they will be discussed from the point of view of empirical tests only.

The key concept in further classification is the application in which the random number sequence will be used. Therefore, let us study the sequence based on the assumption that the application is already known. By means of this approach, we may introduce the concept of *application specific testing*, whose main feature is that such tests mimic the essential features of the application, sometimes even being identical to it. For example, as the exact solution of the two-dimensional Ising model is known [8, 48, 143], for simulations where this model is studied, it can be used as a test for randomness as well. In most cases, however, exact results are not known. Then, one alternative approach is to measure some relevant quantities of the application with several different random number generators, and compare their results with each other. Generators whose results clearly deviate from the general trend should then be avoided, especially if results of some generators known to be good follow this trend. Another alternative is to construct new tests in such a way that only the most important properties of the application are included. An example of the latter are numerical integrations in which uniform distribution (in the desired dimension) is the most important property of random number sequences, and therefore an extensive uniformity test would serve as an application specific test for this purpose. Then, from the point of view of application specific testing, all remaining empirical tests (which are not application specific given a particular application) probe properties that are less significant for this particular application. Using the previous example, for numerical integration bit level studies of the least significant bits would belong to this category.

The importance of application specific testing cannot be overestimated: since the number of empirical tests that can be conceived is practically unlimited and still only few tests are usually applied to any sequence, the chosen tests should include many which mimic the crucial properties of the application as well as possible. Therefore, effective application specific tests are clearly needed to reveal subtle correlations in random number sequences. This is emphasized by recent observations in some high accuracy Monte Carlo simulations in which biased results with some pseudorandom number generators have been found [28, 49, 65, 66, 166].

Finally, let us consider one more category of tests. *Visual tests* are used to test spatial correlations between random numbers by visual means. For example, when a sequence of random bits is put on a two-dimensional lattice, short range correlations are easily observed. Therefore, besides being very useful for illustrative purposes, visual tests offer a possibility to develop more quantitative tests through interpretation of the visualized configurations as representations of physical systems, such as the Ising model. This has been made use of in the cluster test, which will be explained in Section 4.5. Furthermore, it is clear that in some cases visual tests may correspond to application specific testing also.

4.2 Brief review of previous work

In this Section, we will review the development of theoretical and empirical tests until now. Our purpose is not to give a thorough summary but to concentrate on giving a general idea of their characteristic features, the emphasis being on empirical testing. Since the number of empirical tests is vast, we will first consider them from a historical point of view, proceeding from the beginning of the 20^{th} century to the 1960's. Then, more recent developments will be discussed, followed by discussion on recent application specific tests.

Theoretical tests

As we noticed in Section 3.1, there are several commonly used classes of pseudorandom number generators, each with their own set of parameters. Since these generators are based on deterministic algorithms, there must be means to analyse their behavior theoretically. For some classes of generators such theoretical tests have been designed [33, 96, 106, 142, 177].

The purpose of all these tests is to analyse the goodness of an algorithm in terms of its parameters, based on some chosen measure. Several measures such as discrepancy [106], serial correlations [96], lattice structure [114], Walsh functions [177] and period length [96] have been suggested. In the following, we will concentrate on one particular and illustrative example of theoretical testing: the so called spectral test. For more details on this subject see Knuth ([96] pp. 75-110) and L'Ecuyer [106].

The main drawback of LCG generators was first observed in 1967 by Coveyou and MacPherson [33]. They found that LCG generators exhibit a lattice structure in the sense that the points generated by LCG generators lie on a set of equidistant parallel hyperplanes. One example of a such structure is shown on the left in Fig. 2, in which 5 000 2-tuples (consequtive pairs (x_{2i}, x_{2i+1}) of random numbers, $i = 0, 1, 2, \ldots$) of GGL are plotted. Coveyou and MacPherson also gave an algorithm for computing

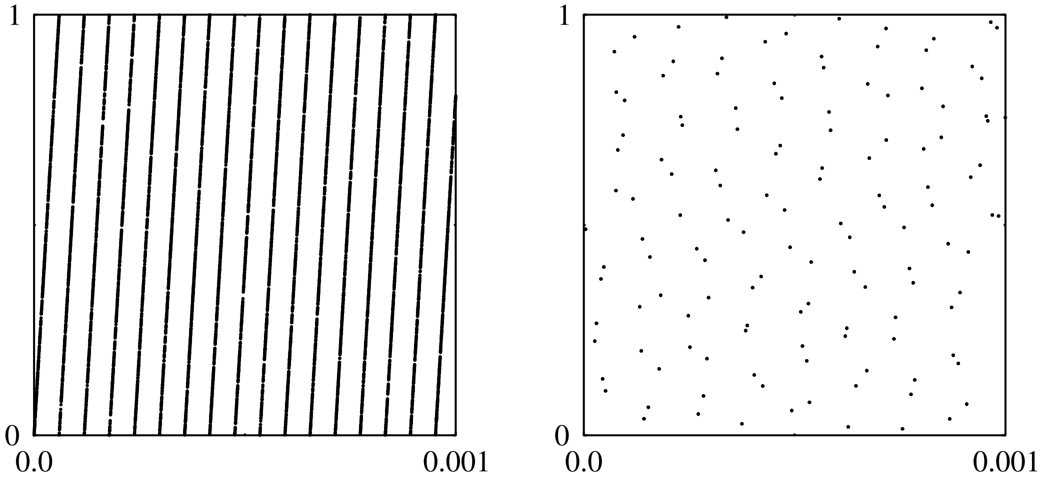

Figure 2: On the left we have 5 000 2-tuples of GGL, and on the right 1000 2-tuples generated by GFSR(17,3,$\oplus$). The horizontal part has been expanded for illustration purposes. Lattice structure in both cases is clearly observed.

the distance l_d between such parallel hyperplanes in $d \geq 2$ dimensions and called it *the spectral test*. In this test, the shorter the distance l_d between the hyperplanes the better the generator. Later these calculations were performed more explicitly by Marsaglia [114], and an algorithm for the computation of l_d was proposed by Knuth ([96] pp. 98-101). For higher dimensions ($d > 10$), a more efficient algorithm has been proposed by Fincke and Pohst [50].

When good LCG generators are being searched for, the number of possible combinations of parameters a, b, and m is practically unlimited. For that reason theoretical support for their choice given by the spectral test is invaluable, and therefore it has been used on many occasions [51, 52, 53] to weed out bad choices. Since similar tests for other classes of generators have not been constructed, the spectral test has remained the most significant single achievement in theoretical testing of pseudorandom number generator algorithms.

Recently, similar lattice structures have been observed for other classes of generators also. The AWC and SWB generators described in Section 3.1 are equivalent to LCG generators, and therefore also have structural properties [32] that can be analysed by studying the lattice structure of their LCG representations [184]. Furthermore, as Tezuka [180] has pointed out, Tausworthe and GFSR generators also exhibit certain lattice structure, as we may notice on the right in Fig. 2.

Empirical tests

Apparently the first empirical tests for randomness were the chi-square (χ^2) and the Kolmogorov-Smirnov (KS) tests, which were introduced by Pearson in 1900 [149] and by Kolmogorov originally in 1933 [96], in respective order. Strictly speaking, these two tests are *methods* rather than tests, since very often they are not used as such but utilized by other empirical tests. In this work, however, we will follow the standard practice and call them tests. Since they are also used in connection with the tests developed in this work, a more detailed description of the chi-square and KS tests will be given in Section 4.3.

Other well-known tests for randomness were proposed in 1938 by Kendall and Babington-Smith [89]. These tests for random digits were the frequency test, the serial test, the poker test, and the gap test. The frequency test is based on the idea that all the numbers in a random number sequence should occur an approximately equal number of times. In the serial test we count the number of times that the pairs of successive random numbers (X_i,X_{i+1}) occur, and then calculate their deviation from the uniform distribution. A correction to the proposed version of the serial test has later been given by Good [64]. These two tests are still widely used, and should be included in any reasonable test bench. The poker test is based on an analogue of the card game having the same name, and the gap test examines the length of "gaps" between occurrencies of X_i in a certain range. Later Gruenberger [69] described a computational approach for some of the tests proposed by Kendall and Babington-Smith.

At about the same time there appeared several other works related to empirical testing. Kermack and McKendrick [92, 93] proposed the "run" test, in which segments of successive digits with increasing or decreasing length are examined. Despite some errors in the original proposition, the run test has later been corrected and used extensively [96]. Nair [137] described a test based on some ideas of Pearson [150], and Yule [201] studied normally distributed random variables which were formed by adding five uniformly distributed random variables together. The next flood of new empirical tests followed the development of ENIAC. In 1951 the d^2 test was proposed by Gruenberger and Mark [71], in 1955 the coupon collector's test by Greenwood [68], and in 1961 the partition test by Butcher [21]. Many of these and several other tests are described in more detail by Knuth ([96] pp. 38-73), and practical implementations of several such tests have been given by Dudewitz and Ralley [39].

Development of pseudorandom number generators has lead to a situation where many modern generators pass most of the aforementioned tests. Therefore, new tests have been designed to detect the deficiencies of many poor generators. This development occurred mostly in the 1980's. A well-known example of this development is Marsaglia's [116] test bench DIEHARD, which contains a set of eight new tests. These tests have been applied in several studies with great success [4, 105, 116, 190], in the sense that many generators known to pass other tests have failed them. In addition, L'Ecuyer [105] has described the nearest pair test, which especially LCG generators tend to fail. For cryptologic purposes Blum and Micali [12] have proposed the next-bit-test, which measures the ability to predict the next bit from the preceeding ones. This work was extended by Schrift and Shamir [163]. Other tests have been proposed *e.g.* by Yuen [200], Garpman and Randrup [60], Ugrin-Šparak [189], and Maurer [127].

Although several of the tests mentioned above can be performed to test random bits as well, specific tests for this purpose have also been proposed. In DIEHARD, two tests are well suited for this purpose. In the rank test [119], samples of the rank of a random binary matrix are taken, and their probability distribution function is then compared with the expected behavior. The d-tuple test (also known as the overlapping m-tuple test) [4, 116, 190] is basically a run test performed on strings of bits in random numbers. These tests are desribed in more detail in Section 4.4.

A rather different way of testing spatial correlations between random numbers is possible by using direct visualization. This can most easily be done in two dimensions by plotting pairs of random numbers on a plane like in Fig. 2, or visualizing the bits of binary numbers [31, 190], as in Fig. 3. In Fig. 3, the development of irregularity of

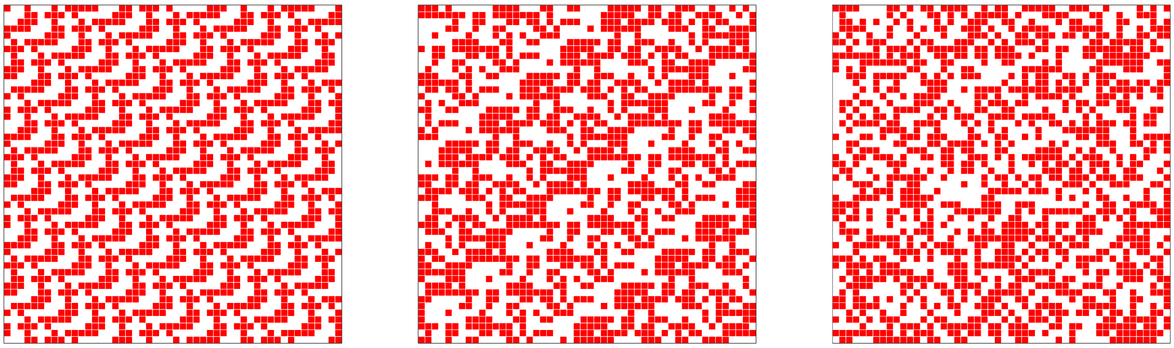

Figure 3: Bits of random numbers generated by RAND, as put on the lattice of size 50×50 one bit after another. Black squares denote ones and white squares zeros. Starting from the left, the figures correspond to the 27^{th}, 24^{th}, and 21^{st} bits of successive random numbers, bit number one denoting the most significant bit.

individual bits is easily seen. Other methods for visual testing have been given *e.g.* in Refs. [67, 111, 160, 190].

Application specific tests

The development of application specific testing started more or less in the 1980's, when several papers on physical simulation models as tests for randomness appeared. Above all, the Ising model has been the most popular in this respect. First, Kirkpatrick and Stoll [94] studied pseudorandom number generators by means of the two-dimensional Ising model with the standard Metropolis scheme, followed by other groups [6, 76, 77, 85, 145] who utilized the three-dimensional Ising model. Other Monte Carlo checks of the quality of pseudorandom number generators include the ϕ^4 model [134], the two-dimensional Ising model with the Wolff algorithm [28, 49, 166], and non-biased [11, 206] and self-avoiding [65, 66] random walks. In addition to these physical models, Lewis and Payne [111] have proposed a simple "scattering simulation test", Landauer [101] has considered queueing systems, and Paulsen [148] has performed Monte Carlo time series simulations. Moreover, some of the tests in DIEHARD [116] such as the rank test and the parking lot test have also a definite application, their results having use *e.g.* in graph theory [116] and lattice simulations, respectively.

Although this brief list is certainly far from complete, it gives certain idea of the approach for application specific testing. Popularity of the Ising model in these tests is not surprising due to its extraordinary status in statistical mechanics. As a matter of fact, two of the new tests proposed in Section 4.5 are also very closely related to the Ising model.

4.3 Chi-square and Kolmogorov-Smirnov tests

In the following, brief descriptions of the chi-square test and the Kolmogorov-Smirnov test will be given. For further details see Refs. [96] pp. 39-56, [161], pp. 26-30, and [190]. Although these two tests can be used to study any random sample, in this work we will consider them from the computational point of view. Therefore, we assume that we have some random number generator, whose output we are about to study.

Chi-square test

One of the most common tests for randomness is *the chi-square test*, which is widely used in connection with other tests. In general, it is performed in the following way.

Let us take N independent observations $X_1, X_2, \ldots, X_N$ from some distribution generated by a random number generator, and assume that every observation X_i can fall into one of μ mutually exclusive categories with probability p_i, $i = 1, 2, \ldots, \mu$. Then let O_i be the number of observations that actually do fall into the category i. Also, let us form the test statistic

$$V = \sum_{i=1}^{\mu} \frac{(O_i - Np_i)^2}{Np_i} \tag{18}$$

with $\nu = \mu - 1$ degrees of freedom. Then consider a null hypothesis H_0 that the generator is good if V obeys the χ^2 distribution, and consider the *observed* descriptive level $\alpha = \mathrm{Prob}(\chi^2 \leq V \,|\, H_0)$, $\alpha \in [0, 1)$. When V (and thus α) is very small, the empirical distribution follows the theoretical one too smoothly in the statistical sense, and the null hypothesis H_0 must be rejected: the random number generator fails the test. On the other hand, if V (and α) is very large, the empirical distribution is too far from the theoretical one and again the generator fails. Choice of the failing criteria is up to the statistician, but usual choices are domains below $\alpha = 0.05$ and above $\alpha = 0.95$. For the purpose of illustration, in Fig. 4 we show the (cumulative) χ^2

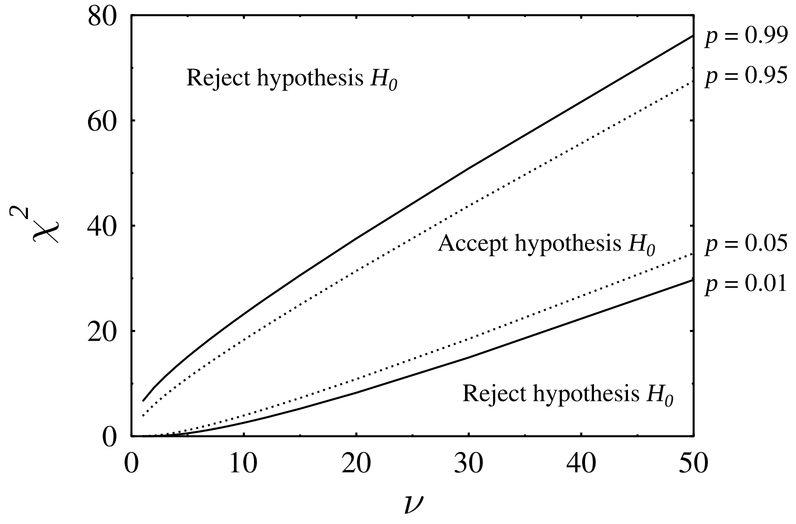

Figure 4: The χ^2 distribution with ν degrees of freedom with four percentage points p and over a range of ν. The acceptance and rejection regions of the two-sided chi-square tests at levels of significance 0.02 (solid lines) and 0.10 (dotted lines) are also shown.

distribution with ν degrees of freedom with four percentage points p and over a range of ν.

When the chi-square test is applied, the statistician has to decide the number of independent observations N. Parameter N must be large enough, since the statistic V in Eq. (18) obeys the χ^2 distribution *approximately*, the approximation being the better the higher the values of products Np_i is. As a rule of thumb, Knuth [96] recommends $Np_i \geq 5$ for all i.

Kolmogorov-Smirnov test

The disadvantage of the chi-square test is that it requires binning of data. This drawback is eliminated in the *the Kolmogorov-Smirnov* (KS) *test*, in which such binning is not needed.

In the KS test, we make N independent observations of some random quantity X and compare their empirical cumulative distribution function (cdf) $F_N(x)$ with the theoretical one $F(x)$ by computing their maximum deviations. When $F(x)$ is continuous, we may form the following test statistics:

$$K^{+} = \sqrt{N}\sup\{F_N(x) - F(x)\}; \tag{19}$$

$$K^{-} = \sqrt{N}\sup\{F(x) - F_N(x)\}. \tag{20}$$

K^+ measures the maximum deviation of $F_N(x)$ from $F(x)$ when $F_N(x) > F(x)$ and K^- measures the respective quantity for $F_N(x) < F(x)$. These test statistics should be distributed according to the KS distribution (with chosen N), which is found in standard statistical textbooks. Let S then denote the KS statistic. As in the chi-square test, a random number generator fails the KS test if under the null hypothesis H_0 (that K^+ and K^- obey the KS distribution) any of the observed descriptive levels $\delta^+ = \mathrm{Prob}(S \leq K^+ \,|\, H_0)$ or $\delta^- = \mathrm{Prob}(S \leq K^- \,|\, H_0)$ is "too small" or "too large". Again, usually chosen limits are 0.05 and 0.95, respectively. This kind of test in which the empirical distribution is compared *once* with the theoretical one is sometimes called a one-level test [105].

Unlike in the chi-square test, calculations of the KS test statistics do not necessarily involve any approximation. Therefore, the KS test can be reliably used with any value of N. However, if $F_N(x)$ does not follow $F(x)$ precisely, fairly large values of N would confirm this difference better than small ones. On the other hand, when a large number of observations are taken, locally nonrandom behavior will tend to average out [96]. This suggests using small values of N. This apparent contradiction is avoided by making a compromise: choose N fairly large and apply another KS test to the previous test statistics K^+ and K^- by means of repeating the test $M \gg 1$ times. When the empirical cdf of the test statistics K^+ and K^- is compared with the expected (approximate) distribution [96] $F_M(x) = 1 - \exp(-2x^2)(1 - \frac{2x}{3\sqrt{M}} + \mathcal{O}(1/N))$, $x \geq 0$, four new test variables K^{++}, K^{+-}, K^{-+}, and K^{--} are found. Then, the generator is considered to fail the test if any of the respective descriptive levels δ^{++}, δ^{+-}, δ^{-+}, and δ^{--} are again "too small" or "too large". This is called a *two-level* test [105], which tends to detect local correlations better than a one-level test [96].

4.4 d-tuple and rank tests

In this Section, we will present brief descriptions of the d-tuple and rank tests, which in this work are the first tests in which the aforementioned chi-square and KS tests are utilized. Both of these tests have been designed to study bit level properties of random numbers. For further details see Refs. [4, 116, 119, 190].

d-tuple test

The d-tuple test realized here is a modified version [4] of the original [116]. We have extended the test [4] further by improving its statistical accuracy by submitting the empirical distribution of the test statistics [4] to a Kolmogorov-Smirnov test. In the d-tuple test, we represent a random integer I_i, $i = 1, 2, \ldots, n$, as a binary sequence of b bits $\tilde{b}_{i,j}, (j = 1, \ldots, b)$:

$$\begin{aligned} I_1 &= \tilde{b}_{1,1}\tilde{b}_{1,2}\tilde{b}_{1,3}\cdots\tilde{b}_{1,b}, \\ I_2 &= \tilde{b}_{2,1}\tilde{b}_{2,2}\tilde{b}_{2,3}\cdots\tilde{b}_{2,b}, \\ &\vdots \\ I_n &= \tilde{b}_{n,1}\tilde{b}_{n,2}\tilde{b}_{n,3}\cdots\tilde{b}_{n,b}, \end{aligned} \tag{21}$$

with an obvious choice for the parameter being $b = 31$ (in testing RAN3, we used $b = 30$). Each of the binary sequences I_i is divided into subsequences of length l which can be used to form n new binary sequences $I_i' = \tilde{b}_{i,1}\tilde{b}_{i,2}\ldots\tilde{b}_{i,l}$. These sequences are then joined into one more binary sequence of length $d \times l$ in such a way that these final sequences $\bar{I}_i$ partially overlap:

$$\begin{aligned} \bar{I}_1 &= \tilde{b}_{1,1}\cdots\tilde{b}_{1,l}\tilde{b}_{2,1}\cdots\tilde{b}_{2,l}\cdots\tilde{b}_{d,1}\cdots\tilde{b}_{d,l}, \\ \bar{I}_2 &= \tilde{b}_{2,1}\cdots\tilde{b}_{2,l}\tilde{b}_{3,1}\cdots\tilde{b}_{3,l}\cdots\tilde{b}_{d+1,1}\cdots\tilde{b}_{d+1,l}, \\ &\vdots \\ \bar{I}_n &= \tilde{b}_{n,1}\cdots\tilde{b}_{n,l}\tilde{b}_{n+1,1}\cdots\tilde{b}_{n+1,l}\cdots\tilde{b}_{n+d-1,1}\cdots\tilde{b}_{n+d-1,l}. \end{aligned} \tag{22}$$

Each of these new integers falls within $\bar{I}_i \in [0, 2^{dl} - 1]$. In the test, the values $\bar{I}_i$ of the new random numbers are calculated as well as the number of respective occurrences. A statistic which follows the χ^2 distribution can be calculated although the subsequent sequences are correlated [4]. The chi-square test is repeated N times, and the results are finally subjected to the KS test.

Based on studies of Altman [4] and Vattulainen *et al.* [190], the d-tuple test detects bit level correlations effectively. In this work, however, our main purpose is not to use this test in testing random number generators but to compare its efficiency with the cluster test.

Rank test

The rank test is one of several tests proposed by Marsaglia [116], who developed it for studies of bit level properties of random numbers. However, since our purpose in this work is not to study generators by means of the rank test but to compare its efficiency (to find known correlations) with the cluster test, we have used a following version of the rank test. Using the notation of Eq. (21), we first studied the bits of the first w columns in the random number sequence I_i, $i = 1, 2, \cdots, n \times v$ by forming n binary matrices of size $(v \times w)$. Since consequtive samples were independent of each other (no overlapping between consequtive matrices), we then calculated n values for the rank R, whose probability to equal one of values $0, 1, 2, \ldots, \min(v, w)$ is [116, 119]

$$p_R = 2^{R(v+w-R)-vw} \prod_{i=0}^{R-1} \left[\frac{(1-2^{i-w})(1-2^{i-v})}{1-2^{i-R}} \right]. \tag{23}$$

The n values for the rank R were then subjected to the chi-square test. In a similar way we studied other sequences of columns starting from bits $2, 3, \ldots, b$, b being the number of bits in a random word. Finally, this procedure was repeated N times in order to perform b KS tests. Bit number i then failed, if the values for the resulting descriptive levels of all the columns $i - w + 1, i - w + 2, \cdots, i$ were smaller than 0.05 or larger than 0.95.

Based on some bit level studies [190], although less efficient than the d-tuple test, also the rank test seems to find correlations effectively.

4.5 Presentation of new tests

In the following, detailed descriptions of the tests developed in this work will be given. The first two, the cluster test and the autocorrelation test, are closely related to the two-dimensional Ising model [8]. These tests are respectively based on studies of the cluster size distribution in a random lattice and on calculations of the integrated autocorrelation times for certain physical quantities. Although we apply these tests to the Ising model, they can be generalized to other models and applications as well. For example, our version of the cluster test is developed for studies of random bits, but nothing restricts its use with random words as well. Moreover, the idea of using autocorrelation functions in testing of random numbers is by no means restricted to the Ising model.

The next two test methods are related to random walks. In the random walk test, we study random walks on a plane as a function of the walk length. The n-block test is based on the idea of renormalizing a sequence of uniformly distributed random numbers. Despite its simplicity, the latter test is especially effective in finding local correlations.

Finally, we present the condition number test which is based on studies of Gaussian distributed random matrices. Moreover, from a more general point of view tests on random matrices are a particularly charming method, since random matrices have applications in a wide area of science: nuclear physics [128], chaotic systems [34], computer image generation [37], and algorithm development [44], for example. For a recent review on random matrices, see Ref. [34].

4.5.1 The cluster test

There is a natural analogy between binary numbers and the Ising model, which can be made use of in constructing a *cluster test* in the following way. We take i^{th} bit from every successive number and put them on a two-dimensional square lattice of size L^2. By identifying ones and zeros with the "up" and "down" spins $\mathcal{S} = \pm 1$ of the Ising model, the resulting configuration — if truly random — should be one of the 2^{L^2} equally weighted configurations corresponding to infinite temperature. The easiest quantity that one can then compute from this analogy is the magnetization. However, a better measure of *spatial* correlations can be obtained if we study the distribution of connected spins, or clusters of size s on the lattice. The cluster size distribution $\langle \mathcal{C}_s \rangle$ is given by [174]

$$\langle \mathcal{C}_s \rangle = sp^s D_s(p), \tag{24}$$

in which $D_s(p)$'s are polynomials in $p = 1/2$. The normalization condition is $\sum_{s=1}^{\infty}\langle \mathcal{C}_s \rangle = 1$. Enumeration of the polynomials $D_s(p)$ has been done up to $s = 17$ [174], and they are listed in Appendix A.

The test procedure we have used is as follows. We first form a L^2 lattice as above and enumerate all the clusters in it [194] by using periodic boundary conditions in both directions ([11] pp. 26-28). For such a configuration we calculate the (unnormalized) average size of clusters within $s = 1, 2, \ldots, 17$, denoted as $S_{17}^{(k)}$. This procedure is repeated N times corresponding to configurational averaging, yielding $S_{17} = \sum_{k=1}^{N} S_{17}^{(k)}/N$. The theoretical counterpart of this quantity is given by $s_{17} = \sum_{s=1}^{17} s\langle \mathcal{C}_s \rangle$. We also compute the empirical standard deviation σ_{17} of the quantities $S_{17}^{(k)}$. For each i^{th} bit the test statistic chosen is:

$$g_i = \frac{S_{17} - s_{17}}{\sigma_{17}}. \tag{25}$$

Using this statistic, the tests were performed comparatively between several pseudorandom number generators, with results from GGL assumed to be independent variables. Comparison of other generators with GGL is justified since GGL has been shown to have excellent properties on bit level [4, 190], and as Fig. 5 shows, the distribution for $S_{17}^{(k)} - s_{17}$ is given by a Gaussian for GGL. Therefore, the mean value of g_i over all the 31 bits of GGL, denoted as g_{GGL} and the corresponding standard deviation σ_{GGL} were computed and the results for all other generators were compared with these values using

$$g_i' = \frac{|g_i - g_{GGL}|}{\sigma_{GGL}}. \tag{26}$$

The bit i in question failed the test if g_i' was consequtively greater than three in two separate tests.

We also considered other similar choices for the test parameters such as using the maximum value of g_i over all the 31 bits of GGL instead of g_{GGL}, and then performing the analysis as above. The results of this approach were consistent with Eq. (26) (results of bits 7 and 12 of RAND being the only exceptions).

4.5.2 Autocorrelation test

In *the autocorrelation test* we consider the autocorrelation function C_A for some physical quantity A. Then, we calculate the integrated autocorrelation time τ_A of C_A. Our

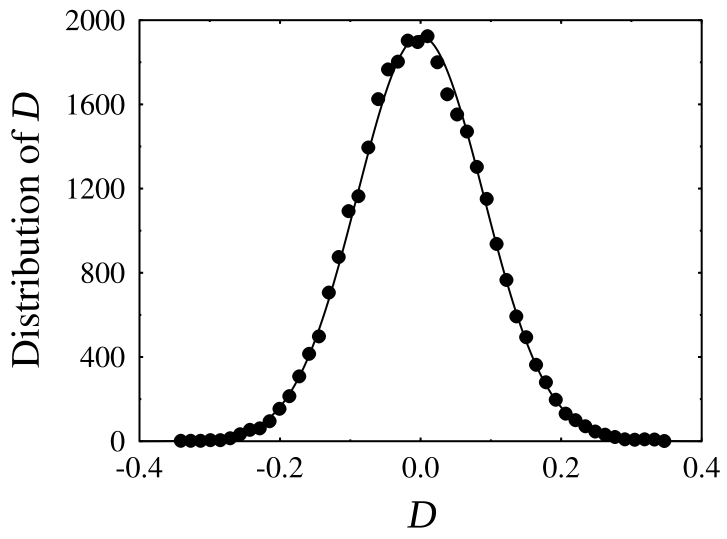

Figure 5: The (unnormalized) distribution (circles) of GGL for $D = S_{17}^{(k)} - s_{17}$ with 31 000 independent samples. Although this Figure is made for illustrated purposes only, we may still observe that the distribution clearly follows Gaussian behavior (indicated with a solid line).

approach follows the procedure given in Ref. [198], whose main details are given below.

The autocorrelation function is defined as

$$C_A(t) = \frac{\langle A(t_0)A(t_0+t)\rangle - \langle A(t_0)\rangle^2}{\langle A(t_0)^2\rangle - \langle A(t_0)\rangle^2}, \tag{27}$$

where t denotes time. In order to calculate an estimator $\tau_A(W)$ for the integrated autocorrelation time τ_A, a truncation window W is used:

$$\tau_A(W) = \frac{1}{2} + \sum_{t=1}^{W-1} C_A(t) + R(W), \tag{28}$$

with the remainder

$$R(W) = \frac{C_A(W)}{1-\gamma(W)} \tag{29}$$

and

$$\gamma(W) = \frac{C_A(W)}{C_A(W-1)}. \tag{30}$$

The convergence of $\tau_A(W)$ must be checked as a function of the window size W. Since noisy contributions from large separations appear after some value W_n, the estimate

τ_A is found by averaging $\tau_A(W)$ between W_c and W_n, in which $W_c < W_n$ denotes the value for which Eq. (28) first converges. An illustration of this procedure is given in Fig. 6. The error estimate for $\tau_A(W)$ is given in Ref. [198].

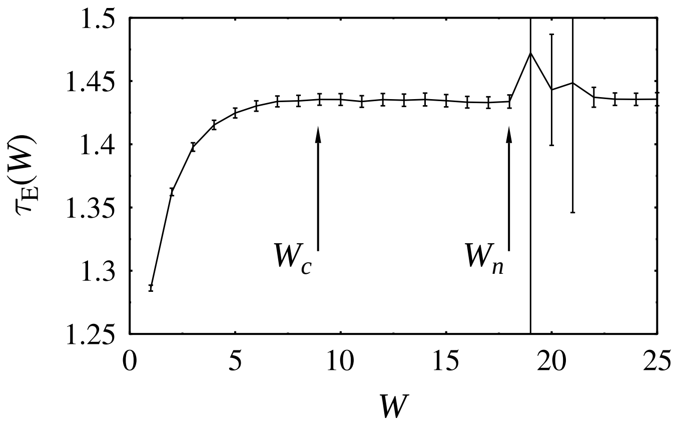

Figure 6: The integrated autocorrelation time τ_E of energy E, when RAND has been employed. The error starts to increase after $W_n = 18$. The error bars in the cases of $W = 19$ and $W = 21$ extend beyond the graph.

In this work, we considered the two-dimensional Ising model. The simulations were carried out on a square lattice with the Wolff algorithm [197] at the well known critical coupling $K_c = \frac{1}{2}\ln(1+\sqrt{2})$. The linear size of the system was $L = 16$. Our implementation of the single cluster search algorithm followed Ref. [194], and the measurements for the calculated quantities the energy E, the magnetic susceptibility $\hat{\chi}$ [198], and the (normalized) size of the flipped clusters c were separated by one single cluster update only. Then, by following the procedure given above we calculated the corresponding *integrated autocorrelation times* $\tilde{\tau}_E$, $\tilde{\tau}_{\hat{\chi}}$, and $\tilde{\tau}_c$ by first calculating their autocorrelation functions $C_E, C_{\hat{\chi}}$, and C_c. Finally, these estimates for the integrated autocorrelation times were scaled to the time unit of one Monte Carlo step; *i.e.* every spin on the lattice is updated once on the average. Therefore, the final results are $\tau_A = \tilde{\tau}_A\langle c\rangle$ [198], in which A is one of the quantities E, $\hat{\chi}$, and c.

In the case of the two-dimensional Ising model, the exact value for the energy $E = 1.45312$ [48] is known, which allows a comparison between results from different pseudorandom number generators. For other quantities, the test provides us with information on the relative performance of the random number generators. Here we assumed the results from GGL and RANMAR to be correct. This assumption is justified because of their results for the energy E, which in our simulations were correct within error limits.

4.5.3 Random walk test

Random walks have applications in a very wide area of science: linear polymer molecules may be modelled with self-avoiding random walks [169], (non-biased) random walks are used in studies of *e.g.* diffusion limited aggregation [193], and so on. Therefore, the idea of using random walks as a test for randomness seems a very interesting and useful idea.

In *the random walk test*, we consider random walks on a two-dimensional lattice, which is divided into four equal blocks, each of which has an equal probability to contain the random walker after a walk of length n. The test is performed N times, and the number of occurrences in each of the four blocks is compared to the expected value of $N/4$, using the chi-square test with three degrees of freedom. The generator fails if the χ^2 value exceeds 7.815 in at least two out of three independent runs. This should occur with a probability of only about 3/400.

For the purpose of completeness, let us mention that other random walk tests have been proposed by Binder and Heermann ([11] p. 76-80) and Ziff [206]. The former is based on the idea of studying the average end-to-end distance which should be a linear function of the walk length n. The test proposed by Ziff is based on random walks in a two-dimensional square lattice, where the random walker starts from one corner and heads towards the opposite one. At every step it may turn either left or right, unless it enters a previously visited site in which case it is forced to turn so as not to retrace its path. Therefore, eventually it hits one of the two opposite boundaries, which should occur with an equal probability.

4.5.4 n-block test

The n-block test is a simplified version of our random walk test, being basically a random walk test in one dimension. In this test we take a sequence $\{x_1, x_2, \ldots, x_n\}$ of uniformly distributed random numbers $0 \leq x_i < 1$, whose average $\bar{x}$ is calculated.

If $\bar{x} \geq 1/2$, we choose $y_i = 1$; otherwise $y_i = 0$. This is repeated N times. We then perform the chi-square test on variables y_i with one degree of freedom. Each test is repeated three times, and the generator fails the test with fixed n if at least two out of three χ^2 values exceed 3.841, which should occur with a probability of about 3/400.

In Ref. [66], Grassberger has proposed a somewhat unspecified "block" test to study the range of correlations for LF$(17, 5, +)$.

4.5.5 Condition number test

An interesting method for studies of pseudorandom number generators is by means of random matrices. We have constructed one test of this kind, concentrating on distributions of condition numbers in Gaussian distributed random matrices. This *condition number test* is based on the theoretical work of Edelman [41].

Consider a real $m \times 2$ matrix B with elements from a standard normal (Gaussian) distribution. We define a Wishart matrix $W = BB^T$, and calculate its eigenvalues λ_{max} and λ_{min} ($\lambda_{max} \geq \lambda_{min} \geq 0$) and the 2-norm condition number $\kappa = \sqrt{\lambda_{max}/\lambda_{min}}$ of B. The probability distribution function of κ for such a Gaussian matrix is [41]

$$f(\kappa) = (m-1)2^{m-1} \frac{\kappa^2 - 1}{(\kappa^2+1)^m} \; \kappa^{m-2}, \tag{31}$$

and its cumulative distribution function (cdf) is given by

$$\begin{aligned} F(b) &= \int_1^b f(\kappa) \; d\kappa \\ &= 1 - \left(\frac{2b}{1+b^2} \right)^{m-1}, \quad b \geq 1. \end{aligned} \tag{32}$$

In the condition number test we proceed as follows. First, by using a pseudorandom number generator we form a Gaussian distributed $m \times 2$ matrix B and calculate its condition number κ. This is repeated N times. Then, we compare the empirical cdf with the one given by Eq. (32) by using a Kolmogorov-Smirnov test. At this point, two test statistics K^+ and K^- are found. Then, the previous procedure is repeated M times, and another KS test is performed for the cdf's of test statistics K^+ and K^-. As a result, we find the final test variables K^{++}, K^{+-}, K^{-+}, and K^{--}. In this work, we considered the generator to fail the test if any of the respective descriptive levels δ^{++},

δ^{+-}, δ^{-+}, and δ^{--} were less than 0.05 or larger than 0.95. In other words, a failure occurred if the empirical cdf followed too closely or was too far from the theoretical one.

Other tests based on the properties of random matrices can also be constructed. For example, one possibility is to use the expected number of real eigenvalues in a Gaussian distributed random matrix [45]. However, our experience shows that this test is not very efficient[1]. Other ideas consist of using distributions of smallest eigenvalues [42] or of scaled condition numbers [43] as a basis for empirical tests. Finally, Marsaglia [116, 119] has proposed a test based on the calculation of the rank of a random matrix (cf. Section 4.4).

In testing Gaussian distributed random variables one important factor must be taken into account: since pseudorandom number generators usually produce uniformly distributed random variables, some method must be used to transform them into Gaussian distributed random variables. This raises an obvious question: are we studying the goodness of the transformation method or the goodness of uniformly distributed random variables? In order to clarify this question, we have studied few different transformation methods, and chosen the best one for further condition number tests on pseudorandom number generators. In the following Section, these transformation methods will be introduced.

4.6 Generation methods for Gaussian distributed random variables

In addition to uniformly distributed random numbers, variables from numerous other distributions are often needed. One of the most frequently used distributions is the normal (Gaussian) distribution, because of its importance in the field of statistics and probability theory [181], for example. In this Section, we consider some generation methods for Gaussian distributed random variables. The algorithms of these methods are given below, where we assume that all x_i, $i = 1, 2, \ldots$ are identically and

[1] We performed few bit level tests based on this idea. In the tests, we formed 31-bit integers of bits i ($i = 1, 2, \ldots, 31$) in the output of a pseudorandom number generator, and transformed them into Gaussian distributed random variables by means of the Box-Muller method [14]. With matrices of size 50×50 no correlations with any generator studied were found.

independently distributed random variables from the $U(0,1)$ distribution (the uniform distribution between zero and one); for other methods see Refs. [18] pp. 134-179, [96] pp. 114-133, and [161] pp. 38-107.

One of the most common techniques in generation of such variables X_i, $i = 1, 2$, is the Box-Muller algorithm [14]:

$\triangleright$ 1 `Generate` x_1 `and` x_2.
$\triangleright$ 2 `Let` $X_1 = \sqrt{-2\ln x_1}\cos(2\pi x_2)$ `and` $X_2 = \sqrt{-2\ln x_1}\sin(2\pi x_2)$.

Despite its popularity, this method has several drawbacks. First, since the Box-Muller method uses several multiplications, one trigonometric and logarithmic function, and a square root per a single random variable, it is very slow. In addition, the tail distribution differs markedly from the true distribution, when LCG or Tausworthe generators are used [181]. Therefore, in this work we have considered three other methods instead of the Box-Muller algorithm. The chosen methods include the so called Marsaglia's polar method [157], the ratio method [157], and a method based on the central limit theorem [159].

The algorithm of *Marsaglia's polar method* [157] is basically a Box-Muller algorithm, in which the sine and cosine computations have been eliminated by a rejection technique:

$\triangleright$ 1 `Generate` x_1, x_2, `and` $y_i = 2x_i - 1$, $i = 1, 2$.
$\triangleright$ 2 `Let` $z = y_1^2 + y_2^2$. `If` $z > 1$ `goto 1.`
$\triangleright$ 3 `Let` $C = \sqrt{(-2\ln z)/z}$, $X_1 = Cx_1$, `and` $X_2 = Cx_2$.

This method generates two Gaussian distributed random variables X_1 and X_2 at a time, and utilizes only one logarithm and a square root. The price it must pay is the efficiency, since only $\pi/4 \approx 0.786$ normal deviates are produced per one uniform random number.

One way to write *the ratio algorithm* for the normal distribution [157] is:

$\triangleright$ 1 `Generate` x_1, x_2, `and` $y = \sqrt{2/e}\,(2x_2 - 1)$.
$\triangleright$ 2 `Form` $X = y/x_1$, $z = X^2/4$.
$\triangleright$ 3 `If` $z > -\ln x_1$ `go to 1, otherwise return` X.

Although this method does not calculate the square root, it is slower than Marsaglia's method since it generates only one Gaussian distributed random variable X with an approximate efficiency of 0.366.

The third method considered in this work relies on the central limit theorem [159], in which n uniform random deviates x_i are used to form one Gaussian distributed random variable X:

▷ 1 Generate $x_1, x_2, \ldots, x_n$.
▷ 2 Form $X = (\sum_{i=1}^{n} x_i - n/2)\sqrt{12/n}$.

The main advantage of this method is its speed, if small values of n are used. Nevertheless, it is a well known fact that this method converges fairly slowly towards the Gaussian distribution, and therefore small values of n should be avoided. In the literature, however, this method is still often mentioned, usually with the choice $n = 12$ [18, 161].

In testing the generation methods for Gaussian distributed random variables we used a simplified version of the condition number test. First, by using RANMAR we formed a Gaussian distributed $m \times 2$ matrix B and calculated its condition number κ. This was repeated N times. Then, we compared the empirical cdf with the expected one (Eq. (32)) by using a Kolmogorov-Smirnov test, and found two test statistics K^+ and K^- and their respective descriptive levels δ^+ and δ^- ($\delta^+, \delta^- \in [0, 1)$). When this procedure was repeated M times, we calculated the total number G of descriptive levels which were less than 0.01 or larger than 0.99. Since the maximum value of G is $2M$, we finally obtained the test statistic $\epsilon = G/2M$. The tested generation method was considered good, if ϵ was 0.02 ± 0.01.

Chapter 5

Results

In this Chapter, results of the tests described in Section 4.5 will be given. Most Tables and Figures are given in the context of the text, although for purposes of clarity two Tables containing the exact values of the test statistics are given in the Appendices. Moreover, in addition to reporting the results and discussing some open problems our results clarify, the properties of the tests developed in this work will be discussed in some extent.

Before proceeding to the results, however, some technical facts must be given. In this test program, the initial seed values for pseudorandom number generators are chosen from the set $\{12345, 667790, 14159, 1415926535, 97766\}$. The tests have been performed on computers of DEC 3000 AXP series and Convex C3840. In some tests routines of the NAG library have been used, other code being written during this work. In the cases where test codes have been used on different machines, portability has been checked. Parallelization has not been utilized.

5.1 Cluster test

In this Section, we will study bit level correlations of some commonly used pseudo-random number generators by means of three tests: the d-tuple test [4, 116], the rank test [116, 119], and *the cluster test*. First, we study their efficiency in finding periodic correlations in random bit sequences, and by this means compare them. Then, we apply the cluster test to several generators and compare these results with previous results of the d-tuple and rank tests.

5.1.1 Studies on efficiency

The d-tuple and rank tests have been shown to find bit level correlations efficiently [190]. In order to determine the quantitative effectiveness of the tests, we first studied their ability to observe correlations inserted into the output of GGL, which has passed the bit level tests in Ref. [190]. The correlations were inserted periodically by setting the i^{th} bit ($i = 1, 2, \ldots, 31$) of every ξ^{th} number always equal to one. By systematically varying ξ, we could then find the maximum approximate distance ξ_c within which the d-tuple and rank tests can detect periodic correlations. The d-tuple test was repeated three times with parameters $d = l = 3$, $n = 5000$ and $N = 1000$. Accordingly, the rank test was repeated three times with parameters $v = w = 2$, $n = 1000$, and $N = 1000$. The results are shown in Table 1, where the parameter p gives the probability of observing correlations. Thus, the d-tuple test can detect periodic correlations up to

Results of the d-tuple test										
ξ	40	43	52	60	70	80	90	100	110	120
p	1.000	0.889	0.778	0.333	0.667	0.222	0.333	0.111	0.222	0.000
Results of the rank test										
ξ	40	45	50	60	70	80	90	100	110	120
p	1.000	.667	0.667	0.333	0.167	0.50	0.167	0.000	0.333	0.000

Table 1: Results of the d-tuple (above) and rank tests (below) with inserted correlations in the bits, with a period of ξ. The probability for the tests to observe correlations is denoted by p, which for the d-tuple and rank tests equals one up to $\xi_c \approx 43$.

$\xi_c \approx 43$ bits apart. The same test was repeated with $d = 9$ and $l = 1$ to consider single bits only, which gave $\xi_c \approx 50$. Similar systematic tests for the rank test also gave a result $\xi_c \approx 43$, but in the range $40 - 70$ for ξ the d-tuple test was found to be slightly more efficient. Hence, based on our results the rank test is slightly inferior to the d-tuple test.

The effectiveness of the cluster test was first scrutinized by inserting periodic correlations as in the previous cases. We chose $L = 200$, $N = 10\,000$ and the study was repeated for all values of $\xi = 1, 2, \ldots, L$. The results are shown in Table 2, where filled squares denote distances where correlations were detected. With this choice of

ξ	1	2	3	4	5	6	7	8	9	10
0+	■	■	■	■	■	■	■	■	■	■
10+	■	■	■	■	■	■	■	■	■	■
20+	■	■	■	■	■	■	■	■	■	■
30+	■	■	■	■	■	■	■	■	■	■
40+	■	■	■	■	■	■	■	■	■	■
50+	■	■	■	■	■	■	■	■	■	■
60+					■	■	■	■		
70+			■		■		■		■	■
80+			■		■	■				
90+							■		■	■
100+	■								■	
110+										■
120+										
130+			■	■						
140+										■
150+										
160+										
170+										
180+										
190+								■		■

Table 2: Results of the cluster test with correlations in the bits, with a period of ξ from one to 200. Black squares denote corresponding distances at which correlations were found as explained in the text.

parameters, the cluster test is able to find all periodic correlations up to $\xi_c \approx 60$. Moreover, due to the periodic boundary conditions (cf. Section 4.5.1) further correlations with larger values of ξ were also observed. This shows that the cluster test performs somewhat better than either the d-tuple or rank tests.

5.1.2 Testing generators using the cluster test

Next, we subjected each bit of the random number generators to the cluster test. It was repeated twice with parameters $L = 200$ and $N = 10\,000$. To confirm exclusion of finite-size effects additional tests with $L = 500$ were carried out. They gave consistent results with $L = 200$. Results are summarized in Table 3, where results of the previous d-tuple and rank tests from Ref. [190] have also been included. More detailed results

Random number generator	Failing bits			
	Cluster test	d-tuple test	Rank test	Equidistribution of bits
GGL	none	none	none	none
R250	none	none	none	none
R1279	none	none	none	none
RANMAR	$25-31$	$25-31$	$25-31$	$25-31$
RAN3	$1-4$, $25-30$	$1-5$, $25-30$	$1-5$, $26-30$	$1-11$, $24-30$
RAND	$7-31$	$13-31$	$18-31$	$22-31$

Table 3: Results of the cluster test ($k = 1$), where bit number one denotes the most significant bit. d-tuple and rank test results are from Ref. [190]. The last column denotes bits which fail in testing the equidistribution of ones.

of the cluster test are given in Appendices B and C. Although more powerful than the other methods, the cluster test still reveals no discernible correlations for either GGL, R250 or R1279. For RANMAR and RAN3, the cluster test gives results consistent with Ref. [190], but for RAND additional correlations are revealed in bits $7-12$, in contrast of passing the d-tuple test [190]. According to the results of RAND, the cluster test is very effective in locating periodic correlations, since the period of bit number 8 of RAND is as large as 2^{24} (see discussion of RAND on page 26).

For completeness, we also tested *the equidistribution of bits*. The bits failed the test if the deviation from the expected number of ones (*i.e.* $L^2/2$) consequtively exceeded three times the standard deviation of the binomial distribution with M samples. The test was repeated twice with $M = 4 \times 10^8$ and its results are also shown in Table 3. No correlations were found for GGL, R250, or R1279. Surprisingly, however, this rather simple test revealed that the first 11 bits of RAN3 fail (with standard deviations larger than 6.7) although only the first four or five bits fail in the other tests. On the other hand, for RAND only bits $22-31$ failed, which produced an exact $50-50$ distribution of zeros and ones.

In conclusion, the cluster test in the form presented here, is very well suited for detection of correlations on bit level, being especially effective for periodic correlations.

5.2 Autocorrelation test

The autocorrelation test was carried out with two sets of parameters. First, 10 000 Monte Carlo steps (MCS's) were performed to equilibrate the system starting from a random initial state, and then $N = 10^7$ samples were taken to test most of the generators once. One MCS denotes updating of each lattice site once on the average. In the second set, 100 000 MCS's were followed by $N = 10^8$ samples to test some generators more extensively. In the results considered here, the linear size of the system was $L = 16$.

A summary of the results in Table 4 shows that based on this test, the generators can be classified into two categories. First, let us consider results with $N = 10^7$ samples. For the energy $\langle E \rangle$, deviations from the exact result of $\langle E \rangle = 1.45312$ [48] for R31, R250, R521, and RAN3 are much larger than 3σ in which σ is the standard deviation [198]. In particular, the average size of flipped clusters $\langle c \rangle$ is very sensitive to correlations in random number sequences, since in the erroneous cases it is clearly biased. Most striking, however, is the behavior of the integrated autocorrelation times τ. For generators, which show no significant deviations in $\langle E \rangle$, $\langle \hat{\chi} \rangle$, or $\langle c \rangle$, results for the τ's agree well with each other. However, for R31 and R250, the integrated autocorrelation times show errors of about 8% compared with results of GGL and RANMAR. We thus propose these quantities as particularly sensitive measures of correlations in pseudorandom number sequences.

Another important point is the behavior of R31 compared with ZIFF31 and PENTA31. Though R31 clearly fails these autocorrelation tests, its 5-decimated sequence ZIFF31 and a generator PENTA31 based on a primitive pentanomial $x^{31}+x^{23}+x^{11}+x^9+1$ give correct results within error limits. This is further emphasized in studies with $N = 10^8$ samples, where R89 fails whereas ZIFF89 and PENTA89 give compatible results with RANMAR. Therefore, these results clearly indicate that k-decimation of GFSR generators with two lags and primitive pentanomials generate (in statistical sense) more

N	RNG	k	Lags	Physical quantities			Integrated autocorrelation times		
			q_i	$\langle E \rangle$	$\langle \hat{\chi} \rangle$	$\langle c \rangle$	τ_E	$\tau_{\hat{\chi}}$	τ_c
10^7	R31		3	**1.46774(7)**	**0.564(2)**	**0.5664(3)**	**1.233(4)**	**1.058(3)**	**0.507(2)**
	PENTA31		23,11,9	1.45300(10)	0.545(2)	0.5454(2)	1.440(6)	1.224(5)	0.627(7)
	ZIFF31		13,8,3	1.45313(10)	0.545(2)	0.5456(2)	1.440(6)	1.223(5)	0.624(4)
	R250		103	**1.45509(7)**	0.548(2)	**0.5474(2)**	**1.333(4)**	**1.143(4)**	**0.589(4)**
		3	103	1.45302(7)	0.545(2)	0.5452(2)	1.446(6)	1.226(5)	0.628(5)
	R521		168	**1.45379(7)**	0.546(2)	**0.5461(2)**	**1.384(5)**	**1.182(5)**	**0.604(4)**
	R1279		418	1.45312(7)	0.545(2)	0.5454(2)	1.426(5)	1.215(4)	0.622(3)
	R2281		1029	1.45311(7)	0.545(2)	0.5456(2)	1.439(5)	1.226(5)	0.627(5)
	R4423		2098	1.45303(7)	0.545(2)	0.5454(2)	1.441(5)	1.226(5)	0.624(4)
	R9689		4187	1.45313(7)	0.546(2)	0.5455(2)	1.444(5)	1.229(5)	0.625(4)
	R19937		9842	1.45294(7)	0.545(2)	0.5452(2)	1.434(5)	1.220(5)	0.624(4)
	R44497		21034	1.45292(7)	0.545(2)	0.5452(2)	1.434(5)	1.219(5)	0.622(2)
10^7	RAN3			**1.45254(7)**	0.545(2)	0.5446(2)	1.447(5)	1.231(5)	0.630(3)
	RAND			1.45304(10)	0.545(2)	0.5454(2)	1.434(5)	1.221(5)	0.620(2)
	GGL			1.45309(7)	0.545(2)	0.5454(2)	1.436(5)	1.221(5)	0.622(4)
	RANMAR			1.45303(7)	0.545(2)	0.5452(2)	1.443(5)	1.227(5)	0.624(4)
10^8	R89		38	**1.45720(3)**	**0.5503(6)**	0.55031(4)	**1.3041(14)**	**1.1134(12)**	**0.5662(9)**
	PENTA89		69,40,20	1.45300(3)	0.5454(6)	0.54532(4)	1.4454(16)	1.2278(15)	0.6260(12)
	ZIFF89		61,38,33	1.45305(3)	0.5454(6)	0.54539(4)	1.4398(19)	1.2243(17)	0.6241(15)
	RANMAR			1.45304(3)	0.5454(6)	0.54539(4)	1.4372(16)	1.2211(14)	0.6231(14)

Table 4: Results of simulations for the Ising model at criticality with the Wolff algorithm. The number of samples is denoted by N and the values of lags q_i are given where needed. The value of the decimation parameter k is one unless stated otherwise. The errors shown correspond to σ [198]; 1.447(5) denotes 1.447 ± 0.005 etc. . The most erroneous results are in boldface. See text for details.

reliable sequences than GFSR generators based on two lags only.

To compare our results with those of Refs. [49, 166] we also used the autocorrelation time test to further study the decimation of the output of R250, *i.e.* we took every k^{th} number of the pseudorandom number sequence. For $k = \{3, 5, 6, 7, 9, 10, 11, 12, 24, 48\}$, the correlations vanish in agreement with Ref. [49] ($k = 5$) and Ref. [166] ($k = 3, 5$). On the other hand, for $k = 2^m$ with $m = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$, the sequences fail. These findings agree with the theoretical result of Golomb ([63] pp. 78-79) who showed that the decimation of a maximum-length GFSR sequence by powers of two results in equivalent sequences.

Our results of the autocorrelation test are in agreement with observations made by various other authors [28, 49, 166], who have also studied the two-dimensional Ising model. Moreover, the errors in the average cluster sizes show that the origin of errors observed in these references lies in local correlations present in the cluster formation process of the Wolff algorithm. The main advantage of our approach is the use of integrated autocorrelation times as measures for nonrandomness, since in the erroneous cases the errors are of the order of several percents, whereas for other quantities such as the energy the error is much smaller. Due to the fact that this test is not restricted to the Ising model only, its use in other problems might be very fruitful.

5.3 Random walk test

Although errors in the average cluster sizes for some of the GFSR generators in the autocorrelation test suggest that the correlations are within the $\mathcal{O}(L^2)$ successive pseudorandom numbers, this result does not give a quantitative support for the argument that the correlation length equals the longer lag p (cf. Section 3.1). Therefore, to quantify the range of correlations empirically, we have performed *the random walk test.*

First, we studied a group of generators with the walk length $n = 1000$. These results are presented in Table 5, and they are in agreement with the autocorrelation test. No correlations for either GGL, RAND or RANMAR were observed. R250 and R521 pass the test with $k = 3$, but fail with $k = \{1, 2, 2^6\}$, whereas R1279 passes with all k's tested. The failure of RAN3 with $k = 1$ is consistent with results of previous tests [190] and the autocorrelation test. It is notable that all the failures in this test

RNG	k	q_i	χ^2 values	Result
R31		3	4094, 4105, 4300	FAIL
R250	1,2,64	103	396.4 – 539.8	FAIL
R521	1,2,64	168	49.01 – 79.16	FAIL
RAN3			40.01, 42.99, 44.53	FAIL
R250	3	103	0.301, 0.873, 1.024	PASS
R521	3	168	1.249, 1.352, 1.735	PASS
R1279	1,2,3,64	418	0.709 – 9.372	PASS
R4423		2098	0.621, 1.226, 8.217	PASS
PENTA31		23,11,9	0.685, 1.587, 2.363	PASS
ZIFF31		13,8,3	2.352, 2.367, 2.632	PASS
RAND			0.304, 0.640, 4.063	PASS
RAN3	2,3		0.033 – 6.877	PASS
GGL			0.090, 0.459, 1.981	PASS
RANMAR			0.293, 1.944, 3.187	PASS

Table 5: Results of the random walk test with $N = 10^6$ samples. The parameter k equals one unless stated otherwise. The fourth column indicates the χ^2 values in three independent tests, or the range of the χ^2 values when results with more than one value of k are included on the same line. The classification of the generators is based on the failing criterion given in Section 4.5.3: a generator fails the test if the χ^2 value exceeds 7.815 in at least two out of three independent runs. See text for details.

were very clear, since even the smallest χ^2 values exceeded 40. However, RAN3 passed the test when every second or third number was used.

The main difference between the failing generators R250 and R521 (with $k = 1$) and the successful ones R1279 and R4423 lies in the lag parameter p which is less than n for the former and larger than n for the latter. We studied this for various values of p with the random walk test by locating the approximate value n_c, above which the generators fail. The test was performed for R31, R250, R521 and R1279 with $N = 10^6$ samples. The results for n_c were 32 ± 1, 280 ± 5, 590 ± 5, and 1515 ± 5, respectively, in which the error estimate is the largest distance between samples close to n_c. For the purpose of illustration, in Fig. 7 we show an example of the χ^2 values for R31 and R250 as a function of the walk length n.

As another illustration of nonrandom behavior we consider the probability distribution functions (pdf's) for the final position of a random walker after n steps for GGL and R250. Since GGL passed the random walk test, we assume that its pdf is approximately correct. Then, we plot the difference between pdf's of R250 and GGL,

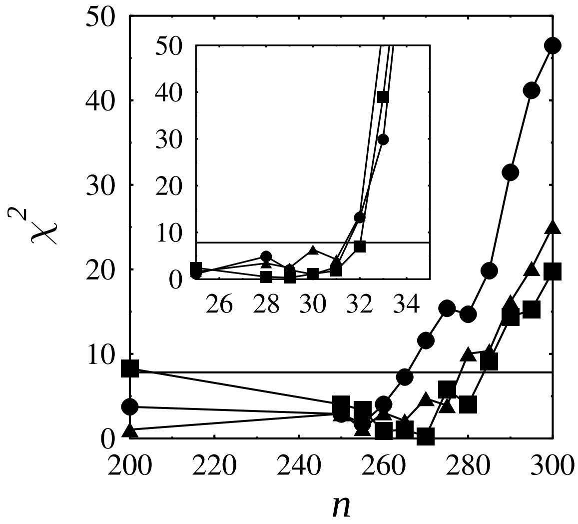

Figure 7: The χ^2 values for R31 (inner figure) and R250 ($k = 1$) in the random walk test as a function of walk length n, when $N = 10^6$ samples have been taken. Three independent runs in both cases are denoted by different symbols. The horizontal lines denote $\chi^2 = 7.815$.

as has been done in Fig. 8. We note that the probability distribution of R250 deviates significantly from that of GGL, since instead of uniform noise we notice two clear peaks. As a matter of fact, the deviation is very significant, since the relative error of R250 compared to GGL is about 6.5 % at these two peaks observed in the figure.

We also studied generators, which are based on primitive pentanomials or decimation of GFSR generators with two lags. As Table 5 indicates, PENTA31 and ZIFF31 pass the random walk test with $N = 10^6$ samples. In these cases, studies to locate n_c were inconclusive, since a small period of these generators did not allow testing them with more than 10^7 samples. Therefore, similar studies for PENTA89 and ZIFF89 with $N = 10^8$ samples were carried out, these results being summarized in Table 6. Although large fluctuations are still present, we may notice that both PENTA89 and ZIFF89 exhibit correlated results with $n_c \approx 95 - 200$. In order to quantify this result more precisely, we devised the n-block test, whose results we will consider next. Because the results of the random walk test and the n-block test are very closely related to each other, discussion of the random walk test will be given together with the n-block test at the end of the following Section.

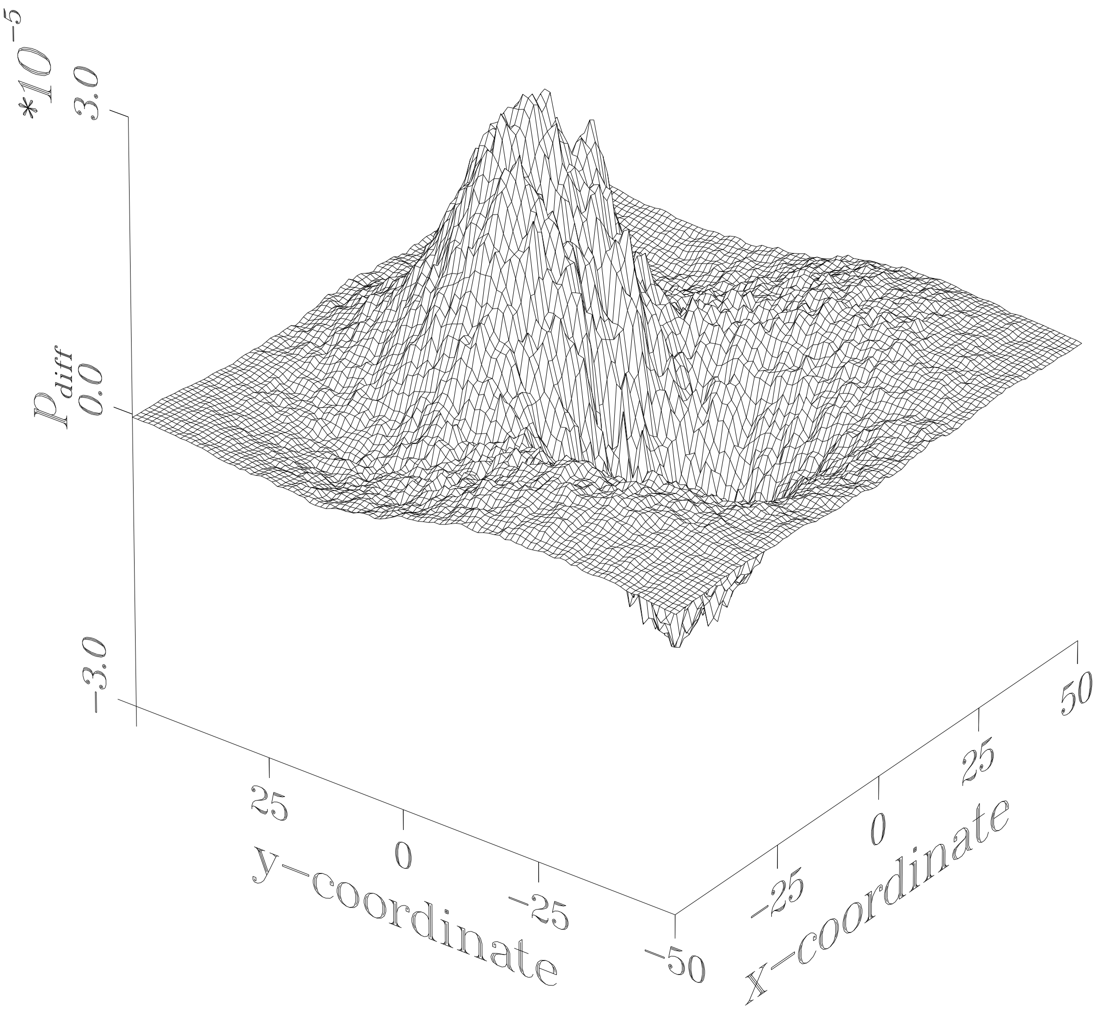

Figure 8: The deviation p_{diff} between (normalized) probability distribution functions of R250 ($k = 1$) and GGL as a function of lattice site, when the pdf of GGL was subtracted from the one of R250. The starting point of the random walk is (0,0). In this study we used $n = 1000$ and $N = 10^6$.

n	χ^2 for ZIFF89			χ^2 for PENTA89		
85	7.785	7.544	**8.131**	0.427	1.132	1.711
90	2.050	3.716	2.165	1.338	3.874	1.509
95	**10.93**	6.642	3.895	1.335	3.563	3.422
100	**9.130**	5.910	**8.770**	3.867	5.611	4.350
200	**8.632**	**15.76**	**13.61**	**25.02**	**18.48**	**17.56**
500	1.007	6.173	**9.822**	**34.90**	**39.74**	**39.51**

Table 6: Some results of the random walk test with $N = 10^8$ samples for PENTA89 and ZIFF89. For both generators three independent runs have been performed. Failing results are shown with bold type. See text for details.

5.4 n-block test

In the n-block test, we used an approach similar to the random walk test. First, we studied various generators with parameters $n = 10^4$ and $N = 10^6$. In the cases of GGL, RAND, RANMAR and RAN3, we observed no correlations. Then, for GFSR generators R31, R250 and R521 we performed an iterative study by varying n. When $N = 10^6$ samples were taken, the resulting correlation lengths n_c were 32 ± 1, 267 ± 5, and 555 ± 5, respectively. With better statistics $N = 10^8$, we observed no change for R31, whereas the estimate for R521 reduced to 525 ± 1, and that of R250 to 251 ± 1. The latter value was confirmed with $N = 10^9$ also. Typical values of χ^2 for R250 are shown in Fig. 9, where a sharp onset of correlations at n_c is visible.

Then we concentrated on studying a recommended generator ZIFF9689 (GFSR(9689,471,314,157,$\oplus$)) [205, 206], which unlike several other generators has performed well in recent simulations of self-avoiding random walks [65, 66]. This generator was extensively tested up to $n = 25\,000$ and $N = 10^7$, but no correlations were found. In order to increase the number of samples N, we tested ZIFF1279 (GFSR(1279,598,299,216,$\oplus$)) which is a 5-decimation of GFSR(1279,216,$\oplus$). With parameters up to $n = 1500$ and $N = 10^9$, no correlations were observed. These results suggest that deviations from random behavior are less significant for ZIFF generators than for GFSR generators with two lags. For quantitative purposes, we have studied this subject in more detail by comparing the results of R89, PENTA89, and ZIFF89. These results are shown in Fig. 10. The figure on the left clearly shows how dramatically inferior GFSR generators based on primitive trinomials are compared to generators which are based on either decimation of such sequences or use of primitive

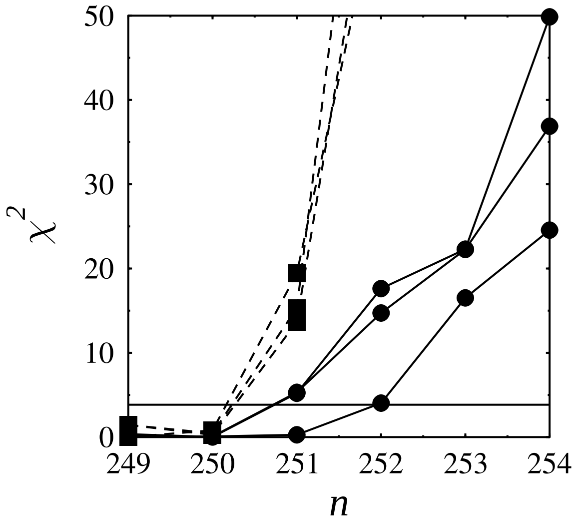

Figure 9: The χ^2 values for R250 in the n-block test. Curves with circles and squares correspond to $N = 10^8$ and $N = 10^9$ samples, respectively. In both cases three independent runs have been performed. The horizontal line denotes $\chi^2 = 3.841$.

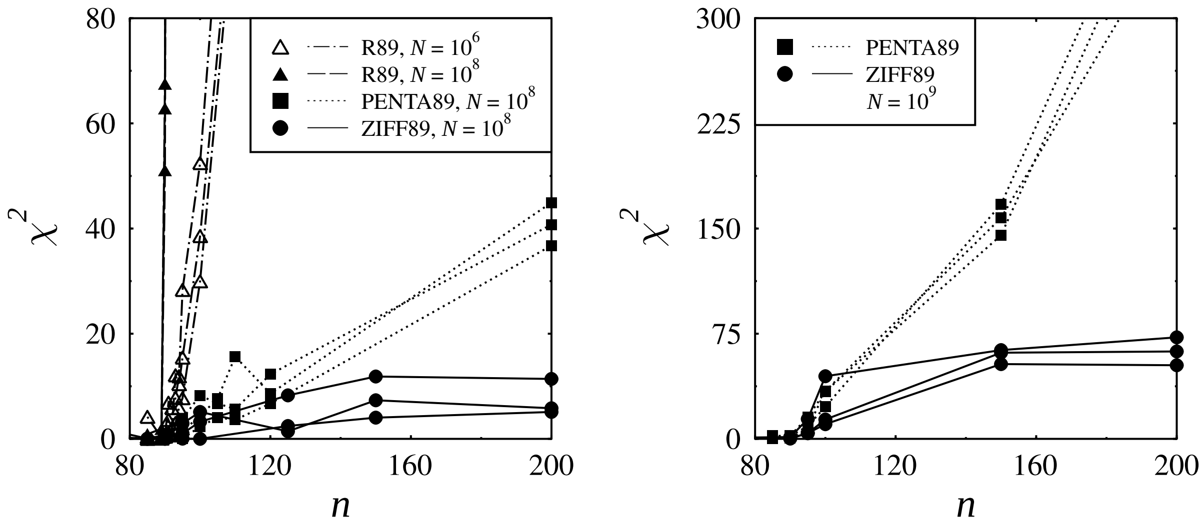

Figure 10: On the left are shown the χ^2 values of R89, PENTA89, and ZIFF89 as a function of block size n, when $N = 10^8$ samples have been taken. In the case of R89 results with $N = 10^6$ samples (open triangles) are also shown. On the right results of PENTA89 and ZIFF89 have been compared with better statistics $N = 10^9$.

pentanomials. Furthermore, when PENTA89 and ZIFF89 are compared to each other with higher statistics $N = 10^9$ (figure on the right), we may notice that at least in this particular case the decimated sequence ZIFF89 performs somewhat better than PENTA89, although correlations in both sequences are clearly present.

The results of the random walk and n-block tests together show that for the GFSR generators with two lags, the origin of the errors in the simulations presented here and in Refs. [28, 49, 65, 66, 166] must be the appearance of local correlations in the probability distribution. Moreover, although some empirical estimates for the correlation length have previously been given [65, 66], our tests are the first which quantitatively show that the correlation length lies very close to the expected value [31, 206] of the longer lag parameter p. Furthermore, for the generators based on a judicious decimation (*e.g.* $k = 3, 5, 7$) of GFSR generators (with two lags) or when primitive pentanomials are used as a basis for a generator, our results show that similar but less clear behavior is observed for these generators as well. Thus, generators using three consequtive exclusive-or operations seem to shuffle bits better than Rp generators in which only one exclusive-or operation is used. This results from the fact that compared with Rp generators, in ZIFFp generators the three-point correlations are much farther apart, and therefore higher order correlations dominate their behavior [206]. Although we are not aware of any theoretical studies for PENTAp generators, we *assume* that this is what happens for them also. In other words, the approach of using multiple exclusive-or's does not remove the correlations but makes them more subtle, as was also observed with the autocorrelation test where both ZIFF89 and PENTA89 passed the test with $N = 10^8$ samples, whereas R89 did not. Therefore, we may conclude this part of our work by saying that when generators based on the exclusive-or operation with a low lag parameter p are used, their results should be taken with a very sceptical attitude. On the other hand, if GFSR generators are still used, dubious results may not appear if multiple exclusive-or's and greater lags such as $p \geq 1279$ are used.

These results convincingly show that both the random walk test and the n-block test are very well suited for detecting local deviations from randomness in pseudorandom number sequences. The n-block test in particular is very efficient due to its simple nature. Moreover, it is important to realize that many other applications such as studies of self-avoiding random walks [169], percolation phenomena [172], and diffusion limited aggregation [193] are based on the use of random walks. Therefore, results of our random walk tests as well as the autocorrelation test are valid for such applications as well.

5.5 Condition number test

The studies of the condition number test consist of two parts. First, we will study the quality for three generation methods of Gaussian distributed random variables. Then, the best method will be used in further studies where several pseudorandom number generators will be tested using the condition number test.

5.5.1 Testing generation methods for Gaussian distributed random variables

The first part of this test consists of testing the quality of three generation methods of Gaussian distributed random variables: Marsaglia's polar method, the ratio method, and a method based on the central limit theorem. We first used parameters $m = 10$, $N = 10\,000$, and $M = 100$. With this choice of parameters, three independent tests for each method were carried out. In the case of Marsaglia's polar method the results for ϵ were 0.01, 0.005, and 0.02. In the case of the ratio method we found values 0.015, 0.015, and 0.03. Therefore, no significant deviations from the expected behavior with these methods were found. In the case of the method based on the central limit theorem, however, the number n of U(0,1) distributed random numbers (used in the generation of one Gaussian distributed random variable) affected the quality of the Gaussian distribution significantly. This is illustrated in Fig. 11 where ϵ is shown as a function of n. Clearly the often suggested value $n = 12$ is not large enough for a good approximation of the Gaussian distribution. Instead of that, values of $n \geq 96$ should be used.

Additional tests with parameters $m = 10$, $N = 10\,000$, and $M = 1000$ were also performed. The results for Marsaglia's polar method were 0.0105, 0.0205, and 0.02, while the ratio method gave 0.0245, 0.0235, and 0.023. Since Marsaglia's polar method is faster than the ratio method, the former was chosen for further studies of the condition number test in this work.

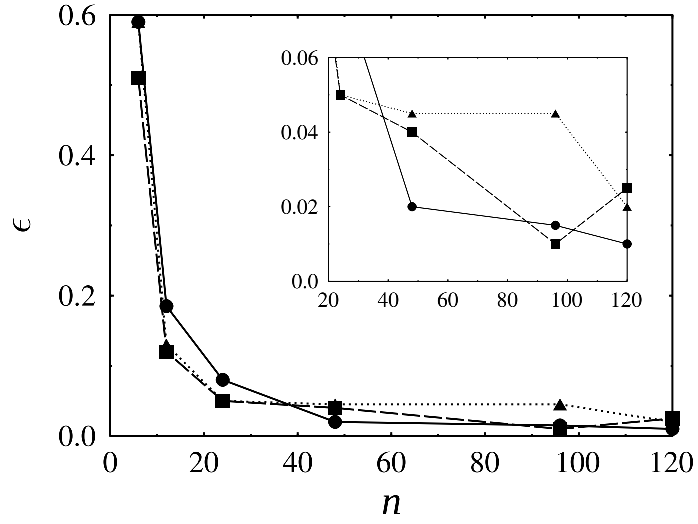

Figure 11: The test statistic ϵ as a function of n used uniform random numbers, when Gaussian distributed random variables have been generated with the method based on the central limit theorem. "Good" values of ϵ are 0.02 ± 0.01. The inner figure contains a small portion of the main figure.

5.5.2 Testing generators

In the condition number test, generators R31, R250, GGL, RAND, RAN3, and RANMAR were tested with two sets of parameters. First, we used $m = 10$, $N = 1000$, and $M = 1000$. None of the generators failed, when a single test was carried out. Then, with another set of parameters $m = 100$, $N = 1000$, and $M = 100$, generators R31 and R250 passed the test, whereas GGL, RAND, RAN3, and RANMAR failed the test. When the same test was repeated with a different initial seed value, all the remaining generators passed the test also.

Based on the results of the condition number test, subtle deviations from randomness in the output of uniform pseudorandom number generators are not very significant when such random numbers are being transformed into Gaussian distributed variables. One may assume that due to calculation of several arithmetic operations and functions, the transformation method reduces the effects of inevitable correlations, and if such a method is good enough, even relatively poor uniform random number generators may do well in applications in which Gaussian distributed random variables are generated.

Chapter 6

Summary and discussion

In addition to traditional applications such as lottery and numerical integration, random numbers are needed in numerous modern applications: high precision Monte Carlo simulations in physical sciences [10], image compression [7], algorithm development [44], and stochastic optimization [1], to name but a few. All these methods need reliable but still practical sources of random numbers, which due to practical reasons are usually generated by means of deterministic algorithms, implemented as pseudorandom number generators. Since the reliability of the results in all these applications depends on the quality of random numbers, there must be efficient means to confirm that. These means are empirical and theoretical tests. Although theoretical tests based on studying some general properties of pseudorandom number sequences give us basic knowledge of the properties of such sequences, random number testing is still mainly an empirical science. Though no empirical test can ever *prove* "goodness" of the quality of random number sequences, such tests give us a valuable insight into their properties. However, since the number of possible tests is practically unlimited, it is important that many tests, which mimic the properties of the application in which the random number sequences will be used, are included in the test program. This idea leads to the concept of application specific testing.

The motivation of this work has been twofold. First, we have wanted to develop efficient tests for random numbers, which are used in high precision physical simulations. This need arises from the fact that presently there are only few efficient empirical tests which mimic the properties of commonly studied simulation models. Second, recent high precision Monte Carlo simulations [28, 49, 65, 66, 166] have revealed anomalous correlations in the output of some commonly used pseudorandom number generators, and in that of the so called generalized feedback shift register (GFSR) generators in particular. Although there is good reason to assume that the erroneous results of GFSR generators are due to the three-point correlations with a correlation length $\xi = p$ (cf. Section 3.1), efficient test methods for confirming this have not been available up to date.

In this work, we have presented five simple tests for detecting correlations in random number sequences. The cluster test is based on the idea of comparing the cluster size distribution of a random lattice with the Ising model at an infinite temperature. Another test based on the use of the Ising model is the autocorrelation test, in which integrated autocorrelation times of some quantities of the Ising model are calculated. Then, in the random walk test we follow a random walker for a certain number of steps on a plane. The n-block test is also based on random walks, although in a simpler way. Finally, the condition number test uses some properties of random matrices, which are widely used in several applications such as nuclear physics [128] and studies of chaotic systems [34].

We implemented the cluster test on bit level and found it to be very effective in localization of periodic correlations in individual bits of random number sequences. It is remarkable, however, that correlations in GFSR generators were not observed, which shows that these generators should be good enough for many applications in which good bit level properties *of their individual bits* are required. However, since crosscorrelations between various bits in random numbers were not studied in this work, we do not want to rule out a possibility of bit level correlations either. Other tests were implemented to study random words. The results of the autocorrelation test, random walk test, and n-block test were especially enlightening, since these tests revealed the reason for poor behavior of GFSR generators in Refs. [28, 49, 166]. First, by means of the autocorrelation test we found that the origin of the errors observed in Ref. [49] lies in local correlations present in the cluster formation process of the Wolff algorithm [197, 198] since the average cluster sizes in these cases were clearly biased. Moreover, the integrated autocorrelation times of some quantities in the Ising model were found very sensitive measures of correlations. Then, based on the arguments in Refs. [31, 206], the correlation length of GFSR generators in the case of three-point correlations equals their longer lag p, which might be the real reason for the aforementioned errors. In this work, by means of the random walk and n-block tests, we have been able to reveal that this is indeed the case. Therefore, the random walks and n-block tests are very efficient in detecting local correlations in random number sequences. Furthermore, the problems observed in Refs. [49, 166] concern mainly GFSR generators (with two lags) using one exclusive-or operation, since similar lagged Fibonacci generators using one of operations $\{+, -, \times\}$ have given correct results in corresponding simulations [28]. One way to avoid this problem is to use four lags instead of two, as we have shown in this work. Such generators can be formed by using tables of primitive pentanomials [99] or by decimating GFSR sequences [205, 206]; *i.e.* taking every third number of their sequence, for example. Such approaches do not eliminate the three-point correlations but make them more subtle (farther apart), and therefore improve the quality of generated random numbers. Such generators have passed all our tests, when the longest lag parameter p has been chosen large enough ($p \geq 1279$). Finally, we have also performed tests for Gaussian distributed random matrices, but our results show that most Gaussian transformation methods reduce the effects of inevitable correlations in (uniformly distributed) random numbers to such an extent, that with our set of test parameters in the condition number test, no deviations from randomness with any of the generators were found. An exception was observed in the case of the transformation method based on the central limit theorem. We found this method to converge very slowly towards Gaussian distribution.

In conclusion, we believe that the tests for randomness presented here form an efficient test bench which can be used to develop better generators for demanding applications in physical sciences. Our results of the decimated sequences of GFSR generators are one step towards this direction. However, we note that it is still of crucial importance to further develop application specific tests along the lines presented here to detect more subtle correlations, which may not be revealed by the present test methods.

Bibliography

[1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines, A Stochastic Approach to Combinatorial Optimization and Neural Computing* (John Wiley & Sons, Chichester, 1989).

[2] S. L. Anderson, SIAM Review **32**, 221 (1990).

[3] D. A. André, G. L. Mullen, and H. Niederreiter, Math. Comp. **54**, 737 (1990).

[4] N. S. Altman, SIAM J. Sci. Stat. Comput. **9**, 941 (1988).

[5] A. C. Arvillias and D. G. Maritsas, J. Assoc. Comput. Mach. **25**, 675 (1978).

[6] M. N. Barber, R. B. Pearson, and D. Toussaint, Phys. Rev. B **32**, 1720 (1985).

[7] M. F. Barnsley and A. D. Sloan, Byte **13**, 1:215 (1988).

[8] R. J. Baxter, *Exactly Solved Models in Statistical Mechanics* (Academic Press, London, 1982).

[9] D. Beaver and N. So, in *Advances in Cryptology — EUROCRYPT '93*, edited by T. Helleseth (Springer-Verlag, Berlin, 1994), p. 424.

[10] K. Binder, *Monte Carlo Methods in Condensed Matter Physics*, edited by K. Binder (Springer-Verlag, Berlin, 1992).

[11] K. Binder and D. W. Heermann, *Monte Carlo Simulation in Statistical Physics* (Springer-Verlag, Berlin, 1988).

[12] M. Blum and S. Micali, SIAM J. Comput. **13**, 850 (1984).

[13] L. Blum, M. Blum, and M. Shub, SIAM J. Comput. **15**, 364 (1986).

[14] G. E. P. Box and M. E. Muller, Ann. Math. Sci. **29**, 610 (1958).

[15] J. Boyar, J. Cryptology **1**, 177 (1989).

[16] J. Boyar, J. Assoc. Comput. Mach. **36**, 129 (1989).

[17] P. Bratley and B. L. Fox, ACM Trans. Math. Softw. **14**, 88 (1988).

[18] P. Bratley, B. L. Fox, and L. E. Schrage, *A Guide to Simulation* (Springer-Verlag, New York, 1983).

[19] M. Brown and H. Solomon, Ann. Stat. **7**, 691 (1979).

[20] B. W. Brown, J. Lovato, and K. Russell, *A Library of Routines for Random Number Generation (C and Fortran)*, RANLIB, is available via anonymous FTP from `odin.mda.uth.tmc.edu` (129.106.3.17) in directory `/pub/unix`.

[21] J. C. Butcher, Math. Comp. **15**, 198 (1961).

[22] G. J. Chaitin, J. Assoc. Comput. Mach. **13**, 547 (1966).

[23] G. J. Chaitin, Scientific American, *May*, 47 (1975).

[24] G. J. Chaitin, J. Assoc. Comput. Mach. **22**, 329 (1975).

[25] G. J. Chaitin, Scientific American, *July*, 52 (1988).

[26] T.-W. Chiu and T.-S. Guu, Comput. Phys. Commun. **47**, 129 (1987).

[27] B. Chor and O. Goldreich, SIAM J. Comput. **17**, 230 (1988).

[28] P. D. Coddington, `cond-mat@babbage.sissa.it` No. `9309017` (unpublished).

[29] A. Compagner, J. Stat. Phys. **63**, 883 (1991).

[30] A. Compagner, Am. J. Phys. **59**, 700 (1991).

[31] A. Compagner and A. Hoogland, J. Computat. Phys. **71**, 391 (1987).

[32] R. Couture and P. L'Ecuyer, Math. Comp. **62**, **799** (1994).

[33] R. R. Coveyou and R. D. MacPherson, J. Assoc. Comput. Mach. **14**, 100 (1967).

[34] A. Crisanti, G. Paladin, and A. Vulpiani, *Products of Random Matrices* (Springer-Verlag, Berlin, 1993).

[35] L.-Y. Deng and Y.-C. Chu, in *Proceedings of the 1991 Winter Simulation Conference*, (IEEE Press, 1991), p. 1034.

[36] L.-Y. Deng and Y.-C. Chu, in *Proceedings of the 1991 Winter Simulation Conference*, (IEEE Press, 1991), p. 1043.

[37] P. Diaconis and M. Shahshahani, in *Random Matrices and Their Applications (Contemporary Mathematics 50)*, edited by J. E. Cohen, H. Kesten, and C. M. Newman (American Mathematical Society, Providence, 1986), p. 173.

[38] E. L. Dodd, Econometrica **10**, 249 (1942).

[39] E. J. Dudewitz and T. G. Ralley, *The Handbook of Random Number Generation and Testing with TESTRAND Computer Code* (American Sciences Press, 1981).

[40] W. F. Eddy, J. Computat. Appl. Math. **31**, 63 (1990).

[41] A. Edelman, SIAM J. Matrix Anal. Appl. **9**, 543 (1988).

[42] A. Edelman, Linear Alg. Appl. **159**, 55 (1991).

[43] A. Edelman, Math. Comp. **58**, 185 (1992).

[44] A. Edelman, Preprint *Eigenvalue Roulette and Random Test Matrices* (unpublished); personal communication (1993).

[45] A. Edelman, E. Kostlan, and M. Shub, J. Amer. Math. Soc. **7**, 247 (1994).

[46] A. Ehrenfeucht and J. Mycielski, The American Mathematical Monthly **99**, 373 (1992).

[47] J. Eichenauer-Herrmann, Int. Stat. Rev. **60**, 167 (1992).

[48] A. E. Ferdinand and M. E. Fisher, Phys. Rev. **185**, 832 (1969).

[49] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, Phys. Rev. Lett. **69**, 3382 (1992).

[50] U. Fincke and M. Pohst, Math. Comp. **44**, 463 (1985).

[51] G. S. Fishman, Math. Comp. **54**, 331 (1990).

[52] G. S. Fishman and L. R. Moore III, J. Amer. Statist. Assoc. **77**, 129 (1982).

[53] G. S. Fishman and L. R. Moore III, SIAM J. Sci. Stat. Comput. **7**, 24 (1986).

[54] L. D. Fosdick, *Methods in Computational Physics* (Academic Press, New York, 1963), p. 262.

[55] M. Fushimi, Inform. Proc. Lett. **16**, 189 (1983).

[56] M. Fushimi, SIAM J. Comput. **17**, 89 (1988).

[57] M. Fushimi, Appl. Math. Lett. **2**, 135 (1989).

[58] M. Fushimi, J. Computat. Appl. Math. **31**, 105 (1990).

[59] M. Fushimi and S. Tezuka, Comm. ACM **26**, 516 (1983).

[60] S. Garpman and J. Randrup, Comput. Phys. Commun. **15**, 5 (1978).

[61] J. E. Gentle J. Computat. Appl. Math. **31**, 119 (1990).

[62] H. H. Goldstine and A. Goldstine, Math. Tables Other Aids Comput. **2**, 97 (1946).

[63] S. W. Golomb, *Shift Register Sequences*, Revised Edition (Aegean Park Press, Laguna Hills, 1982).

[64] I. J. Good, Proc. Camb. Phil. Soc. **49**, 276 (1953).

[65] P. Grassberger, J. Phys. A **26**, 2769 (1993)

[66] P. Grassberger, Phys. Lett. A **181**, 43 (1993).

[67] D. H. Green, IEE Proceedings **132**, 133 (1985).

[68] R. E. Greenwood, Math. Tables Other Aids Comput. **9**, 1 (1955).

[69] F. Gruenberger, Math. Tables Other Aids Comput. **4**, 244 (1950).

[70] F. Gruenberger, Math. Tables Other Aids Comput. **6**, 123 (1952).

[71] F. Gruenberger and A. M. Mark, Math. Tables Other Aids Comput. **5**, 109 (1951).

[72] K. G. Hamilton, Comput. Phys. Commun. **81**, 237 (1994).

[73] J. Helin (unpublished) (1985).

[74] *Helsingin Sanomat*, 1st of February (tuesday), 1994, p. C4 (In Finnish): Previous weekend eight players were found winners in a Belgian lottery session, but the result had to be cancalled due to a technical error. The round was renewed on Monday.

[75] J. R. Heringa, H. W. J. Blöte, and A. Compagner, Int. J. Mod. Phys. C **3**, 561 (1992).

[76] A. Hoogland, A. Compagner, and H. W. J. Blöte, Physica **132**A, 593 (1985).

[77] A. Hoogland, J. Spaa, B. Selman, and A. Compagner, J. Computat. Phys. **51**, 250 (1983).

[78] T. E. Hull and A. R. Dobell, SIAM Review **4**, 230 (1962).

[79] H. Inoue, H. Kumahora, Y. Yoshizawa, M. Ichimura, and O. Miyatake, Appl. Statist. **32**, 115 (1983).

[80] N. Ito, M. Kikuchi, and Y. Okabe, Int. J. Mod. Phys. C **4**, 569 (1993).

[81] F. James, Rep. Prog. Phys. **43**, 1145 (1980).

[82] F. James, Comput. Phys. Commun. **60**, 329 (1990).

[83] F. James, Comput. Phys. Commun. **79**, 111 (1994).

[84] B. Jansson, *Random Number Generators*, (Victor Pettersons Bokindustri Aktiebolag, Stockholm, 1966).

[85] C. Kalle and S. Wansleben, Comput. Phys. Commun. **33**, 343 (1984).

[86] K. Kankaala, T. Ala-Nissila, and I. Vattulainen, Phys. Rev. E **48**, 4211 (1993).

[87] R. Kannan, A. K. Lenstra, and L. Lovász, Math. Comp. **50**, 235 (1988).

[88] M. G. Kendall, Biometrika **32**, 1 (1941).

[89] M. G. Kendall and B. Babington-Smith, J. Roy. Statist. Soc. A**101**, 147 (1938).

[90] M. G. Kendall and B. Babington-Smith, J. Roy. Statist. Soc. B**6** (Supplement), 51 (1939).

[91] M. G. Kendall and B. Babington-Smith, Tracts for Computers No. **24** (Cambridge University Press, London, 1961 Reprint).

[92] W. O. Kermack and A. G. McKendrick, Proc. Roy. Soc. Edinburgh **57**, 228 (1937).

[93] W. O. Kermack and A. G. McKendrick, Proc. Roy. Soc. Edinburgh **57**, 332 (1937).

[94] S. Kirkpatrick and E. P. Stoll, J. Computat. Phys. **40**, 517 (1981).

[95] R. Kleiss, in *Physics Computing '92* (World Scientific, Singapore, 1992), p. 94.

[96] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Vol. 2, 2nd Edition (Addison-Wesley, Reading, 1981).

[97] S. E. Koonin and D. C. Meredith, *Computational Physics, Fortran Version* (Addison-Wesley, Redwood City, 1990), p. 199.

[98] H. Krawczyk, J. Algorithms **13**, 527 (1992).

[99] Y. Kurita and M. Matsumoto, Math. Comp. **56**, 817 (1991).

[100] M. van Lambalgen, J. Symb. Logic **52**, 725 (1987).

[101] E. G. Landauer, Comput. & Indus. Engng. **8**, 65 (1984).

[102] G. P. Learmonth and P. A. W. Lewis, in *Computer Science and Statistics: 7th Annual Symposium on the interaface*, edited by W. J. Kennedy, (Statistical Laboratory, Iowa State University, Ames, Iowa, 1973), p. 163.

[103] P. L'Ecuyer, Comm. ACM **31**, 742 (1988).

[104] P. L'Ecuyer, Comm. ACM **33**, 86 (1990).

[105] P. L'Ecuyer, in *Proceedings of the 1992 Winter Simulation Conference*, (IEEE Press, 1992), p. 305.

[106] P. L'Ecuyer, *Uniform Random Number Generation*, Preprint; to appear in Ann. Oper. Res. in 1994.

[107] P. L'Ecuyer and R. Proulx, in *Proceedings of the 1989 Winter Simulation Conference*, (IEEE Press, 1989), p. 467.

[108] P. L'Ecuyer and S. Tezuka, Math. Comp. **57**, 735 (1991).

[109] D. H. Lehmer, in *Proc. 2nd Symp. on Large-Scale Digital Calculating Machinery* (Harvard University Press, Cambridge, 1951), p. 141.

[110] P. A. Lewis, A. S. Goodman, and J. M. Miller, IBM Syst. J. **8**, 136 (1969).

[111] T. G. Lewis and W. H. Payne, J. Assoc. Comput. Mach. **20**, 456 (1973).

[112] M. Lüscher, Comput. Phys. Commun. **79**, 100 (1994).

[113] M. D. MacLaren and G. Marsaglia, J. Assoc. Comput. Mach. **12**, 83 (1965).

[114] G. Marsaglia, Proc. of the Nat. Acad. Sci. **61**, 25 (1968).

[115] G. Marsaglia, in *Applications of Number Theory to Numerical Analysis*, edited by S. K. Zaremba (Academic Press, New York, 1972), p. 249.

[116] G. Marsaglia, in *Computer Science and Statistics: The Interface*, edited by L. Billard (Elsevier, Amsterdam, 1985), p. 3.

[117] G. Marsaglia, *Yet another RNG*, in newsgroup sci.stat.math (Internet), `1 Aug 1994 13:35:09 GMT` (Message-Id: <`1994Aug1.093312@stat.fsu.edu`>) (unpublished).

[118] G. Marsaglia and T. A. Bray, Comm. ACM **11**, 757 (1968).

[119] G. Marsaglia and L.-H. Tsay, Linear Algebra and its Applications **67**, 147 (1985).

[120] G. Marsaglia and A. Zaman, Stat. Prob. Lett. **8**, 35 (1990)

[121] G. Marsaglia and A. Zaman, Ann. Appl. Prob. **1**, 462 (1991).

[122] G. Marsaglia and A. Zaman, Computers in Physics **8**, 1:117 (1994).

[123] P. Martin-Löf, Information and Control **9**, 602 (1966).

[124] M. Matsumoto and Y. Kurita, ACM Trans. Mod. Comp. Sim. **2**, 179 (1992).

[125] A. De Matteis and S. Pagnutti, Parallel Computing **14**, 207 (1990).

[126] A. De Matteis, J. Eichenauer-Herrmann, and H. Grothe, J. Computat. Appl. Math **39**, 49 (1992).

[127] U. M. Maurer, J. Cryptology **5**, 89 (1992).

[128] M. L. Mehta, *Random Matrices* (Academic Press, New York, 1967).

[129] N. Metropolis, Los Alamos Science, Special Issue, 125 (1987).

[130] N. C. Metropolis, G. Reitwiesner, and J. von Neumann, Math. Tables Other Aids Comput. **4**, 109 (1950).

[131] N. Metropolis and S. Ulam, J. Amer. Statist. Assoc. **44**, 335 (1949).

[132] N. METROPOLIS, A. ROSENBLUTH, M. N. ROSENBLUTH, A. H. TELLER, AND E. TELLER, J. Chem. Phys. **21**, 1087 (1953).

[133] S. MICALI AND C. P. SCHNORR, J. Cryptology **3**, 157 (1991).

[134] A. MILCHEV, K. BINDER, AND D. W. HEERMANN, Z. Physik B **63**, 521 (1986).

[135] K. MOSER AND D. E. WOLF, Preprint *Vectorized and Parallel Simulations of the Kardar-Parisi-Zhang Equation in* $3+1$ *Dimensions* (unpublished).

[136] M. MULLER, Math. Rev. **16**, 749 (1955).

[137] K. R. NAIR, Sankhyā **4**, 65 (1938).

[138] R. E. NANCE AND C. OVERSTREET JR., Computing Reviews **13**, 495 (1972).

[139] J. VON NEUMANN, in *John von Neumann, Collected Works*, Vol. 5, edited by A. H. Taub (Pergamon Press, Oxford, 1963), p. 768.

[140] H. NIEDERREITER, Bull. Am. Math. Soc. **84**, 957 (1978).

[141] H. NIEDERREITER, SIAM J. Sci. Stat. Comput. **8**, 1035 (1987).

[142] H. NIEDERREITER, Math. Comp. **52**, 135 (1989).

[143] L. ONSAGER, Phys. Rev. **65**, 117 (1944).

[144] C. H. PAPADIMITRIOU, *Computational Complexity* (Addison-Wesley, Reading, 1994).

[145] G. PARISI AND F. RAPUANO, Phys. Lett. **157**B, 301 (1985).

[146] S. K. PARK AND K. W. MILLER, Comm. ACM **31**, 1192 (1988).

[147] R. K. PATHRIA, Math. Comp. **16**, 188 (1962).

[148] J. PAULSEN, J. Statist. Comput. Simul. **19**, 23 (1984).

[149] K. PEARSON, Philosophical Magazine **50** (fifth series), 157 (1900).

[150] K. PEARSON, Biometrika **25**, 379 (1933).

[151] S. C. Phatak and S. S. Rao, `cond-mat@babbage.sissa.it` No. 9310004 (unpublished).

[152] W. H. Press, B. P. Flannery, S. A. Tenkolsky, and W. T. Vetterling, *Numerical Recipes, The Art of Scientific Computing*, Fortran version (Cambridge University Press, 1989), p. 199.

[153] M. H. Quenouille, Biometrika **46**, 178 (1959).

[154] J. H. Reif and J. D. Tygar, SIAM J. Comput. **17**, 404 (1988).

[155] G. W. Reitwiesner, Math. Tables Other Aids Comput. **4**, 11 (1950).

[156] M. Richter has generated millions of physical random numbers (known as PURAN II) by measuring thermal noise of a semiconductor device. Statistical properties of these numbers were tested in Ref. [190], where no errors were found. The random numbers by PURAN II are available from an anonymous FTP site at `dfv.rwth-aachen.de`.

[157] B. D. Ripley, Int. Stat. Rev. **51**, 301 (1983).

[158] B. D. Ripley, Proc. Roy. Soc. London A**389**, 197 (1983).

[159] B. D. Ripley, Math. Programming **42**, 53 (1988).

[160] C. S. Roberts, Bell Syst. Tech. Journal **61**, 2053 (1982).

[161] R. Y. Rubinstein, *Simulation and the Monte Carlo Method* (John Wiley & Sons, New York, 1981).

[162] H. Sahai, J. Statist. Comput. Simul. **10**, 31 (1979).

[163] A. W. Schrift and A. Shamir, J. Cryptology **6**, 119 (1993).

[164] H. G. Schuster, *Deterministic Chaos: An Introduction* (Physik-Verlag, Weinheim, 1984).

[165] J. Seberry and J. Pieprzyk, *CRYPTOGRAPHY: An Introduction to Computer security* (Prentice-Hall, New York, 1989).

[166] W. Selke, A. L. Talapov, and L. N. Shchur, JETP Lett. **58**, 665 (1993).

[167] F. Sezgin, J. Computat. Appl. Math. **39**, 383 (1992).

[168] D. Shanks and J. W. Wrench Jr., Math. Comp. **16**, 76 (1962).

[169] A. D. Sokal, `hep-lat@ftp.scri.fsu.edu` No. `9405016` (unpublished).

[170] E. R. Sowey, Int. Stat. Rev. **40**, 355 (1972).

[171] E. R. Sowey, Int. Stat. Rev. **46**, 89 (1978).

[172] D. Stauffer, *Introduction to Percolation Theory* (Taylor & Francis, London, 1985).

[173] D. Stauffer gave us the main part of the Ziff pseudorandom number generator [205], which we extended to take care of initialization and vectorization; personal communication.

[174] M. F. Sykes and M. Glen, J. Phys. A **9**, 87 (1976).

[175] C. B. Tompkins, Math. Tables Other Aids Comput. **10**, 39 (1956).

[176] R. C. Tausworthe, Math. Comp. **19**, 201 (1965).

[177] S. Tezuka, Comm. ACM **30**, 731 (1987).

[178] S. Tezuka, J. Assoc. Comp. Mach. **34**, 939 (1987).

[179] S. Tezuka, Math. Comp. **50**, 531 (1988).

[180] S. Tezuka, in *Proceedings of the 1990 Winter Simulation Conference*, (IEEE Press, 1990), p. 266.

[181] S. Tezuka, in *Proceedings of the 1991 Winter Simulation Conference*, (IEEE Press, 1991), p. 1030.

[182] S. Tezuka, Math. Comp. **62**, 809 (1994).

[183] S. Tezuka and P. L'Ecuyer, ACM Trans. Mod. Comp. Sim. **1**, 99 (1991).

[184] S. Tezuka, P. L'Ecuyer, and R. Couture, *On the Lattice Structure of the Add-With-Carry and Subtract-With-Borrow Random Number Generators*; to appear in ACM Trans. Mod. Comp. Sim. (1994).

[185] W. E. Thomson, J. Roy. Statist. Soc. A**122**, 301 (1959).

[186] L. H. C. Tippett, Tracts for Computers No. **15** (Cambridge University Press, London, 1927).

[187] J. P. R. Tootill, W. D. Robinson, and A. G. Adams, J. Assoc. Comput. Mach. **18**, 381 (1971).

[188] A. M. Turing, Proc. London Math. Soc. **42**, 230 (1936-7); Proc. London Math. Soc. **43**, 544 (1937).

[189] G. Ugrin-Šparak, Computing **46**, 53 (1991).

[190] I. Vattulainen, K. Kankaala, J. Saarinen, and T. Ala-Nissila, University of Helsinki preprint HU-TFT-93-22 (1993); `cond-mat@babbage.sissa.it` No. `9304008` (unpublished).

[191] I. Vattulainen, T. Ala-Nissila, and K. Kankaala, Phys. Rev. Lett. **73**, 2513 (1994).

[192] I. Vattulainen, T. Ala-Nissila, and K. Kankaala, (unpublished) (1994).

[193] T. Vicsek, *Fractal Growth Phenomena* (World Scientific, Singapore, 1989).

[194] J.-S. Wang and R. H. Swendsen, Physica A **167**, 565 (1990).

[195] T. Warnock, Los Alamos Science, Special Issue, 137 (1987).

[196] W. J. Westlake, J. Assoc. Comput. Mach. **14**, 337 (1967).

[197] U. Wolff, Phys. Rev. Lett. **62**, 361 (1989).

[198] U. Wolff, Phys. Lett. B **228**, 379 (1989).

[199] S. Wolfram, Phys. Rev. Lett. **55**, 449 (1985).

[200] C.-K. Yuen, IEEE Trans. Comp. **C-26**, 329 (1977).

[201] G. U. Yule, J. Roy. Statist. Soc. A**101**, 167 (1938).

[202] R. Zielinski, J. Computat. Appl. Math. **31**, 205 (1990).

[203] N. Zierler, Inform. and Control **15**, 67 (1969).

[204] N. Zierler and J. Brillhart, Inform. and Control **13**, 541 (1968).

[205] R. M. Ziff, Phys. Rev. Lett. **69**, 2670 (1992).

[206] R. Ziff, *Reduction of Correlations in Shift-register Sequence Random Number Generators Using a Multiple Feedback Taps*, University of Michigan Preprint, 1992 (unpublished).

[207] *Convex Fortran Guide*, 1st edition (Convex Computer Corp., Richardson, USA, 1991), p. 553.

[208] *Encyclopedic Dictionary of Mathematics*, edited by S. Iyanaga and Y. Kawada (MIT Press, Cambridge, 1977), p. 1095.

[209] *IBM Subroutine Library — Mathematics* (User's Guide program number 5736-XM7, 1971).

[210] *IMSL Stat/Library User's Manual* **3** (IMSL, Houston, Texas, 1989), p. 945.

[211] *MATLAB User's Guide, PRO-MATLAB for VAX/VMS Computers* (The MathWorks Inc., South Natick, MA, 1991), p. 3-158; K. Kankaala, personal communication.

Appendix A

Polynomials $D_s(p)$

$$
\left\{
\begin{array}{lcl}
D_1(p) & = & p^4 \\
D_2(p) & = & p^6 \\
D_3(p) & = & p^9 + p^7 \\
D_4(p) & = & p^{12} + p^{11} + p^9 + p^8 \\
D_5(p) & = & p^{15} + p^{13} + p^{12} + p^{11} + p^8 \\
D_6(p) & = & p^{18} + p^{15} + p^{14} + p^{13} + p^{11} + p^{10} + p^9 \\
D_7(p) & = & p^{21} + p^{19} + p^{18} + p^{14} + p^{13} + p^{12} + p^{11} + p^9 \\
D_8(p) & = & p^{24} + p^{21} + p^{19} + p^{18} + p^{16} + p^{13} + p^{12} + p^9 \\
D_9(p) & = & p^{27} + p^{25} + p^{23} + p^{22} + p^{20} + p^{18} + p^{17} + p^{16} + p^{15} + p^{14} + p^{13} + p^{12} \\
 & & +p^{11} + p^{10} \\
D_{10}(p) & = & p^{30} + p^{27} + p^{26} + p^{23} + p^{22} + p^{20} + p^{19} + p^{18} + p^{14} + p^{13} + p^{11} + p^{10} \\
D_{11}(p) & = & p^{33} + p^{31} + p^{30} + p^{29} + p^{28} + p^{24} + p^{23} + p^{22} + p^{21} + p^{20} + p^{18} + p^{16} \\
 & & +p^{14} + p^{13} + p^{12} + p^{10} \\
D_{12}(p) & = & p^{36} + p^{33} + p^{31} + p^{30} + p^{29} + p^{27} + p^{25} + p^{24} + p^{22} + p^{21} + p^{20} + p^{18} \\
 & & +p^{16} + p^{12} + p^{10} \\
D_{13}(p) & = & p^{39} + p^{37} + p^{33} + p^{32} + p^{29} + p^{27} + p^{26} + p^{23} + p^{19} + p^{17} + p^{16} + p^{15} \\
 & & +p^{14} + p^{10} \\
D_{14}(p) & = & p^{42} + p^{39} + p^{36} + p^{34} + p^{33} + p^{30} + p^{29} + p^{27} + p^{26} + p^{25} + p^{23} + p^{22} \\
 & & +p^{21} + p^{20} + p^{17} + p^{16} + p^{15} + p^{14} + p^{13} + p^{12} + p^{11} \\
D_{15}(p) & = & p^{45} + p^{43} + p^{42} + p^{40} + p^{37} + p^{36} + p^{34} + p^{31} + p^{26} + p^{25} + p^{23} + p^{22} \\
 & & +p^{21} + p^{19} + p^{18} + p^{17} + p^{13} + p^{12} + p^{11} \\
D_{16}(p) & = & p^{48} + p^{45} + p^{44} + p^{42} + p^{41} + p^{40} + p^{39} + p^{38} + p^{29} + p^{28} + p^{27} + p^{26} \\
 & & +p^{25} + p^{21} + p^{18} + p^{16} + p^{15} + p^{12} + p^{11} \\
D_{17}(p) & = & p^{51} + p^{49} + p^{47} + p^{45} + p^{43} + p^{41} + p^{39} + p^{38} + p^{37} + p^{36} + p^{33} + p^{32} \\
 & & +p^{31} + p^{30} + p^{25} + p^{24} + p^{23} + p^{22} + p^{20} + p^{19} + p^{18} + p^{15} + p^{14} + p^{13} \\
 & & +p^{11}
\end{array}
\right.
$$

Table 7: The polynomials $D_s(p)$ in $p = 1/2$ [174].

Appendix B

Results of the first cluster test

Bit	Random number generator					
	GGL	R250	R1279	RANMAR	RAN3	RAND
1	1.906	0.697	0.733	1.640	**18.39**	0.853
2	0.215	0.447	0.185	0.131	**19.85**	0.001
3	1.005	0.259	0.420	0.254	**20.26**	0.804
4	2.160	0.815	0.186	0.649	**11.18**	0.613
5	0.389	0.054	1.523	0.921	0.602	0.959
6	0.846	0.685	0.192	1.548	0.679	1.045
7	1.787	0.447	0.838	0.340	0.599	**4.290**
8	1.666	1.440	0.040	0.234	0.341	**6.929**
9	0.122	1.134	0.768	0.579	0.082	**6.533**
10	1.284	1.221	0.864	0.494	1.107	**7.187**
11	1.734	0.939	0.298	1.274	0.488	**13.29**
12	0.437	0.173	0.457	0.580	0.798	**4.424**
13	0.027	0.244	0.985	0.803	0.207	**8.829**
14	2.557	0.558	1.032	0.635	0.535	**89.09**
15	0.782	1.223	0.740	**3.335**	1.310	**142.2**
16	0.324	0.042	0.231	0.523	0.911	**193.0**
17	1.428	1.572	0.767	0.131	0.470	**128.0**
18	1.959	1.079	0.013	0.132	1.570	**929.7**
19	0.199	0.075	1.097	0.883	0.675	**1485.**
20	0.759	0.164	1.750	1.190	1.276	**2812.**
21	2.266	0.809	0.613	1.098	0.489	∞
22	0.977	2.134	0.433	0.076	0.181	**5968.**
23	0.694	0.907	1.066	0.386	1.520	**8868.**
24	1.663	1.034	1.309	0.396	1.922	∞
25	1.945	0.247	1.540	∞	**4424.**	∞
26	0.153	0.072	1.055	∞	∞	∞
27	0.882	0.226	0.183	∞	∞	∞
28	2.283	1.694	1.161	∞	∞	∞
29	0.416	1.325	0.303	∞	∞	∞
30	0.567	0.675	0.087	∞	∞	∞
31	1.382	1.600	1.726	∞	∞	∞

Table 8: The values of goodness g'_i, $i = 1, 2, \ldots, 31$, for the first run of the cluster test. ∞ denotes a very large number (greater than 10^5). Other failing results are given in bold type. See text for details.

Appendix C

Results of the second cluster test

	Random number generator					
Bit	GGL	R250	R1279	RANMAR	RAN3	RAND
1	1.806	1.218	0.666	1.355	**21.03**	0.606
2	1.888	0.383	0.535	0.219	**20.61**	0.364
3	0.117	0.496	1.678	1.584	**20.15**	0.656
4	1.188	0.038	0.715	1.498	**14.62**	0.564
5	2.364	1.627	1.885	1.995	0.130	0.980
6	0.710	0.059	1.874	0.073	1.338	1.682
7	0.291	0.494	1.829	1.667	0.038	**4.282**
8	1.338	1.275	1.542	1.233	0.121	**7.253**
9	1.171	1.936	0.391	0.916	0.807	**6.374**
10	0.020	1.384	1.079	1.090	2.280	**7.690**
11	1.227	0.055	1.864	0.359	1.356	**12.40**
12	1.876	0.205	0.461	1.948	0.767	**4.348**
13	0.438	1.471	1.277	0.132	2.492	**9.594**
14	0.225	0.290	0.743	1.202	0.039	**93.68**
15	2.408	0.018	0.406	0.546	1.866	**144.8**
16	0.824	0.130	0.903	1.227	0.766	**209.6**
17	0.456	0.472	1.602	0.812	1.429	**141.5**
18	1.922	0.008	1.873	1.548	0.161	**1063.**
19	2.313	1.489	0.460	0.113	0.681	**1343.**
20	0.288	0.362	1.835	0.764	0.111	**2858.**
21	1.304	0.056	0.612	0.789	1.069	∞
22	2.598	0.646	2.704	0.467	1.268	**6458.**
23	1.412	1.370	0.816	0.798	0.285	**5720.**
24	0.851	0.535	1.544	0.755	1.151	∞
25	1.712	0.536	0.797	∞	**4542.**	∞
26	2.198	0.024	0.172	∞	∞	∞
27	0.044	1.372	0.993	∞	∞	∞
28	1.124	0.163	0.544	∞	∞	∞
29	1.995	0.179	0.185	∞	∞	∞
30	0.029	0.250	1.389	∞	∞	∞
31	0.531	0.692	1.230	∞	∞	∞

Table 9: The values of goodness g'_i, $i = 1, 2, \ldots, 31$, for the second run of the cluster test. ∞ denotes a very large number (greater than 10^5). Other failing results are given in bold type. See text for details.