Efficient Class Matrix Congruential Generator Easily Implemented From Multiple Recursive Generators

GWEI-HUNG TSAI*, DER-JIN CHEN, CHIOU-HUA LIN

Department of Applied Statistics and Information Science

Ming Chuan University, Taoyuan County

No.5, Deming Rd., Gueishan Dist., Taoyuan City 33348,

TAIWAN, ROC

herbtsai@mail.mcu.edu.tw

Abstract: - Classical random number generators such as Linear Congruential Generators (LCGs) and Multiple Recursive Generators (MRGs) are commonly employed for modern simulation studies. LCGs have a too short period and a too coarse structure to be used as reliable random number generators. Most Matrix Congruential Generators (MCGs) are less applicable due to the inefficiency and the difficulty of deciding the too many coefficients. We propose a class of Efficient Matrix Congruent Generators (EMCGs) including EMCG-1, EMCG-2, EMCG-D and EMCG-G that simultaneously combine k LCGs to form the multiplier matrix of MCGs. We can easily derive the multiple-matrices for all four EMCGs from the corresponding full fixed minimal primitive characteristic polynomials. This makes them easy to implement the results from MRGs without any restrictions on the dimension of a multiple-matrix. We also provide the portable and efficient MAPLE algorithms without using any matrix multiplications for generating the four EMCGs with some selected coefficients that can achieve the high-dimensional uniformity with extreme long cycles.

Key-words: Linear Congruential Generators, Multiple Recursive Generators, Matrix Congruential Generators, Fast Matrix Congruential Generators.

1 Introduction

Random number generators are employed in various areas in the modern world; not only in the scientific experiments, but also for the huge simulation data in the real life. For example, in a 747 aircraft, we randomly draw 10% of passengers (40 in 400) to check for security. Therefore there will be about 1.9×10^{55} different combination which is further beyond the period of a common used Linear Congruential Generators (LCGs). In the modern era, high dimensions and superior effectiveness huge periodical uniform random number generators are heavily required in scientific researches and the simulation studies by Claudia et al., Li & Zhao, and Myszor & Cyran [1, 11, 15].

Deng and Xu suggested that a superior random number generator should have the following properties: High-dimensional uniformity, Efficiency, Long cycle, and Portability (HELP) [4].

In Section 2, we review some important results of the commonly used random number generators such as LCGs, Multiple Recursive Generators (MRGs), and Matrix Congruential Generators (MCGs).

In Section 3, we propose a system of special efficient MCGs, called EMCG generators that simultaneously combine k LCGs to form the multiplier matrix of MCGs. This special designed multiplier matrix of the MCG can achieve the maximum period, p^k-1 with the given characteristic polynomial corresponding to any minimal primitive polynomial modulo p. By using GMP efficient algorithm introduced by this study, we provide all parameters of the four EMCGs including EMCG-1, EMCG-2, EMCG-G and EMCG-D of some selected dimension k and prime modulo p . These four generators have the properties of HELP that are recommended for high dimensional extreme long period simulations.

In Section 4, we provide users the maple EMCGs algorithms to efficiently generate the random numbers for a high dimensional, uniform, extremely long period simulation used.

In Section 5, we compare the four EMCGs by calculating process and required memory spaces, and give more coefficients of EMCG-D for practical uses.

2 Related reference paper

Linear Congruential Generators (LCGs) was first proposed by Lehmer [10]. The recursive formula is as follows:

$$x_i = (\alpha\ x_{i-1} + c) \ \bmod m, \ i = 1, 2, 3, \cdots, \qquad (1)$$

where α is a multiple, c is an increment, m is a modulus, and x_0 is an initial seed. Let c, m and x_0 be nonnegative integers. Once x_0 is given, the recursive formula can produce a sequence of random numbers. When $x_i = x_0$ for some integer i, this recursive begin to generate the duplicate sequence of random numbers. The length of the non-repeat sequence is called the period.

Knuth pointed out LCG generator to produce a sequence with period less or equal to modulus m [9]. These LCGs have a too short period and a too coarse structure; therefore, they cannot be used as reliable random number generators.

2.1 Multiple Recursive Generators

A Multiple Recursive Generator (MRG) is an extension of LCG with a k-terms polynomial. Its recursive formula is as follows:

$$x_i = (\alpha_1\, x_{i-1} + ... + \alpha_k\, x_{i-k}) \ \bmod p, \qquad (2)$$

where k is the positive integer, p is the prime modulus, $\alpha_1, \alpha_2, \cdots, \alpha_k$ are integer multipliers between 0 to $p-1$, $\alpha_k \neq 0$ and at least one other $\alpha_j \neq 0.$ Given k initial nonzero seeds $(x_0, x_1, \cdots, x_{k-1}),$ it follows by the recursive formula to produce a sequence of random numbers.

Knuth defined the recursive formula with its corresponding characteristic polynomial as [9]:

$$f(x) = x^k - \alpha_1 x^{k-1} - \cdots - \alpha_k. \qquad (3)$$

When $f(x)$ is a minimal primitive polynomial modulo $p,$ the MRG in equation (2) can achieve the maximum period, $p^k - 1.$ Knuth, Lidl and Niederreiter confirmed that a maximum period MRG of order k enjoys a nice equi-distribution property up to k dimensions [9, 12].

These MRGs usually need many multiplications and additions in the recursive formula, and hence they are less efficient for producing random numbers. Marsaglia took all the nonzero multiplier $\alpha_i's$ equal to the same constant α for improving the efficiency [14].

Deng et al. considered a class of DL-k generators with all nonzero multipliers equal, equation (4) or alternately negate equal, equation (5) as the following two recursive formulas [6]:

$$x_i = \alpha\, x_{i-1} + \alpha\, x_{i-2} + \cdots + \alpha\, x_{i-k+1} + \alpha\, x_{i-k} \ \bmod p; \quad (4)$$

$$x_i = \alpha\, x_{i-1} - \alpha\, x_{i-2} + \cdots - \alpha\, x_{i-k+1} + \alpha\, x_{i-k} \ \bmod p. \quad (5)$$

These two generators with corresponding characteristic polynomials carry out in the following forms:

$$f_1(x) = x^k - \alpha\, x^{k-1} \mp \alpha\, x^{k-2} - \cdots \mp \alpha\, x - \alpha. \quad (6)$$

Such generators can be implemented efficiently.

Deng et al. recommended another more efficient MRG called DT generators, which have all nonzero multiples with geometric weights [5]:

$$x_i = \alpha^k x_{i-1} + \alpha^{k-1} x_{i-2} + \cdots + \alpha^2 x_{i-k+1} + \alpha x_{i-k} \bmod p. \quad (7)$$

The corresponding characteristic polynomials are achieved in the following forms

$$f(x) = x^k - \alpha^k x^{k-1} - \alpha^{k-1} x^{k-2} - \ldots - \alpha^2 x - \alpha. \quad (8)$$

DT generators can be efficiently implemented by the (k+1)-order equation below:

$$x_i = ((\alpha^{-1} + \alpha^k) x_{i-1} - x_{i-k-1}) \bmod p, \quad (9)$$

where $D = (\alpha^{-1} + \alpha^k) \bmod p$ can be pre-computed and $\alpha^{-1} = \alpha^{p-2} \bmod p$. DT generators compared with DX-K-s and DL generators are the most efficient with better empirical statistical testing results.

2.2 Matrix Congruential Generators

Franklin, Grothe and Niederreiter suggested a recursive formula involving matrix multiplications and name them Matrix Congruential Generators (MCGs) as follows [7, 8, 16, 17]:

$$x_i = Ax_{i-1} \bmod p. \quad (10)$$

Here $x_i \in F^{k\times 1}$ is a positive integer vector (F is a finite field) and $A \in F^{k\times k}$ is a $k\times k$ multiplier matrix. Select p as a prime modulus, and x_0 as a nonzero k dimensional vector. A MCG produces a vector sequence with maximal period $p^k - 1$. To compute the next vector of the MCG, $k\times k$ integer's size memory is needed for storing the matrix coefficients in addition to memory for storing two seed vectors (x_i and x_{i-1}) of k integers. Determining the coefficients for the multiplier matrix A, is very computing intensive.

To improve the efficiency of MCGs, Deng and Lin designed a special multiplier-matrix A and named it Fast Matrix Congruential Generators (FMCGs) [3]. This matrix with a recursive formula is given as equation (11).

$$\begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,k-1} \\ x_{i,k} \end{bmatrix} = \begin{bmatrix} \alpha_1 & -1 & \cdots & 0 \\ 0 & \alpha_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \\ -1 & 0 & \cdots & \alpha_k \end{bmatrix} \begin{bmatrix} x_{i-1,1} \\ x_{i-1,2} \\ \vdots \\ x_{i-1,k-1} \\ x_{i-1,k} \end{bmatrix} \bmod p. \quad (11)$$

Wikramaratna propose that a special ACORN matrix generator sharing the same set of multipliers of any MRG with minimal primitive polynomial still achieves the maximal period [19]. The multiplier matrix is defined as following:

$$\mathrm{G}_k = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_k & \alpha_{k-1} & \cdots & \alpha_2 & \alpha_1 \end{bmatrix} = \begin{bmatrix} \vec{0}^T & I \\ \alpha_k & \vec{\mathrm{a}} \end{bmatrix} \quad (12)$$

Here $\vec{0}^T$ is a column vector of $(k-1)$ zeroes; I represents the $(k-1)$ by $(k-1)$ identity matrix; $\vec{\mathrm{a}} = [\alpha_{k-1}\, \alpha_{k-2} \cdots \alpha_1]$ is a row vector.

Tang proposes that a MCG using the inverse of the above G_k matrix as the multiplier matrix still achieve the maximal period [18].

3 Main result

In this study we simultaneously combine k LCGs with the selected multipliers putting in the matrix A in equation (10) to design a new class of MCG where the multiplier matrix $A = \mathrm{E}_k$ is such that

$$\mathrm{E}_k = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & \alpha_k \\ 1 & 0 & \cdots & 0 & 0 & \alpha_{k-1} \\ 0 & 1 & \cdots & 0 & 0 & \alpha_{k-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & \alpha_2 \\ 0 & 0 & \dots & 0 & 1 & \alpha_1 \end{bmatrix} = \begin{bmatrix} \vec{0} & \alpha_k \\ I & \vec{\mathrm{a}}^{\mathrm{T}} \end{bmatrix}. \quad (13)$$

Here $\vec{0}$ is a row vector with $k-1$ zeroes, I is an (*k-1*) by (*k-1*) identity matrix; $\vec{\mathrm{a}}^{\mathrm{T}} = [\alpha_{k-1}\ \alpha_{k-2} \cdots \alpha_1]^{\mathrm{T}}$ is a column vector of $k-1$ selected multipliers. Note that the matrix E_k is the transpose of matrix G_k. The corresponding k recursive equations can be expressed as:

$$x_{i,1} = \alpha_k\, x_{i-1,k} \bmod p$$
$$x_{i,2} = \alpha_{k-1}\, x_{i-1,k} + x_{i-1,1} \bmod p$$
$$x_{i,3} = \alpha_{k-2}\, x_{i-1,k} + x_{i-1,2} \bmod p$$
$$\vdots$$
$$x_{i,k} = \alpha_1\, x_{i-1,k} + x_{i-1,k-1} \bmod p \quad (14)$$

Theorem 3.1: The designed matrix E_k as in equation (13) of the MCG can achieve the maximum period, $p^k - 1$ and the corresponding characteristic polynomial of E_k is as follows:

$$f_2(x) = x^k - \alpha_1 x^{k-1} - \alpha_2 x^{k-2} - \dots - \alpha_{k-1} x - \alpha_k \quad (15)$$

Proof: E_k is the transpose matrix of G_k. Hence both E_k and G_k. share the same characteristic polynomial as equation (15).

Considering the following matrix multiplication with suitable partition, we let

$$\mathrm{F}\,\mathrm{G}_k = \begin{bmatrix} P & Q \\ R & s \end{bmatrix} \begin{bmatrix} \vec{0}^T & I \\ \alpha_k & \vec{\mathrm{a}} \end{bmatrix} = \begin{bmatrix} T & U \\ 0 & \mathrm{V} \end{bmatrix} = E_k. \quad (16)$$

Here *P* is a (*k-1*) by (*k-1*) matrix, *Q* is a (*k-1*) by 1 matrix, *R* is a 1 by (*k-1*) matrix, and *s* is a number for the pre-multiplier matrix F. T= $[0 1 \cdots 0]^T$ is a (*k-1*) elements column vector, 0 is a number, $V = [0 \quad 0 \quad \dots \quad 1 \quad \alpha_1]$ is a (*k-1*) elements row vector, and U is a (*k-1*) by (*k-1*) matrix as following,

$$U = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & \alpha_k \\ 0 & 0 & \cdots & 0 & 0 & \alpha_{k-1} \\ 1 & 0 & \cdots & 0 & 0 & \alpha_{k-2} \\ 0 & 1 & \cdots & 0 & 0 & \alpha_{k-3} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & \alpha_2 \end{bmatrix}. \quad (17)$$

Now we can get the following 4 simultaneous equations as:

$$T = P\vec{0}^T + \alpha_k Q \quad (18)$$
$$U = PI + Q\vec{\mathrm{a}} \quad (19)$$
$$0 = R\vec{0}^T + s\alpha_k \quad (20)$$
$$V = RI + s\vec{\mathrm{a}} \quad (21)$$

Then we can find out the solutions as following:

i. $s = 0$,

ii. $Q = [0\ \alpha_k^{-1} \cdots 0]^T$,

iii. R = V,

and finally the $(k-1)$ by $(k-1)$ matrix *P*=

$$\begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & \alpha_k \\ \dfrac{-\alpha_{k-1}}{\alpha_k} & \dfrac{-\alpha_{k-2}}{\alpha_k} & \cdots & \dfrac{-\alpha_3}{\alpha_k} & \dfrac{-\alpha_2}{\alpha_k} & \alpha_{k-1} - \dfrac{\alpha_1}{\alpha_k} \\ 1 & 0 & \cdots & 0 & 0 & \alpha_{k-2} \\ 0 & 1 & \cdots & 0 & 0 & \alpha_{k-3} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & \alpha_2 \end{bmatrix} \quad (22)$$

The multiplier matrix F does exist. Consider the equation for the determinant,

$$|\mathrm{F}\,\mathrm{G}_k| = |F||\mathrm{G}_k| = |E_k| = \alpha_k = |\mathrm{G}_k|.$$

We have $|F| = 1$. Hence F is nonsingular, the kernel is the set of 0 vector, and it is one to one. Now suppose the period of MCG with multiplier matrix G_k achieves maximal period, $p^k - 1$, so does MCG with the given multiplier matrix E_k using the same set of multipliers.

In this way, we can easily select the multiplier matrix of MCG with characteristic polynomial corresponding to any given minimal primitive polynomial modulo *p*.

By carefully selecting the entries of the last column of multiplier-matrix *A*, we can easily generate a non-repeated vector sequence with maximal period, $p^k - 1$. Our special design of choosing the entries of A with a prime dimension k will achieve highly efficiency for matrix multiplication in recursive computing processes.

***Example* 3.1:** We first consider the MCG derived from the DL generators, named here as EMCG-1 and EMCG-2, respectively, with the multiplier-matrix A_1 and A_2 corresponding characteristic polynomials shown in equation (6). The multiplier-matrices with an odd prime order k are as following :

$$A_1 = \begin{bmatrix} \vec{0} & \alpha \\ I & \vec{a} \end{bmatrix}, \text{for EMCG-1,}$$

$$\text{where } \vec{a} \text{ is the } [\alpha\ \alpha \cdots \alpha]^{\mathrm{T}} \text{ vector, and}$$

$$A_2 = \begin{bmatrix} \vec{0} & \alpha \\ I & \vec{a} \end{bmatrix}, \text{for EMCG-2,}$$

$$\text{where } \vec{a} \text{ is the } [-\alpha\ \ \alpha \cdots -\alpha\ \ \alpha]^{\mathrm{T}} \text{ vector.}$$

For an efficient calculation, we can store the results $t_a = \alpha\, x_{i-1,k}$ for later recursively use. We should also consider the special structure of the multiplier matrix by using the reverse order of index to calculate the seed vector. We first calculate the seed vector element $x_{i,k}$, then get $x_{i,k-1}, \ldots, x_{i,3}$, $x_{i,2}$ consequently, and finally obtain the $x_{i,1}$ which is the previously stored $t_a = \alpha\, x_{i-1,k}$. By this rotating algorithm, we only need one seed vector for both vectors x_i and x_{i-1}. This first index, *i* is no longer needed. This generator produce the next *k*-tube vector seed using only one multiplication and (*k*-1) additions. The actually computational process is given as following:

1. Calculate $t_a = \alpha x_{[k]} \bmod p$ and store it.
2. $x_{[k]} = x_{[k-1]} + t_a \bmod p$
3. $x_{[k-1]} = x_{[k-2]} \pm t_a \bmod p$

$\vdots$

k. $x_{[2]} = x_{[1]} \pm t_a \bmod p$

k+1. $x_{[1]} = t_a$ (23)

***Example* 3.2:** As example 3.1, we change the last column coefficients of the multiplier-matrix of A_1 to an arithmetic series with common difference d to increase its variation and name it EMCG-D. The recursive formula with the multiplier-matrix is as follows:

$$A_3 = \begin{bmatrix} \vec{0} & \alpha \\ I & \vec{a} \end{bmatrix} \text{ where } \vec{a} \text{ is the}$$

$$[\alpha + d\ \ \alpha + 2d \cdots\ \ \alpha + (k-1)d]^{\mathrm{T}} \text{ vector.}$$

The corresponding characteristic polynomial of the multiplier-matrix now becomes

$$f_3(x) = x^k - x^{k-1}(\alpha + (k-1)d) - \ldots - x(\alpha + d) - \alpha. \quad (24)$$

For an efficient calculation, we can store the results $t_a = \alpha\, x_{i-1,k}$ and $t_d = d\, x_{i-1,k}$ for later recursively used. The same reason as example 3.1, we should use reverse order to calculate so that two seed vectors x_i and x_{i-1} become one x. The first index, i is no longer needed. We then use another temporarily stored variable $t_v = (k-1)t_d + t_a \bmod p$ for later recursively used. The $x_{[k]}, x_{[k-1]}, \cdots x_{[2]}$ can then be obtained in order by:

$$x_{[j]} = x_{[j-1]} + t_v \bmod p,$$

$$\text{then update } t_v = t_v - t_d \bmod p,$$

and finally obtain the $x_{i,1}$ which is the previously stored t_a. This EMCG-D generator will apply $2\times(k-1)$ additions and only 3 multiplications to produce the next seed vector.

Example 3.3: Now if we apply the corresponding characteristic polynomial, equation (8), results from DT generators, the multiplier matrix A and the recursive formula with the multiplier-matrix is given as follows:

$$A_4 = \begin{bmatrix} \vec{0} & \alpha \\ I & \vec{\mathrm{a}} \end{bmatrix} \text{ where } \vec{\mathrm{a}} \text{ is}$$
$$\left[\alpha^2 \ \alpha^3 \cdots \ \alpha^k\right]^{\mathrm{T}} \text{ vector}.$$

We name this Generator as EMCG-G, for the multiplier matrix with the last column coefficients becoming to a geometric series the first term α and the common ratio α. The same reason as example 3.1, we should use reverse order to calculate so that two seed vectors x_i and x_{i-1} become one and we can omit the first index. We then introduce another varying stored variable $t_v = a^k x_{[k]} \bmod p$ for later recursively used. The $x_{[k]}, x_{[k-1]}, \cdots x_{[2]}$ can then be obtained in order by the $x_{[j]} = x_{[j-1]} + t_v \bmod p$ then update $t_v = t_v \times a^{-1} \bmod p$, and finally obtain the $x_{i,1}$. This procedure will require to find the $a^k \bmod p$ and $a^{-1} \bmod p = a^{p-2} \bmod p$ which should be calculated in the initial procedure once the α is selected. This EMCG-G generator will apply $(k-1)$ additions and only k multiplications to produce the next seed vector. EMCG-Gs have good variations to improve the empirical property.

In this study we first select a fixed prime number term k and choose the largest prime modulus p less than 2^{31} satisfying the Generalized Mersenne Prime (GMP) condition: $R(k,p) = (p^k - 1)/(p-1)$ defined by Deng [2]. By using GMP efficient algorithm, we obtain the following coefficients of α for EMCG-1, EMCG-2, EMCG-G; α and d for EMCG-D of various dimensions k as shown in Table 1. The parameters include prime rank of multiplier matrix (k), prime modulus (p), multiplier (α) and a common difference (d) for EMCG-D, α^k and α^{-1} for EMCG-G.

Table 1: The Used Parameters for All Generators

Rank (K), Modulus (p), Multiplier (α), (d) for EMCG-D, α^k and α^{-1} for EMCG-G

Generator	***k***	p	α	***d***	α^k	α^{-1}
EMCG-1	97	2147482621	1048148			
EMCG-2	97	2147482621	1048501			
EMCG-D	199	2147481173	1048575	10005		
EMCG-G	907	2143082759	2361		1323649103	561867102

4 EMCG MAPLE programs

Note that the coefficients of the last column of the multiplier matrix $\mathbf{E}_k$ of equation (13) can be selected from any previously obtained minimal primitive polynomial modulo *p*. For example, consider the MRG of order 7 with all nonzero coefficients in L'Ecuyer et al. [13]. The characteristic polynomial with $p=2^{31}-1$ is as follows:

$$f(x)=x^7-1975938786x^6-875540239x^5 -433188390x^4-451413575x^3 -1658907683x^2-1513645334x -1428037821 \quad (25)$$

In this case, we will need more memory space to store all the different coefficients which is not suggested compared with all the predefined four EMCGs.

We use the MAPLE software to carry out this research due to the feature of symbolic programing and efficiently calculating in finite field. We can find out the $a^{-1} \bmod p = a^{p-2} \bmod p$ and $a^k \bmod p$ without difficulty, and use it to easily discover all the required coefficients of minimal primitive polynomial. The program for EMCG-1 is almost the same as the program for EMCG-2 except the statement:

```
TA:=A_p - 1*TA;
```

Hence we do not provide this program in this paper. The interested researchers can easily translate these maple programs to other computer program language. The MAPLE program of three generators, EMCG-2, EMCG-D, and EMCG-G with the coefficients shown in table 1 are given as follows:

#MAPLE program: Initial EMCG-2

```
initial_emcg2:=proc()
#initial the global variables for generating the random
    numbers
global A_x, A_j, A_k, A_p, A_a;
#setup the parameters: A_k(dimension)、A_p(modulus)、
    A_a(multiplier)
A_k:=97;A_a:=1048501;A_p:=2147482621;
#initial the index A_j
A_j:=A_k;
#initial the seed vector
A_x:=< seq( irem(rand(), A_p), i=1..A_k)>;
end proc;
```

# EMCG-2 the Generator procedure

```
emcg2:=proc()
global A_x, A_j, A_k, A_p, A_a;
local R,TA, i;
#if all the k random numbers in seed vector are used
        (A_j=A_k),we need to reset the seed vector
if A_j=A_k then
#calcullate the TA keep it as a temp variable
  TA:=irem(A_a* A_x[A_k],A_p);
# in reverse order, we need only one vector seed
  for i from (A_k) to 2 by -1 do
      A_x[i]:= irem(TA+A_x[i-1],A_p);
      TA:=A_p - 1*TA;    # TA:= - 1*TA;
#omit this for EMCG-1
      od;
#reset the index A_j to 0
  A_x[1]:= TA;    A_j:=0:
fi;
#to get a random number we increase the index and take
    one from the seed vector
A_j:=A_j+1:
R:=A_x [A_j]:
end proc;
```

#MAPLE program: Initial EMCG-D

```
initial_emcgD:=proc()
#initial the global variables for generating the random
    numbers
global D_x,D_j,D_k,D_p, D_a, D_d;
#setup the parameters: D_k(dimension)、D_p(modulus)、
    D_a(multiplier)
D_k:=199;D_a:=1048575;D_p:=2147481173;
#initial the index D_j and D_d(parameter for difference)
D_j:=D_k; D_d:=10573;
#initial the seed vector
D_x:= <seq( irem(rand(), D_p), i=1..D_k)>;
end proc;
```

# EMCG-D the Generator procedure

```
emcgD:=proc()
global D_x,D_j,D_k,D_p, D_a, D_d;
local R,TA, TD, Tv, j;
#if all the k random numbers in seed vector are used
     (D_j=D_k), # we need to reset the seed vector
if D_j=D_k then
#calcullate the TA¸TD   keep it as a temp variable
   TA:=irem(D_a* D_x[D_k],D_p);
   TD:= irem(D_d* D_x[D_k],D_p);
   Tv:= irem((D_k-1)*TD+TA,D_p);
#To in reverse order, we need only one vector seed
   for j from D_k to 2 by -1 do
       D_x[j]:= irem(Tv+D_x[j-1],D_p);
       Tv:=irem(D_p + Tv - TD, D_p);
       od;
#reset the index D_j to 0
   D_x[1]:= TA;    D_j:=0:
fi;
#to get a random number we increase the index and take
    one from the seed vector
D_j:=D_j+1:    R:=D_x [D_j]:
end proc;
```

#MAPLE program: Initial EMCG-G

```
initial_emcgG:=proc()
#initial the global variables for generating the random
    numbers
global G_x,G_j,G_k,G_p, G_a, G_a_k,   G_a_inv;
#setup the parameters: G_k(dimension)、G_p(modulus)
   、G_a(multiplier)
G_k:=907;G_a:=2361;G_p:=2143082759;
# G_a_k (a^k mod p=1323649103), G_a_inv (a^(-1) mod
p=561867102 )
G_a_k:= 1323649103; G_a_inv := 2143082759;
G_j:=G_k;    #initial the index G_j
#initial the seed vector
G_x:=<seq( irem(rand(), G_p), i=1..G_k)>;
end proc;
```

# EMCG-G the Generator procedure

```
emcgG:=proc()
global G_x,G_j,G_k,G_p, G_a, G_a_k, G_a_inv;
local R, TV, j;
#if all the k random numbers in seed vector are used
    (G_j=G_k), we need to reset the seed vector
if G_j=G_k then
#calcullate the TV (a^k * X[i-1, k] mod p) keep it as
     a temp variable
     TV:= irem(G_a_k* G_x[G_k],G_p);
# in reverse order, we need only one vector seed
     for j from (G_k) to 2 by -1 do
       G_x[j]:= irem(TV+G_x[j-1],G_p);
       TV:=irem(TV*G_a_inv,G_p);
     od;
     G_x[1]:= TV; G_j:=0: #reset the index G_j to 0
fi;
#to get a random number we increase the index and take
    one from the seed vector
G_j:=G_j+1:   R:=G_x [G_j]:
end proc;
```

5 Comparison and Suggestion

In this study all the proposed four efficient MCGs: EMCG-1, EMCG-2, EMCG-D and EMCG-G are not restricted by the matrix dimension. Table 2 shows the summary resources used for the four EMCGs such as the calculating processes including times of multiplications, “×” and additions, “+”, required memory space for seed, multiplier and modulus-MM, temporarily stored variables and index-TI.

Table 2: The Summary for Comparing Generators

Generators Name	**k**	**Calculating Process**		**required Memory Space**		
		×	+	**Seed**	**MM**	**TI**
EMCG-1	*k*	*1*	*k-1*	*k*	*3*	*4*
EMCG-2	*k*	*1*	*k-1*	*k*	*3*	*4*
EMCG-G	*k*	*k*	*k*	*k*	*4*	*6*
EMCG-D	*k*	*3*	*2*(k-1)*	*k*	*5*	*4*

Considering to enhance the variations of generating sequence, EMCG-D and EMCG-G are recommended. In table 3, we provide some selected coefficients of EMCG-D for various prime rank of multiplier matrix (*k*) with prime modulus (p), multiplier (α), and in each case we offer five common difference (*d*) for practical uses.

Table 3: *S*everal Practical Coefficients of EMCG-D

k	***p***	***a***	**Five *d's***				
47	2147479991	1048540	10027	10045	10114	10176	10308
97	2147482621	1048148	10069	10130	10377	10607	10648
199	2147481173	1048575	10005	10042	10573	10776	10805
293	2147475439	1048575	10008	10775	11895	12505	13544
397	2147472413	1048572	10533	11460	11758	12476	12492

6 Conclusion

In summary, we propose a system of EMCG generators with the following desired properties: high dimensional uniformity, efficiency, long cycle, and portability. With selected parameters our EMCGs will produce an uniformly in k dimension distributed vector sequence with maximal period $p^k - 1$. The EMCGs are almost as efficient as the LCG because only a single multiplication and an addition in average are required to produce the next vector. We provide the MAPLE algorithms not involving any matrix multiplications so that one can produce the same sequence on various platforms. The MCG has postponed for quite a while due to the complicate structure and calculations with quite a few coefficients to be decided. FMCGs could be similar faster random number generators, but are with the indefinite characteristic polynomials required to calculate with the multiplier matrices case by case. After this study we could try to find more patterns of multiplier matrix that can achieve the maximal period. Since the obtained primitive characteristic polynomials of the EMCG-D multiplier-matrix are also suitable for the MRGs and can be further explored for future study. The future investigators can also apply some package such as TESTU01 or DIEHARD to check the empirical properties of the generated random vectors sequence obtained from EMCGs.

References

[1] Claudia C., Concetta G., and Nicola P. (2010) "Fluid flow modeling of a coastal fractured karstic aquifer by means of a lumped arameter approach". Computers And Simulation In Modern Science Volume III, ISBN: 978-960-474-256-1, ISSN: 1792-6882. pp. 473-482.

[2] Deng, L. Y. (2004). "Generalized Mersenne Prime Number and Its Application to Random Number Generation". Monte Carlo and Quasi-Monte Carlo Methods 2002. Springer-Verlag, pp. 167-180.

[3] Deng, L. Y. and Lin, D. K. J. (2000). "Random Number Generation for the New Century". *American statistician,* 54(2), pp. 145-150.

[4] Deng, L. Y. and Xu, H. Q. (2003). "A System of High- dimensional, Efficient, Long-cycle and Portable Uniform Random Number Generators". *ACM transactions on modeling and computer simulation*, 13(4), pp. 299-309.

[5] Deng, L. Y., Shiau, J. J. H., and Tsai, G. H. (2009). Parallel random number generators based on large order multiple recursive generators. Monte Carlo and Quasi-Monte Carlo Methods 2008, Springer-Verlag. 289-296.

[6] 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". MCQMC 2006 Springer-Verlag. pp. 263-273.

[7] Franklin, J. N. (1964). "Equidistribution of matrix - power residues modulo one". *Mathematics of Computation*, 18, pp. 560-568.

[8] Grothe, H. (1987). "Matrix generators for pseudo-random vector generation". *Statistical Papers*, 28, pp. 233-238.

[9] Knuth, D. E. (1998). The Art of Computer Programming. Vol.2: Seminumerical algorithms, 3rd ed. Addison-Wesley: Reading, MA.

[10] Lehmer, D. H. (1951). "Mathematical methods in large-scale computing units", proceeding of a second symposium on large-scale digital calculating machinery. Cambridge: Harvard University Press, Vol. 26.

[11] Li, M. and Zhao W. (2009). "A Note on Simulation of LRD Network Traffic". Proceedings of the 8th WSEAS International Conference on Instrumentation, Measurement, Circuits and Systems, ISBN: 978-960-474-076-5, ISSN: 1790-5117, pp. 25-30.

[12] Lidl, R. and Niederreiter, H. (1994). "Introduction to Finite Fields and Their Applications", revised edition. Cambridge University Press, Cambridge, MA.

[13] L'Ecuyer, P., Blouin, F., Couture, R. (1993). A Search for Good Multiple Recursive Random

Number Generators. *ACM Transactions on Modeling and Computer Simulation*. 3(2), 87-98.

[14] Marsaglia, G. (1996). "The Marsaglia random number CDROM including the DIEHAED battery of tests of randomness". http://stat.fsu.edu/pub/diehard.

[15] Myszor D., Cyran K. A. (2010) "Non-enzymatic template-directed RNA recombination processes in Monte Carlo simulation model of the RNA World", LATEST TRENDS on SYSTEMS Volume II, ISBN: 978-960-474-214-1, ISSN: 1792-4235, pp. 676-681.

[16] Niederreiter, H. (1986). "A pseudorandom vector generator based on finite field arithmetic". *Math. Japonica*, 31, pp. 759-774.

[17] Niederreiter, H. (1995). "The multiple -recursive matrix method for pseudorandom number generation". *Finite Fields and their Applications*, 1, pp. 3-30.

[18] Tang, H. C. (2005), Reverse multiple recursive random number generators, *European Journal of Operational Research*, 164, pp. 402-405.

[19] Wikramaratna, R. C.(2008), Reverse multiple recursive random number generators, *Journal of Computational and Applied Mathematics*, 216, pp. 371-387.