Design, Search and Implementation of Improved Large Order Multiple Recursive Generators and Matrix Congruential Generators

Bryan R. Winter

DESIGN, SEARCH AND IMPLEMENTATION OF IMPROVED LARGE ORDER MULTIPLE RECURSIVE GENERATORS AND MATRIX CONGRUENTIAL GENERATORS

by

Bryan R. Winter

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Mathematical Sciences

The University of Memphis

May, 2014

DEDICATION

I would like to dedicate this manuscript to my loving father, Michael R. Winter, who passed away August 14, 2013. I would also like to dedicate it to my wife, Julie, my two sons, Colin and Corbin, my soon-to-be born daughter, Caroline, and any future children. I worked toward this and completed it thinking of you.

ACKNOWLEDGMENTS

I would like to thank Dr. Lih-Yuan Deng for showing me to climb directly up the mountain instead of going around, for investing in me, for countless hours of guidance, and for unwavering patience in stating and restating the facts for however long it took me to internalize them. To my committee members: Dr. E. Olusegun George, Dr. Dale Bowman Armstrong, and Dr. Su Chen, thank you for your time and valuable suggestions. I would also like to thank Dr. Paul Balister for his tutorial on the spectral test minimization problem and LLL algorithm.

To my mother, Bleeka Owens, and late father, Michael Winter, thank you for the early empowerment that I can do anything that I set my mind to do and for teaching me to work diligently until a job is done. This confidence and tenacity were exactly what I needed to complete this dissertation and the research behind it. To my mother, my mother-in-law, Cheryl Willis, father-in-law, Alan Willis, and several other family members and friends, thank you for countless hours of playing with the boys and filling in gaps so that I could remain engaged in my studies. To my father and brother, Lee Winter, thank you for countless hours of car repair (on my 22-year-old Saturn) when I could not afford the time to slow down and make the repairs myself. To Rusty Wheelington, Ed.D., thank you for encouraging me to get a Ph.D. in statistics.

To my children, thank you for being patient with me and sacrificing countless hours of playtime. To my wife, thank you for believing in me, for supporting me at home, and for holding the rock while I worked towards this manuscript. Most of all, to Providence, thank You for teaching me to fear You and walk in Your ways.

ABSTRACT

Winter, Bryan Ross. Ph.D. The University of Memphis. May, 2014. Design, Search and Implementation of Improved Large Order Multiple Recursive Generators and Matrix Congruential Generators. Major Professor: Dr. Lih-Yuan Deng.

Large order, maximum period multiple recursive generators (MRGs) with few nonzero terms (e.g., DX-k-s generators) have become popular in the area of computer simulation. They are efficient, portable, have a long period, and have the nice property of high-dimensional equi-distribution. The latter two properties become more advantageous as k increases. The performance on the spectral test, a theoretical test that provides some measure of uniformity in dimensions beyond the MRG's order k, could be improved by choosing multipliers that yield a better spectral test value. We propose a new method to compute the spectral test which is simple, intuitive, and efficient for some special classes of large order MRGs. Using this procedure, we list "better" FMRG-k and DX-k-s generators with respect to performance on the spectral test.

Even so, MRGs with few nonzero terms do not perform as well with respect to the spectral test as MRGs with many nonzero terms. However, MRGs with many nonzero terms can be inefficient or lack a feasible parallelization method, i.e., a method of producing substreams of (pseudo) random numbers that appear independent. To implement these MRGs efficiently and in parallel, we can use an equivalent recursion from another type of generator, the matrix congruential generator (MCG), a k-dimensional generalization of a first order linear recursion where the multipliers are embedded in a $k \times k$ matrix. When MRGs are used to construct MCGs and the recursion of the MCG is implemented k at a time for a k-dimensional vector sequence, then the MCG mimics k copies of a MRG in parallel with different starting seeds. Therefore, we propose a method for efficiently finding MRGs with many nonzero terms from an MRG with few nonzero terms and then give an efficient and parallel MCG implementation of these MRGs with many nonzero terms. This method works best for moderate order k.

For large order MRGs with many nonzero terms, we propose a special class called DW-k. This special class has a characteristic polynomial that yields many nonzero terms and corresponds to an efficient and parallel MCG implementation.

Contents

List of Tables

Chapter 1

Introduction

Good pseudo random number generators must be efficient, have sufficiently long periods, and have high-dimensional uniformity, that is, when taking t successive numbers at a time, these t-dimensional subsequences approximate a uniform distribution in a t-dimensional space. They must also have satisfactory empirical performance and a strong mathematical support. Furthermore, today's computing environment demands a good pseudo random number generator also have a method of parallelization, i.e., a method of producing substreams of pseudo random numbers that appear independent.

For a long time, the maximum period multiple recursive generator (MRG) has been thought to have many of the qualities of a good RNG except for efficiency and a feasible method of parallelization. The MRG generates pseudo random numbers sequentially with a k-th order linear recurrence under a prime modulus p. These generators have a strong statistical justification and excellent empirical performance. Furthermore, maximum-period MRGs have two very desirable properties: a huge period of $p^k - 1$ and the nice equi-distribution property over high dimensional spaces (up to order k). For a given modulus p, as order k increases, the large period and equi-distribution properties become more advantageous.

However, finding a maximum-period MRG can be very time consuming for large order k and prime modulus p. Furthermore, even for moderate order k, efficiently implementing maximum period MRGs with many non-zero coefficients or implementing them in parallel is non-trivial. Most research in this area has tried to side-step these difficulties by searching and implementing MRGs with some special structure such that the linear recurrence can be implemented by an MRG with few non-zero terms. Fewer non-zero terms requires fewer costly multiplications when generating the recursion.

Therefore, large order, maximum period MRGs with few nonzero terms (e.g., DX-k-s generators) have become popular in the area of computer simulation. However, the performance on the spectral test, a theoretical test that provides some measure of uniformity in dimensions beyond the MRG's order k, could be improved by choosing multipliers that yield a better spectral test value. We propose a new method to compute the spectral test which is simple, intuitive, and efficient for some special classes of large order MRGs. Using this procedure, we list "better" FMRG-k and DX-k-s generators with respect to performance on the spectral test.

Even so, MRGs with few nonzero terms do not perform as well with respect to the spectral test as MRGs with many nonzero terms. However, MRGs with many nonzero terms can be inefficient or lack a feasible parallelization method. To implement these MRGs efficiently and in parallel, we can use an equivalent recursion from another type of generator, the matrix congruential generator (MCG), a k-dimensional generalization of a first order linear recursion where the multipliers are embedded in a $k \times k$ matrix. When MRGs are used to construct MCGs and the recursion of the MCG is implemented

k at a time for a k-dimensional vector sequence, then the MCG mimics k copies of a MRG in parallel with different starting seeds. Therefore, we propose a method for efficiently finding MRGs with many nonzero terms from an MRG with few nonzero terms and then give an efficient and parallel MCG implementation of these MRGs with many nonzero terms. This method works best for moderate order k.

For large order MRGs with many nonzero terms, we propose a special class called DW-k. This special class has a characteristic polynomial that yields many nonzero terms and corresponds to an efficient and parallel MCG implementation.

Chapter 2

Literature Review

In this chapter, we will cover several developments in the research of the class of maximum period k-th order MRGs with prime modulus p, which this chapter denotes as MRG(k, p). Section 2.1 describes notation that will be used throughout the chapter. Section 2.2 will briefly discuss qualities of a "good" pseudo random number generator. Section 2.3 briefly defines the multiple recursive generator (MRG) and some of its important characteristics. Section 2.4 discusses the equi-distribution property and its relationship to the spectral test. Section 2.5 includes a brief summary of how to search for generators in MRG(k, p). Included in this section are several references for generators in MRG(k, p) already found. Section 2.6 details some generators in MRG(k, p) with few nonzero terms or generators in MRG(k, p) whose linear recurrence can be implemented with a higher-order maximum period MRG with few nonzero terms. Section 2.7 lays out how generators in MRG(k, p) can be implemented for parallel simulation. Finally, Section 2.8 describes two connections between generators in MRG(k, p) and another kind of random number generator called the matrix congruential generator (MCG). These connections explain how generators in MRG(k, p) can be implemented with MCGs.

2.1 Notation

Throughout this chapter, p is a large prime number and $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ denotes the finite field of p elements under the usual modulus operations of addition and multiplication. $\mathbb{Z}_p^k$ and $\mathbb{Z}_p^{k \times k}$ denotes the set of k-dimensional vectors and the set of $k \times k$ matrices, respectively, with elements in $\mathbb{Z}_p$. As already stated, MRG(k, p) denotes the class of maximum period k-th order MRGs with prime modulus p. The function $\phi(x)$ denotes the Euler totient function, which gives the number of integers between 1 and x that are relatively prime to x.

2.2 Good Random Number Generators

Several computer applications require a sequence of numbers that at least appear random. The examples are numerous: randomly sampling from a population, randomly assigning treatment(s) to subjects within a scientific experiment, simulating a probability distribution, and various statistical methods (for example, Monte Carlo methods or re-sampling methods). The majority of computers use some deterministic function to generate sequences of integers that appear as if they were from a sample of identically and independently distributed uniform random variables. Usually, these integers range from 0 to some integer $p-1$ and are later transformed to integers between 0 and 1 so that the sequence appears to mimic a sample of random variables from the uniform $U(0, 1)$ distribution.

Since the numbers are generated from a deterministic function, they are neither truly random nor truly independent and are therefore called *pseudo random numbers* and the generating functions are called *pseudo random number generators*. For the remainder of this dissertation, we will leave off the word "pseudo," and simply call the generating functions *random number generators* (RNGs), which we will do from this point forward. Most RNGs are periodic in that they can only generate so many numbers in successive sequence before the sequence repeats. The length of this sequence prior to repeating is called the period length of the RNG.

Not all RNGs are created equal (see, e.g., Hellekalek, 1998). Good RNGs must be efficient, have sufficiently long periods, and have high-dimensional uniformity, that is, when taking t successive numbers at a time, these t-dimensional subsequences approximate a uniform distribution in a t-dimensional space. They must also have satisfactory empirical performance. Furthermore, today's computing environment demands a good RNG also have a method of parallelization, that is, a method to generate substreams of random numbers that appear independent.

Several decades have been invested in searching for the “best” RNGs. Early on, the system of linear congruential generators (LCGs) proposed by Lehmer (1951) was established as the RNG of choice. This first order linear recurrence is indeed one of the most efficient RNGs to date. The search for the “best” RNG (inevitably) led to the “minimal standard” LCG published in the widely cited paper *Random Number Generators: Good Ones are Hard to Find* by Park and Miller (1988). However, as a class, LCGs have poor empirical performance, short periods (by today's standards), and inadequate uniformity in higher dimensions. These properties and others have earned the LCG a reputation as an unsuitable generator for modern simulation. Though there has been some movement away from using the LCG as the default generator, it nonetheless remains firmly established in many software programs.

For a long time, the multiple recursive generator (MRG) has been thought by many as the great contender for replacing the entrenched LCG. The MRG generates pseudo random numbers sequentially with a k-th order linear recurrence under a prime modulus p. Maximum-period MRGs have two very desirable mathematical properties: an extremely long period of $p^k - 1$ and the nice equi-distribution property up to order k. MRGs also has a strong statistical justification (Deng & George, 1990; Deng et al., 1997) and excellent empirical performance (Deng, 2005; Deng et al., 2012a, 2012b; L'Ecuyer & Simard, 2007). Indeed, concerning the search for maximum period MRGs, Knuth (1998) said "all known evidence indicates that the result will be a very satisfactory source of random numbers."

In the next section, we will discuss the development of maximum period MRGs.

2.3 Development of Multiple Recursive Generators

The k-th order linear recurrence for a MRG can be defined as

$$X_i = \alpha_1 X_{i-1} + \alpha_2 X_{i-2} + \cdots + \alpha_k X_{i-k} \mod p, \quad i \geq k, \tag{2.1}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_k$ are integers in $\mathbb{Z}_p$, $\alpha_k \neq 0$, and we can choose any k non-zero values as starting seeds, $\mathbf{X}_0 = (X_0, X_1, \cdots, X_{k-1}) \neq (0, 0, \cdots, 0)$. We remark that when the order

$k = 1$, the MRG reduces to a LCG:

$$X_i = BX_{i-1} \mod p, \quad i \geq 0. \tag{2.2}$$

As suggested in Deng and Xu (2003), the generated output X_i can be transformed to U_i in the interval $(0, 1)$ by performing the additional operation $U_i = (X_i + 0.5)/p$.

Checking whether a MRG defined in (2.1) has the maximum period $p^k - 1$ is equivalent to checking whether its characteristic polynomial

$$f(x) = x^k - \alpha_1 x^{k-1} - \alpha_2 x^{k-2} - \cdots - \alpha_k \tag{2.3}$$

is a k-th degree primitive polynomial over $\mathbb{Z}_p$ (see, e.g., L'Ecuyer, 1990). Alanen and Knuth (1964) and Knuth (1998) gave necessary and sufficient conditions for determining whether $f(x)$ is primitive or not. Section 2.5 will briefly describe how to search for generators in MRG(k, p).

As stated in the previous section, generators in MRG(k, p) have strong statistical justification and excellent empirical performance. Over their extremely long period $p^k - 1$, generators in MRG(k, p) also have the equi-distribution property up to order k. In the next section, we will formerly define the equi-distribution property and will discuss its implications as it relates to the spectral test, an important theoretical test that provides some measure of uniformity in dimensions beyond the MRGs order k. In fact, when testing the quality of a random number generator with at linear recurrence, Knuth (1998) said that the spectral test “is by far the most powerful test known.”

2.4 Equi-distribution Property and the Spectral Test

Generators in MRG(k, p) have the equi-distribution property up to order k, that is, over its entire period of $p^k - 1$, every t-tuple $(1 \leq t \leq k)$ of integers in $\mathbb{Z}_p^t$ appears exactly the same number of times (p^{k-t}), with the exception of the all-zero tuple which appears one time less (see, e.g., Lidl & Niederreiter, 1994, Theorem 7.43). We would expect a true t-dimensional multivariate uniform distribution to produce all p^t t-tuples in $\mathbb{Z}_p^t$ with equal frequency for any dimension t. Therefore, as k increases, the period length and equi-distribution property become more advantageous. In fact, for $t \leq k$, large-order generators in MRG(k, p) are pretty close to an "ideal" generator: only the all-zero tuple is generated one less time than the other t-tuples.

Consider the successive sequences of output from a generator in MRG(k, p), $\mathbf{S}_n = (X_n, X_{n+1}, X_{n+2}, \ldots, X_{n+t-1})$ for $n = 0, 1, \cdots, \rho - 1$ where $\rho = p^k - 1$ (the period length). $\mathbf{S}_n$ is state of the MRG at step n and t is the number of variates generated beyond step n. Let I be a set of fixed, nonnegative integers. The set of integers $I = \{0, 1, 2, 3, \ldots, t-1\}$ could be thought of as the indices selected from the state to create all the possible t-tuples over all steps n in the period of the MRG. Note that the number of elements in each tuple is t, because the number of indices in I are equal to t.

This index I does not necessarily have to be successive integers. More generally, we can consider any set of nonnegative integers $I = \left\{j_1, j_2, \ldots, j_r\right\}$ where $j_1 < j_2 < \cdots < j_r$, r is the number indices in set I, and $t = (j_r - j_1 + 1)$ is the number of variates generated beyond step n. Note that choosing numbers according to index set I generates

r-tuples. To select the r-tuples, t numbers are generated beyond each step n of the MRG state $\mathbf{S}_n$. Of these t numbers at each step, r numbers are selected according to index set I, that is, some numbers in $\mathbf{S}_n$ are selected and some are skipped.

Now consider the following set of all possible r-tuples that a maximum period MRG can generate from some general index $I = \{j_1, j_2, \ldots, j_r\}$

$$L_r(I) = \left\{ (X_{n+j_1}, X_{n+j_2}, \cdots, X_{n+j_r}) | n = 0, 1, \cdots, \rho - 1 \right\}. \tag{2.4}$$

According to the equi-distribution property, for *any* choice of I where $j_r - j_1 < k$, or equivalently, $t = (j_r - j_1 + 1) \leq k$, every nonzero r-tuple will appear the same number of times in the lattice $L_r(I)$. However, when $j_r - j_1 \geq k$, that is, $t = (j_r - j_1 + 1) > k$, the equi-distribution property is impossible to achieve. Again, ideally, we would want to generate all p^t possible t-tuples in $\mathbb{Z}_p^t$ with equal frequency over the period of the MRG and only choose r elements from those t-tuples according the index set I. However, generating all t-tuples is impossible, because the MRG can only generate $p^k - 1$ numbers before the sequence repeats. Therefore, there will be many t-tuples, or subsequences, that the MRG can never generate and consequently, many r-tuples that can never be selected.

Geometrically, these t-tuples can be thought of as t-dimensional points or vectors such that $L_t(I)$ forms a lattice of points in a t-dimensional space. Let $\Lambda_t(I)$ be the scaling (by dividing element-wise by p) of each t-tuple in $L_t(I)$ to a lattice of points in

$[0,1)^t$, that is, let

$$\Lambda_t(I) = \left\{ \left(\frac{X_{n+j_1}}{p}, \frac{X_{n+j_2}}{p}, \cdots, \frac{X_{n+j_r}}{p} \right) | n = 0, 1, \cdots, \rho - 1 \right\}. \tag{2.5}$$

Like the well-known problem for the LCG (Marsaglia, 1968), when $t > k$, these t-dimensional points form a lattice in a t-dimensional hypercube where we can find several families of equidistant parallel $(t-1)$-dimensional hyperplanes to cover all the points in the lattice. The space between parallel hyperplanes represent all the t-tuples that the MRG could never produce.

The spectral test computes the largest distance between adjacent parallel hyperplanes among families of parallel hyperplanes that cover all the points (see, e.g., Knuth, 1998; L'Ecuyer, 1997). We will call this largest distance the spectral distance and denote it as $d_t(k)$, since the spectral distance is influenced by dimension t and order k. The spectral distance is a measure of uniform spread of the t-tuples across a t-dimensional space. A small spectral distance implies a more uniform spread of the t-tuples, or t-dimensional points, across the t-dimensional space. Therefore, a large $d_t(k)$ is considered "bad," because a relative small number of parallel $(t-1)$-dimensional hyperplanes can cover all the t-dimensional points. Consequently, the MRG is said to have a "bad" lattice structure in dimension t. Clearly, if the dimension t is much larger than k, the spectral distance $d_t(k)$ becomes so large that no MRG (of fixed order k) can be considered "good."

In Chapter 3, we will consider the problem of computing the spectral test for an MRG of order k, which we also call computing the spectral distance $d_t(k)$ for $t > k$.

Traditionally, the spectral distance $d_t(k)$ is computed using a index set $I = \{0, 1, 2, 3, \ldots, t-1\}$ of successive indices (see, e.g., Knuth, 1998). However, the spectral test can also be computed for more general index sets $I = \left\{j_1, j_2, \ldots, j_r\right\}$ (see, e.g., L'Ecuyer, 1997). We remark that one can easily find a set of indices J such that the equi-distribution property cannot be achieved for an MRG.

For example, for the sake of generating efficiency, it is common to consider a MRG with few, say s, non-zero terms. Consider the following set of indices

$$J = \{k - i \mid \alpha_i \neq 0, i = 1, 2, \cdots, k\} \cup \{k\}. \tag{2.6}$$

The set J will always contain $r = s + 1$ indices where $j_1 = 0$ and $j_r = k$ such that $t = (j_r - j_1 + 1) = k + 1$. The other indices are merely k minus the index of the remaining nonzero multipliers α_i. This set J was considered in L'Ecuyer and Touzin (2004), L'Ecuyer and Simard (2014), and Tang and Kao (2002). Generating $(s+1)$-tuples according to this sequence requires generating $k+1$ variates at each step n of the MRG state and then only choosing $s+1$ of the variates according the indices in J. Notice that this nonsuccessive or lacunary sequence is purposefully chosen to exploit the structure of the few nonzero terms. Furthermore, selecting the $(s+1)$-tuples according to index set J essentially requires generating all possible $(k+1)$-tuples of which only $s+1$ elements are chosen. As explained above, under these conditions, the equi-distribution clearly cannot hold.

In the next section, we will consider how to search for maximum period MRGs.

2.5 Search for MRG(k, p)

As previously stated, checking whether a MRG defined in (2.1) belongs to the class MRG(k, p) is equivalent to checking whether its characteristic polynomial $f(x)$ in (2.3) is a k-th degree primitive polynomial over $\mathbb{Z}_p$. There are exactly $\phi(p^k - 1)/k$ primitive polynomials of degree k (see, e.g., Knuth, 1998). Therefore, $\phi(p^k - 1)/k$ is also the number of generators in MRG(k, p). Section 2.7 states a theorem that explains how to find every generator in MRG(k, p) from just one generator in MRG(k, p).

A set of necessary and sufficient conditions under which $f(x)$ is a primitive polynomial has been given in Alanen and Knuth (1964) and Knuth (1998). In order to apply these conditions, we need to find the complete factorization of $R(k, p) = (p^k - 1)/(p - 1)$ which can be hard when k or p is large. There are two common approaches to by-pass the difficulty of the factorization: (a) one can consider a prime order k and then find prime p such that $R(k, p)$ is also a prime number, or (b) for a given p, say $p = 2^{31} - 1$, one can find k such that $R(k, p)$ is (relatively) easy to factor, usually because $R(k, p)$ has only one huge prime factor and the rest are (relatively) small prime factors.

For examples of the first approach, see L'Ecuyer, et al. (1993) for $k \leq 7$, L'Ecuyer (1999) for $k \leq 13$, Deng (2004) for several $k \leq 1511$, Deng (2008) for several $k \leq 10007$, and Deng et al. (2012a) for several $k \leq 25013$. The largest period of the MRG found with this method is approximately 10^{233361} with the property of equi-distribution up to 25013 dimensions. For examples of the second approach, see Deng and Xu (2003) for

$k \in \{102, 120\}$, Deng (2005) for $k \in \{47, 643, 1597\}$, and Deng et al. (2012b) for $k \in \{7499, 20897\}$. The largest period of the MRG found using this method is approximately 10^{195009} with the property of equi-distribution up to 20897 dimensions.

2.6 Efficiently Implementing Generators in MRG(k, p) with Few Nonzero Terms

For even a moderate sized k, a generator in MRG(k, p) can be inefficient because it may require k multiplications to compute the next variate. Research in finding efficient generators in MRG(k, p) has generally involved finding those MRGs whose implementation only requires a small number of nonzero terms and thus, a small number of multiplications. Either the recursion in (2.1) is restricted to few nonzero terms or many terms are allowed but the structure is simple enough such that there exists an equivalent higher-order recurrence with only a few number of nonzero terms. If the former, then the MRG is implemented directly; if the latter, then the MRG is implemented via an equivalent higher-order MRG with few nonzero terms. In this section, we describe both of these special classes of generators in MRG(k, p).

Efficient MRGs with Only a Few Nonzero Terms

To increase efficiency, several authors have suggested restricting the number of coefficients to a small number of nonzero terms (usually, two). For examples, see Grube (1973), Kao and Tang (1997a, 1997b), L'Ecuyer and Blouin (1988), L'Ecuyer

(1990), L'Ecuyer et al. (1993). To further increase the generating efficiency, several special forms of the MRG recurrence requiring at most one multiplication per iteration were proposed. See Deng (2004), Deng (2005), Deng and Lin (2000), and Deng and Xu (2003). Specifically, Deng and Xu (2003) proposed the FMRG-k and DX-k-s generators:

- FMRG-k ($\alpha_t = 1, \alpha_k = B$):

$$X_i = X_{i-1} + BX_{i-k} \mod p, \quad i \geq k. \tag{2.7}$$

- DX-k-2 ($\alpha_t = \alpha_k = B$):

$$X_i = B(X_{i-1} + X_{i-k}) \mod p, \quad i \geq k. \tag{2.8}$$

- DX-k-3 ($\alpha_t = \alpha_{\lceil k/2 \rceil} = \alpha_k = B$):

$$X_i = B(X_{i-1} + X_{i-\lceil k/2 \rceil} + X_{i-k}) \mod p, \quad i \geq k. \tag{2.9}$$

- DX-k-4 ($\alpha_t = \alpha_{\lceil k/3 \rceil} = \alpha_{\lceil 2k/3 \rceil} = \alpha_k = B$):

$$X_i = B(X_{i-1} + X_{i-\lceil k/3 \rceil} + X_{i-\lceil 2k/3 \rceil} + X_{i-k}) \mod p, \quad i \geq k, \tag{2.10}$$

where $\lceil x \rceil$ is the ceiling function denoting the smallest integer $\geq x$. Some will note that FMRG-k has been previously denoted as DX-k-1. However, as noted in Deng and Xu (2003), FMRG-k does not formerly belong to the DX-k-s class and was only referred to as DX-k-1 for convenience. Technically, the class of DX-k-s generators should have s nonzero terms where each term shares the same multiplier B. As can be seen in (2.7), FMRG-k strays from this design and as such the performance of FRMG-k does not necessarily generalize to the DX-k-s class. For the remainder of this dissertation, DX-k-s will refer only to those generators given in (2.8), (2.9), and (2.10). FMRG-k will refer specifically to (2.7).

Generators in MRG(k, p) with few nonzero terms in the generating equation can be very efficient. And several have performed very well when subjected to stringent empirical tests (Deng, 2008; Deng et al., 2012a, 2012b; L'Ecuyer & Simard, 2007). However, the spectral distance $d_{k+1}(k)$ is worse than what would be expected from a MRG with many nonzero terms (Kao & Tang, 1997a). In Chapter 3, we discuss computing the spectral test for DX-k-s generators and observe a special property concerning the spectral test for this class.

MRGs with Many Nonzero Terms Efficiently Implemented with a Higher-Order Recurrence

According to L'Ecuyer (1997), generators in MRG(k, p) with small spectral distances $d_t(k)$ will necessarily have multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$ in (2.1) such that their sum of squares $\sum_{i=1}^{k} \alpha_i^2$ is large. Therefore, Deng, Li, Shiau, and Tsai (2008) and Deng, Shiau, and Tsai (2009b) proposed special classes of generators in MRG(k, p) with many nonzero multipliers: DL-k-t, DS-k-t, and DT-k. A direct implementation of these special classes is inefficient, because several costly multiplications are required. Instead, these special classes have a simple structure across the multipliers such that an equivalent higher-order recurrence of a maximum period MRG with few nonzero terms can be utilized for an efficient implementation.

- DL-k-t ($\alpha_i = B$ for $i = 1, 2, \ldots, k$), $1 \le t < k$):

$$X_i = B(X_{i-t} + X_{i-t-1} + \cdots + X_{i-k}) \mod p, \quad i \ge k \tag{2.11}$$

  efficiently implemented by

$$X_i = X_{i-1} + B(X_{i-t} - X_{i-k-1}) \mod p, \quad i \ge k+1. \tag{2.12}$$

- DS-k-t ($\alpha_t = 0$, $\alpha_i = B$ for $i = 1, 2, \ldots, t-1, t+1, \ldots, k$), $1 \le t < k$:

$$X_i = B \sum_{j=1, j \ne t}^{k} X_{i-j} \mod p \tag{2.13}$$

  efficiently implemented by

$$X_i = X_{i-1} + B(X_{i-1} - X_{i-t} + X_{i-t-1} - X_{i-k-1}) \mod p, \quad i \ge k+1. \tag{2.14}$$

- DT-k ($\alpha_i = B^{k+1-i}$ for $i = 1, 2, \ldots, k$):

$$X_i = (B^k X_{i-1} + B^{k-1} X_{i-2} + \cdots + B X_{i-k}) \mod p, \quad i \ge k \tag{2.15}$$

  efficiently implemented by

$$X_i = ((B^{-1} + B^k) X_{i-1} - X_{i-k-1}) \mod p, \quad i \ge k+1 \tag{2.16}$$

  where $D \equiv (B^{-1} + B^k) \mod p$ can be pre-computed.

For the above generators, $\mathbf{X}_0 = (X_0, X_1, \ldots, X_{k-1})$ is the initial seed. Since the higher-order recursions additionally require X_k before initialization, X_k is computed from the recursion in (2.11) for DL-k-t, from the recursion in (2.13) for DS-k-t, and from the recursion in (2.15) for DT-k. According to Kao and Tang (1997a), these generators should have larger spectral distances in dimension $k+2$ than would be expected from a MRG with many nonzero terms, since the recursion corresponds to a $(k+1)$-th order MRG with few nonzero terms.

2.7 Running MRG(k, p) in Parallel

To generate streams of random numbers that appear independent, there are currently two ways of running MRGs in parallel: (a) change the starting seeds or (b) change the multipliers (Deng et al., 2009a). The former uses only one generator in MRG(k, p) to generate each stream of random numbers; the goal of this method is to skip sufficiently far enough ahead in the large period of the MRG so that there is no overlap in the streams of random numbers. Thus, each starting seed is assigned to a different central processing unit (CPU), but the same generator in MRG(k, p) is used across CPUs. The latter allows each stream to come from a different MRG with the same order k and prime modulus p. Thus, each CPU receives its own unique maximum-period MRG.

Philosophically speaking, parallelization using one generator in MRG(k, p) with different starting seeds is a good strategy if the generator at hand is known to have excellent empirical performance and strong mathematical justification. For example, a general MRG without any special structure would be suitable for such a parallelization strategy. However, for a generator in MRG(k, p) with some special structure, a flaw may exist that is not yet been discovered; if so, the entire parallel simulation could be compromised. In contrast, if each parallel CPU is given its own generator in MRG(k, p), then only a part of the parallel simulation would suffer if one of the MRGs were found to have some flaw. When choosing a parallelization strategy, these concepts should be kept in mind.

Parallelization by Changing Starting Seeds for One MRG via Jump-Ahead

For any generator in MRG(k, p), the characteristic polynomial $f(x)$ in (2.3) always has a corresponding $k \times k$ companion matrix $\mathbf{M}_f$ defined as

$$\mathbf{M}_f = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ \alpha_k & \alpha_{k-1} & \alpha_{k-2} & \dots & \alpha_1 \end{pmatrix}. \tag{2.17}$$

Thus, $f(x)$ can be rewritten as

$$f(x) = \det(x\mathbf{I} - \mathbf{M}_f) \mod p. \tag{2.18}$$

Using this companion matrix, we can compute seeds that are sufficiently far apart within the period of the generator in MRG(k, p). As explained in Deng et al. (2009a), to compute a new seed vector, $\mathbf{X}_{new0}$, that is "m-apart" from the initial seed vector, $\mathbf{X}_{old0} = (X_0, X_1, \dots, X_{k-1})$, we can calculate

$$\mathbf{X}_{new0} = \mathbf{M}_f^m \mathbf{X}_{old0} \mod p. \tag{2.19}$$

Several seeds can be pre-computed and saved for later use. L'Ecuyer et al. (2002) use this jump-ahead technique in their set of software utilities. Clearly, this method becomes increasingly difficult for larger orders of k. This method is also not scalable; it requires the simulation scientist to pre-determine how many streams are needed for her parallel simulation. The more streams that are needed, the shorter each stream sequence must be. This limitation will be problematic for parallel simulations requiring many (or some unknown amount of) parallel CPUs where each CPU demands a very long sequence of random numbers.

Parallelization by Creating Different MRGs On-demand via AGM

Starting with a base generator in MRG(k, p), Deng (2004) proposed the Automatic Generation Method (AGM) to quickly find numerous generators in MRG(k, p) with the same number of nonzero multipliers as the base MRG. Therefore, if the base generator has few nonzero terms, then all the MRGs spawned from AGM will also have few nonzero terms. However, the spawned MRGs will (most likely) not share a multiplier even if the base generator does, which is the case when using a base generator from the DX-k-s class. Thus, spawned MRGs can be slightly less efficient than their base generator.

Deng (2004) showed that for any k-th degree primitive polynomial $f(x)$ in (2.3) with modulus p such that $(p^k - 1)/(p - 1)$ is prime, a transformation of $f(x)$ could be made such that the transformed polynomial is also a k-th degree primitive polynomial with the same number of nonzero coefficients. For any nonzero integer $z \in \mathbb{Z}_p$, the transformation of $f(x)$ is

$$\begin{aligned} G(x) &= z^{-k} f(zx) \mod p \qquad (2.20) \\ &= x^k - G_1 x^{k-1} - G_2 x^{k-2} - \cdots - G_k \mod p, \end{aligned}$$

where

$$G_j = z^{-j} \alpha_j \mod p, \quad j = 1, 2, \ldots, k,$$

and $G(x)$ is a k-th degree primitive polynomial only if $G_k = z^{-k}\alpha_k$ is a primitive root modulo p. Since we know how many primitive roots exist, we also know that AGM can produce $\phi(p-1)$ generators in MRG(k,p) from one base generator in MRG(k,p). Note also that the number of nonzero terms in the base generator is preserved in the spawned generators, because G_j will be nonzero only when α_j is nonzero.

Deng (2005) extended AGM to base generators in MRG(k,p) where $R(k,p)$ was not necessarily prime. Deng et al. (2009a) improved AGM by adding conditions so that z was chosen distinctly and randomly while still ensuring that $z^{-k}\alpha_k$ is a primitive root modulo p. Deng et al. (2009b) showed how AGM could be applied to DL-k-t and DT-k generators.

With this method, each CPU can be assigned its own, unique generator in MRG(k,p). Thus, the entire period of each MRG is available to its corresponding CPU. Since each CPU has a different generator in MRG(k,p), each CPU can have its own seed or the same seed can be given to all (or some subset of) the CPUs. Furthermore, this method is scalable. The simulation scientist does not need to know *a priori* the number

CPUs needed for the simulation. As the simulation demands a new CPU, AGM can quickly find a new generator in MRG(k, p) which can be assigned to the new CPU. If the base generator in MRG(k, p) has few nonzero terms sharing the same multiplier, then the spawned MRGs will have the same number of nonzero terms but will not share the same multiplier. Thus, the spawned MRGs require a few more multiplications and will be slightly less efficient than the base MRG.

Parallelization by Finding All MRGs for Fixed k, p

If we are not concerned about the number of nonzero coefficients, then we can quickly find all $\phi(p^k - 1)/k$ generators in MRG(k, p) without having to check the time-consuming conditions mentioned in Section 2.5. Since finding a k-th degree primitive polynomial modulo p is equivalent to finding a generator in MRG(k, p), the following theorem shows how to find all the generators in MRG(k, p) from one generator in MRG(k, p).

Theorem 1. *Let $f(x)$ in (2.3) be a k-th degree primitive polynomial with companion matrix $\mathbf{M}_f$ as in (2.17). Define*

$$f_r(x) = \det(x\mathbf{I} - \mathbf{M}_f^r) \mod p. \tag{2.21}$$

1. *$f_r(x)$ is a primitive polynomial if and only if r is relatively prime to $p^k - 1$.*

2. *For any k-th degree primitive polynomial $h(x)$, there exists r relatively prime to $p^k - 1$ such that $h(x) = f_r(x) = \det(x\mathbf{I} - \mathbf{M}_f^r) \mod p$.*

3. *If q, r are relative prime to $p^k - 1$, then the following conditions are equivalent:*

   (a) $f_r(x) = f_q(x)$.

   (b) $r = qp^i \mod (p^k - 1)$, *for some integer* $i \geq 0$.

Theorem 1 follows easily using arguments similar to those by Golomb (1967, Theorem 2, p. 78) and by Zierler (1959, Theorem 11).

The first part of this theorem shows once any primitive polynomial $f(x)$ is found, a second one $f_r(x)$ can be found by taking the companion matrix $\mathbf{M}_f$ of the first, raising it to a power r coprime to $p^k - 1$, and finding $\det(x\mathbf{I} - \mathbf{M}_f^r) \bmod p$. The next part of the theorem states that, in fact, each of the $\phi(p^k - 1)/k$ possible k-th degree primitive polynomials can found by using the companion matrix $\mathbf{M}_f$ from just one primitive polynomial $f(x)$. The last part of the theorem instructs that although all $\phi(p^k - 1)/k$ possible k-th degree primitive polynomials can found using one companion matrix $\mathbf{M}_f$, there are some integers q and r coprime to $p^k - 1$ that produce duplicate primitive polynomials (thus duplicate coefficients: $\alpha_1, \alpha_2, \ldots, \alpha_k$), and these duplicate primitive polynomials will occur whenever $r = qp^i \mod (p^k - 1)$, for some integer $i \geq 0$.

The fact that all primitive polynomials can be generated from one primitive polynomial has been known for a long time. And it is perhaps self-evident that each of these unique MRGs could be assigned to a different CPU for a parallelization of independent sources of random numbers. However, we cannot control the number of nonzero coefficients that will result from Theorem 1. As k increases, the spawned MRGs become increasingly inefficient to implement directly, because a direct implementation may often require many multiplications.

Nonetheless, this important theorem will be drawn upon in Chapter 4 to produce an abundant source of general maximum period MRGs with many nonzero terms.

2.8 Some Connections between MRGs and Matrix Congruential Generators (MCGs)

Grothe (1987) showed that the matrix congruential generator (MCG), a k-dimensional generalization of the first order linear recurrence where the multipliers are embedded in a $k \times k$ matrix, can be constructed from a generator in MRG(k, p). Then, using the Cayley-Hamilton theorem, Grothe (1987) showed that the MCG can be used to implement this maximum period MRG in parallel. We describe both of these connections next.

The MCG is a natural k-dimensional extension of the LCG:

$$\mathbf{X}_i = \mathbf{B}\mathbf{X}_{i-1} \mod p, \quad i \geq 1 \tag{2.22}$$

where $\mathbf{X}_i$ is a k-dimensional vector in $\mathbb{Z}_p^k$ and the multiplier matrix $\mathbf{B}$ is a $k \times k$ matrix in $\mathbb{Z}_p^{k \times k}$. The characteristic polynomial of a MCG is defined as

$$f_{\mathbf{B}}(x) = \det(x\mathbf{I} - \mathbf{B}) \mod p. \tag{2.23}$$

An integer matrix $\mathbf{B} \in \mathbb{Z}_p^{k\times k}$ has order n if

$$n = \min_{j>0} \left\{ j : \mathbf{B}^j \mod p = \mathbf{I} \right\}.$$

As a vector sequence $(\mathbf{X}_i)_{i\geq 0}$, a MCG has the maximum period of $p^k - 1$ *if and only if* the matrix $\mathbf{B}$ has the order of $p^k - 1$. It is well known that $\mathbf{B}$ has the order of $p^k - 1$ *if and only if* its corresponding characteristic polynomial $f_{\mathbf{B}}(x)$ defined in (2.23) is a primitive polynomial.

As shown in Section 2.7, the primitive characteristic polynomial $f(x)$ of a generator in MRG(k, p) can be rewritten as $f(x) = \det(x\mathbf{I} - \mathbf{M}_f)$, where $\mathbf{M}_f$ is the companion matrix defined in (2.17). This implies every companion matrix $\mathbf{M}_f$ of a generator in MRG(k, p) has order $p^k - 1$. Knowing this, Grothe (1987) proposed a class of MCGs with $\mathbf{B} = \mathbf{T}\mathbf{M}_f\mathbf{T}^{-1}$, where $\mathbf{T}$ is an invertible $k \times k$ matrix over $\mathbb{Z}_p^{k\times k}$ and $\mathbf{M}_f$ is the companion matrix to a generator in MRG(k, p). Hence, our first connection between MCGs and MRGs is that MCGs can be constructed from the companion matrix $\mathbf{M}_f$ of a generator in MRG(k, p). Since $f_{\mathbf{B}}(x) = \det(x\mathbf{I} - \mathbf{T}\mathbf{M}_f\mathbf{T}^{-1}) = \det(x\mathbf{I} - \mathbf{M}_f) = f(x)$ over $\mathbb{Z}_p$, then the MCG shares the same characteristic polynomial as the generator in MRG(k, p).

Using the Cayley-Hamilton Theorem, Grothe (1987) gave a parallel MCG implementation of a generator in MRG(k, p) that shared the same primitive characteristic polynomial as the MCG. He showed that, when taken as a k-dimensional vector sequence $(\mathbf{X}_i)_{i\geq 0}$, the MCG satisfies the following recursion

$$\mathbf{X}_i = \alpha_1\mathbf{X}_{i-1} + \alpha_2\mathbf{X}_{i-2} + \cdots + \alpha_k\mathbf{X}_{i-k} \mod p, \quad i \geq k. \tag{2.24}$$

Therefore, each of the k sequences taken from each of the k rows in (2.24) can be viewed as k copies of the same MRG with different starting seeds. In other words, as a vector sequence, maximum period MCGs can be viewed as running k copies of a generator in MRG(k, p). This is our second connection.

Due to the equi-distribution property, every k-tuple of integers in $\mathbb{Z}_p^k$ (except the all-zero tuple) appears exactly once. Hence, these k MRG sequences (with different starting seeds) are simply shifts within the same MRG cycle. Clearly, different choices of $\mathbf{T}$ will result in different shifts. Typically, due to the huge period length for MRG(k, p), a "random" choice of $\mathbf{T}$ tends to yield k sequences that are far apart in the MRG cycle. It can be shown that, on average, each of the k sequences will be about $O((p^k - 1)/k^2)$ numbers apart. With this in mind, we can consider a MCG as a parallel implementation of k copies of the same MRG with some "random" jump-ahead scheme.

From this parallel MCG implementation, it is important to choose matrix $\mathbf{T}$ such that the MCG's generating recursion is efficient. The matrix $\mathbf{B} = \mathbf{T}\mathbf{M}_f\mathbf{T}^{-1}$ has $O(k^2)$ nonzero terms which requires at most k^2 multiplications to produce a single k-dimensional vector. On average, the number of multiplications needed per computed value is of the order $O(k)$. This is clearly not efficient.

If we choose $\mathbf{T}$ to be the identity matrix, then the MCG with multiplier matrix $\mathbf{B} = \mathbf{M}_f$ requires only k multiplications to produce a single k-dimensional vector. However, as a vector sequence, the first $k - 1$ rows in (2.24) are simple one-step shifts of each other, which is clearly unsuitable for parallel simulation. The last row, with all the k multiplications, is no more efficient than implementing the MRG directly.

In Chapter 4, we show how to dramatically increase the efficiency of the parallel

MCG implementation. In Chapter 5, we give an efficient and parallel MCG implementation of a special class of large order maximum period MRGs with many nonzero terms.

Chapter 3

Spectral Tests For Some Large Order Multiple Recursive Generators

3.1 Introduction

Large order, maximum period multiple recursive generators (MRGs) (with few nonzero terms) have become popular in the area of computer simulation. They are efficient, portable, have a long period, and have the nice property of high-dimensional equi-distribution. Recently, several large order MRGs have been proposed in the literature. Of these specified generators, the performance on the spectral test, a theoretical test that provides some measure of uniformity in dimensions beyond the MRG's order k, could be improved by choosing multipliers that yield a better spectral test value. While there has been some published work on the problem of computing the spectral test, the procedure can be quite tedious and inefficient for large order MRGs. In this chapter, we propose a new method which is simple, intuitive, and efficient for some special classes of large order MRGs including the FMRG-k and DX-k-s generators. Using this procedure, we propose a list of "better" FMRG-k and DX-k-s generators with respect to performance on the spectral test.

In Section 3.2, we will give an intuitive, geometric method for computing the spectral test, an important theoretical test explained in Section 2.4. For computing the spectral test of an maximum period MRG in (2.1) for one dimension beyond k, a simple algorithm is given. However, computing the spectral test for several dimensions beyond k requires a more sophisticated algorithm, which we give in Section 3.3. Using this algorithm to study FMRG-k and DX-k-s generators, we observed in Section 3.4 that these special classes of MRG have a property that we term the consistency property, where some generators in this class maintain the same spectral test value for many dimensions beyond k. In Section 3.5, we tabulate several generators in FMRG-k and DX-k-s with spectral test values below a pre-specified cutoff and each possessing the consistency property.

Notation

Throughout this chapter, p is a large prime number and $\mathbb{Z}_p = \{0, 1, 2, \cdots, p-1\}$ denotes the finite field of p elements under the usual modulus operations of addition and multiplication. $\mathbb{Z}_p^k$ denotes the set of k-dimensional vectors with elements in $\mathbb{Z}_p$. For any integer x, we define

$$[x]_p = \begin{cases} (x \bmod p), & \text{if } (x \bmod p) < p/2 \\ (x \bmod p) - p, & \text{otherwise.} \end{cases} \tag{3.1}$$

Therefore, $[x]_p$ denotes the symmetric representation with respect to modulus p such that $-p/2 < [x]_p < p/2$. For a t-dimensional integer vector $\mathbf{x} = (x_1, x_2, \cdots, x_t)$, we define

$[\mathbf{x}]_p = ([x_1]_p, [x_2]_p, \cdots, [x_t]_p)$ and $||\mathbf{x}||^2 = \sum_{i=1}^{t} x_i^2$. All MRGs considered in this chapter are maximum period MRGs, that is, generators in MRG(k,p), where MRG(k,p) is the class of all maximum period MRGs of order k and modulus p.

3.2 Spectral Tests for Multiple Recursive Generators

In this section, we consider the problem of computing the spectral test for an MRG of order k. As explained in Section 2.4, the spectral distance $d_t(k)$ is traditionally computed using successive indices in an index set $I = \{0, 1, 2, 3, \ldots, t-1\}$ (see, e.g., Knuth, 1998). However, the spectral test can also be computed for nonsuccessive index set $I = \left\{j_1, j_2, \ldots, j_r\right\}$ (see, e.g., L'Ecuyer, 1997). For the remainder of this chapter, we will only refer to the traditional spectral test or spectral distance $d_t(k)$.

Throughout this chapter, we will need to apply the symmetric modulo function $[x]_p$ to some integers x for some minimization problems, and we will apply the regular modulo function for the computation of the next number X_i $(i \geq k)$ in the sequence for a MRG. Without loss of generality, for some "constant multiplier" c and for the MRG multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$, we will assume that they are expressed in "symmetric-modulus" format with their absolute value less than $p/2$.

In the next section, we will propose a simple, intuitive, and geometric interpretation of the spectral test as a minimization problem of finding a "normal vector" with the shortest length. Then in Section 3.3, we will outline a new method for computing the spectral test.

Computing the Spectral Test for LCGs

To motivate our method of computing the spectral test for MRGs of order k, we will first start with computing the spectral distance $d_2(1)$ for a LCG in (2.2), which is the same as a MRG of order $k = 1$. The maximum period of a LCG is $\rho = p - 1$. Let $\Lambda_2(I)$ in (2.5) be the set of all successive (overlapping) pairs $(X_n/p, X_{n+1}/p)$ for $n = 0, 1, 2, \cdots, \rho - 1$. Clearly, the ordered pairs in $\Lambda_2(I)$ can be covered by several parallel lines of $Bx - y = r$, where x corresponds to X_n/p, y corresponds to X_{n+1}/p, and r is the integer multiple of p such that $BX_n - X_{n+1} = rp$. The "normal vector" to these parallel lines is $\mathbf{V} = [B, -1]'$. The distance between two adjacent parallel lines is $\frac{1}{||\mathbf{V}||} = \frac{1}{\sqrt{1+B^2}}$. Shorter $||\mathbf{V}||$ imply a larger distance between some pair of adjacent parallel lines.

For an integer c, every generated output pair in $L_2(I)$ will also satisfy the equation

$$cX_i = [cB]_p X_{i-1} \bmod p.$$

Hence, the points in lattice $\Lambda_2(I)$ can also be covered by several parallel lines of $[cB]_p x - cy = r$ with the corresponding "normal vector" $\mathbf{N}_c = [c\mathbf{V}]_p = [[cB]_p, -c]$. The distance between two adjacent parallel lines is $1/||\mathbf{N}_c|| = \frac{1}{\sqrt{c^2+[cB]_p^2}}$. It is easy to see that $||[-c\mathbf{V}]_p|| = ||[c\mathbf{V}]_p||$. Hence, we can restrict the range of c to $0 < c < p/2$. Each c in this range will define a (not necessarily unique) family of parallel lines that cover all the points in $\Lambda_2(I)$. For each of these families, we can compute the distance between adjacent parallel lines $1/||\mathbf{N}_c|| = \frac{1}{\sqrt{c^2+[cB]_p^2}}$ use that family's "normal vector" $\mathbf{N}_c$.

To compute the spectral distance $d_t(k)$, we must find the value of c such that

$1/||\mathbf{N}_c|| = \frac{1}{\sqrt{c^2+[cB]_p^2}}$ is the largest among all the families of parallel lines that cover all the points in $\Lambda_2(I)$. Equivalently, we can search for the value of c for such that $||\mathbf{N}_c||$ is the smallest among all the families of parallel lines.

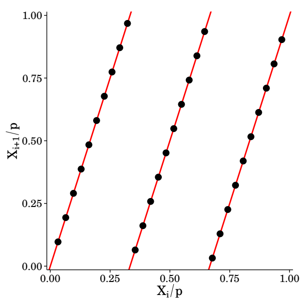
Simple Example for LCG

For the purpose of illustration, in Figure 1 we plot the successive, overlapping pairs from a LCG with multiplier $B = 3$ and modulus $p = 31$, which we denote as LCG(3,31). Each subfigure in Figure 1 represents a unique family of parallel lines that cover all the points in $\Lambda_2(I)$. Each subcaption gives the "normal vector" $\mathbf{N}_c = [c\mathbf{V}]_p$ (and integer c) corresponding to that family. The family corresponding to integer $c = 1$ in Figure 1a is the same for $1 \le c \le 5$; therefore, we only include the plot for $c = 1$. Families corresponding to $-p/2 < c < 0$ are also duplicates of the families already shown in Figure 1. Clearly, the family corresponding to $c = 1$ is the one with the largest spectral distance between adjacent, parallel lines. Consequently, it is also the family with the shortest "normal vector" $\mathbf{N}_1 = [1\mathbf{V}]_p = (3, -1)$.

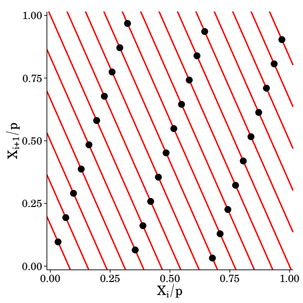
Computing the Spectral Test for MRGs

Spectral test for $t = k+1$

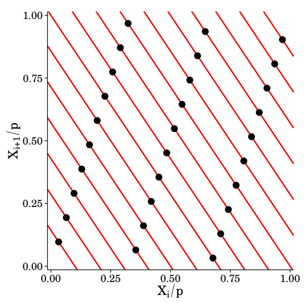
Naturally, we can extend the above procedure for computing the spectral test (for dimension $t = k+1$) to a MRG of order k defined in (2.1). First, consider $L_{k+1}(I)$ in (2.4)



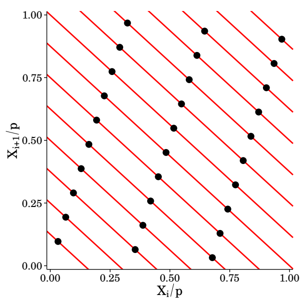
(a) $\mathbf{N}_1 = [1\mathbf{V}]_p = (3, -1)$



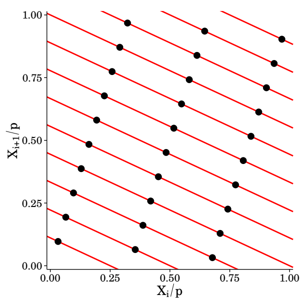
(b) $\mathbf{N}_6 = [6\mathbf{V}]_p = (-13, -6)$



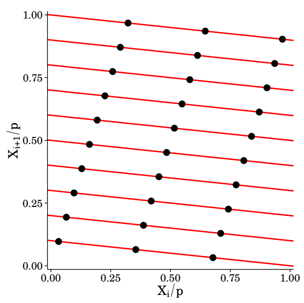
(c) $\mathbf{N}_7 = [7\mathbf{V}]_p = (-10, -7)$



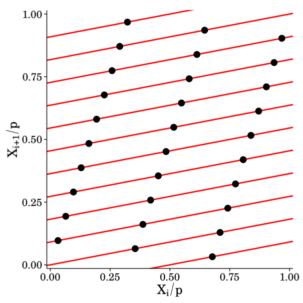
(d) $\mathbf{N}_8 = [8\mathbf{V}]_p = (-7, -8)$



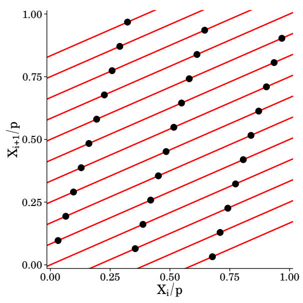
(e) $\mathbf{N}_9 = [9\mathbf{V}]_p = (-4, -9)$



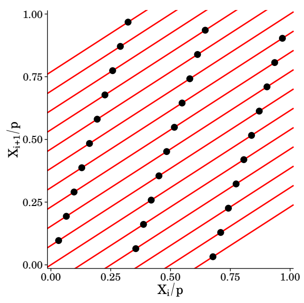
(f) $\mathbf{N}_{10} = [10\mathbf{V}]_p = (-1, -10)$



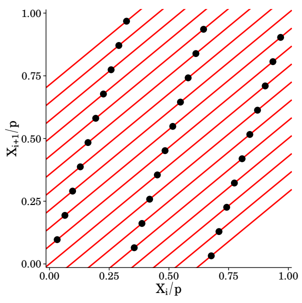
(g) $\mathbf{N}_{11} = [11\mathbf{V}]_p = (2, -11)$



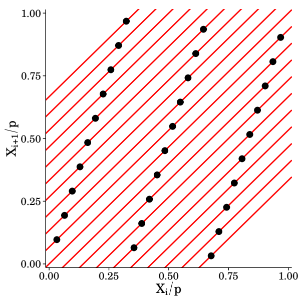
(h) $\mathbf{N}_{12} = [12\mathbf{V}]_p = (5, -12)$



(i) $\mathbf{N}_{13} = [13\mathbf{V}]_p = (8, -13)$



(j) $\mathbf{N}_{14} = [14\mathbf{V}]_p = (11, -14)$



(k) $\mathbf{N}_{15} = [15\mathbf{V}]_p = (14, -15)$

Figure 1: Families of parallel lines covering successive, overlapping pairs from LCG(3,31)

where every $(k+1)$-tuple in $L_{k+1}(I)$ in generated from a k-th ordered MRG. Next, note that the points in $\Lambda_{k+1}(I)$ form a $(k+1)$-dimensional lattice over $[0,1)^{k+1}$ where all these points can be covered by several parallel k-dimensional hyperplanes whose "normal vector" is

$$\mathbf{V} = [\alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1]'.$$

Clearly, for many integers c, every $(k+1)$-tuple in $L_{k+1}(I)$ will also satisfy the equation

$$cX_i = ([c\alpha_1]_p X_{i-1} + \cdots + [c\alpha_k]_p X_{i-k}) \bmod\ p, \quad i \geq k.$$

Hence, several families of parallel k-dimensional hyperplanes (associated with chosen c) can cover all the points in $\Lambda_{k+1}(I)$. For each family, the corresponding "normal vector" is $[c\mathbf{V}]_p$. We can find the shortest "normal vector" $[c\mathbf{V}]_p$, by computing

$$v^2_{k+1}(k) = \min_{c\neq 0} ||[c\mathbf{V}]_p||^2 = \min_{0<c<p/2} \left(\sum_{i=1}^{k} [c\alpha_i]^2_p + c^2 \right). \tag{3.2}$$

Because of the symmetric property, we can limit our search space for c by half to $0 < c < p/2$. A similar result is given in L'Ecuyer and Simard (2014), but the authors only applied it to MRGs with few nonzero terms and were only concerned with lacunary sequences dictated by the index set J in (2.6). However, this minimization problem is true for MRGs in general and can be applied even to spectral tests of successive sequences.

Once $v^2_{k+1}(k)$ is computed, the spectral distance for dimension $k+1$ is given by

$$d_{k+1}(k) = \frac{1}{v_{k+1}(k)}.$$

For any dimension t, a large $v_t(k)$ corresponds to a small $d_t(k)$, which implies more uniform coverage of the t-tuples. Therefore, for a given MRG of order k, we desire $v_t(k)$ to be as large as possible.

Since computing the spectral test is primarily concerned with finding the "normal vector" with the shortest squared length, $v_t^2(k)$, for the rest of this chapter, we will focus on $v_t^2(k)$ as the value representing the spectral test, where a larger $v_t^2(k)$ is better. We will simply call $v_t^2(k)$ the *spectral test value*.

To compute spectral test value $v_{k+1}^2(k)$ for a MRG of order k, the following algorithm is straightforward.

Algorithm to compute the spectral test value $v_{k+1}^2(k)$ for a MRG of order k

1. Initially, set $v_{\min}^2 = 1 + \sum_{i=1}^{k} \alpha_i^2$ with $c = 1$.
2. For $c = 2, 3, \cdots$, do
   (a) compute $v_c^2 = c^2 + \sum_{i=1}^{k} [c\alpha_i]_p^2$
   (b) if $v_{\min}^2 > v_c^2$, then reset $v_{\min}^2 = v_c^2$.
   (c) if $v_{\min}^2 \le (c+1)^2$, then break; else continue with next c;
3. Deliver $v_{k+1}^2(k) = v_{\min}^2$.

This algorithm is simple to implement when calculating $v_{k+1}^2(k)$ for a MRG of order k. However, as we will see next, it is not easy to generalize for dimensions larger than $k+1$.

Spectral test for $t > k$

In the previous section, the method for computing the spectral test value was to find the "normal vector" then set up the minimization problem which is simple enough to solve with a straightforward algorithm. We can extend this method to computing the spectral test in dimension $t = k + d$ for some integer d, where d stands for dimensions added to k and not the spectral distance $d_t(k)$. However, as we will see, we will need a more sophisticated algorithm to solve the minimization problem.

First, we need to define d "normal" vectors of dimension t:

$$\begin{aligned}
\mathbf{V}_1 &= [\alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1, 0, 0, \ldots, 0]', \\
\mathbf{V}_2 &= [0, \alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1, 0, \ldots, 0]', \\
&\vdots \\
\mathbf{V}_d &= [0, 0, \ldots, 0, \alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1]',
\end{aligned} \tag{3.3}$$

where $\mathbf{V}_{i+1}$ is merely a simple rotation of $\mathbf{V}_i$ for $i = 1, 2, \ldots, d-1$. We then solve the following minimization problem

$$v^2_{t=k+d}(k) = \min_{(c_1, c_2, \ldots, c_d) \neq (0,0,\ldots,0)} \left(|| \sum_{i=1}^{d} [c_i \mathbf{V}_i]_p ||^2 \right). \tag{3.4}$$

The actual value of the minimization, $v_t^2(k)$, depends on the choices of the MRG's multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$. Clearly, $v^2_{k+1}(k) \geq v^2_{k+2}(k) \geq \cdots \geq v^2_{k+d}(k)$, that is, as we calculate spectral test values $v_t^2(k)$ for dimensions t that are farther and farther away from k, we expect the number of possible t-tuples that the MRG can produce to either stay the same or to get smaller. Solving this minimization problem becomes increasingly difficult for MRGs of larger order k or for large values of d. Even more difficult is searching for the "best" multipliers (α_i's) such that $v_t^2(k)$ is largest for a given order k and modulus p.

Computing the spectral test for MRGs can be thought of as equivalent to solving the above minimization problem. In the next section, using a basis reduction algorithm, we will give a new method for computing the spectral test. As we will explain, this method is easier to implement than the one currently used.

3.3 Using Basis Reduction to Compute the Spectral Test for $t > k$

To solve the minimization problem of finding $v_t^2(k)$ for some dimension $t = k + d$, we first create a matrix whose rows correspond to a lattice basis of "normal vectors". From this basis of "normal vectors", we can find another one whose basis vectors are relatively short and nearly "orthogonal." This process is called lattice basis reduction (see, e.g., Cohen, 1993). The vector with the shortest squared length will give the spectral value $v_t^2(k)$.

To perform this lattice basis reduction, we can use the help of the popular LLL Algorithm which was proposed by Lenstra, Lenstra and Lovász (1982). Essentially, the LLL Algorithm is an integer lattice version of Gram-Schmidt orthogonalization. Several

LLL implementations are available in many software packages such as MAPLE, NTL and Sage. Entacher et al. (2002) used the LLL algorithm to compute the spectral test for the LCG. However, others (see, e.g., L'Ecuyer et al., 1993; L'Ecuyer & Couture, 1997) have used the Fincke-Pohst algorithm (Fincke & Pohst, 1985) to compute the spectral test. For discussion on differences between these two algorithms see Entacher et al. (2002) and the references cited there. Our below algorithm for finding $\nu^2_{t=k+d}(k)$ can be used with either basis-reduction algorithm. In Section 3.3, we will discuss how our method for creating a matrix of basis (row) vectors differs from the current method in the preceding references.

Algorithm for finding $\nu^2_{t=k+d}(k)$

1. [Create the initial d normal vectors] Let $\mathbf{V}_1 = [\alpha_k, \alpha_{k-1}, \dots, \alpha_1, -1, 0, \dots, 0]'$, where its last $d-1$ entries are all zero. Compute the remaining $d-1$ normal vectors, $\mathbf{V}_i = \mathbf{R}^{i-1}(\mathbf{V}_1)$, for $i = 2, 3, \dots, d$, where $\mathbf{R}^{i-1}(\mathbf{V}_1)$ denotes $i-1$ simple rotations of $\mathbf{V}_1$ as shown in (3.3).

2. [Creation of initial matrix.] Let $\mathbf{M}_0$ be an initial $d \times t$ matrix whose d rows are $\mathbf{V}'_1, \mathbf{V}'_2, \cdots \mathbf{V}'_d$.

3. [Remove columns of zeros.] Remove any columns of zeros from the initial matrix $\mathbf{M}_0$. Call the new matrix $\mathbf{M}_1$ whose dimensions will be $d \times t^*$, where t^* is the number columns left in the matrix. If there are no columns of only zeros in $\mathbf{M}_0$, then $t^* = t$ and $\mathbf{M}_1 = \mathbf{M}_0$.

4. [Create final matrix for basis reduction.] Let $\mathbf{M}$ be a $t^* \times t^*$ matrix whose first

$(t^* - d)$ rows are $p\mathbf{e}_{i(t^*)}$, where p is the modulus, $\mathbf{e}_{i(t^*)}$ is the i-th unit vector of dimension t^*, and i corresponds to the row number $i = 1, 2, \ldots, (t^* - d)$. Let the remaining d rows correspond to the rows in $\mathbf{M}_1$.

5. [Basis reduction.] Apply the LLL algorithm (or some other basis reduction procedure) to matrix $\mathbf{M}$ which yields a reduced matrix $\mathbf{M}^* = \text{LLL}(\mathbf{M})$. The spectral test value $v_t^2(k)$ is squared length of the shortest row vector in $\mathbf{M}^*$.

Together, Steps 1 and 2 create our initial $d \times t$ matrix of "normal vectors" which are merely rotations of $\mathbf{V}_1$. If any columns of zeros are present in $\mathbf{M}_0$, then we remove them in Step 3, because the zero columns do not contribute to solving the minimization problem. Step 4 adds the necessary number of unit vectors (times p) such that $\mathbf{M}$ is a basis of "normal vectors." Finally, Step 5 uses LLL (or preferred algorithm) to reduce the basis $\mathbf{M}$ to $\mathbf{M}^*$ such that the shortest row vector in $\mathbf{M}^*$ yields the $v_t^2(k)$.

Simple example for MRG

To illustrate this algorithm, consider computing the spectral test for three dimensions beyond $k = 5$ for the following small order MRG.

$$X_i = \alpha_1 X_{i-1} + \alpha_2 X_{i-2} + \alpha_3 X_{i-3} + \alpha_4 X_{i-4} + \alpha_5 X_{i-5} \bmod p, \quad i \geq 5. \tag{3.5}$$

The first corresponding "normal vector" will be

$$\mathbf{V}_1 = [\alpha_5, \alpha_4, \alpha_3, \alpha_2, \alpha_1, -1, 0, 0]' \tag{3.6}$$

$\mathbf{V}_2$ and $\mathbf{V}_3$ can be easily found by rotating the elements in $\mathbf{V}_1$ once and twice, respectively. Then we will have

$$\mathbf{M}_0 = \begin{bmatrix} \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 & 0 \\ 0 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 \\ 0 & 0 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 \end{bmatrix}$$

Since $\mathbf{M}_0$ has no columns of all zeros to remove, then $\mathbf{M}_1 = \mathbf{M}_0$. The final matrix to submit to LLL will be

$$\mathbf{M} = \begin{bmatrix} p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 0 & 0 & 0 \\ \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 & 0 \\ 0 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 \\ 0 & 0 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 \end{bmatrix}$$

This example showcases the simplicity of the above algorithm. For larger order MRGs (with many nonzero terms), a large matrix of basis vectors will need to be created. The above algorithm can create this rather efficiently. In contrast, creating a large matrix of basis vectors using the current method is very tedious as we will see in the next section.

Comparison with Current Method

Kao and Tang (1997a), Knuth (1981, 1998), L'Ecuyer (1997), and L'Ecuyer and Couture(1997) all use a similar method for computing the spectral test. Currently, the method for computing t-dimensional spectral tests for MRGs requires finding a dual basis of the lattice basis for $L_t(I)$ (see, e.g., L'Ecuyer & Couture, 1997).

First, let **A** equal the transpose of the MRG's companion matrix in (2.17)

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & \dots & 0 & \alpha_k \\ 1 & 0 & \dots & 0 & \alpha_{k-1} \\ 0 & 1 & \dots & 0 & \alpha_{k-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & \alpha_1 \end{bmatrix},$$

and let $[\mathbf{A}]_{ij}$ denote the ij^{th} element of matrix **A**. Next, for $t = k + d$, we define $(t \times t)$ matrix **V** whose rows are t-dimensional basis vectors of $L_t(I) + p\mathbb{Z}^t$, which is the periodic continuation of $L_t(I)$ in (2.4) with period p. Here, $\mathbb{Z}^t$ denotes the t-dimensional vector whose elements are any integer in set of all integers $\mathbb{Z}$. The matrix **V** is defined as

$$\mathbf{V} = \begin{bmatrix} 1 & 0 & \cdots & 0 & \alpha_k & [\mathbf{A}^2]_{1k} \bmod p & \cdots & [\mathbf{A}^d]_{1k} \bmod p \\ 0 & 1 & \cdots & 0 & \alpha_{k-1} & [\mathbf{A}^2]_{2k} \bmod p & \cdots & [\mathbf{A}^d]_{2k} \bmod p \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & \alpha_1 & [\mathbf{A}^2]_{kk} \bmod p & \cdots & [\mathbf{A}^d]_{kk} \bmod p \\ 0 & 0 & \cdots & 0 & p & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & p & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & p \end{bmatrix}.$$

From basis matrix **V** construct dual basis matrix **W** such that

$\mathbf{V}_i'\mathbf{W}_j = p * \text{Kroenecker's } \delta_{ij}$

$$\mathbf{W} = \begin{bmatrix} p & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & p & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p & 0 & 0 & \cdots & 0 \\ \alpha_k & \alpha_{k-1} & \cdots & \alpha_1 & -1 & 0 & \cdots & 0 \\ [\mathbf{A}^2]_{1k} \bmod p & [\mathbf{A}^2]_{2k} \bmod p & \cdots & [\mathbf{A}^2]_{kk} \bmod p & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ [\mathbf{A}^d]_{1k} \bmod p & [\mathbf{A}^d]_{2k} \bmod p & \cdots & [\mathbf{A}^d]_{kk} \bmod p & 0 & 0 & \cdots & -1 \end{bmatrix}.$$

Finally, solve the following minimization problem

$$\nu_{k+d}^2(k) = \min_{(z_1,z_2,\ldots,z_d)\neq(0,0,\ldots,0)} \left(|| \sum_{i=1}^{d} z_i \mathbf{W}_i \pmod p ||^2 \right),$$

where $z_i \in \{-(p-1),\ldots,(p-1)\}$ and

$$\mathbf{W}_1 = [\alpha_k, \alpha_{k-1}, \ldots, \alpha_1, -1, 0, 0, \ldots, 0]',$$

$$\mathbf{W}_2 = [[\mathbf{A}^2]_{1k}, [\mathbf{A}^2]_{2k}, \ldots, [\mathbf{A}^2]_{kk}, 0, -1, \ldots, 0]' \pmod p,$$

$$\vdots$$

$$\mathbf{W}_d = [[\mathbf{A}^d]_{1k}, [\mathbf{A}^d]_{2k}, \ldots, [\mathbf{A}^d]_{kk}, 0, 0, \ldots, -1]' \pmod p.$$

When $t = k+1$, the proposed method solves a similar minimization problem as the current method. For $t > k+1$, the current procedure requires raising a $k \times k$ matrix to a some power. When k is large, this procedure will be difficult to implement. Furthermore, the current method lacks an intuitive explanation for what is being minimized and how the minimization problem relates to the spectral test. Overall, our approach is similar, except we propose a simpler way to find the dual basis to the lattice points $L_t(I)$ in (2.4). Lastly, as we will see in the next section, for some special classes of large order MRGs with few nonzero terms, the proposed method can be substantially more efficient.

3.4 Spectral test for DX-k-s Generators

As discussed in Chapter 2.6, FMRG-k was proposed by Deng and Lin (2000) and Deng and Xu (2003) proposed DX-k-s generators as two classes of portable, efficient, and maximum-period MRGs. For the DX-k-s class, s is the number of terms sharing the multiplier B.

Spectral test for DX-k-s when $t = k+1$

When computing the spectral test value $v^2_{k+1}(k)$ for FMRG-k, the "normal vector" **V** will consist of one entry of 1, one entry of B, and the rest zeros except the last entry which will be -1. For DX-k-s the "normal vector" **V** will consist of s entries of B and the rest zeros except for the final entry which will be -1.

Thus, the minimization problem in (3.2) can be further reduced. For FMRG-k, we have

$$v^2_{k+1}(k) = \min_{0<c<p/2}\left([cB]^2_p + 2c^2\right)$$

and for DX-k-s, we have

$$v^2_{k+1}(k) = \min_{0<c<p/2}\left(s[cB]^2_p + c^2\right). \tag{3.7}$$

For convenience, we will mainly discuss the minimization problem for DX-k-s in (3.7). However, the results in this section that apply to DX-k-s generators will also apply to FMRG-k generators. Note that $v_{k+1}^2(k)$ in (3.7) depends only on the parameters B and s, not on the order k.

Note also that for LCG in (2.2), we have a very similar form:

$$v_2^2 = \min_{0<c<p/2} \left([cB]_p^2 + c^2 \right).$$

The lattice structure for LCG has been studied extensively in the literature (see, e.g., Entacher et al., 2005; Sezgin 1996, 2004, 2006). From this extensive study, we know that the LCG with multiplier B tends to have a bad (small) v_2^2 whenever B is close to the value of pN/D with small numerator N (see, e.g., Sezgin, 2004). The study of "bad" lattice structures for LCGs can also be useful to avoid choosing a bad multipliers B for FMRG-k and DX-k-s generators. We remark that for higher values of $t = k + d$, the lattice structure for a LCG is getting worse whereas the lattice structure for FMRG-k and DX-k-s generators remains steady as will be shown in Section 3.4. But first, we will need to show how to compute the $(k+2)$-dimensional spectral test value for DX-k-s generators which is similar for FMRG-k generators.

Spectral test for DX-k-s when $t = k+2$

For DX-k-s generators, the minimization problem in (3.4) for $t = k+2$ can be reduced to

$$v_{k+2}^2(k) = \min_{(c_1,c_2)\neq(0,0)} \left((s[c_1B]_p^2 + c_1^2) + (s[c_2B]_p^2 + c_2^2) - 2c_1[c_2B]_p \right).$$

Similar to the case for finding $v^2_{k+1}(k)$, it is pretty straight forward to solve this minimization problem. However, we can also compute $v^2_{k+2}(k)$ using the algorithm given in Section 3.3.

Simple Example for DX-k-2

For example, consider a DX-k-2 generator for some k. The normal vectors will be

$$\mathbf{V}_1 = [B, 0, 0, \ldots, 0, B, -1, 0]'$$

$$\mathbf{V}_2 = [0, B, 0, 0, \ldots, 0, B, -1]'$$

where there are many zeros between B's. Therefore, there will be many zero columns in $\mathbf{M}_0$ that we can remove resulting in

$$\mathbf{M}_1 = \begin{bmatrix} B & 0 & B & -1 & 0 \\ 0 & B & 0 & B & -1 \end{bmatrix}$$

whose number of columns have been drastically reduced from the initial matrix $\mathbf{M}_0$, especially if k was large. Next, to compute $v^2_{k+2}(k)$, we simply apply LLL to the much smaller matrix

$$\mathbf{M} = \begin{bmatrix} p & 0 & 0 & 0 & 0 \\ 0 & p & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 \\ B & 0 & B & -1 & 0 \\ 0 & B & 0 & B & -1 \end{bmatrix}.$$

Thus, the special structure of DX-k-s generators allows us to compute the spectral test efficiently for even large orders of k. As we will show next, this special structure also allows for another special property.

Consistency property for DX-k-s

As noted earlier, $v_{k+1}^2(k) \geq v_{k+2}^2(k)$ for any MRG of order k. For many (but not all) DX-k-s generators with multiplier B and k not too small, we have observed that

$$v_{k+1}^2(k) = v_{k+2}^2(k).$$

We will call this property the *consistency* of the spectral test value for the DX generators. As a simple experiment, we randomly selected 5000 B's as possible multipliers for the DX-k-s generators. For $s = 2$, 82.92% of the DX-k-2 generators had the consistency property. For DX-k-3, 96.24% of the generators had this property. For DX-k-4 generators, all 5000 generators had the consistency property. Since there are (rare) exceptions, it is impossible to have a general theory. However, we can provide some intuitive explanations for these observations:

1. For DX-k-s generators, two vectors $[c_1\mathbf{V}_1]_p$ and $[c_2\mathbf{V}_2]_p$ are "nearly" linearly "orthogonal" to each other because there is only one common entry with both nonzero values. The inner product between these two vectors $[c_1\mathbf{V}_1]_p$ and $[c_2\mathbf{V}_2]_p$ is $-2c_1[c_2B]_p$.

2. The value of $v_{k+2}^2(k)$ can be further reduced if we can choose c_1 and c_2 so that

$-2c_1[c_2B]_p$ is a small enough negative number so that $2c_1[c_2B]_p > s[c_1B]_p^2 + [c_1]_p^2$ or $2c_1[c_2B]_p > s[c_2B]_p^2 + [c_2]_p^2$. As s changes from 2 to 3 or 4, the "chance" of finding such c_1 and c_2 is greatly reduced. This is consistent with our empirical finding above where the percentage of DX-k-s with $v_{k+2}^2(k) < v_{k+1}^2(k)$ decreased as s increased, i.e, the percentage DX-k-s where $v_{k+2}^2(k) < v_{k+1}^2(k)$ was 17.08% for $s = 2$, 3.76% for $s = 3$, and 0% for $s = 4$. Clearly, the actual percentages could vary if we performed the above simple experiment with a different sample of multipliers B.

3. Geometrically speaking, for any two vectors, $\mathbf{V}$ and $\mathbf{W}$, $||\mathbf{V}+\mathbf{W}||^2$ can be smaller than $||\mathbf{V}||^2$ and $||\mathbf{W}||^2$ only when the "angle" between two vectors is larger than 90 degrees. Since the two vectors $[c_1\mathbf{V}_1]_p$ and $[c_2\mathbf{V}_2]_p$ are almost "orthogonal" for DX generators, it is harder to find c_1 and c_2 such that $||[c_1\mathbf{V}_1]_p + [c_2\mathbf{V}_2]_p||^2$ is shorter than $||[c_1\mathbf{V}_1]_p||^2$ or $||[c_2\mathbf{V}_2]_p||^2$, where $v_{k+2}^2(k)$ corresponds to minimizing $||[c_1\mathbf{V}_1]_p + [c_2\mathbf{V}_2]_p||^2$ with respect to some (c_1, c_2) and $v_{k+1}^2(k)$ is the same as minimizing $||[c_1\mathbf{V}_1]_p||^2$ with respect to c_1 or, equivalently, the same as minimizing $||[c_2\mathbf{V}_2]_p||^2$ with respect to c_2.

We remark that some FMRG-k generators also exhibit the consistency property, and this can be seen by using similar arguments listed above. When computing the spectral test value $v_{k+3}^2(k)$ for MRGs of order k, we would expect the following decreasing property for the spectral values:

$$v_{k+1}^2(k) \geq v_{k+2}^2(k) \geq v_{k+3}^2(k).$$

Using the same geometric arguments, however, the consistency property extends to $v_{k+3}^2(k)$ for DX-k-s generators. Indeed, among the 5000 B's tested, there is 100% "consistency" between the values of $v_{k+2}^2(k)$ and $v_{k+3}^2(k)$ for all DX-k-s generators. In fact, the consistency property is expected to hold for $v_{k+d}^2(k)$ for several dimensions beyond k until there is a change in relative relationship on the "angles" (inner-products) among these vectors of $\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_d$.

For several dimensions beyond k, MRGs with the consistency property will have the exact same spectral value. Therefore, MRGs with the consistency property are easier to rank for a given order k (and modulus p). If a set of MRGs all have the consistency property, then the one with the best spectral test value in dimension $k+1$ will be the best for several dimensions beyond k. This is not true for MRGs in general where an MRG with the best spectral test value in dimension $k+1$ can have a worse spectral value in dimension $k+2$ (see, e.g., Knuth, 1998).

In the next section, we search for "better" FMRG-k and DX-k-s generators by screening for multipliers B such that the generator has a spectral test value smaller than a pre-specified cutoff and possesses the consistency property.

3.5 List of better FMRG-k and DX-k-s generators

We use the proposed algorithm in Section 3.3 to search for "better" FMRG-k and DX-k-s generators. Recently, Deng et al. (2012a, 2012b) found several FMRG-k and DX-k-s generators of large orders. While these generators have the equi-distribution property up to dimension k, several of these generators have poor spectral test values for dimensions t beyond k. Using the algorithm in Section 3.3, we searched for a new set of FMRG-k and DX-k-s generators with reasonably good spectral distances $d_{k+1}(k)$. We remark that computing the spectral distance is much easier than finding a generator with the maximum period. Therefore, we first screened potential multipliers for a spectral distance $d_{k+1}(k)$ below a pre-specified bound (say, 3e-05 or 2e-05) and verified that the generator possessed the consistency property. After passing these two checks, then we proceeded to check whether the generator had the maximum period using methods described in Deng et al. (2012a, 2012b). Table 1 through Table 4 list "better" FMRG-k and DX-k-s generators.

Table 1: List of FMRG-k and DX-k-2 with $B < 2^{20}$ and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)$

k	p	FMRG-k				DX-k-2			
		$B < 2^{20}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$	$B < 2^{20}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$
47	2147483647	1047527	1.87e-05	1073719468	2.24e-05	1047104	1.98e-05	1073718369	1.90e-05
643	2147483647	1047252	1.91e-05	1073718413	2.14e-05	1048207	2.00e-05	1073690591	1.91e-05
1597	2147483647	1016724	2.22e-05	1073716054	1.94e-05	777051	1.90e-05	1073718474	1.97e-05
7499	2147483647	967501	1.91e-05	1073708416	2.16e-05	336838	1.87e-05	1073381825	1.84e-05
20897	2147483647	820866	2.17e-05	1073616009	2.75e-05	1028880	2.57e-05	1073658320	2.28e-05
101	2147400803	1047864	2.16e-05	1073678105	2.24e-05	1048093	1.90e-05	1073677156	1.99e-05
211	2146642319	1047749	2.02e-05	1073721711	1.97e-05	1044526	1.83e-05	1073719650	1.81e-05
307	2147431103	1045846	1.87e-05	1073691791	2.10e-05	1047799	1.92e-05	1073692149	1.99e-05
401	2147426459	1048196	2.14e-05	1073689547	2.11e-05	1033380	1.92e-05	1073689977	1.91e-05
503	2147309159	1048331	2.14e-05	1073720383	2.12e-05	1022698	1.79e-05	1073679109	1.84e-05
601	2146156163	1043822	1.91e-05	1073719452	1.92e-05	1042172	2.00e-05	1073712188	1.96e-05
701	2147262983	1046874	1.86e-05	1073717689	2.21e-05	1040476	1.89e-05	1073680615	1.82e-05
809	2145472859	1043034	2.08e-05	1073719786	2.08e-05	985209	1.89e-05	1073706375	2.00e-05
907	2143082759	1046522	1.83e-05	1073721128	2.14e-05	1032051	1.98e-05	1073718871	1.92e-05
1009	2145114779	1045259	2.15e-05	1073716427	1.99e-05	1014011	1.95e-05	1073683017	1.98e-05
1103	2140167287	1045590	2.12e-05	1073708831	2.04e-05	1028217	1.91e-05	1073718798	1.93e-05
1201	2146369943	1044395	2.16e-05	1073711360	2.18e-05	1034308	1.86e-05	1073634611	1.96e-05
1301	2146412747	1038695	2.08e-05	1073714389	2.03e-05	1019650	1.91e-05	1073694173	1.84e-05
1409	2143163459	1023942	1.89e-05	1073721338	1.93e-05	1015803	1.89e-05	1073653955	2.00e-05
1511	2144712443	1027601	1.91e-05	1073703279	2.16e-05	1033442	1.99e-05	1073711638	1.96e-05
1601	2147114687	1048172	2.15e-05	1073706909	1.93e-05	1030332	1.93e-05	1073631425	1.89e-05
1709	2146451207	1037856	1.96e-05	1073711506	2.17e-05	1001582	1.99e-05	1073379700	1.99e-05
1801	2141694407	1026509	2.22e-05	1073702501	1.97e-05	1040074	1.99e-05	1073688661	1.93e-05
1901	2147216327	1047198	1.92e-05	1073719969	1.76e-05	1002056	1.88e-05	1073576432	1.95e-05
2003	2147438687	1040900	1.95e-05	1073694634	2.02e-05	964935	1.86e-05	1073695597	1.84e-05
2111	2143947263	1048318	1.90e-05	1073719627	1.96e-05	1023929	1.85e-05	1073516503	1.84e-05
2203	2141440559	1041675	1.89e-05	1073709809	1.77e-05	1034145	1.88e-05	1073687710	1.95e-05
2309	2147143463	1046953	1.76e-05	1073719650	2.16e-05	980142	1.79e-05	1073716624	1.91e-05
2411	2138227199	1046643	2.05e-05	1073720277	2.02e-05	960701	1.91e-05	1073681521	1.84e-05

Table 2: List of FMRG-k and DX-k-2 with $B < 2^{20}$ and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)$

k	p	FMRG-k				DX-k-2			
		$B < 2^{20}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$	$B < 2^{20}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$
2503	2133944399	1045562	2.10e-05	1073719199	1.90e-05	1039452	1.87e-05	1073702140	1.92e-05
2609	2138671967	1032235	2.15e-05	1073719690	1.91e-05	921424	1.94e-05	1073719923	1.89e-05
2707	2146370063	1048221	1.80e-05	1073697994	2.19e-05	923007	2.00e-05	1073642864	1.99e-05
2801	2146388039	1040902	2.07e-05	1073721289	1.87e-05	1029014	1.87e-05	1073715771	1.98e-05
2903	2133427823	1048504	1.97e-05	1073720598	1.88e-05	620667	2.00e-05	1073655527	1.89e-05
3001	2144425247	1048008	2.18e-05	1073693367	2.24e-05	973895	1.90e-05	1073531231	1.85e-05
3109	2140742519	1028657	1.88e-05	1073702701	2.14e-05	842603	1.91e-05	1073609758	1.98e-05
3203	2142764759	1038126	2.19e-05	1073707984	2.19e-05	1045174	1.83e-05	1073494301	1.87e-05
3301	2132602463	1029102	1.98e-05	1073700259	2.21e-05	927480	1.94e-05	1073599925	1.84e-05
3407	2141240639	1025440	1.83e-05	1073720463	1.85e-05	1036658	1.82e-05	1073570507	1.97e-05
3511	2146070687	1044201	1.76e-05	1073699334	1.87e-05	931391	1.82e-05	1073696749	1.96e-05
3607	2146457063	1016279	2.02e-05	1073716724	1.77e-05	1000578	2.00e-05	1073593347	1.98e-05
3701	2135907023	1025699	2.21e-05	1073715028	2.00e-05	1002227	1.98e-05	1073619234	1.86e-05
3803	2115425519	1012956	1.78e-05	1073713780	2.07e-05	1044969	1.96e-05	1073715220	1.89e-05
3907	2130101999	1017573	1.88e-05	1073697705	1.84e-05	980525	1.96e-05	1073545500	1.89e-05
4001	2143071167	1044560	1.85e-05	1073666248	2.18e-05	694433	1.98e-05	1073431583	1.99e-05
5003	2146224359	1033689	1.81e-05	1073727083	2.11e-05	1033485	2.47e-05	1073741516	2.14e-05
6007	2137498943	1035429	2.16e-05	1073737595	1.94e-05	1015366	1.84e-05	1073682415	1.87e-05
7001	2146873559	1010706	2.15e-05	1073709808	1.79e-05	954617	2.69e-05	1073720260	2.30e-05
8009	2142326903	1041446	2.05e-05	1073717208	2.46e-05	960779	2.18e-05	1073724902	2.35e-05
9001	2140247399	1045508	2.37e-05	1073737583	2.44e-05	1029903	2.37e-05	1073682825	2.30e-05
10007	2147051903	954436	1.90e-05	1073688561	2.09e-05	987064	2.15e-05	1073702542	2.17e-05
11003	2146207223	1006998	1.93e-05	1073664067	1.81e-05	1047362	2.74e-05	1073730907	2.47e-05
12007	2109950867	990386	2.75e-05	1073714070	1.93e-05	879930	2.13e-05	1073692289	2.41e-05
13001	2147191153	1034426	2.06e-05	1073566489	1.91e-05	827268	2.06e-05	1073621150	2.34e-05
14009	2146857347	1046516	2.66e-05	1073730141	2.74e-05	1041819	2.20e-05	1073738204	2.13e-05
15013	2138487383	1030728	1.98e-05	1073724930	1.85e-05	993180	2.13e-05	1073723417	1.93e-05
20011	2121351707	938904	2.32e-05	1073528611	2.44e-05	687847	2.24e-05	1073460800	2.11e-05
25013	2135944739	1007372	1.91e-05	1073707771	2.06e-05	969323	2.15e-05	1073692717	2.56e-05

Table 3: List of DX-k-3 and DX-k-4 with $B < 2^{19}$ and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)$

k	p	DX-k-3				DX-k-4			
		$B < 2^{19}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$	$B < 2^{19}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$
47	2147483647	523431	1.71e-05	1073721764	1.64e-05	500675	1.50e-05	1073710999	1.45e-05
643	2147483647	515035	1.71e-05	1073720605	1.62e-05	374621	1.46e-05	1073630573	1.48e-05
1597	2147483647	494876	1.58e-05	1073676583	1.73e-05	351486	1.49e-05	1073555198	1.49e-05
7499	2147483647	458125	1.67e-05	1073512119	1.72e-05	519708	1.83e-05	1072885813	1.46e-05
20897	2147483647	88085	1.81e-05	1073546709	2.14e-05	514809	1.86e-05	1073718732	1.97e-05
101	2147400803	519060	1.73e-05	1073721359	1.74e-05	508951	1.47e-05	1073719687	1.49e-05
211	2146642319	524256	1.67e-05	1073716159	1.66e-05	522067	1.49e-05	1073716794	1.48e-05
307	2147431103	522571	1.73e-05	1073695949	1.72e-05	523974	1.49e-05	1073605522	1.46e-05
401	2147426459	521532	1.55e-05	1073683219	1.68e-05	520121	1.43e-05	1073695179	1.49e-05
503	2147309159	517960	1.71e-05	1073718398	1.64e-05	484552	1.46e-05	1073687532	1.50e-05
601	2146156163	516820	1.70e-05	1073718565	1.74e-05	515881	1.48e-05	1073623410	1.49e-05
701	2147262983	503998	1.65e-05	1073705319	1.72e-05	493716	1.43e-05	1073712974	1.48e-05
809	2145472859	498800	1.65e-05	1073718045	1.63e-05	392650	1.50e-05	1073687422	1.48e-05
907	2143082759	496916	1.67e-05	1073717140	1.72e-05	467541	1.45e-05	1073528220	1.46e-05
1009	2145114779	521562	1.62e-05	1073713141	1.68e-05	409391	1.45e-05	1073606414	1.49e-05
1103	2140167287	499560	1.71e-05	1073707666	1.66e-05	443332	1.49e-05	1073663872	1.46e-05
1201	2146369943	495592	1.67e-05	1073708180	1.72e-05	429035	1.46e-05	1073700353	1.49e-05
1301	2146412747	518942	1.62e-05	1073708245	1.72e-05	263182	1.49e-05	1073675769	1.48e-05
1409	2143163459	523322	1.69e-05	1073662387	1.62e-05	257155	1.47e-05	1073402202	1.47e-05
1511	2144712443	507002	1.73e-05	1073708454	1.61e-05	504720	1.48e-05	1073222617	1.50e-05
1601	2147114687	427322	1.65e-05	1073695970	1.58e-05	485162	1.45e-05	1073511216	1.47e-05
1709	2146451207	512537	1.66e-05	1073703115	1.70e-05	158921	1.50e-05	1073570307	1.45e-05
1801	2141694407	505746	1.67e-05	1073710164	1.61e-05	392186	1.49e-05	1073687233	1.47e-05
1901	2147216327	467917	1.67e-05	1073708038	1.71e-05	193527	1.44e-05	1073402948	1.48e-05
2003	2147438687	519539	1.74e-05	1073697033	1.74e-05	33799	1.48e-05	1073081401	1.49e-05
2111	2143947263	517247	1.69e-05	1073634269	1.72e-05	511415	1.48e-05	1073414834	1.47e-05
2203	2141440559	463037	1.70e-05	1073649719	1.73e-05	403080	1.49e-05	1073599725	1.44e-05
2309	2147143463	524185	1.71e-05	1073643091	1.74e-05	240647	1.44e-05	1073633650	1.46e-05
2411	2138227199	497168	1.65e-05	1073696807	1.69e-05	126843	1.47e-05	1073708619	1.49e-05

Table 4: List of DX-k-3 and DX-k-4 with $B < 2^{19}$ and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)$

k	p	DX-k-3				DX-k-4			
		$B < 2^{19}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$	$B < 2^{19}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$
2503	2133944399	469703	1.67e-05	1073721366	1.63e-05	429166	1.48e-05	1073440776	1.45e-05
2609	2138671967	478085	1.64e-05	1073657472	1.61e-05	493819	1.49e-05	1073687130	1.48e-05
2707	2146370063	522221	1.63e-05	1073713190	1.56e-05	465462	1.44e-05	1073575014	1.49e-05
2801	2146388039	516134	1.61e-05	1073665827	1.69e-05	470279	1.44e-05	1073556130	1.46e-05
2903	2133427823	499182	1.72e-05	1073700199	1.70e-05	504081	1.49e-05	1073542942	1.46e-05
3001	2144425247	522512	1.60e-05	1073620638	1.60e-05	98371	1.47e-05	1073610508	1.50e-05
3109	2140742519	425914	1.75e-05	1073572260	1.67e-05	164379	1.47e-05	1073613673	1.45e-05
3203	2142764759	502470	1.72e-05	1073694874	1.71e-05	516915	1.47e-05	1073458794	1.45e-05
3301	2132602463	375801	1.60e-05	1073697276	1.71e-05	455177	1.48e-05	1073371940	1.50e-05
3407	2141240639	419455	1.71e-05	1073652217	1.66e-05	483510	1.49e-05	1073649297	1.48e-05
3511	2146070687	486256	1.64e-05	1073623359	1.65e-05	479412	1.48e-05	1073525158	1.48e-05
3607	2146457063	515337	1.75e-05	1073658421	1.65e-05	485302	1.48e-05	1073397131	1.48e-05
3701	2135907023	421586	1.71e-05	1073559996	1.60e-05	447339	1.50e-05	1073428871	1.49e-05
3803	2115425519	439856	1.64e-05	1073631777	1.65e-05	209040	1.44e-05	1073455831	1.49e-05
3907	2130101999	491181	1.62e-05	1073648923	1.74e-05	131958	1.46e-05	1073464874	1.47e-05
4001	2143071167	481746	1.73e-05	1073721166	1.65e-05	520508	1.50e-05	1072551829	1.48e-05
5003	2146224359	479985	2.00e-05	1073727160	2.00e-05	483758	2.11e-05	1073729995	1.55e-05
6007	2137498943	474028	2.19e-05	1073719733	1.64e-05	519501	2.04e-05	1073733401	1.60e-05
7001	2146873559	512813	1.58e-05	1073714433	1.91e-05	499426	2.09e-05	1073735327	1.73e-05
8009	2142326903	469107	1.95e-05	1073638001	1.94e-05	489163	2.10e-05	1073719175	2.05e-05
9001	2140247399	507398	1.96e-05	1073733156	1.78e-05	375998	1.67e-05	1073717006	2.05e-05
10007	2147051903	515671	2.42e-05	1073654940	1.80e-05	493723	1.97e-05	1073713517	2.11e-05
11003	2146207223	468559	2.19e-05	1073663378	1.85e-05	474179	1.70e-05	1073698910	1.46e-05
12007	2109950867	382970	1.91e-05	1073740063	1.68e-05	521891	1.65e-05	1073718486	1.47e-05
13001	2147191153	352845	1.57e-05	1073695479	1.63e-05	385071	1.62e-05	1073737394	1.54e-05
14009	2146857347	380859	1.69e-05	1073736613	1.69e-05	467248	1.89e-05	1073685011	1.99e-05
15013	2138487383	419740	2.34e-05	1073642398	2.36e-05	481133	1.64e-05	1073711351	2.22e-05
20011	2121351707	404150	1.82e-05	1073727681	1.88e-05	481952	1.89e-05	1073717310	1.60e-05
25013	2135944739	222202	1.82e-05	1073684077	1.76e-05	490509	1.90e-05	1073565806	1.53e-05

3.6 Discussion

Concerning testing the quality of a random number generators with a linear recurrence, Knuth (1998) said that the spectral test "is by far the most powerful test known." In this chapter, we give an intuitive, geometric explanation of how to compute the t-dimensional spectral test using "normal vectors" that are perpendicular to families of equidistant, parallel $(t-1)$-dimensional hyperplanes that cover all the t-dimensional lattice points, which are created by taking successive subsequences of output from the MRG. An algorithm is given in Section 3.3 for computing the spectral test for MRGs. This algorithm is easy to implement and (for some special classes of large order MRGs with few nonzero terms) can be more efficient than the current method for computing the spectral test. Using this algorithm we observe that FMRG-k and DX-k-s generators have a consistency property where the spectral test value remains the same for several dimensions beyond k. We list "better" FMRG-k and DX-k-s generators that have the consistency property and relatively small spectral test values for their respective class and order k.

Chapter 4

Efficient and Parallel Implementation of General Multiple Recursive Generators with Many Nonzero Terms

4.1 Introduction

As explained in Section 2.2, maximum period MRGs are thought to be good random number generators. They have great empirical performance, strong statistical justification, long periods, and the equi-distribution up to order k. However, maximum period MRGs can be inefficient and difficult to implement in parallel. A direct implementation of the recursion could require up to k multiplications to produce one number. The general parallelization method in (2.19) can be difficult for large k as it requires raising a $k \times k$ matrix to some power. If the recursion requires up to k multiplications, then the AGM parallelization method in (2.20) will require the same.

As we saw in Section 2.6, most research in this area has tried to side-step these difficulties by searching and implementing MRGs with some special structure such that the linear recurrence can be implemented by an MRG with few non-zero terms. Fewer non-zero terms requires fewer costly multiplications when generating the recursion. However, as Kao and Tang (1997) point out, MRGs with few nonzero have spectral distances $d_{k+1}(k)$ that are worse than what is expected from MRGs with many nonzero terms. To have small spectral distances $d_t(k)$ for dimension $t > k$, L'Ecuyer (1997) gave the necessary but not sufficient condition that the sum of squares of the MRG multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$ in (2.1) must be large.

By Theorem 1 in Section 2.7, we can find numerous maximum period MRGs with many nonzero terms from the companion matrix $\mathbf{M}_f$ of one maximum period MRG, even if the starting MRG has few nonzero terms. However, without an efficient implementation of these MRGs, this fecund theorem has not frequently been drawn upon to find new general MRGs with many nonzero terms. As explained in Section 2.8, maximum period MCGs can be constructed from the companion matrix $\mathbf{M}_f$ of a maximum period MRG. Generating numbers k at a time from the recursion of a MCG contructed this way will yield a parallel implementation of the MRG. However, this parallel MCG implementation can be very inefficient (see Section 2.8). In this chapter, we will describe a procedure to find numerous maximum period MRGs with many nonzero terms and give an efficient and parallel MCG implementation for these MRGs.

Some Notation

Throughout this chapter, we let p be a large prime number and $\mathbb{Z}_p = \{0, 1, 2, \cdots, p-1\}$ be the finite field of p elements under the usual modulus operations of addition and multiplication. We will use $\mathbb{Z}_p^k$ and $\mathbb{Z}_p^{k\times k}$ to denote the set of k-dimensional vectors and the set of $k \times k$ matrices, respectively, with elements in $\mathbb{Z}_p$. We let MRG(k, p) denote the class of maximum period k-th order MRGs with prime modulus p. We use $\phi(x)$ to denote the Euler totient function, which gives number of integers between 1 and x that are relatively prime to x.

4.2 Finding Numerous MRGs with Many Nonzero Terms

To use Theorem 1 in Section 2.7, we need to start with a generator in MRG(k, p). Section 2.6 lists many references to generators in MRG(k, p) with few nonzero terms whose primitive polynomials are already given. Starting with one of these primitive polynomials $f(x)$, we can raise its corresponding companion matrix $\mathbf{M}_f$ to some power r with r coprime to $p^k - 1$, and find $f_r(x) = \det(x\mathbf{I} - \mathbf{M}_f^r)$. According to Theorem 1, $f_r(x)$ is a primitive polynomial. The primitive polynomial $f_r(x)$ will define a new generator in MRG(k, p). The resulting maximum period MRG is very likely to have many nonzero terms. Let MRG-k denote the class of generators in MRG(k, p) with many nonzero terms that are found using Theorem 1.

Now we need an efficient and parallel MCG implementation of MRG-k. In the next section, we will give such an implementation.

4.3 Efficient and Parallel MCG Implementation of MRG(k, p)

In this section, we will modify the parallel MCG implementation given in Section 2.8. The method requires constructing a MCG from a MRG-k such that the parallel MCG implementation is very efficient. Consider the companion matrix $\mathbf{M}_f$ corresponding to some generator in MRG(k, p) regardless of the number of nonzero terms. It is interesting to note that $\mathbf{M}_f$ and its transpose $\mathbf{M}'_f$ have the same characteristic polynomial, since $\det(x\mathbf{I} - \mathbf{M}_f) = \det(x\mathbf{I} - \mathbf{M}'_f)$, where

$$\mathbf{M}'_f = \begin{pmatrix} 0 & 0 & \dots & 0 & \alpha_k \\ 1 & 0 & \dots & 0 & \alpha_{k-1} \\ 0 & 1 & \dots & 0 & \alpha_{k-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & \alpha_1 \end{pmatrix}.$$

If we choose the invertible matrix $\mathbf{T}$ with the following form

$$\mathbf{T} = \begin{pmatrix} t_1 & 0 & 0 & \dots & 0 \\ 0 & t_2 & 0 & \dots & 0 \\ 0 & 0 & t_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & t_k \end{pmatrix} \tag{4.1}$$

and consider the multiplier matrix

$$\mathbf{B} = \mathbf{T}\mathbf{M}_f'\mathbf{T}^{-1} = \begin{pmatrix} 0 & 0 & \dots & t_1/t_k\,\alpha_k \\ t_2/t_1 & 0 & \dots & t_2/t_k\,\alpha_{k-1} \\ 0 & t_3/t_2 & \dots & t_3/t_k\,\alpha_{k-2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & t_k/t_{k-1} & \alpha_1 \end{pmatrix} = \begin{pmatrix} 0 & \dots & 0 & \alpha_k^* \\ \gamma_1 & \dots & 0 & \alpha_{k-1}^* \\ 0 & \gamma_2 & 0 & \alpha_{k-2}^* \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & \gamma_{k-1} & \alpha_1^* \end{pmatrix} \qquad (4.2)$$

where

$$\alpha_1^* = \alpha_1, \quad \alpha_{i+1}^* = (t_{k-i}/t_k)\alpha_{i+1}, \quad \gamma_i = t_{i+1}/t_i, \quad i = 1,2,\dots,(k-1),$$

then the MCG recursion equation in (2.22) can be rewritten as

$$\begin{pmatrix} X_{i,1} \\ X_{i,2} \\ \vdots \\ X_{i,k} \end{pmatrix} = \begin{pmatrix} \alpha_k^* X_{i-1,k} \\ \gamma_1 X_{i-1,1} + \alpha_{k-1}^* X_{i-1,k} \\ \vdots \\ \gamma_{k-1} X_{i-1,k-1} + \alpha_1^* X_{i-1,k} \end{pmatrix} \bmod p, \quad i \geq 1. \qquad (4.3)$$

Each row in (4.3) requires at most two multiplications. However, if there were k multipliers in $\mathbf{M}_f$, a direct implementation of MRG-k would have require k multiplications. Clearly, this implementation is dramatically more efficient. We call this an efficient and parallel MCG implementation of a MRG-k. This implementation requires only $2k-1$ multiplications to produce k numbers whereas a direct implementation with k multipliers would requires k^2 multiplications.

Since the k-dimensional vectors must be taken as a sequence $(\mathbf{X}_i)_{i\geq 0}$, we must generate k numbers at a time and assign each number in sequence to one of k parallel processors, which is equivalent to assigning each of the k rows in (2.24) to a different

CPU. However, we recommend always skipping the first element in (4.3) and only assigning the remaining $k-1$ elements to one of $k-1$ processors. The reason being that $X_{i,1}$ for $i \geq 1$ is simply a multiplication of the previous number just computed. With this efficient and parallel MCG implementation, the k copies of the MRG will mimic some "random" jump-ahead scheme.

Suggestion for Choices of T

When choosing invertible matrix $\mathbf{T} = \text{Diag}(t_1, t_2, \ldots, t_k)$ in (4.1), there are many selections that can be had by randomly choosing nonzero t_i for $i = 1, 2, \cdots, k$. If one wants to create a program to choose the t_i “randomly”, perhaps, the following could be helpful. If we let g be a primitive root over $\mathbb{Z}_p$, then for any nonzero t_i in $\mathbb{Z}_p$, we can find an integer s_i such that $t_i = g^{s_i} \mod p$. Therefore, we can rewrite

$$\alpha_1^* = \alpha_1, \quad \alpha_{i+1}^* = (t_{k-i}/t_k)\alpha_{i+1} = (g^{s_{k-i}-s_k})\alpha_{i+1} \mod p,$$

$$\gamma_i = t_{i+1}/t_i = g^{s_{i+1}-s_i} \mod p, \quad i = 1, 2, \ldots, (k-1).$$

For the modulus $p = 2^{31} - 1$, $g = 7$ is the smallest primitive root, and $g = 7^5 = 16807$ is the best known primitive root. If using a different modulus, then, of course, a different primitive root g could easily be found.

4.4 Procedure to find and implement a general MRG with many nonzero terms

The following procedure summarizes the previous two sections. Steps 1 and 2 show how to find MRG-k with many nonzero terms. Step 3 and 4 show how to implement MRG-k efficiently and in parallel.

1. Start with the primitive characteristic polynomial $f(x)$ of a generator in MRG(k,p). Several references are given in Sections 2.5 and 2.6.

2. Find the primitive polynomial $f_r(x) = \det(x\mathbf{I} - \mathbf{M}_f^r)$, where $\mathbf{M}_f$ is the companion matrix to $f(x)$ and r is coprime to $p^k - 1$. If $f_r(x)$ has many nonzero terms, then let it be the corresponding characteristic polynomial to MRG-k. If not, then find $f_r(x)$ for a new r coprime to $p^k - 1$ until it does have many nonzero terms.

3. Using companion matrix $\mathbf{M}_{f_r}$ corresponding to $f_r(x)$, construct the MCG whose multiplier matrix is given in (4.2) and implement its efficient recursion

$$\begin{pmatrix} X_{i,1} \\ X_{i,2} \\ \vdots \\ X_{i,k} \end{pmatrix} = \begin{pmatrix} \alpha_k^* X_{i-1,k} \\ \gamma_1 X_{i-1,1} + \alpha_{k-1}^* X_{i-1,k} \\ \vdots \\ \gamma_{k-1} X_{i-1,k-1} + \alpha_1^* X_{i-1,k} \end{pmatrix} \bmod\ p, \quad i \geq 1,$$

   where $\alpha_1^* = \alpha_1,\ \alpha_{i+1}^* = (t_{k-i}/t_k)\alpha_{i+1},\ \gamma_i = t_{i+1}/t_i$, for $i = 1,2,\ldots,(k-1)$ and $t_1, t_2, \ldots, t_k$ are diagonal elements of the invertible matrix $\mathbf{T}$ in (4.1).

4. Generate k numbers at a time, skip the first number, and assign the remaining $k-1$ numbers to one of $k-1$ processors.

4.5 Spectral Test for MRG-k

For this section, we only find MRG-k for small order k. Finding MRGs with larger order k requires raising a large $k \times k$ matrix, $\mathbf{M}_f$, to some power r, then finding $f_r(x) = \det(x\mathbf{I} - \mathbf{M}_f^r)$. Finally, the efficient and parallel MCG implementation requires that the $2k-1$ multipliers be stored in a computer program. Finding MRG-k generators for larger order k is achievable, but computationally and programmatically cumbersome.

The traditional spectral test with successive indices (see Section 2.4) has often been used to compare several generators in MRG(k, p) for a given order k and modulus p (see, e.g., Kao & Tang, 1997a; L'Ecuyer et al., 1993). However, we do not attempt to find the "best" generator in MRG(k, p), because it is not yet clear how to rank which generators in MRG(k, p) are better than others for a given k and p. This is especially true for generators in MRG(k, p) with orders $k \geq 8$. Nonetheless, we welcome more research in this area. For this chapter, we simply report spectral distances $d_t(k)$. Recall from Section 2.4, that smaller spectral distances $d_t(k)$ for a given dimension t and order k are better.

Since we need a starting primitive polynomial to find several MRG-$k$generators for a given order k, we first found a generator in DX-k-2 for orders $k = 5, 6, \ldots, 24$ and modulus $p = 2^{31} - 1$ using methods described in Deng (2005). From these primitive polynomials, we found $f_r(x) = \det(x\mathbf{I} - \mathbf{M}^r_{\mathrm{DX}-k-2})$ for several r and each order k, where $\mathbf{M}_{\mathrm{DX}-k-2}$ is the companion matrix to each corresponding DX-k-2 generator for each order k. From methods described in Chapter 3, we then calculated $d_t(k)$ for dimensions $t = k+1$ and $t = k+10$ for each DX-k-2 generator and each MRG-$k$generator.

Table 5 tabulates the ranges of $d_{k+1}(k)$ and $d_{k+10}(k)$ for the MRG-k. For reference, the multiplier of each DX-k-2 is included along with its corresponding $d_{k+1}(k)$. As explained in Chapter 3, we remark that $d_{k+1}(k) = d_{k+10}(k)$ for each specified generator in classes DX-10-2 through DX-24-2. For each order k, Figures 2 and 3 use box plots to depict the spread of $d_{k+1}(k)$ and $d_{k+10}(k)$, respectively, on a $\log_{10}$ scale for the MRG-k. Thus, the minimum and maximum $d_{k+1}(k)$ for each order k in Table 5 are graphically represented in Figure 2 (on a $\log_{10}$ scale). The same goes for $d_{k+10}(k)$ and Figure 3.

First, note from Table 5 that all the spectral distances $d_{k+1}(k)$ for all generators in DX-k-2 were O(1e-05). Figures 2 confirms the results of Kao and Tang (1997), that is, for every order k in the figure, DX-k-2 (with few nonzero terms) has much larger spectral distances $d_{k+1}(k)$ than MRG-k (with many nonzero terms).

Second, from the table and figures, we see a general trend: MRG-k for larger k tend to have smaller and more closely clustered $d_{t>k}(k)$, especially when you consider the $\log_{10}$ scaling. Although not shown here, this trend persisted across $d_{k+2}(k), \ldots, d_{k+9}(k)$. For a given order k, there are many MRG-k with small $d_{t>k}(k)$. Therefore, MRG-k (for a given k) tend to perform about the same on the spectral test.

Table 5: Range of $d_t(k)$ for DX-k-2 vs. MRG-k

k	DX-k-2		MRG-k			
	B	$d_{k+1}(k)$	$\min d_{k+1}(k)$	$\max d_{k+1}(k)$	$\min d_{k+10}(k)$	$\max d_{k+10}(k)$
24	1011139	2.0169e-05	5.4681e-10	1.1147e-09	4.2325e-08	1.1493e-07
23	1010738	8.5545e-05	6.5150e-10	1.1357e-09	7.9450e-08	1.5698e-07
22	1010598	1.9042e-05	5.4479e-10	1.3708e-09	9.8564e-08	1.9183e-07
21	1010675	6.8704e-05	7.0677e-10	1.1904e-09	1.3134e-07	2.6068e-07
20	1011093	3.6123e-05	7.0100e-10	1.4115e-09	1.4896e-07	3.3066e-07
19	1011385	2.1093e-05	7.8290e-10	1.4744e-09	2.5707e-07	4.7395e-07
18	1010715	2.9729e-05	8.8826e-10	1.5527e-09	3.0252e-07	6.4520e-07
17	1010547	2.8752e-05	9.2199e-10	2.1985e-09	4.6684e-07	9.0989e-07
16	1011236	3.1188e-05	1.0428e-09	2.1323e-09	6.8931e-07	1.2099e-06
15	1010677	4.3985e-05	1.2033e-09	2.0694e-09	9.1564e-07	1.7059e-06
14	1010568	2.6494e-05	1.3987e-09	2.9181e-09	1.6032e-06	2.7158e-06
13	1010667	4.2776e-05	1.6456e-09	2.8490e-09	2.4466e-06	4.6174e-06
12	1010866	9.3999e-05	1.7686e-09	3.2046e-09	4.1708e-06	6.2978e-06
11	1010397	2.6414e-05	2.3066e-09	3.4175e-09	6.5306e-06	1.4636e-05
10	1010571	3.4786e-05	3.0760e-09	5.8136e-09	1.1926e-05	1.9623e-05
9	1010274	2.0613e-05	3.6131e-09	8.2099e-09	2.3410e-05	3.4665e-05
8	1010661	3.9999e-05	4.5075e-09	8.1566e-09	4.5410e-05	9.0088e-05
7	1010167	1.9872e-05	6.1129e-09	1.2974e-08	8.7102e-05	0.00017899
6	1010233	1.7798e-05	1.0093e-08	2.0482e-08	0.00022300	0.00033965
5	1010152	3.3185e-05	1.6960e-08	3.7570e-08	0.00049972	0.00087323

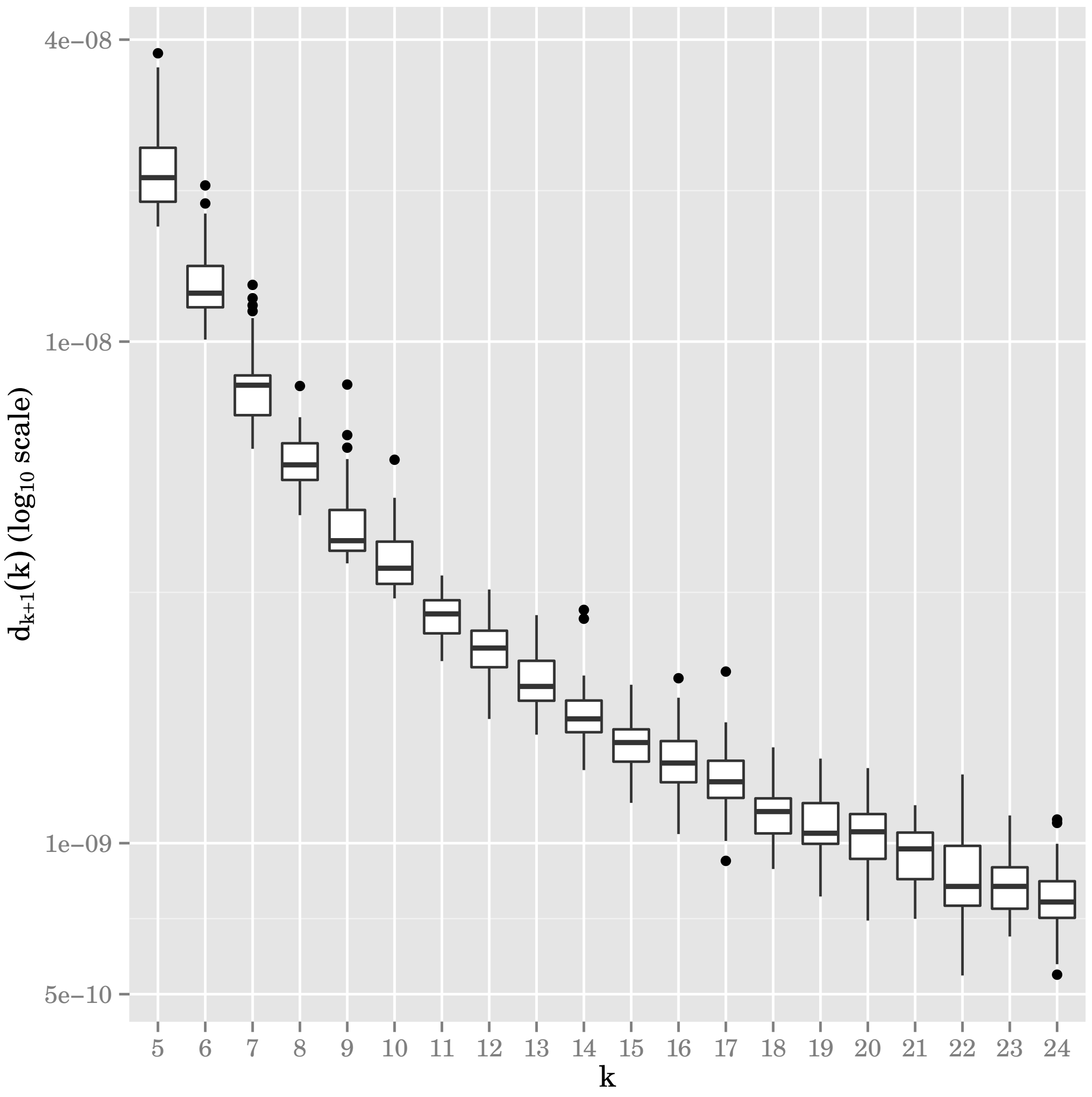


Figure 2: Box plots of $d_{k+1}(k)$ for different MRG-k

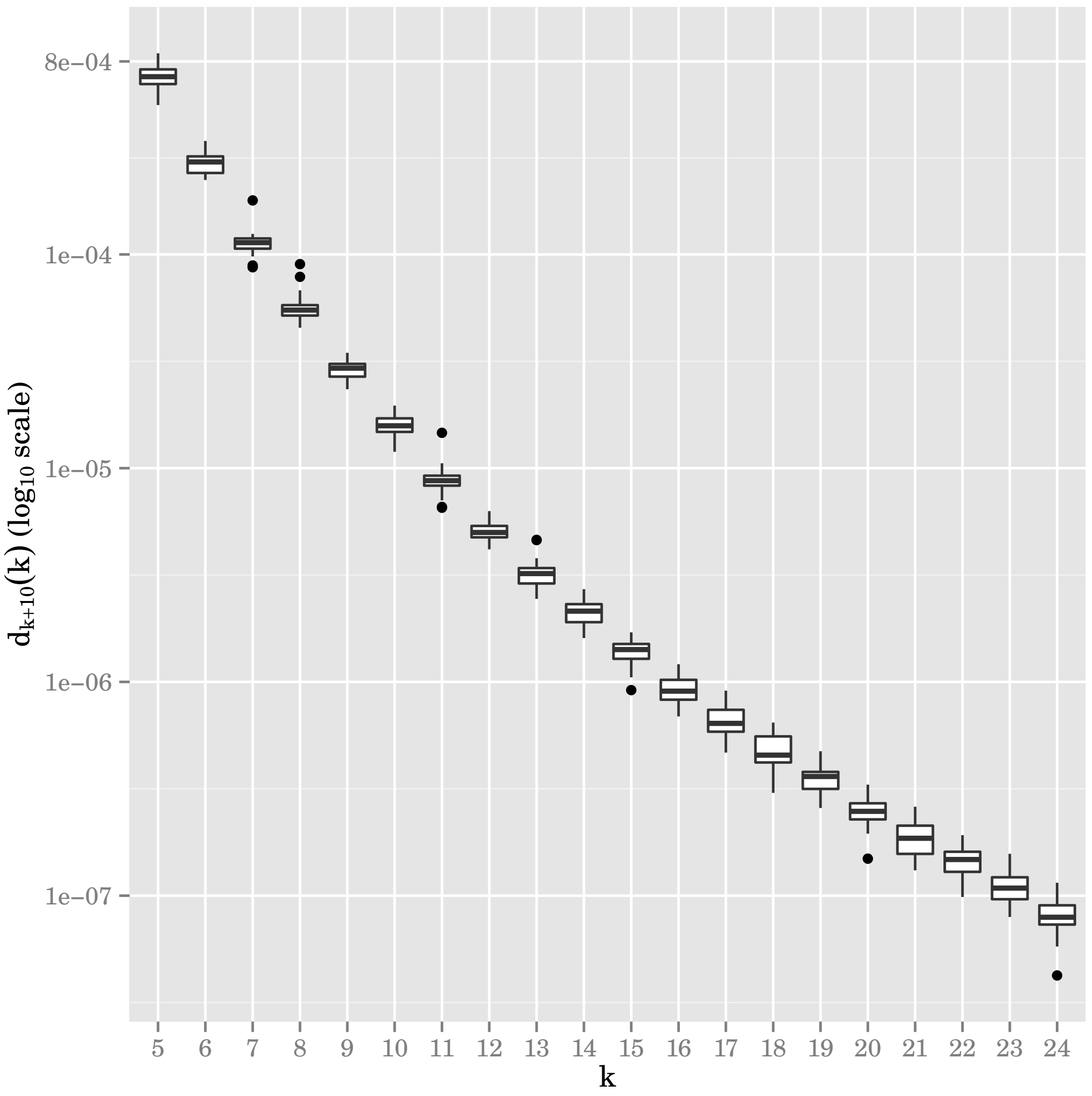


Figure 3: Box plots of $d_{k+10}(k)$ for different MRG-k

4.6 Discussion

The general maximum period MRG with many nonzero terms has all the qualities of a good random number generator except that its direct implementation is inefficient and parallel implementation grows increasingly difficult as the order k increases. Current maximum period MRGs utilize some simple structure in the multipliers to gain a significant increase in their efficiency. However, there is a trade off. These MRGs with few nonzero terms will have larger spectral distances than expected from a MRG with many nonzero terms.

Using a well known theorem (Theorem 1, Section 2.7), this chapter shows how to find numerous general MRGs from one with a simple structure. This abundance source of general MRGs has remained untapped without an efficient and parallel implementation. To solve this problem, we give an efficient and parallel MCG implementation of MRGs with many nonzero terms.

As discussed in Section 4.5, this implementation requires raising a $k \times k$ matrix to some power and then finding the determinant of another $k \times k$ matrix. The efficient and parallel MCG implementation then requires that $2k - 1$ multipliers be stored in a computer program. For large order k, this method will certainly be time-consuming and computer intensive. Perhaps, this can be seen as a limitation to the method. However, the reward for all this work is a general maximum period MRG with an extremely long period and equi-distribution property up to large order k—a very desirable random number generator indeed!

General maximum period MRGs have long been hailed as the ideal random number generator if only they could be implemented efficiently and in parallel. This chapter outlines such an implementation.

Chapter 5

Design and Efficicent, Parallel Implementation of a Special Class of Large Order Multiple Recursive Generators with Many Nonzero Terms

5.1 Introduction

In the previous chapter, we proposed implementing maximum period MRGs with many nonzero terms efficiently and in parallel by using a MCG constructed from the MRG. However, the method in Chapter 4 demands raising a $k \times k$ matrix to a power of r coprime to $p^k - 1$, then finding the determinant of a $k \times k$ matrix to obtain up to $2k - 1$ coefficients that will have to then be stored in a program. When k is large, this method is achievable but computationally and programmatically cumbersome. Therefore, there is a need for a special class of large order MRGs with many nonzero terms that also have an efficient and parallel implementation. In this chapter, we propose a such a class.

Notation

Throughout this chapter, we let p be a large prime number and $\mathbb{Z}_p = \{0, 1, 2, \cdots, p-1\}$ be the finite field of p elements under the usual modulus operations of addition and multiplication. Also, all MRGs discussed in this chapter are of maximum period $p^k - 1$ for given order k and modulus p. Where appropriate, we will from time to time reemphasize that the MRG being discussed has maximum period.

5.2 Design of DW-k: a new class of large order general MRGs

In this chapter, we consider a special class of large order MRGs with many nonzero terms defined by the following characteristic polynomial

$$f(x) = (x - B)(x - C)^{k-1} - ABx^{k-2} \bmod p \tag{5.1}$$

where A, B, and C are suitably chosen nonzero integers over $\mathbb{Z}_p$ such that $f(x)$ is a k-th degree primitive polynomial modulo p. By expanding this polynomial using the binomial theorem, we obtain many nonzero multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$, each of which will be a function of the order k and the parameters A, B, or C over $\mathbb{Z}_p$:

$$\alpha_i = \begin{cases} ((k-1)C + B) \bmod p, & \text{for } i = 1 \\ \left(AB - \binom{k-1}{2}C^2 - (k-1)BC\right) \bmod p, & \text{for } i = 2 \\ \left((-1)^{i-1}(\binom{k-1}{i}C^i + \binom{k-1}{i-1}BC^{i-1})\right) \bmod p, & \text{for } i = 3, 4, \ldots, k-1 \\ (-1)^{k-1}BC^{k-1} \bmod p, & \text{for } i = k \end{cases} . \tag{5.2}$$

We refer to this special class of large order MRGs with many nonzero terms defined by the characteristic polynomial $f(x)$ in (5.1) as DW-k generators. Since the multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$ are fully specified by the order k and the parameters A, B, C, we denote DW-k as DW$(k; A, B, C; p)$ when specifying the order k, parameters A, B, C, and prime modulus p. To avoid confusion, we refer to A, B, C as the *parameters* of the $\alpha_1, \alpha_2, \ldots, \alpha_k$ *multipliers*.

The characteristic polynomial of DW-k in (5.1) has three special features. First, it yields many nonzero multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$ for the MRG recursion in (2.1). Second, there is an efficient MCG that shares the same characteristic polynomial of DW-k. In Section 5.3, we will define this MCG and will show that its characteristic polynomial is the same as DW-k. In Section 5.4, we will give this MCG's efficient implementation and will explain how to use it for an efficient and parallel implementation of DW-k. Lastly, only the multiplier α_2 in (5.2) is a function of A. In Section 5.5, we capitalize on this feature to simplify the search for maximum period DW-k, and we list several DW-k for many orders k and moduli p. Section 5.6 describes how to quickly find a new DW-k from one listed in Section 5.5. Finally, in Section 5.7, we evaluate DW-k in terms of timing, empirical, and theoretical performance. We also offer some evidence that the MCG used to implement DW-k may have merits as a standalone generator.

5.3 MCG Sharing Same Characteristic Polynomial as DW-k

In this section we will define the MCG that shares the same characteristic polynomial as DW-k. Consider a MCG with the following multiplier matrix

$$\mathbf{B} = \begin{pmatrix} B & 0 & 0 & \dots & 0 & A \\ B & C & 0 & \dots & 0 & 0 \\ B & C & C & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ B & C & C & \dots & C & 0 \\ B & C & C & \dots & C & C \end{pmatrix} \tag{5.3}$$

whose recursion is given in (2.22) and whose characteristic polynomial is defined by $f_{\mathbf{B}}(x) = \det(x\mathbf{I} - \mathbf{B}) \bmod p$ in (2.23).

Next, we will prove that the MCG used with multiplier matrix **B** in (5.3) has the same corresponding characteristic polynomial $f_{\mathbf{B}}(x)$ as the one that defines DW-k in (5.1). Before showing this, we first give the characteristic polynomial $f_{\mathbf{M}}(x)$ of a more general matrix **M** of which **B** is a special case.

Lemma 2. *For a given matrix* $\mathbf{M}$ *of the form*

$$\mathbf{M} = \begin{pmatrix} M_1 & 0 & 0 & \cdots & 0 & A \\ M_1 & M_2 & 0 & \cdots & 0 & 0 \\ M_1 & M_2 & M_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ M_1 & M_2 & M_3 & \cdots & M_{k-1} & 0 \\ M_1 & M_2 & M_3 & \cdots & M_{k-1} & M_k \end{pmatrix} \tag{5.4}$$

where A and M_i (for $i = 1, \dots, k$) are integers, the corresponding characteristic polynomial $f_{\mathbf{M}}(x) = \det(x\mathbf{I} - \mathbf{M})$ is

$$f_{\mathbf{M}}(x) = (x - M_1)(x - M_2)\cdots(x - M_k) - AM_1 x^{k-2}.$$

Proof. Let $f_{\mathbf{M}}(x) = \det(x\mathbf{I} - \mathbf{M})$, then after expanding the first row,

$$f_{\mathbf{M}}(x) = (x - M_1) \begin{vmatrix} x - M_2 & 0 & 0 & \cdots & 0 \\ -M_2 & x - M_3 & 0 & \cdots & 0 \\ -M_2 & x - M_3 & x - M_4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -M_2 & -M_3 & -M_4 & \cdots & x - M_k \end{vmatrix}_{(k-1)\times(k-1)}$$

$$+ (-1)^{k+1}(-A) \begin{vmatrix} -M_1 & x - M_2 & 0 & \cdots & 0 \\ -M_1 & -M_2 & x - M_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -M_1 & -M_2 & -M_3 & \cdots & x - M_{k-1} \\ -M_1 & -M_2 & -M_3 & \cdots & -M_{k-1} \end{vmatrix}_{(k-1)\times(k-1)}$$

The determinant of the first (lower triangular) matrix is simply the product of the diagonals. The second matrix can be transformed into an upper triangular matrix via row operations: $R_i - R_{i-1} \to R_i$ for $i = 2, \dots, k-1$. Hence,

$$f_{\mathbf{M}}(x) = (x - M_1)(x - M_2) \cdots (x - M_k)$$

$$+ (-1)^{k+1}(-A) \begin{vmatrix} -M_1 & x - M_2 & 0 & \dots & 0 \\ 0 & -x & x - M_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & x - M_{k-1} \\ 0 & 0 & 0 & \dots & -x \end{vmatrix}_{(k-1)\times(k-1)}$$

$$= (x - M_1)(x - M_2) \cdots (x - M_k) - AM_1 x^{k-2}. \qquad \square$$

Theorem 2. *Given the matrix* $\mathbf{B}$ *defined in (5.3), the corresponding characteristic polynomial* $f_{\mathbf{B}}(x) = \det(x\mathbf{I} - \mathbf{B})$ *is*

$$f_{\mathbf{B}}(x) = (x - B)(x - C)^{k-1} - ABx^{k-2}.$$

Proof. The multiplier matrix $\mathbf{B}$ is a special case of $\mathbf{M}$ in (5.4) where $M_1 = B$ and $M_i = C$ for $i = 2, \dots, M_k$. Therefore, by Lemma 2, the result follows. $\square$

5.4 Efficient and Parallel MCG Implementation of DW-k

Now that we have proved that MCG defined by multiplier matrix **B** in (5.3) shares the same characteristic polynomial as the one that defines DW-k in (5.1), we will show in this section that DW-k can be implemented efficiently and in parallel using this MCG. Given the multiplier matrix **B** in (5.3), the linear recursion in (2.22) can be rewritten as

$$\begin{pmatrix} X_{i,1} \\ X_{i,2} \\ X_{i,3} \\ \vdots \\ X_{i,k} \end{pmatrix} = \begin{pmatrix} BX_{i-1,1} + AX_{i-1,k} \\ BX_{i-1,1} + CX_{i-1,2} \\ BX_{i-1,1} + CX_{i-1,2} + CX_{i-1,3} \\ \vdots \\ BX_{i-1,1} + CX_{i-1,2} + \ldots + CX_{i-1,k} \end{pmatrix} \mod p, \quad i \geq 1 \tag{5.5}$$

and efficiently implemented as

$$\begin{pmatrix} X_{i,1} \\ X_{i,2} \\ X_{i,3} \\ \vdots \\ X_{i,k} \end{pmatrix} = \begin{pmatrix} BX_{i-1,1} + AX_{i-1,k} \\ BX_{i-1,1} + CX_{i-1,2} \\ X_{i,2} + CX_{i-1,3} \\ \vdots \\ X_{i,k-1} + CX_{i-1,k} \end{pmatrix} \mod p, \quad i \geq 1. \tag{5.6}$$

Notice in (5.6) that the generated output $X_{i,1}$ and $X_{i,2}$ are solely generated from numbers in the previously generated output vector $\mathbf{X}_{i-1}$. However, $X_{i,j}$ for $j = 3, 4, \ldots, k$ is the sum of the previous number just generated in the current output vector $\mathbf{X}_i$ and a multiple of a number in the previous output vector $\mathbf{X}_{i-1}$. Hence, the recursion in (5.6) does not require as many multiplications and additions as the one in (5.5). Also, we note that additional efficiency on some compilers might be gained if we let C be a power of 2, that is, we can let $C = 2^e$ for some positive integer e. If this approach is taken, it appears from empirical testing (not shown) that 2^e for $e \leq 4$ should be avoided, which also implies that C in general should not be too small for given A, B, and order k.

As explained in Section 2.8, since the MCG defined by $\mathbf{B}$ in (5.3) and DW-k share the same characteristic polynomial, then the generated vector sequence $(\mathbf{X}_i)_{i\geq 0}$ from this MCG satisfies the recurrence for DW-k

$$\mathbf{X}_i = \alpha_1 \mathbf{X}_{i-1} + \alpha_2 \mathbf{X}_{i-2} + \cdots + \alpha_k \mathbf{X}_{i-k} \bmod p, \quad i \geq k, \tag{5.7}$$

where the multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$ are defined in (5.2). Therefore, each of the k sequences taken from each of the k rows in (5.7) can be viewed as k copies of the same DW-k with different starting seeds. Therefore, we recommend generating k numbers at a time from the efficient recursion in (5.6) and assigning each number to one of k processors. As we will see in the Section 5.7, empirical performance suggests that the MCG defined by $\mathbf{B}$ in (5.3) has its own merits as a standalone generator where numbers can be generated one number at a time.

In the next section, we turn our attention to finding A, B, C such that $f(x)$ is a primitive polynomial over $\mathbb{Z}_p$.

5.5 Search for DW-k

As explained in Section 2.5, Alanen and Knuth (1964) and Knuth (1998) gave necessary and sufficient conditions for determining whether $f(x)$ is primitive or not. The main difficulty is finding the complete factorization of $R(k,p) = (p^k - 1)/(p-1)$ when k or p is large. There are two common approaches to by-pass the difficulty of the factorization: (a) for a given p one can find k such that $R(k,p)$ is (relatively) easy to factor, usually because $R(k,p)$ has only one huge prime factor and the rest are (relatively) small prime factors or (b) one can consider a prime order k and then find prime p such that $R(k,p)$ is also a prime number.

Once we know the complete factorization of $R(k,p) = (p^k - 1)/(p-1)$ for a given order k and modulus p, then we need only search for multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$ such that the rest of the necessary and sufficient conditions of a primitive polynomial are met. For DW-k this process can be simplified. First, only multiplier α_2 is a function of order k and parameters A, B, and C over $\mathbb{Z}_p$. The rest are completely specified by order k and only parameters B and C. Therefore, for a given order k and modulus p, once we know $\alpha_k = (-1)^{k-1}BC^{k-1}$ is a primitive root modulo p, we can fix multipliers $\alpha_1, \alpha_3, \ldots, \alpha_k$ and search for α_2 until the characteristic polynomial of DW-k is primitive. Equivalently, once we we find B and C such that $(-1)^{k-1}BC^{k-1}$ is a primitive root modulo p, we can fix B and C and just search for A until $f(x)$ in (5.1) is a primitive polynomial.

Deng (2005) and Deng et al. (2012b) found k-th degree primitive polynomials for $k \in \{47, 643, 1597, 7499, 20897\}$ and modulus $p = 2^{31} - 1$ such that $R(k,p)$ is (relatively) easy to factor. This modulus is a popular choice, because the modulus operation can be replaced with more efficient logical operations. Furthermore, it is also the largest (signed) integer that can be stored in a 32-bit computer word. The largest period length for this modulus is approximately $10^{195009.3}$ with equi-distribution up to 20897 dimensions. Following the methods described in these references, for order k, prime modulus $p = 2^{31} - 1$, C in the form of 2^e for $5 \le e \le 9$ (for additional efficiency), and four values of B, we search for A such that $\mathrm{DW}(k; A, B, C; p)$ achieves the maximum-period. Table 6 tabulates 100 $\mathrm{DW}(k; A, B, C; p = 2^{31} - 1)$: 5 values of k, 4 values of B, and 5 values of C.

Deng (2004, 2008) and Deng et al. (2012a) found different k ranging from 101 to 25013 and moduli of the form $p = 2^{31} - w$, where w is a positive integer, such that $R(k, p)$ is also a prime. The largest period length, $p^k - 1$, found so far has reached approximately 10^{233361} with equi-distribution up to 25013 dimensions. Again, following the methods in these references, we search for $\mathrm{DW}(k; A, B, C; p)$ with the more flexible form modulus, $p = 2^{31} - w$. For order k, prime modulus p, C in the form of 2^e for $5 \le e \le 9$, and one value of B, we search for A such that $\mathrm{DW}(k; A, B, C; p)$ achieves the maximum period. Tables 7, 8, and 9 tabulate 265 $\mathrm{DW}(k; A, B, C; p = 2^{31} - w)$: 53 values of k and 5 values of C.

Table 6: List of A for DW($k; A, B, C = 2^e; p = 2^{31} - 1$)

k	B	32	64	128	56	512
47	5005	65536	65524	65516	65490	65499
47	10001	65460	65434	65447	65461	65533
47	15090	65455	65519	65472	65505	65468
47	20006	65479	65457	65523	65445	65428
643	5005	65495	63564	64444	63932	62930
643	10001	65326	64001	64562	64797	65510
643	15090	65386	65480	64825	65406	64380
643	20006	63411	65239	64072	63090	64505
1597	5005	64577	65235	64934	65508	63884
1597	10001	63954	63471	64674	63495	65486
1597	15090	63832	65074	63227	65239	64294
1597	20006	65518	65014	64296	61968	65152
7499	5005	64797	59943	63816	61707	61396
7499	10001	58921	49795	52984	55685	60937
7499	15090	56680	53060	64547	64620	53885
7499	20006	62277	59547	58067	59030	50003
20897	5005	21951	59691	49564	33494	62280
20897	10001	27020	53061	35759	29904	37956
20897	15090	34619	36325	58692	49618	31093
20897	20006	47762	44292	57165	13979	62931

Table 7: List of A for DW($k; A, B, C = 2^e; p = 2^{31} - w$); $k \leq 2003$

k	w	B	32	64	128	256	512
101	82845	20000	20028	20066	20093	20050	20217
211	841329	20001	20227	20606	20052	20205	20067
307	52545	20001	20077	20053	20809	20303	20317
401	57189	20000	20184	20266	21862	20106	20131
503	174489	20001	21144	22448	20179	20286	20089
601	1327485	20000	20142	20937	20746	21791	20870
701	220665	20002	20074	21035	20579	20433	20172
809	2010789	20000	20744	21516	20558	20820	20129
907	4400889	20004	22482	20986	20139	20727	20295
1009	2368869	20000	20734	20258	22517	21765	20255
1103	7316361	20002	20637	20813	21623	20535	20201
1201	1113705	20005	21289	22405	21161	20762	20793
1301	1070901	20000	22028	21108	21694	20618	20511
1409	4320189	20000	21545	22719	22407	20942	23501
1511	2771205	20000	21718	20724	22864	21286	21147
1601	368961	20001	20877	21155	22095	22037	20485
1709	1032441	20001	21518	20501	20504	20552	26690
1801	5789241	20002	21325	21222	20049	20892	21539
1901	267321	20002	20336	20779	21070	20050	22125
2003	44961	20001	20499	20606	29363	27302	22033

Table 8: List of A for DW($k; A, B, C = 2^e; p = 2^{31} - w$); $2111 \leq k \leq 4001$

k	w	B	32	64	128	256	512
2111	3536385	20005	20326	21705	23674	20213	23241
2203	6043089	20002	23370	24475	25871	24984	21072
2309	340185	20003	22094	20623	20372	26437	22412
2411	9256449	20006	20975	21355	22754	25674	20429
2503	13539249	20001	21325	20477	22106	21521	25097
2609	8811681	20001	20761	22499	23285	21895	20796
2707	1113585	20004	20521	20016	20403	20994	24184
2801	1095609	20002	20363	25844	20090	20993	25253
2903	14055825	20003	20159	20246	23773	20718	23791
3001	3058401	20001	28132	20617	21599	21230	23706
3109	6741129	20002	20692	22733	23316	21300	25581
3203	4718889	20001	22833	25436	38018	21850	21251
3301	14881185	20002	28598	26152	20943	22581	20220
3407	6243009	20001	20904	29014	31949	22005	20327
3511	1412961	20001	21747	25713	20470	26032	23604
3607	1026585	20003	20773	22329	20577	28986	29789
3701	11576625	20002	21346	22672	29143	27596	21854
3803	32058129	20002	21522	21082	20997	21753	27771
3907	17381649	20001	23800	20442	24531	24191	20518
4001	4412481	20001	24549	20465	24707	22823	20072

Table 9: List of A for DW($k; A, B, C = 2^e; p = 2^{31} - w$); $k \geq 5003$

k	w	B	32	64	128	256	512
5003	1259289	20001	27764	20766	20859	24425	21082
6007	9984705	20001	20113	24411	30649	21365	26515
7001	610089	20003	23538	30769	31220	20721	24699
8009	5156745	20001	24596	30764	29030	28750	26236
9001	7236249	20002	34768	24425	22364	32772	24052
10007	431745	20003	20258	63815	33089	47201	28169
11003	1276425	20001	45421	51250	25568	21659	23189
12007	37532781	20000	37739	27242	30994	34999	22036
13001	71128005	20003	52978	21790	24522	38227	28803
14009	626301	20000	63410	26334	29856	77138	27117
15013	8996265	20003	72834	22198	38433	37498	20490
20011	26131941	20000	43502	37043	31430	69099	24899
25013	11538909	20000	25118	24538	57529	87512	66488

5.6 Finding new DW-k via AGM

Utilizing Deng's (2004) Automatic Generation Method (AGM) described in Section 2.7, once we know the parameters A, B, C for one DW-k, we can quickly find another DW-k of the same order k and modulus p. Recall that, starting with a k-th degree primitive polynomial, AGM quickly finds numerous k-th degree primitive polynomials with the same number of nonzero multipliers as the base primitive polynomial.

To use AGM to find new DW-k, first, choose the desired order k and modulus p and corresponding parameters A, B, C, from Tables 6 through 9. A defining feature of DW-k is that its primitive characteristic polynomial $f(x)$ in (5.1) is equal to $f_{\mathbf{B}}(x) = \det(x\mathbf{I} - \mathbf{B})$

where $\mathbf{B}$ is defined by (5.3). For any nonzero integer z in $\mathbb{Z}_p$, define

$$\begin{aligned}
G_{\mathbf{B}}(x) &= z^k f_{\mathbf{B}}(z^{-1}x) \bmod p \\
&= z^k \det(z^{-1}x\mathbf{I} - \mathbf{B}) \bmod p \\
&= \det(x\mathbf{I} - z\mathbf{B}) \bmod p \\
&= (x - zB)(x - zC)^{k-1} - (zA)(zB)x^{k-2} \quad \text{(by Theorem 2 for } z\mathbf{B}\text{)}.
\end{aligned}$$

Letting $A' = zA$, $B' = zB$, and $C' = zC$, then $\alpha_1, \alpha_2, \ldots, \alpha_k$ are given in (5.2). Specifically, if $\alpha_k = (-1)^{k-1} z^k BC^{k-1} \bmod p$ is a primitive root modulo p, then $G_{\mathbf{B}}(x)$ is the primitive characteristic polynomial of a new DW-k whose multipliers $\alpha_1, \alpha_2, \ldots, \alpha_k$ are defined by A', B', C' (Deng, 2004, Theorem 4). By the Cayley-Hamilton theorem, the new DW-k can be implemented efficiently and in parallel by generating numbers k at a time from the recursion in (5.6). Since we know how many primitive roots exist, then for each modulus p and order k, we know that AGM can produce $\phi(p-1)$ new DW-k from one base DW-k, where $\phi(x)$ denotes the Euler totient function, which gives number of integers between 1 and x that are relatively prime to x.

One advantage of applying AGM to DW-k is that, for a given order k and modulus p, it gives the user the liberty to find a new DW-k if for some reason she needs more parameters in addition to those in Tables 6 through 9.

5.7 Evaluation of DW-k

Timing

DW-k is very efficient. When the modulus is the largest 32-bit signed integer word, $p = 2^{31} - 1$, this class of generators can be employed even faster by replacing the modulus operation with logical operations (see, e.g., Deng & Xu, 2003). We compared the DW$(20897; 62931, 20006, 512; 2^{31} - 1)$ against the Mersenne-Twister (MT19937; Matsumoto & Nishmura, 1998) and L'Ecuyer's (1999) combined MRG (MRG32k3a). Table 10 tabulates the average of 100 timings (in seconds) to generate 100 million numbers for each generator. The starred version of DW$(20897; 62931, 20006, 512; 2^{31} - 1)$ replaces the modulus operation with logical operations. The timings program was compiled using `gcc` and the `-O3` optimizer flag on a 64-bit AMD Opteron 6274 16 core processor with clock speed of 2.2 GHz. When using the modulus operation, DW-k is approximately as fast as MT19937 and more than 3 times faster than MRG32k3a. However, when replacing the modulus operation with logical operations (for $p = 2^{31} - 1$), DW-k is twice as fast as MT19937 and more than 6 times faster than MRG32k3a.

Empirical Testing

Efficiently implementing DW-k in parallel across k CPUs requires generating k numbers at a time from the recursion in (5.6) and assigning each of these numbers to one of the k CPUs. It is widely known that maximum period MRGs have excellent

Table 10: Average of 100 timings (in seconds) to generate 100 million numbers

Random Number Generator	C (-O3 flag)
DW(20897;62931,20006,512;$2^{31}-1$)	0.97
DW(20897;62931,20006,512;$2^{31}-1$)*	0.52
MT19937	0.93
MRG32k3a	3.24

* indicates modulus operations were replaced with logical operations

empirical performance when the generated output are taken in successive sequence. Indeed, these MRGs are one of a few kinds or random number generators that are able to pass all the empirical tests implemented in the TestU01 package (L'Ecuyer & Simard, 2007). Concerning output from general maximum period MRGs, Knuth (1998) noted that "all known evidence indicates that the result will be a very satisfactory source of random numbers." Therefore, the k sequences of DW-k has excellent empirical performance.

We note here that empirical performance is also satisfactory when generating numbers one at a time from the MCG recursion in (5.6). For each combination of order k, modulus p and parameters A, B, C, listed in Table 6 through Table 9, we generated the numbers one at a time and subjected the sequential output to the *Crush* battery of stringent empirical tests in the TestU01. For each generator tested, the battery produced 144 p-values from 144 statistical tests performed. Small p-values indicate that the generator fails that particular test. On the other hand, p-values is too close to 1 are considered "too good to be truly random" (L'Ecuyer & Simard, 2007). At the end of each test, those 144 p-values produced outside the range of $[10^{-3}, 1-10^{-3}]$ are

Table 11: Percentages of p-values (in specified ranges) of *Crush* tests on $\mathrm{DW}(k; A, B, C; 2^{31} - 1)$ listed in Table 6

k	$< 10^{-7}$	$< 10^{-5}$	$< 10^{-4}$	$< 10^{-3}$	$> 1-10^{-3}$	$> 1-10^{-4}$	$> 1-10^{-5}$	$> 1-10^{-7}$
47	0.00000	0.00007	0.00014	0.00125	0.00083	0.00014	0.00014	0.00014
643	0.00000	0.00000	0.00000	0.00014	0.00014	0.00000	0.00000	0.00000
1597	0.00000	0.00000	0.00007	0.00035	0.00021	0.00000	0.00000	0.00000
7499	0.00000	0.00000	0.00007	0.00090	0.00104	0.00007	0.00000	0.00000
20897	0.00000	0.00000	0.00021	0.00139	0.00111	0.00014	0.00014	0.00014

reported.

For the prime modulus of $p = 2^{31} - 1$, the total number of runs for for the 100 generators in Table 6 is 500: each of the 100 generators was tested with 5 different starting seeds. The total number of p-values produced for generators in Table 6 is $72000 = 100 \times 5 \times 144$ and the percentages of these p-values in the specified ranges are summarized in Table 11. Similarly, for the prime modulus of $p = 2^{31} - w$ with $(p^k - 1)/(p - 1)$ also a prime, the total number of runs for generators in Tables 7, 8, and 9 is 1325: each of the 265 generators was tested with 5 different starting seeds. The total number of p-values produced for generators in Tables 7, 8, and 9 is $190800 = 265 \times 5 \times 144$ and the percentages of these p-values in the specified ranges are summarized in Table 12.

From these tables, we have strong empirical evidence that even generating numbers one at a time from the MCG recursion in (5.6) yields a satisfactory source of uniform random numbers. However, to implement DW-k, numbers must be generated k at a time. Therefore, we recommend only generating numbers k at a time.

Table 12: Percentages of p-values (in specified ranges) of *Crush* tests on DW$(k; A, B, C; 2^{31} - w)$ listed in Tables 7 through 9

C	$< 10^{-7}$	$< 10^{-5}$	$< 10^{-4}$	$< 10^{-3}$	$> 1-10^{-3}$	$> 1-10^{-4}$	$> 1-10^{-5}$	$> 1-10^{-7}$
32	0.00000	0.00000	0.00021	0.00144	0.00077	0.00010	0.00005	0.00005
64	0.00000	0.00005	0.00026	0.00136	0.00100	0.00013	0.00008	0.00008
128	0.00000	0.00000	0.00008	0.00141	0.00116	0.00021	0.00008	0.00008
256	0.00000	0.00000	0.00008	0.00095	0.00095	0.00026	0.00013	0.00013
512	0.00000	0.00003	0.00008	0.00116	0.00080	0.00021	0.00010	0.00010

Theoretical Testing

The spectral test for a MRG as described in Section 2.4 is a theoretical test that gives some measure of uniformity in dimensions beyond order k. For dimensions $t < k$, the equi-distribution property for maximum period MRGs guarantees very good uniformity. As described in Chapter 3, current methods for computing the spectral test in dimension $t = k + d$ can be very inefficient when order k is as large as many of those in Tables 6 through 9. Furthermore, since DW-k has many nonzero terms, it is unlikely that the proposed method in Chapter 3 will be any more efficient. Therefore, computing the spectral test for large order MRGs with many nonzero terms is computationally intensive.

For maximum period MRGs with good spectral test performance in dimensions beyond k, L'Ecuyer (1997) stated a necessary but not sufficient condition that the sum of the squares of the multipliers, $\sum_{i=1}^{k} \alpha_i^2$, should be large. Given that DW-k has many nonzero multipliers, then it is likely to have excellent spectral test performance.

Furthermore, we remark that since there is no simple relationship among these nonzero multipliers and since k copies of DW-k are implemented in parallel, it appears very difficult to find a lacunary index set (see Section 2.4) such that the k copies perform poorly on the spectral test.

5.8 Discussion

This chapter defines DW-k, a special class of large order MRGs with many nonzero terms whose recurrence can be efficiently implemented in parallel via a k-th order MCG sharing the same characteristic polynomial. The implementation requires generating k numbers at a time from the MCG recurrence in (5.6) and assigning each number to one of k CPUs. For DW-k with modulus $p = 2^{31} - 1$, the implementation is twice as fast as MT19937 and six times as fast as MRG32K3A; for other DW-k the implementation is about as fast as MT19937 and three times faster than MRG32k3a. Of the DW-k given, DW-25013 has the longest period length of 10^{233361} with equi-distribution up to 25013 dimensions. Furthermore, new DW-k can be quickly found by applying Deng's (2004) Automatic Generation Method (AGM) to a base DW-k.

Chapter 6

Conclusion

The general maximum period MRG with many nonzero terms has all the qualities of a good random number generator except that its direct implementation is inefficient and parallel implementation grows increasingly difficult as the order k increases. To significantly increase efficiency, current maximum period MRGs usually have few nonzero terms or can be implemented by a higher order maximum period MRG with few nonzero terms. However, the spectral test performance of these MRGs with few nonzero terms is poorer than would be expected from a MRG with many nonzero terms.

The current method for computing the spectral test requires raising a $k \times k$ matrix to some power, and it is unclear how this method relates to the spectral distance. In Chapter 3, we offer a geometric, intuitive, and easier method for calculating the spectral test. Using this procedure, we list "better" FMRG-k and DX-k-s generators with respect to performance on the spectral test.

Chapter 4 illustrates that k-th ordered MRGs with few nonzero terms indeed have larger spectral distances than those of k-th ordered MRGs with many nonzero terms. Furthermore, a method is provided (a) for finding numerous MRGs with many nonzero terms from one with few nonzero terms and (b) for implementing these MRGs efficiently and in parallel using a MCG.

This implementation requires raising a $k \times k$ matrix to some power and then finding the determinant of another $k \times k$ matrix. Finally, $2k-1$ multipliers must be stored in a computer program. For large order k, this method will certainly be time-consuming and computer intensive for large order k. This method works well for small or moderate order k. When k is large, this method is still achievable but computationally and programmatically cumbersome.

To extend efficient and parallel MCG implementation to large order, maximum period MRGs with many nonzero terms, Chapter 5 proposes a new class of MRGs called DW-k. This class of MRGs has a unique characteristic polynomial that yield many nonzero terms and directly corresponds to an efficient and parallel MCG implementation.

In the past, computing spectral test for large order MRGs was tedious and non-intuitive. Also, maximum period MRGs with many nonzero terms were thought to be good random number generators if only they could be implemented efficiently and in parallel. For the former, the dissertation describe an easier, more intuitive way to compute spectral tests of MRGs. For the latter, this dissertation proposes a solution: the efficient and parallel MCG implementation of a maximum period MRG with many nonzero terms for small to large order k.

References

1. ALANEN, J. D., AND KNUTH, D. E. (1964). Tables of finite fields. *Sankhyā, Series A, 26*, 305–328.

2. COHEN, H. (1993). *A Course in Computational Algebraic Number Theory* (Vol. 138). Springer-Verlag.

3. DENG, L. Y. (2004). Generalized Mersenne Prime Number and Its Application to Random Number Generation, in *Monte Carlo and Quasi-Monte Carlo Methods 2002* (H. Niederreiter, ed.), Springer-Verlag, 167–180.

4. DENG, L. Y. (2005). Efficient and portable multiple recursive generators of large order, *ACM Transactions on Modeling and Computer Simulation, 15*(1), 1–13.

5. DENG, L. Y. (2008). Issues on computer search for large order multiple recursive generators. in *Monte Carlo and Quasi-Monte Carlo Methods 2006* (S. Heinrich, A. Keller, and H. Niederreiter, eds.), Springer-Verlag, 251–261.

6. DENG, L. Y. AND GEORGE, E. O. (1990). Generation of uniform variates from several nearly uniform distributed variables. *Communications in Statistics-Simulation and Computing, 19*(1), 145–154.

7. DENG, L. Y., LI, H. AND SHIAU, J.J.H. (2009a). Scalable parallel multiple recursive generators of large order, *Parallel Computing, 35*, 29–37.

8. DENG, L. Y., LI, H., SHIAU, J. J. H., AND TSAI, G. H. (2008). Design and implementation of efficient and portable multiple recursive generators with few zero coefficients. In *Monte Carlo and Quasi-Monte Carlo Methods 2006* (pp. 263-273). Springer Berlin Heidelberg.

9. DENG, L. Y., AND LIN, D. K. J. (2000). Random number generation for the new century. *American Statistician, 54*, 145–150.

10. DENG, L. Y., LIN, D. K. J., WANG, J., AND YUAN, Y. (1997). Statistical justification of combination generators. *Statistica Sinica, 7*, 993–1003.

11. DENG, L. Y., SHIAU, J. J. H. AND LU, H. H. S. (2012a). Efficient computer search of large-order multiple recursive generators for pseudo-random number generators, *Journal of Computational and Applied Mathematics, 236*, 3228–3237.

12. DENG, L. Y., SHIAU, J. J. H. AND LU, H. H. S. (2012b). Large-order multiple recursive generators with modulus $2^{31} - 1$, *INFORMS Journal on Computing, 24*, 636–647.

13. DENG, L. Y., SHIAU, J. J. H., AND TSAI, G.H. (2009b). Parallel random number generators based on large order multiple recursive generators. In *Monte Carlo and Quasi-Monte Carlo Methods 2008* (pp. 289–296). Springer Berlin Heidelberg.

14. DENG, L. Y., AND XU, H. (2003). A system of high-dimensional, efficient, long-cycle and portable uniform random number generators, *ACM Transactions on Modeling and Computer Simulation, 13*(4), 299–309.

15. ENTACHER, K., SCHELL, T., UHL, A. (2002). Efficient lattice assessment for LCG and GLP parameter searches. *Mathematics of Computation, 71*(239), 1231–1242.

16. ENTACHER, K., SCHELL, T., UHL, A. (2005). Bad lattice points. *Computing, 75*, 281–295.

17. FINCKE, U. AND POHST M. (1985). Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation, 44*(170), 463–471.

18. GOLOMB, S. W. (1967). *Shift Register Sequences*, Holden-Day, San Francisco, CA.

19. GROTHE, H. (1987). Matrix generators for pseudo-random vector generation, *Statist. Papers, 28*, 233–238.

20. GRUBE, A. (1978). Mehrfach rekursiv-erzeugte Pseudo-Zufallszahlen. *Zeitschrift für Angewandte Mathematik und Mechanik, 53*, 223–225.

21. HELLEKALEK, P. (1998). Good random number generators are (not so) easy to find. *Mathematics and Computers in Simulation, 46*, 485–505.

22. KAO, C., AND TANG, H. C. (1997a). Upper bounds in spectral test for multiple recursive random number generators with missing terms. *Computers and Mathematical Applications, 33*, 119–125.

23. KAO, C., AND TANG, H. C. (1997b). Systematic searches for good multiple recursive random number generators. *Computers and Operations Research, 24*, 899–905.

24. KNUTH, D. E. (1981). *The Art of Computer Programming, Vol 2: Seminumerical Algorithms.* 2nd ed. Addison-Wesley, Reading, MA.

25. KNUTH, D. E. (1998). *The Art of Computer Programming, Vol 2: Seminumerical Algorithms.* 3rd ed. Addison-Wesley, Reading, MA.

26. L'ECUYER, P. (1990). Random numbers for simulation, *Communications of the ACM, 33*, 85–97.

27. L'ECUYER, P. (1997). Bad lattice structures for vectors of nonsuccessive values produced by some linear recurrences. *INFORMS Journal on Computing, 9*(1), 57–60.

28. L'ECUYER, P. (1999). Good parameters and implementations for combined multiple recursive random number generators, *Operations Research, 47*, 159–164.

29. L'ECUYER, P. AND BLOUIN, F. (1988). Linear congruential generators of order k > 1. In *Winter Simulation Conference: Proceedings of the 20th conference on Winter simulation* . (Vol. 12, No. 14, pp. 432–439).

30. L'ECUYER, P., BLOUIN, F., AND COUTURE, R. (1993). A search for good multiple recursive random number generators. *ACM Transactions on Modeling and Computer Simulation, 3*(2), 87–98.

31. L'ECUYER, P. AND COUTURE, R. (1997). An implementation of the lattice and spectral tests for multiple recursive linear random number generators, *INFORMS Journal on Computing, 9*, 206–217.

32. L'ECUYER, P. AND SIMARD, R. (2007). TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software, 33*(4), Article 22, 1–40.

33. L'ECUYER, P. AND SIMARD, R. (2014). On the Lattice Structure of a Special Class of Multiple Recursive Random Number Generators. *INFORMS Journal on Computing,* to appear.

34. L'ECUYER, P., SIMARD, R., CHEN, E.J., AND KELTON, W.D. (2002). An object-oriented random-number package with many long streams and substreams, *Operations Research, 50,* 1073–1075.

35. L'ECUYER, P. AND TOUZIN, R. (2004). On the Deng-Lin random number generators and related methods. *Statistics and Computing, 14*(1), 5–9.

36. LEHMER, D. H. (1951). Mathematical methods in large-scale computing units. *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery,* Harvard University Press, Cambridge, MA, 141–146.

37. LENSTRA, A.K. LENSTRA, H.W. JR., LOVASZ, L. (1982). Factoring polynomials with rational coefficients, *Mathematische Annalen, 261,* 515–534.

38. LIDL, R., AND NIEDERREITER, H. (1994). *Introduction to Finite Fields and Their Applications.* Revised Edition. Cambridge University Press, Cambridge, MA.

39. MARSAGLIA, G. (1968). Random numbers fall mainly in the planes. *Proceedings of the National Academy of Sciences, 61,* 25–28.

40. MATSUMOTO, M. AND NISHIMURA, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation, 8*(1), 3–30.

41. PARK, S. AND MILLER, K. (1988). Random number generators: good ones are hard to find. *Communications of the ACM, 31*, 1192–1201.

42. SEZGIN, F. (1996). Some improvements for a random number generator with single-precision floating-point arithmetic. *Computers & Geosciences, 22*(4), 453—455.

43. SEZGIN, F. (2004). A method of systematic search for optimal multipliers in congruential random number generators. BIT Numerical Mathematics, 44(1), 135-149.

44. SEZGIN, F. (2006). Distribution of lattice points. *Computing*, 78, 173–193.

45. TANG, H. C. AND KAO, C. (2002). Lower bounds in spectral tests for vectors of nonsuccessive values produced by multiple recursive generator with some zero multipliers. *Computers and Mathematics with Applications, 43*, 1153–1159.

46. ZIERLER, N. (1959). Linear recurring sequences. *Journal of the Society for Industrial & Applied Mathematics, 7*(1), 31–48.