This version: February 4, 2024

Proposal for a standard file format for lattice rules and digital nets

by Pierre L'Ecuyer [1]

1 General ideas and goals

We propose *standard formats* to specify lattice rules, polynomial lattice rules, and digital nets, in simple text files. We want the formats to be simple and relatively compact, with no more than one line per dimension, so they can easily be used for point sets in several thousand dimensions if desired. Ordinary text files with decimal numbers are good enough. They are easy to read by both humans and computers in any language. Other specialized formats (Json or Parquet, for example) can be more compact but then the files are not as easy to read without extra tools.

Joe and Kuo (2008) provide a file for up to 21,201 dimensions for Sobol' points, and this seems to be the most widely used set of parameters for RQMC points at this time. They use a fixed and very simple format, which requires no special software to read. We want to provide similar types of files for other types of point sets, for an arbitrarily large number of dimensions. The SSJ simulation library (L'Ecuyer, 2016) can already read some of our proposed formats. Latnet Builder (L'Ecuyer et al., 2022) can (or will) produce output files in these formats. The choice of output format for Latnet Builder can be specified on the command line, using the following keywords:

`lattice`	A lattice rule: give the modulus and the generating vector.
`dnet`	A digital net: give the generating matrices, one per line.
`plattice`	A polynomial lattice rule: give the polynomial modulus and the generating vector.
`sobol`	Sobol' points (default format), give only the direction numbers.
`soboljk`	Sobol' points, the format used by Joe and Kuo (2008).

The most important formats are the first two, since the point sets covered by the other formats are special cases of digital nets, so they can all be described by the `dnet` format. We propose them because they provide alternative representations that are either more compact or commonly used.

All the point sets that we consider have the form

$$P_n = \{\boldsymbol{u}_i \in [0,1)^s,\ i = 0, \ldots, n-1\}$$

[1] This document was improved by several comments and suggestions from Fred Hickernell, Alexander Keller, Dirk Nuyens, and Aleksei Sorokin.

where n is the number of points and s is the number of dimensions. The dimension j goes from 1 to s and there are n points enumerated by i going from 0 to $n-1$. (In computer code, j usually starts at 0, whereas in math papers, it starts at 1; one must be careful about this discrepancy.) The .txt files that contain the parameters have one line per dimension, preceded by a few lines that contain general parameters, such as s, n, etc. We shall call these lines the *header* of the file. In the header, additional lines that start with "`# `" can be used for comments and descriptions; these lines are totally optional and should be just skipped by the program that reads the file. Anything that starts with "`# `" on any given line in the header should also be skipped. All these comments are only for human readers to better see what is in the file, they are not for the computer. One exception: the first line of the file must be a comment that contains the keyword for the file type; for example "`# dnet`" for a digital net. This line *may* be read by software, but may also be skipped; reading it is not compulsory. The number of dimensions (number of lines after the header) can be much larger than what we usually need; it suffices to use the number of rows that are needed.

The point sets can be extensible in the number of points n or not (they can be constructed for a single n only). Sobol points are extensible ad infinitum, although they are very good only when n is a power of 2. Other types of point sets can also also be extensible, but are usually constructed to be good only for $n = b^k$ for k in a given integer range, e.g., from 10 to 20, where $b \geq 2$ is the base. They satisfy the property that P_{b^k} contains $P_{b^{k-1}}$ for all k in this range. We call them *embedded* point sets. For these types of point sets it is highly recommended to specify the range in a comment in the header of the file. [2]

In the proposed formats, the files do not assume a given computer word size (e.g., 32 bits or 64 bits). The format is exactly the same regardless of the word size. Of course, if the file contains integers of more than 32 bits, the corresponding points cannot be generated properly on a 32-bit computer. A comment in the file header can say it.

Some users might prefer input files with no header at all, only the s lines that give the generating vector or generating matrices. In some languages (e.g., MATLAB), such a file can be read into a matrix by a simple "load file" command, so there is no need to to write any code to read the file. Users who want that can simply strip out the header from the files in standard format and use these naked files privately. We think that the header with human-readable comments as imposed by the standard will be very useful to many users.

The following sections describe the proposed text-file formats for the different point sets.

2 Parameters for ordinary lattice rules: `lattice`

For an ordinary *lattice rule of rank 1*, we have

$$P_n = \{\boldsymbol{u}_i = (i\boldsymbol{a} \bmod n)/n,\ i = 0, \ldots, n-1\}$$

where $\boldsymbol{a} = (a_1, \ldots, a_s)$ is the generating vector. We must specify s, n, and $\boldsymbol{a}$.

[2] From Pierre L: The range for which the points were built could be given in the file, but this would make things more complicated for some point sets. For example, for ordinary lattice rules with a prime number of points, this additional info might be confusing for users. For Sobol points, the range has no limit.

In a "`lattice`" file, the first line must start with "`# lattice`". After that, not counting the comment lines, the first line gives the number s of dimensions, the second line gives the number n of points, and lines 3 to $s+2$ give the coefficients $a_1, \ldots, a_s$, one value per line. In the case of embedded lattice rules, n would usually be a power of 2, say $n = 2^k$, and the smaller embedded lattices will contain the first 2^{k-1} points, the first 2^{k-2} points, etc. Additional comments in the file should tell when the lattice is embedded, which figure of merit and what weights were used, the construction method, etc. [3]

One example of a parameter file for an ordinary lattice rule, in 'lattice' format is given below. In this file, the first line is skipped, only the number "8" is read on the second line, only the number "65536" is read on the second line, etc.

```
# lattice
# A lattice rule, non-embedded, in 'lattice' format
8         # 8 dimensions
65536     # modulus = n = 65536 points
# coordinates of the generating vector, starting at j=1:
1
19463
17213
5895
14865
31925
30921
26671
```

3 Parameters for digital nets: `dnet`

A *digital net in base b* with $n = b^k$ points is defined by selecting integers $s \geq 1$, $r \geq k \geq 1$, and s matrices $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_s$ of size $r \times k$ with entries in $\mathbb{Z}_b$, called the generating matrices. For $i = 0, \ldots, n-1$, let $i = \sum_{\ell=0}^{k-1} a_{i,\ell} b^\ell$ be the expansion of i in base b, and for $j = 1, \ldots s$, let

$$(y_{i,j,1}, \ldots, y_{i,j,r})^T = \boldsymbol{C}_j \cdot (a_{i,0}, \ldots, a_{i,k-1})^T \quad \text{and} \quad u_{i,j} = \sum_{\ell=1}^{r} y_{i,j,\ell} b^{-\ell}.$$

The points $\boldsymbol{u}_i$ are defined by $\boldsymbol{u}_i = (u_{i,1}, \ldots, u_{i,s})$. Digital nets are usually in base $b = 2$, but we allow a general (typically prime) base $b \geq 2$.

The proposed format to specify digital nets is as follows. The first line must start with "`# dnet`". Then the first four non-comment lines give b (the base), s (the number of dimensions), k (the number of columns), and r (the number of rows in the generating matrices in base b). Thus, the output values will have "precision" b^{-r} (they will be integer multiples of b^{-r}). For $b = 2$, a common value in the past has been $r = 31$ when using 32-bit integers, but going forward we should use 64-bit integers and $r = 63$ or 64, or perhaps $r = 53$ to exploit the full

[3] From Pierre L: My suggestion is not to force the computer to read this information, but just put it as comments for humans. Otherwise, it will force additional lines in the file and make the reading more complicating. What would we put for example if n is prime and the rule is not embedded? Should we have a `lattice2` format for embedded rules in base 2? Putting more options makes things more complicated.

accuracy of a "`double`". By looking at r, one can see right away whether this file is good for 64-bit computers only or for 32-bit computers as well.

The s lines after this header will contain the s generating matrices, one per line. Each of these lines contains k integers smaller than b^r giving the k columns of $\boldsymbol{C}_j$, using by default the same encoding as in the class `DigitalNetBase2` in SSJ for $b = 2$. That is, the base-b representation of the integer gives the r digits in the corresponding column, with the digit on the first row of the matrix (row 0) being the most significant, and the one on the last row (row $r-1$) being the least significant. For example, if $b = 2$, $r = 31$, and the first column has a 1 in the first row and 0 in all other rows, as is always the case for Sobol points, then the integer representation of this column will be $2^{30} = 1\,073\,741\,824$. If there is a 1 in the last row and 0 elsewhere, the representation will be $2^0 = 1$. If all 31 elements of the column are 1, the representation will be $2^{31} - 1$.

One example of a file for a digital net in "`dnet`" format:

```
# dnet
# A digital net in base 2, in 'dnet' format
2    # basis b = 2
8    # s = 8 dimensions
10   # k = 10, so n = 2^10 = 1024 points
31   # r = 31 digits
# The columns of gen. matrices C_1, ..., C_s, one matrix per line:
1073741824   536870912  268435456  134217728  ...
2012537125  1382645254  ...
   ...
```

This differs from Joe and Kuo (2008), where the c-th column (for $c \geq 1$) is represented by an integer smaller than 2^c (in base 2) and the least significant bit is the one on the diagonal. Their representation works when $\boldsymbol{C}_j$ is upper triangular, which is true for Sobol point sets, but not for digital nets in general.

Recall that coordinate j of the i-th point is obtained by multiplying the base-b matrix $\boldsymbol{C}_j$ by the vector of digits of the representation of i in base b, with the least significant digits of i at the top. That is, the least significant digit of i goes with the first column of $\boldsymbol{C}_j$. And the first row of $\boldsymbol{C}_j$ is for the most significant digit of output. With our representation of $\boldsymbol{C}_j$ by k integers, the points are easy and fast to generate in base 2. We obtain `u[i,j]`, coordinate j of point i, with the following code snippet, in which `C[j,c]` is the integer that represents column c of $\boldsymbol{C}_j$:

```
normFactor = 1.0 / (1 << r)          // 2^{-r}
coord = 0
for c = 0 to k - 1
   coord ^= ((i >> c) & 1) * C[j,c]
u[i,j] = coord * normFactor
```

4 Parameters for polynomial lattice rules: `plattice`

Polynomial lattice rules are a special type of digital nets with generating matrices of a special form. For a polynomial lattice rule of rank 1 in a prime base b, we have

$$P_n = \left\{ \left(\varphi\left(\frac{h(z)a_1(z)}{Q(z)}\right), \ldots, \varphi\left(\frac{h(z)a_s(z)}{Q(z)}\right)\right) : h(z) \in \mathbb{F}_b[z],\ \text{degree}(h(z)) < k \right\}. \qquad (1)$$

where $\mathbb{F}_b[z]$ is the space of polynomials with coefficients in $\mathbb{F}_b$, the *modulus* $Q(z) \in \mathbb{F}_b[z]$ is a polynomial of degree k, the *generating vector* $\boldsymbol{a}(z) = (a_1(z), \ldots, a_s(z)) \in \mathbb{F}_b[z]^s$ is a vector of s polynomials of degrees less than k, and the mapping φ is defined by

$$\varphi\left(\sum_{l=w}^{\infty} x_l z^{-l}\right) = \sum_{l=\max(w,1)}^{\infty} x_l b^{-l}. \qquad (2)$$

This point set has $n = b^k$ points.

We must specify the polynomial modulus $Q(z)$ and the polynomial generating vector $\boldsymbol{a}(z)$. The polynomial modulus will be represented as an integer that has $(k+1)$ digits in base b, and all the other polynomials will be represented as integers that have no more than k digits in base b. All these integers will be given in base 10 in the file, one per line. In practice, we usually have $b = 2$, so k represents the number of bits. The integer that represents a polynomial is obtained simply by replacing the formal variable by b. For example, if the polynomial is $Q(z) = z^4 + z^3 + 1$ and $b = 2$, its coefficients are "1 1 0 0 1" and its integer representation is $2^4 + 2^3 + 1 = 25$. This is the usual representation, as used in Goda and Dick (2015), for example. In the case of embedded point sets, the modulus should be $Q(z) = z^k$ for $n = b^k$ points, and its integer representation is b^k. In particular, $Q(z) = z$ is represented by the integer b.

As usual, the first line is a comment that tells the type of file. Then the first four non-comment lines give the base b, the number s of dimensions, the degree k of the polynomial modulus, and the integer representation of this polynomial. Lines 5 to $s + 4$ give the polynomials that form the generating vector, one per line, using the integer representation just explained. One example of a file for a polynomial lattice in the "`plattice`" format:

```
# plattice
# A polynomial lattice rule in base 2, in 'plattice' format
2       # base b = 2
8       # s = 8 dimensions
16      # n = 2^16 = 65536 points
45781   # polynomial modulus
# coordinates of the generating vector, starting at j=1:
1
17213
5895
14865
31925
30921
26671
17213
```

A polynomial lattice rule in base b can also be represented as a digital net in base b, so its parameters can also be provided in a file in the "`dnet`" format, as for general digital net in base b. But the generating matrices have a special form and the above representation is much more compact (a single integer per row instead of k integers per row). On the other hand, generating the points is faster with the generating matrices than with the polynomial representation, so the software that will use the "`plattice`" files and generate the points would usually first convert the polynomials into the corresponding generating matrices. LatNet Builder (L'Ecuyer et al., 2022) is also able to make the conversion and produce a file in the "`dnet`" format, for more convenience and better flexibility, so the user can select the format she/he prefers.

5 Parameters for Sobol nets: `sobol` and `soboljk`

The Sobol' construction provides another special case of digital nets (and sequences), in base 2. They are defined in many places, including Joe and Kuo (2008). For each coordinate j, we select a primitive polynomial $p_j(z)$ of degree c_j, and c_j integers $m_{j,1}, \ldots, m_{j,c_j}$ which are used to define the generating matrix $\boldsymbol{C}_j$. The real numbers $2^{-c} m_{j,c}$ are called the initial *direction numbers*. More details are given in Joe and Kuo (2008) and at `http://umontreal-simul.github.io/ssj/docs/master/classumontreal_1_1ssj_1_1hups_1_1SobolSequence.html`.

One obvious option for these point sets is to adopt exacly the same format as Joe and Kuo (2008), because it is already used in many places. The only difference is that we now allow comment lines in the file. In the format of Joe and Kuo (2008), only the first line is skipped. In the proposed format, other comment lines can be added at the beginning of the file, e.g., to give the maximum number of dimensions in the file, the criterion and weights that were used, etc. Note that Sobol' sequences have an infinite number of points and an unlimited number of dimensions, although the file will give parameters for a finite number of dimensions.

The other lines of the file specify the primitive polynomials and the initial direction numbers for each dimension $j \geq 2$, one line per dimension. For dimension $j = 1$, the generating matrix is the identity and is not given in the file (it is implicit). The columns of this matrix are not obtained via a recurrence based on a primitive polynomial, so this matrix is handled separately.

The first number on each line is the dimension j. The second number is the degree c_j of the primitive polynomial $p_j(x)$ used for this dimension. The third number is the integer that corresponds to the binary representation of the inner coefficients of this polynomial (we ignore the first and last coefficients, they are always 1). For example, if the polynomial is $p_j(x) = x^4 + x^3 + 1$, the coefficients are "1 1 0 0 1", and after removing the first and last "1", we get 100 in base 2, which is 4, so the third column would contain the number 4. (Without removing the first and last "1", the number would be 25 instead.) After these three numbers, there are c_j integers $m_{j,1}, \ldots, m_{j,c_j}$ where $m_{j,c}$ is the cth (real-valued) initial direction number for this coordinate, multiplied by 2^c to obtain an integer. This $m_{j,c}$ is the integer formed by taking the bits in row 1 to row c of column c, in this order. The last bit is the bit on the diagonal, which is always 1, so all $m_{j,c}$'s are odd integers. I think this format comes from Bratley and Fox (1988).

We denote this format for Sobol parameters by the "`soboljk`" keyword. One example of a file in this format is shown below. The first line gives the type of file and the next three lines are comments that must be skipped by the reading program.

```
# soboljk
# Parameters for Sobol points, in 'soboljk' format
# 8 dimensions
# c_j p_j m_{j,c}
2  1   0  1
3  2   1  1 3
4  3   1  1 3 1
5  3   2  1 1 1
6  4   1  1 1 3 3
7  4   4  1 3 5 13
8  5   2  1 1 5 5 17
```

The `soboljk` format can be simplified as follows. First, removing the first and last "1" in the representation of the primitive polynomials saves a bit of memory, but it also makes thinks slightly more complicated. In the default representations of the primitive polynomials in the code that generates the points, these bits are usually not removed. In SSJ, the first thing we do when reading a file in `soboljk` format is to add them back. Also, the primitive polynomials can be in a separate file, since they never change, and only the (initial) direction numbers (those depend on the selected FOM and weights) would be given to specify the Sobol' points. That is, we remove the first three columns of the `soboljk` format. The Magic Point Shop (Nuyens, 2020) also produces files that contain only the direction numbers.

One example of a file in this "`sobol`" format:

```
# sobol
# Parameters m_{j,c} for Sobol points, in 'sobol' format
# 8 dimensions
1   # This is  m_{j,c} for the second coordinate
1 3
1 3 1
1 1 1
1 1 3 3
1 3 5 13
1 1 5 5 17
```

A list of the first few primitive polynomials in base 2 is given here: `https://mathworld.wolfram.com/PrimitivePolynomial.html`. If we *do not* remove the first and last 1's in their representations, the first primitive polynomials are: `3, 7, 11, 13, 19, 25, ...`. Their degrees are `1, 2, 3, 3, 4, 4, ...`. This representation is the one used in the code of SSJ, for example. We can have a separate file that gives these polynomials, one per line, exactly as in the first three columns of the "`soboljk`" format. We may also want to remove the first column.

Another, perhaps more convenient, way of storing Sobol' constructions is to just use the general "`dnet`" format, in which the generating matrices are given explicitly. This `dnet` format is easier to use. On the other hand, it requires specifying a (maximum) value of k, and k integers per row to specify the generating matrices, which leads to larger files. From a file in `sobol` format, one can construct a digital net with an arbitrarily large k.

When $n = 2^k$ is fixed, so we use exactly $n = 2^k$ points and there is there no embedding, we can add one *extra dimension* at the beginning by using the reflected identity as a generating matrix. The successive values for this coordinate will then be $0, 1/n, 2/n, 3/n, \ldots$ in this order. This matrix will not be given in the file for Sobol' points; the QMC/RQMC software must handle it. For lattice rules and general digital nets with fixed n (non-embedded), the file could give a first coordinate with this behavior.

6 Files that contain randomizations

The idea of proposing a format for storing specific randomizations was suggested by Fred Hickernell. This can be useful for verification purposes, for example.

We can store randomizations in the following file formats:

`shiftmod1`	A (random) shift modulo 1. It corresponds to a single point in $[0,1)^s$.
`dshift`	A digital shift in base b.
	Also a single point in $[0,1)^s$, but with r digits in base b.
`nuscramble`	A nested uniform scramble in base b.
`lmscramble`	A (linear) left matrix scramble in base b.

For a `shiftmod1` in s dimensions, the file will contain s in the first line, followed by s real numbers between 0 and 1, one per line.

```
# shiftmod1
# A shift modulo 1, in 'shiftmod1' format
3   # s = 3 dimensions
0.32638741823951621
0.91325392536931693
0.1530364040t106301
```

For a `dshift` with r digits of accuracy in base b, in s dimensions, the file will contain b in the first line, s in the second line, r in the third line, and then s integers from 0 to $b^r - 1$, one per line. For the latter, the digits of the base-b representation of the integer divided by b^r will be added modulo b to the corresponding digits of the base-b representation of the coordinate. For example, if $b = 2$ and $r = 31$, the randomization makes a xor of the 31 bits of this integer with the 31 most significant bits of the corresponding coordinate of each point.

```
# dshift
# A digital shift in base 2, in 'dshift' format
2       # b = 2
3       # s = 3
31      # r = 31
2146832861
1084390381
963462828
```

For a `lmscramble` with r digits of accuracy, for b^k points in base b in s dimensions, we need to store s lower-triangular invertible $r \times r$ matrices with entries in $\{0, \ldots, b-1\}$. For $b = 2$, each matrix must have only 1's on the diagonal and 0's above the diagonal. Each such matrix

can be stored in one line of the file, in exactly the same format as the generating matrices in the `dnet` format, using one integer for each column. We want them in this format for the fast LMS implementation we have in SSJ, for example. The file will contain b in the first non-comment line, s in the second line, r in the third line, and then s square lower-triangular and invertible $r \times r$ matrices, one per line, with each column represented as an integer as in the `dnet` format. Thus, each scrambling matrix is represented by r integers on the same line. Here is an example:

```
# lmscramble
# A left matrix scramble in base 2, with 31 bits of resolution.
2    # basis b = 2
8    # s = 8 dimensions
31   # r = 31 digits
# Columns of the s lower-triangular r x r scrambling matrices,
# one matrix per line:
1673741824   906870912  615843556  213427728  ...
2012537125  1012645254  ...
   ...
```

For a `nuscramble` of the first $r \geq k$ digits, for $n = b^k$ points in base b in s dimensions, with the implementation proposed in Section 3 of Friedel and Keller (2002) and used for $b = 2$ in class `DigitalNetBase2` of SSJ, we need sn blocks of r random digits in base b. Each such block can be represented as an integer in the range $\{0, 1, \ldots, b^r - 1\}$. For $b = 2$, these are r-bit integers. We can store these integers one row per dimension, n integers per row. This gives the following `nuscramble` file format. The first non-comment line contains the base b, the second line gives the number s of dimensions, the third line gives the scramble resolution (the number of digits that are scrambled), and the following s lines give the sn integers used for the scrambling, n integers per line. Note that this is the same amount of random numbers that we would need if we use plain Monte Carlo instead of RQMC. [4] [5]

```
# nuscramble
# A nested uniform scramble in base 2, with 30 bits of resolution.
2    # basis b = 2
8    # s = 8 dimensions
10   # k = 10, so n = 2^10 = 1024 points
30   # r = 30 digits
# The following s rows contain n = 1024 30-bit integers per row:
1173741824   906870912  615843556  213427728  ...
1012537125  1001975254  ...
   ...
```

[4] From Pierre L: Another way of storing the NUS is as follows. For each coordinate j, each point can be identified by a k-bit integer, and the NUS maps each such k to a r-bit integer that corresponds to the scrambled coordinate j of this point. So we can simply store this map in an array of size b^k whose entry i contains the corresponding r-bit integer. Applying this NUS is then fast and straightforward.

[5] From Pierre L: Alternative implementations of NUS that use a hashing function in place of a RNG are proposed in Burley (2020) and Laine and Karras (2011). These methods might be faster and the is much less information to store to reproduce a given scramble, but the hashing function must be fixed, known, and reliable. This essentially amount to fixing the RNG and storing only its seed.

7 File names and other recommendations

It is strongly recommend that all file names start with the corresponding keyword, like `plattice` for a polynomial lattice rule, `sobol` for a Sobol point set, and `lmscramble` for a left matrix scramble, for example.

It is also recommended to put enough relevant comments in each file for a knowledgeable human to find what the file is for (type of point set, figure of merit and weights that were used to construct it, range of values of n for embedded point sets, etc.).

[We also want some unit tests: some specific parameter files together with the correct output that should be observed when generating the points from these files.]

References

P. Bratley and B. L. Fox. Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software*, 14(1):88–100, 1988.

Brent Burley. Practical hash-based Owen scrambling. *The Journal of Computer Graphics Techniques*, 9(4):1–20, 2020.

I. Friedel and A. Keller. Fast generation of randomized low-discrepancy point sets. In K.-T. Fang, F. J. Hickernell, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 257–273, Berlin, 2002. Springer-Verlag.

T. Goda and J. Dick. Construction of interlaced scrambled polynomial lattice rules of arbitrary high order. *Foundation of Computational Mathematics*, 15:1245–1278, 2015.

S. Joe and F. Y. Kuo. Constructing Sobol sequences with better two-dimensional projections. *SIAM Journal on Scientific Computing*, 30(5):2635–2654, 2008.

S. Laine and T. Karras. Stratified sampling for stochastic transparency. *Computer Graphics Forum*, 30(4):1197–1204, 2011.

P. L'Ecuyer. SSJ: Stochastic simulation in Java. `http://simul.iro.umontreal.ca/ssj/`, accessed 9th August 2021, 2016.

P. L'Ecuyer, P. Marion, M. Godin, and F. Puchhammer. A tool for custom construction of QMC and RQMC point sets. In A. Keller, editor, *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC 2020*, pages 51–70, Berlin, 2022. Springer. `https://arxiv.org/abs/2012.10263`.

D. Nuyens. The magic point shop, 2020. `https://people.cs.kuleuven.be/~dirk.nuyens/qmc-generators/`.