Random Number Generators Based on Linear Recurrences in $\mathbb{F}_{2^w}$

François Panneton and Pierre L'Ecuyer

Département d'informatique et de recherche opérationnelle
Université de Montréal
C.P. 6128, Succ. Centre-Ville,
Montréal (Québec), H3C 3J7, CANADA
panneton@iro.umontreal.ca and lecuyer@iro.umontreal.ca

Summary. This paper explores new ways of constructing and implementing random number generators based on linear recurrences in a finite field with 2^w elements, for some integer w. Two types of constructions are examined. Concrete parameter sets are provided for generators with good equidistribution properties and whose speed is comparable to that of the fastest generators currently available. The implementations use precomputed tables to speed up computations in $\mathbb{F}_{2^w}$.

1 Generators Based on Linear Recurrences in $\mathbb{F}_{2^w}$

Let $q = 2^w$ for some integer $w > 1$ and $\mathbb{F}_q$ the finite field with q elements. Consider a linear recurrence of order r in $\mathbb{F}_q$:

$$m_n = \sum_{i=1}^{r} b_i m_{n-i}, \tag{1}$$

where r is a positive integer, $b_1, \ldots, b_r$ and $m_0, m_1, \ldots$ are in $\mathbb{F}_q$, $b_r \neq 0$, and all arithmetic is performed in $\mathbb{F}_q$. The polynomial $P(z) = z^r - \sum_{i=1}^{r} b_i z^{r-i}$ is a *characteristic polynomial* of this recurrence. It is well-known that (1) has period length $q^r - 1 = 2^{rw} - 1$ (full period) if and only if $P(z)$ is primitive over $\mathbb{F}_q$. See, e.g., [8, 11] for an account of linear recurrences in finite fields. Consider also the recurrence

$$q_n(z) = z\, q_{n-1}(z) \bmod P(z), \tag{2}$$

where for each n, $q_n(z) = q_{n,1} z^{r-1} + \cdots + q_{n,r-1} z + q_{n,r} \in \mathbb{F}_q[z]/(P(z))$, the ring of polynomials in $\mathbb{F}_q[z]$ modulo $P(z)$. Dividing this recurrence by $P(z)$ yields

$$q_n(z)/P(z) = z\, q_{n-1}(z)/P(z) \bmod 1, \tag{3}$$

from which it is easy to see that one can write

$$q_n(z)/P(z) = \sum_{j=1}^{\infty} x_{n+j} z^{-j} \tag{4}$$

(a formal Laurent series), where $\{x_j,\, j \geq 1\}$ is a sequence that follows the recurrence (1) in $\mathbb{F}_q$. Moreover, by multiplying the infinite sum in (4) by $P(z)$ and equating the coefficients in $q_n(z)$ to the corresponding ones in this product, one obtains the following one-to-one linear correspondence between the vectors $(q_{n,1}, \ldots, q_{n,r})$ and $(x_{n+1}, \ldots, x_{n+r})$ in $\mathbb{F}_q$ (see [7]):

$$\begin{pmatrix} q_{n,1} \\ q_{n,2} \\ \vdots \\ q_{n,r} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ b_1 & 1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ b_{k-1} & \ldots & b_1 & 1 \end{pmatrix} \begin{pmatrix} x_{n+1} \\ x_{n+2} \\ \vdots \\ x_{n+r} \end{pmatrix}. \tag{5}$$

One can see from this that for each j, $\{q_{n,j},\, j \geq 0\}$ also follows the recurrence (1). This means that in a sense, (1), (2), and (3) are just different representations of the same recurrence.

To construct a random number generator (RNG) from such a recurrence, we must define a mapping from the generator's state space $\mathbb{F}_q^r$ to the real interval $[0, 1)$. This requires an explicit representation of the elements of $\mathbb{F}_q$. In this paper, we represent these elements in terms of an ordered polynomial basis, defined as follows. Let $M(z) = z^w + \sum_{i=1}^{w} a_i z^{w-i} \in \mathbb{F}_2[z]$ be an irreducible polynomial over $\mathbb{F}_2$. Then there exists an algebraic element ζ of $\mathbb{F}_q$ whose minimal polynomial over $\mathbb{F}_2$ is $M(z)$ and the ordered set $(1, \zeta, \ldots, \zeta^{w-1})$ is an *ordered polynomial basis* of $\mathbb{F}_q$ over $\mathbb{F}_2$ (see [8], Chapter 1.4). This means that any element $v \in \mathbb{F}_q$ can be written uniquely as a linear combination $v = v_1 + v_2\zeta + \cdots + v_w\zeta^{w-1}$ where $\mathbf{v} = (v_1, \ldots, v_w)^T \in \mathbb{F}_2^w$. Here, we identify $\mathbb{F}_2$ with the set $\{0, 1\}$ in which addition and multiplication are performed modulo 2. Thus, after $M(z)$ (or the w-bit vector $\mathbf{a} = (a_1, \ldots, a_w)$) and ζ have been chosen, each element v of $\mathbb{F}_q$ can be represented by its corresponding binary column vector $\mathbf{v}$, called its *vector representation*. Adding two elements in $\mathbb{F}_q$ corresponds to adding their vector representations componentwise in $\mathbb{F}_2$, i.e., performing a bitwise exclusive-or. The vector representation of the product of $v \in \mathbb{F}_q$ by a fixed element $b \in \mathbb{F}_q$ can be computed as $A_b\mathbf{v}$, where $\mathbf{v}$ is the vector representation of v, whereas A_b is a $w \times w$ matrix with elements in $\mathbb{F}_2$ and whose ith column is the vector representation of $b\zeta^{i-1}$, for $i = 1, \ldots, w$. For any given choice of $M(z)$, the matrix A_b turns out not to depend on the choice of ζ; it depends only on $M(z)$. Addition and multiplication in $\mathbb{F}_q$ can then be implemented efficiently via vector/matrix operations in $\mathbb{F}_2$.

In particular, the recurrence (1) can be implemented by

$$\mathbf{m}_n = \sum_{i=1}^{r} A_{b_i} \mathbf{m}_{n-i} \tag{6}$$

where $\mathbf{m}_n$ is the vector representation of m_n and A_{b_i} performs the multiplication by b_i in the vector representation, for $1 \leq i \leq r$. Under this representa-

tion, the *state* of the generator at step n can be written as the rw-bit column vector $\mathbf{s}_n = (\mathbf{m}_n^T, \mathbf{m}_{n-1}^T, \ldots, \mathbf{m}_{n-r+1}^T)^T$.

If $\mathbf{q}_{n,j}$ denotes the vector representation of $q_{n,j}$ under the chosen polynomial basis, the recurrence (2) can be implemented as

$$\mathbf{q}_n = \begin{pmatrix} \mathbf{q}_{n,1} \\ \vdots \\ \mathbf{q}_{n,r} \end{pmatrix} = \begin{pmatrix} \mathbf{q}_{n-1,2} \\ \vdots \\ \mathbf{q}_{n-1,r} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} A_{b_1} \\ \vdots \\ A_{b_r} \end{pmatrix} \mathbf{q}_{n-1,1}. \tag{7}$$

Here the state is represented by the rw-bit vector $\mathbf{q}_n$.

Both (6) and (7) are actually special cases of the matrix linear recurrence

$$\mathbf{x}_n = A\mathbf{x}_{n-1} \tag{8}$$

in $\mathbb{F}_2$, with k-bit state vector $\mathbf{x}_n = (x_{n,0}, \ldots, x_{n,k-1})^T \in \mathbb{F}_2^k$ at step n and $k \times k$ *transition matrix* A with elements in $\mathbb{F}_2$ where $k = rw$. For (6), one has $\mathbf{x}_n = \mathbf{s}_n$ and

$$A = \begin{pmatrix} A_{b_1} & A_{b_2} & \ldots & A_{b_{r-1}} & A_{b_r} \\ I_w & & & & \\ & I_w & & & \\ & & \ddots & & \\ & & & I_w & \end{pmatrix} \tag{9}$$

where I_w is the $w \times w$-bit identity matrix and the blank areas are blocks of zeros. For (7), one has $\mathbf{x}_n = \mathbf{q}_n$ and A is the transpose of the matrix in (9).

A random number generator can be constructed from a linear recurrence of the form (8) by defining a linear output function of the form

$$\mathbf{y}_n = B\mathbf{x}_n\,, \tag{10}$$

$$u_n = \sum_{i=1}^{L} y_{n,i-1} 2^{-i} \tag{11}$$

for some positive integer L, where $\mathbf{y}_n = (y_{n,0}, \ldots, y_{n,L-1})^T \in \mathbb{F}_2^w$ is the L-bit *output vector* at step n, B is an $L \times k$ matrix with elements in $\mathbb{F}_2$, the operations in (10) are performed in $\mathbb{F}_2$, and $u_n \in [0, 1)$ is the output at step n. The matrix B is called the *output transformation matrix* (or *tempering matrix*) and we will assume that none of its lines is zero. Several types of generators fit this framework, including the Tausworthe, GFSR, TGFSR, and Mersenne twister, for example [3, 5, 10, 13, 14]. Note that each coordinate of $\mathbf{x}_n$ and of $\mathbf{y}_n$ follows a linear recurrence in $\mathbb{F}_2$ whose characteristic polynomial

$$f(z) = z^k - \alpha_1 z^{k-1} - \cdots - \alpha_{k-1} z - \alpha_k = \det(A - zI), \tag{12}$$

is that of the matrix A [11]. The period length of this recurrence is $2^k - 1$ (i.e., maximal) if and only if f is primitive over $\mathbb{F}_2$. It is easy to jump ahead from

$\mathbf{x}_n$ to $\mathbf{x}_{n+\nu}$ for any large value of ν with this type of generator: it suffices to multiply the state by A^ν, which can be precomputed. This is convenient for partitioning the generator's sequence into multiple streams, as in [6].

In this paper, we consider two types of output matrices B, which we denote by T_L and M_L. The matrix T_L simply defines $\mathbf{y}_n$ as the first L bits of $\mathbf{x}_n$; it is an $L \times k$ matrix with the identity in its first L columns and zeros elsewhere. We call it the *L-bit truncation* output. The matrix M_L implements the Matsumoto-Kurita tempering, defined as follows [9]:

$$\begin{aligned}
\mathbf{y}_n &\leftarrow T_L \mathbf{x}_n \\
\mathbf{y}_n &\leftarrow \mathbf{y}_n \oplus ((\mathbf{y}_n \ll s_1) \,\&\, \mathbf{b}) \\
\mathbf{y}_n &\leftarrow \mathbf{y}_n \oplus ((\mathbf{y}_n \ll s_2) \,\&\, \mathbf{c})
\end{aligned}$$

where the operators $\oplus$, $\ll$, and & perform a bitwise XOR, a left-shift, and a bitwise AND, respectively, $\mathbf{b}$ and $\mathbf{c}$ are carefully selected L-bit vectors, and s_1 and s_2 are integer between 0 and L.

We call an RNG implemented via (6) and (10)–(11) a *linear feedback shift register* (LFSR) generator in $\mathbb{F}_q$. The tempered TGFSR generator of [9] is a special case of an LFSR in $\mathbb{F}_q$, with $B = M_L$, $b_r = \zeta$, $b_t = 1$ for some $t < r$, and all other b_i's equal to zero. The *multiple recursive matrix method* (MRMM) introduced by Niederreiter[12] uses a generalization of the recurrence (6) and a different output mapping than we do. With $L = w$ and $B = T_L$, one obtains the 2-adic digital method described in [12], Eq. (34).

The implementation (7) and (10)–(11), with $\mathbf{x}_n = \mathbf{q}_n$ instead of $\mathbf{x}_n = \mathbf{s}_n$, yields a related but different generator (because A is transposed). It can be viewed as an LCG in the polynomial space $\mathbb{F}_q[z]/(P(z))$, with the output constructed directly from the vector representations $\mathbf{q}_{n,j}$ of the polynomial coefficients. In this paper, the expression *polynomial LCG* refers to this implementation. Such polynomial LCGs were considered in [4, 5, 13] for the case where $w = 1$. (There are other places, e.g., [7] and section 4.1 of [14], where the expressions "polynomial LCG" or "LCG using polynomial arithmetic" refer to the case where $\mathbf{x}_n = \mathbf{s}_n$. We apologize for the potential confusion.)

In the special case where $L = w$ and $B = (\tilde{B}\ 0)$ where $\tilde{B}$ is an arbitrary $w \times w$ matrix, the generators based on $\mathbf{x}_n = \mathbf{s}_n$ and $\mathbf{x}_n = \mathbf{q}_n$ turn out to be equivalent, because $\{q_{n,j},\ n \geq 0\}$ follows the same recurrence as $\{m_n,\ n \geq 0\}$. This means that in this case, good parameters for LFSR generators in $\mathbb{F}_q$ are also good parameters for polynomial LCGs and vice-versa.

It is customary (e.g., [3, 5]) to assess the quality of a linear generator of the form (8) and (10)–(11) via measures of *equidistribution* of the point set

$$\Psi_t = \{\mathbf{u}_{0,t} = (u_0, \ldots, u_{t-1}) : \mathbf{x}_0 \in \mathbb{F}_2^k\}, \tag{13}$$

which contains all vectors of t successive output values produced by the generator from all 2^k possible initial states, for several values of t up to a pre-selected limit. Certain low-dimensional projections over non-successive coordinates can

also be examined [7]. The quality of the generators proposed in this paper is assessed by such equidistribution criteria detailed in section 3.

In the rest of the paper, we examine two ways of constructing efficient implementations of LFSR generators and polynomial LCGs in $\mathbb{F}_q$, using precomputed tables, with and without tempering. We then present the results of a search for parameters of generators having good equidistribution properties and a fast implementation. The search was made using the software library REGPOLY [5]. Finally, we compare the equidistribution and speed of the generators found with that of other generators proposed elsewhere. The goal is to see if we can improve on the equidistribution without paying a significant speed penalty.

2 Multiplication in $\mathbb{F}_{2^w}$ Using Precomputed Tables

This section discusses two efficient methods of implementing the multiplications in $\mathbb{F}_{2^w}$ needed in the recurrences (6) and (7). The idea of both methods is to sacrifice some memory in the interest of speed, by using precomputed tables of multiplication in $\mathbb{F}_{2^w}$. Suppose we need to multiply arbitrary w-bit vectors by the matrix A_b.

In the first method, we decompose $A_b = [A_b^{(1)}, \ldots, A_b^{(c)}]$, where $A_b^{(j)}$ is $w \times w_j$ for $1 \le j \le c$, with $w_1 + \cdots + w_c = w$. Let $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(c)}$ be the corresponding decomposition of an arbitrary w-bit vector $\mathbf{v}$. The product $A_b\mathbf{v}$ can be written as $A_b^{(1)}\mathbf{v}^{(1)} + \cdots + A_b^{(c)}\mathbf{v}^{(c)}$. For each j, one can tabulate the values of $A_b^{(j)}\mathbf{v}^{(j)}$ for all 2^{w_j} possibilities for $\mathbf{v}^{(j)}$. These tables require $(2^{w_1} + \cdots + 2^{w_c})w$ bits of storage. Then, c table lookups and $c-1$ bitwise exclusive-ors are needed to compute $A_b\mathbf{v}$ for an arbitrary $\mathbf{v}$. A smaller c means faster multiplications but more memory usage, so a compromise must be made. For example, if $w = 32$, $c = 3$, $w_1 = w_2 = 11$, and $w_3 = 10$, we need 20 kilobytes for the tables, a reasonable number. We also need a single copy of these tables, regardless of how many streams (copies of the generator, with different states) are running in parallel, in contrast with the space required to store the state, which must be multiplied by the number of streams. A nice feature of this technique is that if $B = (\tilde{B}, 0)$ where $\tilde{B}$ is $w \times w$, we can incorporate the tempering in the tables at no extra cost. This is achieved by replacing each A_{b_i} by $\tilde{B}^{-1}A_{b_i}\tilde{B}$ in the transition matrix, and B by T_L, and storing the tables of multiplication by the $\tilde{B}^{-1}A_{b_i}\tilde{B}$'s rather than by the A_{b_i}'s.

For the second method, we write

$$b = \sum_{\gamma=0}^{s_b} c_\gamma \zeta^\gamma \tag{14}$$

where $s_b < w$. Let $\Phi_b = \{\gamma : c_\gamma \neq 0 \text{ in } (14)\}$ and d_b the cardinality of Φ_b (i.e., the number of nonzero coefficients c_γ). This method is appropriate when d_b

and s_b are small. Observe that multiplying some $v \in \mathbb{F}_q$ by ζ^γ corresponds to multiplying the vector representation of v by the matrix A_ζ^γ, where A_ζ is the companion matrix

$$A_\zeta = \begin{pmatrix} & & a_w \\ 1 & & a_{w-1} \\ & \ddots & \vdots \\ & & 1 \quad a_1 \end{pmatrix}.$$

For $0 < \gamma < w$, we have

$$A_\zeta^\gamma = \begin{pmatrix} 0\,0\ldots 0 & p_{11} & \ldots & p_{1\gamma} \\ 0\,0\ldots 0 & p_{21} & \ldots & p_{2\gamma} \\ \vdots \quad\quad \vdots & \vdots & & \vdots \\ 1\,0\ldots 0 & p_{\gamma 1} & \ldots & p_{\gamma\gamma} \\ 0\,1\ldots 0 & p_{\gamma+1,1} & \ldots & p_{\gamma+1,\gamma} \\ \quad \ddots & & & \\ 0\,0\ldots 1 & p_{w1} & \ldots & p_{w\gamma} \end{pmatrix} = R_\gamma + T_\gamma$$

where R_γ performs a right shift by γ positions to the right and T_γ is the matrix with zeros in the first γ columns and whose last $w-\gamma$ columns are the same as those of A_ζ^γ. The matrix A_b can then be written as $A_b = T_b + \sum_{\gamma\in\Phi_b} R_\gamma$ where $T_b = \sum_{\gamma\in\Phi_b} T_\gamma$ is nonzero only in its last $w - s_b$ columns. The multiplication table by those $w - s_b$ nonzero columns of T_b can be stored in $2^{s_b} w$ bits. Multiplication by b is then implemented via one table lookup, d_b right shifts, and d_b bitwise exclusive-ors. The idea here is to choose characteristic polynomials $P(z)$ whose coefficients b_i have small values of d_{b_i} (for speed) and s_{b_i} (for economy of storage).

3 Search for Good Generators

Using the REGPOLY software [5], we searched for good generators with respect to equidistribution criteria, within the class of generators that can be implemented efficiently as described in the previous sections. Before giving the search results, we recall some basic definitions regarding equidistribution (see, e.g., [1, 14]), and define the selection criteria we used.

For a given integer $\ell \geq 0$, partitioning each axis of the unit hypercube $[0,1)^t$ into 2^ℓ equal parts determines a partition of this hypercube into $2^{\ell t}$ small cubes of equal volume. The RNG is called (t,ℓ)-*equidistributed*, or t-*distributed with* ℓ *bits of accuracy*, if each of these small cubes contains exactly $2^{k-\ell t}$ points from Ψ_t. This property can be verified by expressing the $t\ell$ bits of interest as linear combinations of the bits of $\mathbf{x}_0$ and checking if the matrix of the corresponding transformation has full rank, as explained in [1]. For a given ℓ, the largest t for which Ψ_t is (t,ℓ)-equidistributed is called the *dimension in*

resolution ℓ and is denoted by t_ℓ. This value has the upper-bound $t_\ell \le t_\ell^* = \lfloor k/\ell \rfloor$. The *dimension gap in resolution* ℓ, defined as

$$\Delta_\ell = t_\ell^* - t_\ell, \tag{15}$$

gives the difference between the best possible dimension in resolution ℓ and the one that is achieved. If $\Delta_\ell = 0$ for $1 \le \ell \le L$, the RNG is called *maximally equidistributed* (ME) for the word size L [1].

All search results in the paper are for $L = w = 32$. To guide our search, we looked for generators having a small value of $\sum_{\ell=1}^{32} \Delta_\ell$. For maximally equidistributed generators, this value is 0. We also tried to obtain generators for which $\Delta_\ell = 0$ for the most significant bits.

We considered full period generators for the following values of r: 3, 8, 13, 25. The corresponding period lengths are $2^{96} - 1$, $2^{256} - 1$, $2^{416} - 1$, and $2^{800} - 1$, respectively. For each r, we looked for generators having primitive characteristic polynomials of the form $P(z) = z^r + b_{r-t}z^t + b_{r-q}z^q + b_r$, sometimes with $b_{r-q} = 0$ (i.e., polynomials with only 3 or 4 nonzero coefficients). We found generators with general coefficients b_i with $B = T_L$ and generators whose coefficients b_i can be written as in (14) with small s_{b_i} and with $B = M_L$. The generators using the latter special form with $B = T_L$ appear to be limited in the quality of their equidistribution. Denoting $b_i = \sum_{\gamma=0}^{s} c_{i,\gamma}\zeta^\gamma$ for $i = 1, \ldots, r$ and $\gamma^* = \max\{\gamma : 1 \le \gamma \le s \text{ and } c_{i,\gamma} \ne 0 \text{ for some } i\}$, we have been able to find generators with $t_\ell = t_\ell^*$ for $\ell \le \gamma^*$, but we have observed empirically that $t_\ell \le r$ for $\ell > \gamma^*$. A similar limitation holds for the equidistribution of TGFSR generators without tempering [9]. To get around this limitation, we used $B = M_L$, as in [9]. In the definition of M_L, we took $s_1 = 7$, $s_2 = 15$, and used the same algorithm as in [9] to find good vectors $\mathbf{b}$ and $\mathbf{c}$.

In Table 1, we list the best generators we found with general coefficients. The coefficients are given using the polynomial basis $(1, \zeta, \ldots, \zeta^{w-1})$ where ζ is a root of the irreducible polynomial $M(z) = z^w + \sum_{i=1}^{w} a_i z^{w-i}$. In the Table, we express $M(z)$ by the bit vector $\mathbf{a} = (a_1, \ldots, a_w)$. All vectors are represented in hexadecimal notation. The period length of each generator is $\varrho = 2^{32r} - 1$. We also give the values of $E = \max\{\ell \ge 0 : \Delta_1 = \cdots = \Delta_\ell = 0\}$ and $S = \sum_{\ell=1}^{32} \Delta_\ell$, which are good indicators of the quality of equidistribution. Ideally, we want a large E and a small S.

In Table 2, we list the best generators found with coefficients of the special form (14) with $d_{b_i} \le 2$ and $s_{b_i} \le s$, for $s = 3$ and $s = 7$. The columns $\mathbf{b}$ and $\mathbf{c}$ give the vectors used for the tempering.

Some of the generators have been given explicit names in the tables, on the line that precedes their parameters. These names have the form F2wLFSRm_s_k for LFSR generators and F2wPolyLCGm_s_k for polynomial LCGs, where $m = 2$ if $b_{r-q} = 0$ and $m = 3$ otherwise, $s_{b_i} \le s$ in (14) for the b_i's that define the recurrence and $k = rw$.

In addition to E and S, we also looked at the equidisdistribution of point sets of the form

$\log_2 \rho$	r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$\mathbf{a}$	E	S
96	3	1	–	30a72fa7	–	537a531f	ccb06f34	21	3
96	3	1	–	04a87b98	–	4dd5e06e	ccb06f34	21	3
96	3	2	1	bbf58bb6	bd0c7735	b7c5019c	d53c36b9	ME	
96	3	2	1	db3bd1c3	ffbaad94	2f55958b	d53c36b9	ME	
256	8	6	3	fba454a9	045861d5	c5fb7653	ce023b3b	22	6
256	8	5	2	623a6e23	de6f829f	17600ef0	ce023b3b	22	6
416	13	8	–	2be45a08	–	b4816b12	f9820db6	17	29
416	13	5	–	7a64a92e	–	c0643058	f9820db6	17	29
416	13	10	5	99e34535	f09bf592	9803caf7	9f26eaa3	22	13
416	13	10	5	62a42238	e765704a	2f95dc0e	9f26eaa3	20	14
F2wLFSR2_31_800 or F2wPolyLCG2_31_800									
800	25	7	–	e6a68d20	–	287ab842	fa4f9b3f	15	74
800	25	18	–	26dc0579	–	88fc8c8a	fa4f9b3f	15	77
F2wLFSR3_31_800 or F2wPolyLCG3_31_800									
800	25	20	14	0001e6f1	1d5e07e3	3e433359	f70211b8	16	42
800	25	24	16	be1ed999	e21e9910	e09361e8	f70211b8	19	54

Table 1. Generators with general coefficients

$$\tilde{\Psi}_{t,i_2,\ldots,i_t} = \{\mathbf{u}_{0,t} = (u_0, u_{i_2}, \ldots, u_{i_t}) : \mathbf{x}_0 \in \mathbb{F}_2^k\}, \tag{16}$$

where $0 < i_2 < \ldots < i_t$ are positive integers. For a given dimension t, define i_t^* as the largest value of i such that whenever $i_t \leq i$, $\tilde{\Psi}_{t,i_2,\ldots,i_t}$ is $(t, \min(\lfloor k/t \rfloor, L))$-equidistributed (i.e., has optimal equidistribution in t dimensions). This is closely related to the criterion $\Delta_{t_1,\ldots,t_d}$ defined [7], which quantifies the quality of pre-selected point sets (or "projections") $\tilde{\Psi}_{t,i_2,\ldots,i_t}$. Empirically, with $L = w = 32$, $B = (\tilde{B}\ 0)$ where $\tilde{B}$ is $w \times w$ and non-singular, and $t = 2, \ldots, r-2$, i_t^* was larger than any value of i_t that we tried, for any generator. For example, with $t = 2$ and $r = 25$, we looked at all projections with $i_2 < 5000$ and they were all $(2, 32)$-equidistributed (i.e., had the best equidistribution in 2 dimensions). For some generators, we verified the equidistribution of all the point sets $\tilde{\Psi}_{t,i_2,\ldots,i_t}$ for $i_t = 16$ and $t = 2, \ldots, 16$, and found that they were all optimally equidistributed. These examples lead us to believe that the low-dimensional projections of these generators behave very well with respect to the equidistribution criterion.

4 Comparisons and Timings

In Table 3, we compare the equidistribution of two of our generators with that of tempered TGFSR generators of comparable period lengths, taken from [9]. For each generator, we give the value of t_ℓ for $\ell = 1, \ldots, 32$ and $S = \sum_{\ell=1}^{32} \Delta_\ell$. The values of t_ℓ given in boldface attain the upperbound t_ℓ^*. Our generators have better equidistribution. In particular, the last two generators in Table 2,

s	r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$M(z)$	$\mathbf{b}$	$\mathbf{c}$	E	S
3	3	2	–	30000000	–	a0000000	f6b5876b	5ccce080	71d7800c	21	3
3	3	2	–	30000000	–	a0000000	f6b5876b	792b3701	9fe700b6	21	3
7	3	2	–	0c000000	–	41000000	958357a6	8c5f6000	f00e8066	21	3
7	3	2	–	a0000000	–	12000000	958357a6	1d768200	d1e701c2	21	3
3	3	2	1	90000000	a0000000	50000000	8a81f5f4	24b97381	f9d98000	ME	
3	3	2	1	90000000	30000000	50000000	8a81f5f4	b9b76401	b24b0001	ME	
7	3	2	1	03000000	48000000	18000000	fcb5f714	a4d07c01	be2f8001	ME	
7	3	2	1	21000000	12000000	0a000000	fcb5f714	77f22481	57eb8001	ME	
3	8	5	3	a0000000	c0000000	30000000	d3e9de82	a13a9c81	5e6d801b	21	7
3	8	7	3	c0000000	50000000	60000000	d3e9de82	4c0ad481	ebd30053	21	10
3	8	7	4	60000000	90000000	c0000000	d3e9de82	b39e2581	36f30072	21	10
3	8	7	4	c0000000	90000000	30000000	d3e9de82	98fd4c01	eea3003c	21	10
7	8	5	2	03000000	44000000	28000000	ae397b58	05bf4081	eb67000c	22	6
7	8	6	3	41000000	05000000	60000000	ae397b58	1360c281	f3eb8004	22	6
3	13	5	–	50000000	–	30000000	ae8b80e1	c55b6000	fcbd0015	17	32
3	13	5	–	50000000	–	30000000	ae8b80e1	360d4401	eb31803f	17	32
7	13	8	–	0c000000	–	28000000	c65a6fe2	977e1101	fac78000	17	29
7	13	5	–	21000000	–	44000000	c65a6fe2	df850601	e3758001	17	29
F2wLFSR2_7_416 or F2wPolyLCG2_7_416											
7	13	9	6	06000000	41000000	05000000	92bb39c1	5f9bca01	fd9d8006	22	13
7	13	8	5	11000000	0c000000	30000000	92bb39c1	b8404581	22e30003	22	13
F2wLFSR2_3_800 or F2wPolyLCG2_3_800											
3	25	11	–	30000000	–	50000000	e307bc0e	f7b31a80	af530001	13	72
3	25	11	–	30000000	–	50000000	e307bc0e	f0ba1601	ab4b0000	10	75
F2wLFSR2_7_800 or F2wPolyLCG2_7_800											
7	25	11	–	05000000	–	12000000	f282ea95	a6ea0881	4de58000	9	67
7	25	9	–	09000000	–	28000000	f282ea95	fa3cc981	6cf88000	9	68
F2wLFSR3_3_800 or F2wPolyLCG3_3_800											
3	25	21	6	30000000	c0000000	a0000000	e397e5c4	994aa401	5a9d8001	9	45
3	25	19	7	c0000000	60000000	90000000	e397e5c4	b3965001	2b6c8001	13	49
F2wLFSR3_7_800 or F2wPolyLCG3_7_800											
7	25	18	13	42000000	21000000	50000000	9f1f0184	c19ee400	7e778000	21	36
7	25	13	5	12000000	28000000	06000000	9f1f0184	9e60e080	736b0000	21	37

Table 2. Generators with the form (14) with Matsumoto-Kurita tempering

one of which is the first generator in Table 3, are maximally equidistributed for up to 21 bits of resolution.

We have implemented eight of our generators in C and tested their speed by generating and adding 10^8 numbers. The test was performed on a AMD Athlon 750Mhz processor running Linux, using the `gcc` compiler with the optimisation flag `-O3`. The timings are given in Table 4. The code is available at `http://www.iro.umontreal.ca/~panneton/GenF2w.html`. For comparison, we also provide timings for three well-known generators: TT800 [9], MT19937 [10], and MRG32k3a [2]. For these generators, we used the codes given in the original papers. MRG32k3a uses integer arithmetic implemented in floating

Generator	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	
	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	$\sum_{\ell=1}^{32} \Delta_\ell$
	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	
	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	
F2wLFSR3_7_800	**800**	**400**	**266**	**200**	**160**	**133**	**114**	**100**	
F2wPolyLCG3_7_800	**88**	**80**	**72**	**66**	**61**	**57**	**53**	**50**	36
	47	**44**	**42**	**40**	**38**	34	30	25	
	25	25	25	25	25	25	**25**	**25**	
TT800	**800**	**400**	250	**200**	150	125	100	**100**	
	75	75	50	50	50	50	50	**50**	261
	25	25	25	25	25	25	25	25	
	25	25	25	25	25	25	**25**	**25**	
F2wLFSR3_7_416	**416**	**208**	**138**	**104**	**83**	**69**	**59**	**52**	
F2wPolyLCG3_7_416	**46**	**41**	**37**	**34**	**32**	**29**	**27**	**26**	13
	24	**23**	**21**	**20**	**19**	**18**	16	16	
	13	13	13	13	13	**13**	**13**	**13**	
TT403	**403**	195	130	91	78	65	52	39	
	39	39	26	26	26	26	**26**	13	140
	13	13	13	13	13	13	13	13	
	13	13	13	13	13	**13**	**13**	*	

Table 3. Comparison of equidistribution

Generator	time (seconds)
F2wLFSR3_7_800	8.2
F2wPolyLCG3_7_800	8.9
F2wLFSR2_7_800	7.4
F2wPolyLCG2_7_800	8.0
F2wLFSR3_3_800	8.1
F2wPolyLCG3_3_800	8.8
F2wLFSR2_31_800	8.0
F2wPolyLCG2_31_800	7.6
F2wLFSR3_31_800	9.9
F2wPolyLCG3_31_800	9.7
TT800	7.1
MT19937	6.6
MRG32k3a	29.7

Table 4. Time to generate and add 10^8 numbers on a AMD Athlon 750Mhz

point. All other generators use operations on bit vectors and are generally faster. Our generators are slightly slower than MT19937 and TT800, but they have better equidistribution than TT800 and a much smaller state than MT19937. The latter can become an issue when multiple streams of random numbers are maintained in parallel. Jumping ahead in the sequence is also easier for our generators than for MT19937, because the corresponding matrix A in (8) is much smaller.

5 Acknowledgments

This work has been supported by NSERC-Canada and FCAR-Québec scholarships to the first author and by NSERC-Canada grant No. ODGP0110050, NATEQ-Québec grant No. 02ER3218, and a Killam Research Fellowship to the second author.

References

1. P. L'Ecuyer. Maximally equidistributed combined Tausworthe generators. *Mathematics of Computation*, 65(213):203–213, 1996.
2. P. L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1):159–164, 1999.
3. P. L'Ecuyer. Tables of maximally equidistributed combined LFSR generators. *Mathematics of Computation*, 68(225):261–269, 1999.
4. P. L'Ecuyer and F. Panneton. A new class of linear feedback shift register generators. In J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, editors, *Proceedings of the 2000 Winter Simulation Conference*, pages 690–696, Pistacaway, NJ, 2000. IEEE Press.
5. P. L'Ecuyer and F. Panneton. Construction of equidistributed generators based on linear recurrences modulo 2. In K.-T. Fang, F. J. Hickernell, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 318–330, Berlin, 2002. Springer-Verlag.
6. P. L'Ecuyer, R. Simard, E. J. Chen, and W. D. Kelton. An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6):1073–1075, 2002.
7. C. Lemieux and P. L'Ecuyer. Randomized polynomial lattice rules for multivariate integration and simulation. *SIAM Journal on Scientific Computing*, 24(5):1768–1789, 2003.
8. R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, Cambridge, revised edition, 1994.
9. M. Matsumoto and Y. Kurita. Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation*, 4(3):254–266, 1994.
10. M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.
11. H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1992.
12. H. Niederreiter. Factorization of polynomials and some linear-algebra problems over finite fields. *Linear Algebra and its Applications*, 192:301–328, 1993.
13. F. Panneton. Générateurs de nombres aléatoires utilisant des récurrences linéaires modulo 2. Master's thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, 2000.
14. S. Tezuka. *Uniform Random Numbers: Theory and Practice*. Kluwer Academic Publishers, Norwell, Mass., 1995.