Implementation of the Pseudo-Random Numbers Generators and the Low Discrepancy Sequences

Anne GILLE-GENEST

March 1, 2012

Contents

Theoretical aspect for generators is described in the following parts:
Pseudo-Random Numbers Generators
Low Discrepancy Sequences

Some of the algorithms we detail in this part are explained in [4].

1 Implementation

We use the generators implemented in PNL. These generators are now thread-safe and stored in the following structure

```
typedef struct _PnlRng PnlRng;
struct _PnlRng
{
  PnlObject object;
  PnlRngType type; /*!< generator type */
  void (*Compute)(PnlRng *g, double *sample); /*!< the function to compute the
                                                  next number in the sequence */
  int rand_or_quasi; /*!< can be MC or QMC */
  int dimension; /*!< dimension of the space in which we draw the samples */
  int counter; /*!< counter = number of samples already drawn */
  int has_gauss; /*!< Is a gaussian deviate available? */
  double gauss; /*!< If has_gauss==1, gauss a gaussian sample */
  int size_state; /*!< size in bytes of the state variable */
  void *state; /*!< state of the random generator */
};
```

We refer the reader to the documentation of the PNL for the list of implemented generators and how to call them.

2 Pseudo-Random Numbers Generators

2.1 Schrage Method

We first present the Schrage Method. It will be used for several implementations of Pseudo-Random Number generators.
Let x_n be a MLCG defined by the recurrence relation :

$$x_{n+1} = ax_n \mod m$$

For a large prime modulus m, the implementation may lead to overflow in the product ax for $x < m$. To avoid this problem, we can use the Schrage Method, based on the fact that $a(m \mod a) < m$ under the condition $a^2 \leq m$. It is described as follows :
We consider the decomposition $m = aq + r$ where $r = m \mod a$ and $q = [m/a]$. Thus, we have :

$$(ax) \mod m = \begin{cases} a(x \mod q) - r[x/q] & \text{if it is } \geq 0 \\ a(x \mod q) - r[x/q] + m & \text{if it is } \leq 0 \end{cases}$$

If $r < q$ and $0 < x < m-1$, then both $a(x \mod q)$ and $r[x/q] \in \{0, \dots, m-1\}$.

2.2 Park & Miller algorithm

• **Description:** CMRG description
We consider the random number generator U_n of Park & Miller with Bayes and Durham shuffling procedure. It is based on a Multiplicative Linear Congruential Generator (MLCG) X_n, with parameters:

$$a = 7^5 = 16807,\ m = 2^{31} - 1 \simeq 2,15.10^9.$$

The implementation uses the Schrage decomposition $m = aq + r$ with $q = 127773$ and $r = 2836$.
The period of this generator is $\rho = 2^{31} - 2 \simeq 2,1.10^9$.

• **Algorithm:**

- /* Initialisation */

Choice for the initial value X_0

- /* First call to the sequence */

/* Ensure that X_0 differs from 0 */

- /* After 8 "warm-ups", initialisation of the shuffle table */

/* Schrage's method to avoid overflows */
The table $t[\ ]$ contains the first 32 values of the sequence.

- /* For each call to the sequence, computation of a new point */

The next term U_{n+1} of the sequence is computed as follows:
The value X_{n+1} is computed by the Schrage method.
/* Shuffling procedure of Bayes & Durham */
/* Index j dependent on the last point */
/* Next point dependent on j */
We select the value $y = t[j]$ and return $U_{n+1} = y/m$, where the index j depends on the last value y: $j = y/N_1$ with $N_1 = (1 + \frac{m-1}{32})$.
We place the value X_{n+1} in $t[j]$.

2.3 L'Ecuyer generator

• **Description:** CMRG description

We consider the random number generator of L'Ecuyer with Bayes and Durham shuffling procedure. It is based on the combination of 2 LCG, X_n and Y_n with parameters:

$$a_1 = 40014,\ m_1 = 2^{31} - 1;$$
$$a_2 = 40692,\ m_2 = 2^{31} - 249.$$

We use the Schrage decomposition with $q_1 = 53668$, $r_1 = 12211$ and $q_2 = 52774$, $r_2 = 3791$ to compute the values X_n and Y_n.
Shuffle procedure is applied only to the first sequence X_n. The computed value is

$$Z_n = (X_n(j) - Y_n) \mod m_1$$

where $X_n(j)$ is the value found in the shuffle table. If the value is negative, add m_1.
The output value is $U_n = \frac{Z_n}{m_1}$.
The period of this generator is given by:

$$\rho = (m_1 - 1)(m_2 - 1)/2 \simeq 2, 3.10^{18}$$

• **Algorithm:**

- /* First call to the sequence */

Choice of the initial values X_0 and Y_0.

- /* After 8 "warm-ups", initialisation of the shuffle table */

/* Park & Miller's generator */
The table $t[\ ]$ contains the first 32 values of the sequence X_n.

- /* For each call to the sequence, computation of a new point */

The next term Z_{n+1} is computed as follows:
/* First generator */ X_{n+1}
/* Second generator */ Y_{n+1}
/* Shuffling procedure of Bayes & Durham */
/* Index j dependent on the last point */
/* Next point dependent on j */
We calculate the value of j and we select $t[j]$ in the shuffle table.
We return $Z_{n+1} = (t[j] - Y_{n+1})/m_2$
We replace $t[j]$ by the value X_{n+1}.

- /* To avoid 0 value */

2.4 Knuth generator

- **Description:** MRG description

The generator of Knuth is based on a MRG. Its recurrence relation is given by:

$$X_n = (X_{n-24} + X_{n-55}) \mod m$$

with $m = 10^9$.

- **Algorithm:**

- /* Initializes the sequence with a positive seed */
Choice of the initial value X_0.

- /* First call to the sequence */
/* Initialization of the table */
The table $t[\,]$ contains the first 55 values of the sequence. Indices 1 to 55.
/* Randomization of the elements of the table */
The table $t[\,]$ contains now the first 55 values of the sequence in a random order.

/* Initialization of the indices inc1 and inc2 */
Indices i_1 and i_2 are used to find X_{n-55} and X_{n-24} in the table $t[\,]$.
- /* For each call to the sequence, computation of a new point */
The term X_{n+1} is computed as follows:
Indices i_1 and i_2 are incremented at each step (until 55, that is the dimension of the table $t[\,]$) then their value is 1. They allow to select the appropriate values instead shifting all the elements of the table.
/* Substractive method*/
We take values X_{n-24} and X_{n-55} in table $t[\,]$.
We compute $X_{n+1} = X_{n-24} - X_{n-55}$ and if the result is negative, we add m.
We put X_{n+1} in the table at the index i_1.
We return the normalized value $U_n = X_n/m$.

2.5 Combined MRG with Order 3

- **Description:** CMRG description

This generator is a combination of two MRG of order 3.

$$z_n = \left(\sum_{j=1}^{J} \delta_j x_{j,n}\right) \mod m_1, \quad u_n = \frac{z_n}{m_1}$$

with the following parameters:
$J = 2$, $k = 3$ and $\delta_1 = -\delta_2 = 1$;
$m_1 = 2^{32} - 209$, $a_{1,1} = 0$, $a_{1,2} = 1403580$, $a_{1,3} = -810728$;
$m_2 = 2^{32} - 22853$, $a_{2,1} = 527612$, $a_{2,2} = 0$, $a_{2,3} = -1370589$.

$$X_{1,n} = a_{1,2}X_{1,n-2} + a_{1,3}X_{1,n-3}$$

$$X_{2,n} = a_{2,1}X_{2,n-1} + a_{2,3}X_{2,n-3}$$
$$Z_n = (X_{1,n} - X_{2,n}) \mod m_1$$

The period ρ is equal to $(m_1^3 - 1)(m_2^3 - 1)/2 \simeq 2^{191}$.

- **Algorithm:**

- /* First call to the sequence */
/* Initialization */
Initialization of the values $X_{1,0}, X_{1,1}, X_{1,2}, X_{2,0}, X_{2,1}, X_{2,2}$.

- /* For each call to the sequence, computation of a new point */
/* First generator */
/* Second generator */
/* Combination of the two generators */
$Z_n = X_{1,n} - X_{2,n}$.
If the obtained value is negative, we add m_1.
We return the normalized value $U_n = Z_n \times \frac{1}{m_1}$.

2.6 Combined MRG with Order 5

- **Description:** CMRG description

This generator is a combination of two MRG of order 5.

$$z_n = (\sum_{j=1}^{J} \delta_j x_{j,n}) \mod m_1, \quad u_n = \frac{z_n}{m_1}$$

with the following parameters:
$J = 2$, $k = 5$ and $\delta_1 = -\delta_2 = 1$;
$m_1 = 2^{32} - 18269$, $a_{1,1} = 0$, $a_{1,2} = 1154721$, $a_{1,3} = 0$, $a_{1,4} = 1739991$, $a_{1,5} = -1108499$;
$m_2 = 2^{32} - 32969$, $a_{2,1} = 1776413$, $a_{2,2} = 0$, $a_{2,3} = 865203$, $a_{2,4} = 0$, $a_{2,5} = -1641052$.

$$X_{1,n} = a_{1,2}X_{1,n-2} + a_{1,4}X_{1,n-4} + a_{1,5}X_{1,n-5}$$

$$X_{2,n} = a_{2,1}X_{1,n-1} + a_{1,3}X_{1,n-3} + a_{2,5}X_{2,n-5}$$
$$Z_n = (X_{1,n} - X_{2,n}) \mod m_1$$

The period ρ is equal to $(m_1^5 - 1)(m_2^5 - 1)/2 \simeq 2^{319}$.
The steps of the algorithm are the same than for the one with $k = 3$.

• **Algorithm:**

- /* First call to the sequence */
/*Initialization*/
Initialization of the values $X_{1,j}$ and $X_{2,j}$.

- /* For each call to the sequence, computation of a new point */
/* First generator */
/* Second generator */
/* Combination of the two generators */
$Z_n = X_{1,n} - X_{2,n}$.
If the obtained value is negative, we add m_1.
We return the normalized value $U_n = Z_n \times \frac{1}{m_1}$.

2.7 Tausworthe generator

• **Description:** LFSR description
The Tausworthe generator corresponds to the LFSR Generator and it is included in the Digital Methods for Random Number Generator (see [3]).
The generator we now consider is a combination of 3 Tausworthe generators, with parameters:

$$J = 3;$$
$$k_1 = 31,\ q_1 = 13,\ s_1 = 12;$$
$$k_2 = 29,\ q_2 = 2,\ s_2 = 4;$$
$$k_3 = 28,\ q_3 = 3,\ s_3 = 17.$$

The period length ρ is $\rho = (2^{31} - 1)(2^{29} - 1)(2^{28} - 1) \simeq 2^{88}$.

The program allows to take into account a combination of J generators with $J \leq TAUS\_MAX$ (value fixed here to 10).
We particularly delail the algorithm implemented for a single Tausworthe Generator, because it is not completly obvious.

Let s_0 be the initial state. $s_0 = (x_0, \ldots, x_{k-1}) \in \{0,1\}^k$. We define

$$u_n = \sum_{i=1}^{L} x_{ns+i-1} 2^{-i} = \frac{1}{2^L} \sum_{i=1}^{L} x_{ns+i-1} 2^{L-i}$$

(The second expression is used in the implementation.)
Condition (C) : $P(z) = z^k - az^q - 1$ is a primitive trinomial with $0 < 2q < k$, $0 < s \leq k - q < k \leq L$ and $\gcd(s, 2^k - 1) = 1$.
Under the condition (C), we give an efficient algorithm to compute the Tausworthe generator.

Let $r = k - q$ and A, B and C three bit vectors of size L. Initially, A contains $\widetilde{s}_{n-1}$ and C contains k ones followed by $(L-k)$ zeros.

Notation
$\oplus$ denotes the bitwise exclusive OR, that is addition in base 2.
& denotes multiplication in base 2.
These both operators lead to a very fast implementation because they use only binary representation for integers.

• Tausworthe's algorithm

```
Step 1. $B \leftarrow q$-bit left-shift of $A$
Step 2. $B \leftarrow A \oplus B$
Step 3. $B \leftarrow (k-s)$-bit right-shift of $B$
Step 4. $A \leftarrow A \& C$
Step 5. $A \leftarrow s$-bit left-shift of $A$
Step 6. $A \leftarrow A \oplus B$
```

Values of A and B during the algorithm are detailed here for $n = 1$.
Step 0. $A : \widetilde{s}_0 = (x_0, \ldots, x_{L-1})$
Step 1. $B = (x_q, \ldots, x_{L-1}, 0, \ldots, 0)$
Step 2. We use that $x_n = x_{n-r} \oplus x_{n-k}$. $B = (x_k, \ldots, x_{r+L-1}, x_{L-q}, \ldots, x_{L-1})$
Step 3. $B = (0, \ldots, 0, x_k, \ldots, x_{s+L-1})$
Step 4. $A = (x_0, \ldots, x_{k-1}, 0, \ldots, 0)$
Step 5. $A = (x_s, \ldots, x_{k-1}, 0, \ldots, 0)$
Step 6. $A = (x_s, \ldots, x_{k-1}, x_k, \ldots, x_{s+L-1}) = \widetilde{s}_1$

• Initialization of the algorithm
A must be initialized with a value $\widetilde{s}_0$, which agrees the recurrence. We choose the first k bits arbitrarly, such that $s_0 = (x_o, \ldots, x_{k-1}) \neq 0$; the other $(L-k)$ bits are computed with s_o and the recurrence relation. Assumed that $k + r \geq L$, we can use the following algorithm.

Step 1. $B \leftarrow q$-bit left-shift of A
Step 2. $B \leftarrow A \oplus B$
Step 3. $B \leftarrow k$-bit right-shift of B
Step 4. $A \leftarrow A \oplus B$

Step 0. $A = (x_0, \dots, x_{k-1}, 0, \dots, 0)$
Step 1. $B = (x_q, \dots, x_{k-1}, 0, \dots, 0)$
Step 2. We use that $x_n = x_{n-r} \oplus x_{n-k}$. $B = (x_k, \dots, x_{r+k-1}, x_{k-q}, \dots, x_{k-L}, 0, \dots, 0)$
Step 3. $B = (0, \dots, 0, x_k, \dots, x_{sL-1})$
Step 4. $A = (x_0, \dots, x_{k-1}, x_k, \dots, x_{L-1})$

- **Algorithm:**

- /* First call to the sequence. Initialisation */
/* Choice of the parameters to have ME-CF generators (cf L'Ecuyer) */
Choice of $k[j]$, $q[j]$, and $s[j]$. Computation of $r[j]$ and $t[j]$.

- /* Constant c : k bits to one and $(L-k)$ bits to zero */
$c[j] = 2^{32} - 2^{(L-k[j])}$

- /* Initialisation of each generator */
/* The first k bits are chosen randomly */
Call to functions random-word and random-bit. Algorithm for this last function is based on a prime polynomial : here we choose $x^{18} + x^5 + x^2 + x + 1$. It is described in 'Numerical Recipes in C' page 296.
/* The next $L - k$ bits are initially fixed to zero */
/* Now they are computed with the recurrence on u */
See the previous description about the initialization of the algorithm.

- /* For each call to the sequence, computation of a new point */
/* Calculus of the next point for the J generators */
/* 6 steps explained by L'Ecuyer */
See the previous description about the Tausworthe algorithm.
/* Combination : XOR between the J generators */

/* Normalisation by $\frac{1}{2^{32}}$ */

3 Low Discrepancy Sequences

3.1 SQRT sequence

• **Description:** SQRT description
The *SQRT sequence* is a particular case of the Tore sequence with $\alpha = (\sqrt{p_1}, \ldots, \sqrt{p_d})$ and where $(p_1, \ldots, p_d)$ are the first d prime numbers. The d-dimensional sequence is given by:

$$\xi_n = (\{n.\alpha\} = (\{n.\alpha_1\}, \ldots, \{n.\alpha_d\})$$

where $\{x\} = x - [x]$ is the fractional part of x.

• **Algorithm:**

- /* Verification of the dimension. It must not change without reinitializing */
If the dimension d changed, a reinitialization must be done.

- /* First call : initialization */
Computation of the first d prime numbers.

- /* For each call to the sequence, computation of a new point */
$x_m[i] = m\sqrt{p_i} - [m\sqrt{p_i}]$.

3.2 Halton sequence

• **Description:** Halton description
The Halton sequence is a d-dimensional generalization of the Van Der Corput sequence. It is defined by:

$$\xi_n = (\varphi_{p_1}(n), \ldots, \varphi_{p_d}(n))$$

where $(p_1, \ldots, p_d)$ are the d first prime numbers and where $\varphi_p(n) = \sum_{i=0}^{R(n)} \frac{a_i}{p^{i+1}}$ with a_i given by the digit expansion in base p of n.

• **Algorithm:**

- /* Verification of the dimension. It must not change without reinitializing */
If the dimension d changed, a reinitialization must be done.

- /* First call : initialization */
Computation of the first d prime numbers.

- /* For each call to the sequence, computation of a new point */
/* Radical inverse function */
Van der Corput sequence for each p_i.

3.3 Faure sequence

• **Description:** Faure description
The d-dimensional Faure sequence is given by:

$$\xi_n = \Big(\varphi_r(n), T(\varphi_r(n)), \ldots, T^{d-1}(\varphi_r(n))\Big)$$

where φ_r is the Van der Corput sequence in base r.
The transformation T depends on the binomial coefficients and the r-digit expansion of n.

Regarding implementation, it should be noticed that indices i, j of the binomial coefficients $Comb[i][j]$ we need, depend on the r digit expansion of n, the highest coefficient goes to infinity with n. Since we store the binomial coefficients as LONG integers, we can not go further than $Comb[32][16]$. The corresponding value of n is stored in the (macro) MAX_SAMPLE_FAURE. Binomial coefficients are computed at the launching of Premia (routine InitMC()). They are stored this way:

```
#define  MAXI 33
long Comb[MAXI][MAXI];
```

• **Algorithm:**

- /* Verification of the dimension. It must not change without reinitializing */
If the dimension d changed, a reinitialization must be done.

- /*First call to the sequence */
Computation of the first d prime numbers and the binomial coefficients;
Research of the smallest odd prime r such that $r \geq d$.

- /* Initialization */

- /* For each call to the sequence, computation of a new point */
/* r-digit expansion of n -> first term of the sequence */

/* Other terms of the sequence */
/* Successive transformations of the r-digit expansion.*/
T uses the binomial coefficients $Comb[i][j]$.

3.4 Sobol Sequence

• **Description:** Sobol description
The d-dimensional Sobol sequence is defined by

$$\xi_n = \left(a_1 V_1^{(1)} \oplus \cdots \oplus a_{r(n)} V_{r(n)}^{(1)}; \ldots ; a_1 V_1^{(d)} \oplus \cdots \oplus a_{r(n)} V_{r(n)}^{(d)}\right)$$

where the $V_i^{(j)}$ are direction numbers obtained from d primitive polynomials and the a_i are the coefficients from the binary decomposition of n.

To reduce the computing time, we will use the *Gray Code* of n instead its digit expansion in base 2. Antonov and Saleev proved in [1] that it does not affect the asympotic discrepancy of the sequence.

Result: The set of all binary segments of length 2^s is transformed one-to-one into itself if we use the Gray Code transformation. Then the uniformity property is preserved.

Gray Code : (see Numerical Recipes [4], chapter 20)
A Gray code is a function G, such that for each integer $i \geq 0$, the binary representation of $G(i)$ differs from the representation of $G(i+1)$ in exactly one bit (the position of the rightmost zero bit of i in its digit decomposition in base 2)
To obtain the Gray Code of n, we apply a XOR between n and $n/2$, that is:

$$G(n) = n \oplus (n >> 1)$$

where $>>$ is the right shift operator.

• **Sobol implementation**:
Using the properties of the Gray Code, we can compute the term ξ_{n+1} from the term ξ_n and only one direction number V_k (see Antonov and Saleev[1] or Bratley and Fox [2]). We have

$$\xi_{n+1} = \xi_n \oplus V_k$$

where k is the index of the rightmost zero bit in the binary representation of n.

Finally we use the representation of ξ in the following form :

$$\xi_{n+1}(j) = \gamma_{n+1}(j) \times \frac{1}{2^{BIT\_MAX}}$$

$$\gamma_{n+1}(j) = \gamma_n(j) \oplus (C[k][j] \ll (BIT\_MAX - k))$$

The initial value is fixed to $\gamma_0 = 0$. This implementation allows to keep integer values.

BIT_MAX denotes the maximum number of bits allowed to a digit expansion in base 2. $N = 2^{BIT\_MAX}$ is the greater integer for which we can compute ξ_N.
DIM_MAX denotes the maximal dimension allowed for this sequence. At present, it is equal to 39.
We can increase this value, but we have to complete the initialization of the $C_{i,j}$. It is not an easy step to select good constants $C_{i,j}$ (as explained in the presentation of the Sobol sequence ????).

- **Algorithm:**

- /* Degree of the DIM_MAX primitive polynomials */
Degree of the primitive polynomials are stored in a table of size DIM_MAX, the maximal dimension allowed for this sequence. Remark about this parameter is given in the description of the sequence. We recall that we cannot simulate Sobol Sequence for dimensions greater than DIM_MAX because of the unknown initialization of constants Cij.

- /* Index of each primitive polynomial. It determines coefficients b_i */
Indices of primitive polynomials are stored in a table of size DIM_MAX. Each index associated with the degree of the polynomial characterizes its coefficients. Coefficients b_i are obtained from the decomposition in base 2 of its index.
The index of the following polynomial

$$P(j) = x^{s(j)} + b_1 x^{s(j)-1} + \cdots + b_{s(j)-1} x + 1$$

with degree $s(j)$, is given by

$$\sum_{i=1}^{s(j)-1} b_i 2^{s(j)-i-1}$$

Terms of order $s(j)$ and 0 always have a coefficient equals to 1 (else the considered polynomial can not be primitive). They are not included in the coefficients b_i.

- /* Sobol's constants C_i_(j) */

Initialization of the first values, that are the ones for $i < s(j)$. The next ones are computed during the initialization algorithm at the first call of the sequence.
Initialization of the constants C_i_(j) is stored in a two-dimensional table of size DIM_MAX*8 (actually, dimensions are DIM_MAX+1 and 9, but we do not use the first element of each vector, that is the one with index 0). The value 8 corresponds to the highest degree for the considered polynomials.
Values Ci(j) in this table are choosen to satisfy the condition

$$c_i^{(j)} < 2^i, \quad 1 \le i \le s(j)$$

They were found on the web page *www.math.hkbu.edu.hk/qmc/software.html.* They are supposed to be good and to satisfy conditions for uniformity property A defined by Sobol but we don't have tested this criterion.
For $i > s(j)$ and $i \le 8$, we first put 0; the exact values will be computed in the next step.

- /* Test of the dimension */

Test that the required dimension is not greater than DIM_MAX, the maximum dimension allowed for the algorithm.

- /*First call to the sequence */

/* Initialization of the full array $C[i][j]$ */
We now fill the full table of size DIM_MAX * BIT_MAX we will use to generate terms of the Sobol sequence.
/* Recurrence relation to compute the other $C[i][j]$ */
For $i > s(j)$, the computation is based on the recurrence relation given in the description of Sobol sequence. Firstly, terms without coefficients b_k are added with the operator $\oplus$.
/* Test for the coefficient $b(k)$ */
Then, coefficients b_k are computed from the decomposition in base 2 of theindex of the appropriate primitive polynomial. If $b_k = 1$ a factor is added in the recurrence.
Computation of the first vector.

- /* Calculation of a new quasi-random vector on each call */

/* Research of the rightmost 0 bit */

We find an index i.
/* Computation of the term n from the term $(n-1)$ */
With the index i and the recurrence formula $\gamma_{n+1}(j) = \gamma_n(j) \oplus (C[i][j] << (BIT\_MAX - i))$
/* Normalization */
We obtain a value in $[0,1]$ by normalizing with 2^{-BIT_MAX}.

3.5 Niederreiter sequence

- **Description:**

When $b = 2$, we can use the Gray Code of n instead its binary decomposition. Then we obtain a recurrence relation between ξ_{n+1} and ξ_n and the implementation goes faster.

$$\xi_{n+1}(j) = \gamma_n(j) \times \frac{1}{2^{BIT\_MAX+1}}$$

$$\gamma_{n+1} = \gamma_n \oplus (C[k][j])$$

where k is the index of the rightmost zero bit in the binary representation of n.

At the moment, Niederreiter sequence is only available for base $b = 2$.

- **Algorithm:**

- /* Niederreiter's constants */

- /* Test of the dimension */

Test that the required dimension is not greater than DIM_MAX, the maximum dimension allowed for the algorithm, particularly for the $C[i][j]$ constants.
DIM_MAX is equal to 12. We can increase this value, but we have to complete the initialization of the $C_{i,j}$.

- /* First call to the sequence */

/* Initialization of initX_n[ ] */
/* Gray code of saut */
/* XOR sum */

- /* Calculation of a new quasi-random vector on each call */

/* Research of the rightmost 0 bit */
We find an index i.

/* Computation of the term n from the term $n-1$ */
Normalization to obtain ξ_{n+1}.
Calculation of γ_{n+1} with the index i and the recurrence formula $\gamma_{n+1} = \gamma_n \oplus (C[i][j])$.

References

[1] I.A. ANTONOV and V.M. SALEEV. An economic method of computing lp_τ-sequences. *USSR Comput. Maths. Math. Phys*, 19:252–256, 1980. 12

[2] P. BRATLEY and B.L. FOX. Algorithm 659. implementing sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software*, 14(1):88–100, 1988. 12

[3] P. L'ECUYER. Maximally equidistributed combined tausworthe generators. 7

[4] W.T. VETTERLING W.H. PRESS, S.S. TEUTOLSKY and B.P. FLANNERY. *Numerical Recipes in C. The art of scientific computing.* Cambridge University Press, 1992. 1, 12