МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2021
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2021
Proceedings of the Fiftieth Spring Conference
of the Union of Bulgarian Mathematicians
2021

DETERMINISTIC ALGORITHM FOR OPTIMISING THE DIRECTION NUMBERS OF THE SOBOL SEQUENCE[*]

Emanouil Atanassov

The quasi-Monte Carlo (QMC) methods use deterministic, usually low-discrepancy sequences, to improve the order of convergence when compared with Monte Carlo methods dealing with the same problem. Although this improved convergence depends on the smoothness properties of the functions under consideration and sometimes cannot be established theoretically, good practical results were obtained in many areas, notably in Financial Mathematics. Most of the QMC methods do not depend on particular properties of the sequences used beyond their good distribution in a suitable multidimensional space. Many families of low-discrepancy sequences have been studies, with the family of the Sobol sequences being not only one of the oldest but also the most popular in practical applications. Since these sequences have many free parameters, it is desirable to have methods to obtain a good set of parameters, suitable for a wide class of problems. In this work we propose a fully deterministic method for optimisation of the direction numbers of the Sobol sequences that is flexible in taking into account smoothness properties of the potential sub-integral functions. We demonstrate how by implementing this algorithm on modern GPUs it is possible to obtain *good sets* of direction numbers within a reasonable timeframe. The numerical tests demonstrate the significant advantage in terms of accuracy that is observed, which makes the corresponding QMC methods even more competitive.

1. Introduction.

1.1. Quasi-Monte Carlo methods and low-discrepancy sequences. The quasi-Monte Carlo methods are based upon using specially designed deterministic sequences in order to improve on the accuracy of the typical Monte Carlo methods. While in most cases the convergence of the Monte Carlo methods is $O(1/\sqrt{N})$, the quasi-Monte Carlo (QMC) methods can achieve in practice convergence close to $O(1/N)$, depending on the situation. Usually the problem under consideration can be expressed in terms of numerical integration problem, for a suitable sub-integral function. In such case the Koksma-Hlawka inequality (cf. [6] for a discussion) provides some theoretical justification for the improved convergence, as there are sequences with a rate of convergence of their discrepancy of $O\left(N^{-1}\log^s N\right)$. This rate of convergence is considered to be the best

[*]This work has been supported by project National Science Program "Information and Communication Technologies for Unified Digital Market in Science, Education and Security financially supported by the Ministry of Education and Science.

2020 Mathematics Subject Classification: Primary 11K36; secondary 11K38.

Key words: quasi-Monte Carlo methods, Sobol' sequences, High Performance Computing.

possible and sequences that achieve it are called *low-discrepancy sequences.* Most of the sequences used in practice in quasi-Monte Carlo methods are low-discrepancy sequences, although some sequences without this property are also useful in some algorithms, e.g., Korobov nets [7]. The most important class of low-discrepancy sequences are the Sobol sequences [8], which have proven to be useful in a variety of practical applications, most notably in Financial Mathematics, where they are de-facto standard. Most of the quasi-Monte Carlo algorithms can be formulated in terms of numerical integration, where an integral of a suitably smooth function f in the s-dimensional unit cube is approximated with a sum

$$\frac{1}{N}\sum_{i=0}^{N} f\left(x_i\right). \tag{1}$$

Apart from the usual discrepancy, many other measures for the uniformity of distribution of the sequence $\sigma = \{x_i\}$ are studied. The L_2 discrepancy is also extensively studied. However, the L_2 discrepancy is expensive to calculate and the usual or star-discrepancy are even more challenging. Other interesting measures, related to families of orthogonal functions are used in theory and practice. The usual diaphony [15, 16] is related to the family of trigonometric polynomials. For sequences that are defined using binary number system, as is the case with the Sobol sequences, it is natural to consider the family of the Walsh functions. In that case we obtain the dyadic diaphony, when $\lambda = 2$. A theoretical estimate of the rate of convergence to 0 of the dyadic diaphony for the Sobol sequences can be seen in [10]. We should point out that usually there is a big gap between the theoretical estimates and the practically observed convergence. For a large dimension s a term such as $\log^N$ is much larger than N for all practical values of N. Also estimates that involve discrepancy are usually "worst-case" estimates. Through various randomisation techniques it is possible to obtain better estimates of the "average" case. Various methods of "scrambling" the sequence have been studied. We can think of a scrambling method as a transformation that acts on the fully deterministic sequence σ and produces a transformed sequence $\phi(\sigma, \xi)$, depending on a random variable ξ. Usually such transformations achieve that the estimate from 1 becomes unbiased. The Owen scrambling procedure [3], which is applicable to a wider class than the Sobol sequences, has the additional theoretical advantage that with suitable smoothness assumption it achieves a higher rate of convergence. This advantage can also be observed in practice and that is why our experiments always include Owen scrambling. Since it is well known that quasi-Monte Carlo methods are competitive when the so-called *effective dimensionality* of the sub-integral function is low, it may be interesting to consider measures of equidistribution where only low-dimensional combinations are allowed, e.g., considering only terms where Walsh functions from maximum of 3 dimensions are multiplied. Weighted measures, where different dimensions are taken with different weights, are also well studied.

1.2. The Sobol sequences. The sequences, defined by Sobol, are in fact a family because there are many parameters that can be freely chosen in order to improve their equidistribution properties. Every coordinate, s, of the sequence is determined starting from an infinite matrix $A = a_i j$ that consists of 0s and 1s. For each coordinate there is a primitive polynomial p of degree d over the field GF(2):

$$p(x) = x^d + b_1 x^{d-1} + \cdots + b_{d-1}x + 1.$$

Then the sequence $\{a_{i,j}\}$ must satisfy the property that

$$a_{k,j} = b_1 a_{k-1,j} \oplus b_2 a_{k-2,j} \oplus \cdots \oplus b_{d-1} a_{k-d+1,j} \oplus a_{k-d,j} \oplus a_{k-d,j-d} \tag{2}$$

where the last term is present only if $k - d \geq 0$, when we count the rows and columns starting from zero. Since our algorithm will be implemented using the C programming language it is easier to follow this notation. We set all the diagonal elements of A to be 1 and all the elements above the diagonal to be 0, i.e., when $j > i$, $a_{ij} = 0$. Thus we have $d(d-1)/2$ degrees of freedom in selecting the elements of A.

2. Problem formulation. The goal of this work is to describe algorithms that can optimise the direction numbers of the Sobol sequence depending on known smoothness properties of the subintegral function. Effective dimensionality. Sensitivity analysis gives rise to the problem of appropriate selection of direction numbers, using weights inspired from those known properties of the sub-integral functions. The goal is to develop a sufficiently versatile algorithm that also runs in appropriate wall clock time, so that the numbers can be obtained in reasonable time.

In our previous work [1] we described a method which uses effectively the computing power of GPUs and an essentially Monte Carlo technique for assessing the fitness of the direction numbers with respect to our chosen measure of smoothness. Another approach which uses genetic algorithms is described in the work of Kovaleva under the supervision of Lemieux [13]. However, there are certain practical considerations that were not taken into account during the development of our algorithm. Namely, when the number of dimensions becomes sufficiently high compared with the number of points of the sequence, by the Dirichlet principle it becomes impossible to have distinct first columns of the matrices of direction numbers. When two dimensions have the same first columns, their correlation is necessarily high, since the first bit of the sequence will be always the same. Even when the theoretically very well supported Owen scrambling is applied to the resulting sequence, it does not break the correlation significantly. Similar problems arise for the next columns of the direction numbers, but they are of lesser importance and are mitigated by Owen scrambling. Another issue is that it is expected that the first dimensions of the Sobol sequences have better distribution compared with the higher dimensions and this property is used in planning and executing quasi-Monte Carlo algorithms. Therefore it is desirable to have an algorithm that yields direction numbers dimension by dimension, as opposed to all at once. Our algorithm was designed to work for a fixed number of points to be generated, which does not cover the popular use case when certain number of points is generated and then if the desired accuracy is not reached, more points are used. This is a significant restriction that is desirable to be removed.

3. Optimisation Algorithms.

3.1. Description of the algorithm. Based on our experience with the design and application of the algorithm from [1], which employed swarm optimisation, we devised a new algorithm that avoids the drawbacks mentioned earlier and opens new possibilities for use in practical applications. The algorithm is entirely deterministic, based on a greedy approach to the optimisation, and has a variety of options that allow it to be tailored for the particular use case at hand. It fills consecutively the direction numbers dimension-by-dimension and column by column. It is based on minimising a fitness function for each column. The parameters of the algorithm are as follows:

- Total number of dimensions: N
- Starting index m and ending index n
- Smoothness parameter λ
- Dimensionality parameter k.

The starting index m and the ending index n indicate that the smallest number of points that we consider important for the use-case is 2^m and the largest number of points is 2^n. Although we will be able to compute integrals with more than 2^n points, the optimisation procedure will not guarantee a good performance in such case. The smoothness parameter λ means that we consider the norms of the Walsh functions that we consider to be elevated to a power λ. Effectively we have tested λ equal to 1, 1.5 and 2, as well as hybrid variants where λ changes with the increase in dimension. The dimensionality parameter k indicates the maximum dimensionality of the Walsh functions under consideration. The case of $k = 1$ is not interesting, while the cases of $k = 2$ and $k = 3$ have practical importance, since they cover integrals of functions with the corresponding effective dimensionality. The case of infinite k has also been considered, since it covers the dyadic diaphony (when $\lambda = 2$) and also the case of functions that are of so-called "Type C". This means that a limitation on dimensionality is not imposed, although it is not possible to have dimensionality larger than the total number of dimensions. The quantity that we desire to minimize is the following:

$$\sum_{m\le l\le n}\left(\sum_{t\in R^N}\delta\left(l,t,A\right)|t|^{\lambda}\right)^2+\sum_{m\le l\le n}P(l,A). \tag{3}$$

The numbers t encode the Walsh functions for each dimension with a number t_i between 0 and 2^n-1. The norm $|t|$ is then defined as $\prod_{i=1}^{n}2^{-q_i}$, where q_i is the smallest power of two so that $t_i < 2^{q_i}$. By δ we denote a function that is 0 except when the result of the bitwise exclusive or of the columns chosen by the numbers t_i for each dimension is exactly 0 for the first l bits. The bits are counted from the top of the matrix A of the direction numbers. The term $P(l,A)$ is a penalty which sums over the dimensions the number of times when the first column for a given dimension is exactly equal to another first column in a previous dimension, in the first l bits. The exact value of the penalty is a suitably chosen relatively big number. We used simply 10^6. We employ a greedy algorithm along the dimensions and columns. We keep several data structures. The weights w are indexed with the dimensionality index (integer between 1 and k, but no indexing if k is infinity), the power between m and n, and an index between 0 and 2^n-1. Only the sum of these weights $\bar{w}$ along this dimensionality index is used in the computation of the fitness score, while this index is important only for the updating of these weights when a new dimension is started. Once all the direction numbers for dimensions less than s are fixed, the weights w are re-computed and fixed and are used for the computation of the fitness for all columns in dimension s. They serve to aggregate the contributions of these previous dimensions. The "combinations" data structure c is indexed again by the power l between m and n and by a number between 0 and 2^p-1, where p is the column index, starting from 0. For each such number, t, we compute the combination (the bitwise exclusive or sum) of the columns that form a subset corresponding to t (the 1s in t indicate which columns to combine). For each power l between m and n we take only the first l bits and save them in the array c. We evaluate all the possible columns in parallel. Only columns that satisfy the relation 2 are accepted. The verification of this property is simple. Then for the given column we have to add the contributions for all the powers under consideration, i.e., between m and n. For a power l, $m \leq l \leq n$, the contribution to the fitness function for the column p with content t will be obtained by cycling over all subsets of previous columns and then taking the corresponding weights from the weight structure. Effectively we have

$$\sum_{0 \leq t \leq 2^p - 1} \bar{w}(l, c(l,t)) 2^{-\lambda p}. \tag{4}$$

For the first column only, we additionally count the number of coincidences with first columns in the already computed dimensions and add a penalty in the case when this number is bigger than 0. The coincidences are again considered for the first l bits only. In this way for each power l between m and n we obtain a separate fitness result. We take the sum of squares to be the final fitness function. We always select the column with the smallest fitness function. In case of a tie we can chose any of the candidates. To achieve determinism, we can say that we select the smallest candidate column.

3.2. Extension to cover larger number of points. Although the algorithm produces direction numbers that are optimised for a number of points that is not larger than 2^n, it is desirable that the algorithm is able to produce any number of points. The natural limitation here is 2^{52}, because when floating point numbers are represented in the most popular double precision (binary64) format as specified in IEEE 754-2008 revision [5], the number of significant bits is 52. For the dimensions where the degree of the respective polynomial used is less than n, the direction numbers are continued following the formula 2. As is usual the diagonals are filled with 1s and the numbers above the diagonal are filled with 0s. For the rest where we have a choice we simply fill with 0 or 1 with equal probability, using a pseudo-random number generator. For the first dimensions we can expect good results even for powers that are larger than 2^n, because we are following the construction of the Sobol sequences and we maintain the LP_τ property. For the larger dimensions we expect worse results compared with direction numbers optimised for this number of points.

4. On the complexity of the algorithm. The algorithm for optimising the direction numbers described above has polynomial complexity in terms of dimensions and number of points to be generated (which is at most 2^n). Although the re-computing of the weights and combinations structures is not negligible and in some cases can even dominate the total computational time, for large dimensions and large number of points, the actual computation of the fitness of the columns is dominant. As we index the columns starting with 0, the number of possibilities for a given column p is limited by 2^{n-p-1} (since the diagonal has $1s$ and above the diagonal we have $0s$). Considering that such a column should also satisfy the property 2, we can have even less possibilities to consider. If the column is fixed and the index of the power-of-two is l, $m \leq l \leq n$, then for each subset of previous columns encoded as t, $0 \leq t \leq 2^p - 1$ we only combine the corresponding weight from the weight structure w using the combination from the c array and add to the fitness function. Therefore the number of operations is for this column will be less than $2^{n-1}(n-m+1)$. For each dimension we consider only the first n columns. Thus the total number of operations is not more than $2^{n-1}(n-m+1)nN$. Since the maximum number of points Q is 2^n, we see that the number of operations is $O(NQ\log^2 Q)$, N being the total number of dimensions. The re-computation of the weights structures w and $\bar{w}$ is done before each new dimension is started. From the dimension that was finished we have 2^n possible Walsh functions to consider. When we fix one of these, we compute the *xor* sum of the corresponding columns in this dimension and then compute new weights. For this we need approximately n operations. Once we have this summed column, we update the weights, so that from weights of order q we obtain weights of order $q+1$. Consider, for example, that a 3-dimensional combination that contains the dimension s stems from a 2-dimensional combination of two previous dimensions. Thus the total number of operations is in order of $kn2^n$, where k was the maximum dimensionality. When k is infinity, we simply do not keep track of dimensions, so we have only one weight array and we update it, so the number of operations is in the order of $kn2^n$. Multiplying by the number of columns and number of dimensions, we obtain the total operations in this part to be in the order of $Nkn^2 2^n$. The updating of the combinations structure is done before starting the optimisation for each column. The number of operations is proportional to the number of possible subsets (at most 2^n). At most n operations are needed for the bitwise or. Then we have to mask correspondingly to the power between m and n, so not more than $2n$ operations. Since the maximum number of columns is n, we need about $n^2 2^n$ operations for one dimension and thus $n^2 2^n N$ in total. In practice the updating of the combinations structure takes considerably less time, while the updating of the weights may sometimes require more time then the optimisation of the columns, depending on the implementation. Still, for large enough n the optimisation of columns dominates the whole computation. The evaluation of the penalty for coincidences of columns is done only for the first column and thus is of lower order of magnitude.

We note that the computation of an integral that uses Q points and also meaningfully uses all the coordinates will require at least $O(QN)$ operations just in order to generate all the coordinates, so we see that the extra factor is just $\log^2 Q$, which is to be expected. Nevertheless, in a complex computation or in the case when many integrals are to be computed consecutively, it may well happen that the optimisation will be dominated by the actual computations of integrals. Thus although we expect that the optimisation will be performed once and then the direction numbers will be fixed and used in many computations, it may be suitable to generate direction numbers on-the-fly and then use them in one or more computations, especially considering that the use of GPUs for the optimisation leads to a very fast implementation, as we will see in the next section and then in the examples.

5. Implementation of the optimisation algorithm using CUDA. The algorithm has natural parallelism and as such can be implemented using NVIDIA CUDA [2] with sufficient efficiency. While the driver routine runs on the CPU, most of the computations are performed on the GPU launching appropriate kernels. The data structures mentioned above – for the weights and the counts, are kept on the GPU and updated only there. The results from the fitness function evaluation are transferred to the CPU in case more complex processing is to be applied. Three CUDA kernels are developed where the bulk of the computations happens:

- weights update

- counts update
- fitness evaluation.

The optimisation is performed in a cycle over dimensions, starting after the first dimension, where the result is fixed as the classic Van der Corput sequence. Based on the columns computed for the previous dimension, the "weights" w and their sum $\bar{w}$ are updated as described above. The computation is naturally parallel over the possible Walsh functions for this dimension. As they are encoded with an index between 0 and $2^n - 1$, the computation is done in parallel. Where there is addition, atomic operations (*atomicAdd*) are used in order to ensure correctness.

The update for the combinations c is performed for each new column. Since the code for this kernel is relatively concise, we show it as an example here:

```
__global__ void gpu_kernel_right_counts(uint64_t *comb,uint64_t *columns,
     int column_idx,int m,int n){
     size_t tid =threadIdx.x + blockIdx.x * blockDim.x;
     size_t block_size = blockDim.x*gridDim.x;
     int numpowers = n-m+1;
     size_t  total_work  = (1<<column_idx) ;//possible subsets
     for(size_t  w_idx = tid ; w_idx < total_work; w_idx+=block_size){
               uint64_t temp=0;
               for(int i=0;i<column_idx;i++){
                    if ((1ULL<<i) & w_idx){
                              temp ^= columns[i];
                    }
               }
               for(int i=0;i<numpowers;i++){
                    comb[ i + w_idx * numpowers ] = temp & ((1ULL << (m+i))-1);
               }
     }
}
```

A given subset of columns is encoded as a binary number between 0 and 2^p, where p is the new column index *(column_ idx)*, and then for this subset the bitwise exclusive or is computed in *temp* and then masked appropriately for each of the powers between m and n and stored in the array c. The more complex computation of the fitness function is parallelized in a similar manner. One source of parallelism is the number of candidate column values. The other is the choice of subset of the previous columns, which is encoded as a number between 0 and $2^p - 1$. If we do not take into account the requirement to obey the equation 2, for a column of index p we have 2^{n-p-1} columns to evaluate. We also compute in parallel the fitness with respect to each power between m and n. Thus we have $2^{n-p-1}(n - m + 1)2^p$, i.e., $2^{n-1}(n - m + 1)$ so-called "work items", which we can evaluate in parallel. As is usual in CUDA, we have high number of parallel threads and this work is divided between them. In order to take into account equation 2, we verify it and reject those candidate columns that do not satisfy it by stopping further evaluation and entering very high score for the fitness (since we search for a minimum). If the equation is satisfied, then we proceed to compute the fitness for the corresponding power l, $m \leq l \leq n$. Effectively, the appropriate value c from the combinations structure is multiplied with the appropriate weight from the weights structure $\bar{w}$ and then with the norm for the Walsh function in this dimension, i.e., with $2^{-\lambda p}$, as in 4.

6. Timing results. Our algorithm was developed mainly for the use case when one desires to generate a set of direction numbers once and then use them in multiple applications. For such a use case it is important to be able to reach high number of dimensions and high number of points for reasonable time, like a few days. This is possible when using a powerful GPU card, like Nvidia's V100. Another possible use case is when one desires to obtain the direction numbers fast in order to apply them in a particular problem. When the number of terms of the sequence to be generated is not that high, this would also be possible. In the next figure one can see timing results, when the dimension increases. The lower limit of the number of points is fixed at 2^{10} and we vary the upper limit.

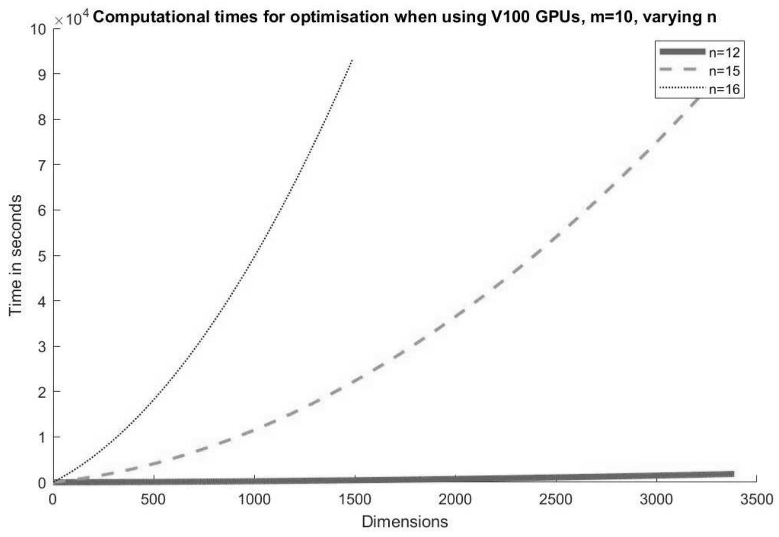

Fig. 1. Computational times for varying number of points and dimensions

One can see that the computations complete very fast for up to 2^{12} points and in reasonable time for 2^{16}. Note that through the use of the relation 2 the direction numbers can be extended indefinitely for the dimensions shown, since the degrees of the corresponding polynomials are below 16. By running the optimisation executable for days one can reach much higher dimensions. We have effectively reached even 65536 dimensions.

7. Numerical results. Typically the performance of the Sobol sequence or any low-discrepancy sequence is evaluated in computing of test integrals. Since we have the option to scramble the sequence, for a fixed number of points we perform sufficiently high number of evaluations M with different seed of the pseudorandom generator that we use for the scrambling, in order to obtain a situation with M samples that are identically distributed and have mathematical expectation equal to the integral to be evaluated. In this way we can compute the mean-squared error as a measure of the accuracy. Here we show results for several integrals that are popular among practitioners. In all cases the integration is performed over the s-dimensional unit cube E^s and the exact value of the integral is made to be 1 by multiplying with an appropriate constant when required. Thus we need to compute:

$$I_i = \int_{E^s} F_i(x)dx, \tag{5}$$

where the sub-integral functions are as follows:

- $F_1(x) = \left(\frac{1}{\sqrt{s}} \sum_{i=1}^{s} \Phi^{-1}(x_i) \right)^2$, where $\Phi(x)$ is the cumulative distribution function of normal distribution and $\Phi^{-1}(x)$ is its inverse.
- $F_2(x) = \left(1 + \frac{1}{s}\right)^s \prod_{i=1}^{s} x_i^{\frac{1}{s}}$.
- $F_3(x) = \prod_{i=1}^{s} \frac{s - x_i}{s - \frac{1}{2}}$.

In the next figures we can see a comparison of the accuracy when computing integrals with the optimised direction numbers and direction numbers as given by Joe and Kuo [11], when using criterion 5 and also as given in the CUDA toolkit, under the so-called *randomized version*, which seems to give better results in tests. In order to have the same conditions, we used our implementation of Owen scrambling for all the sets of direction numbers. One can see that the optimised direction numbers offer a significant advantage. The number of points is fixed to 2^{12} and the dimensions vary. Note that when the number of dimensions becomes 4096, it is equal to the number of points and thus there are definitely cases when the first columns will coincide.

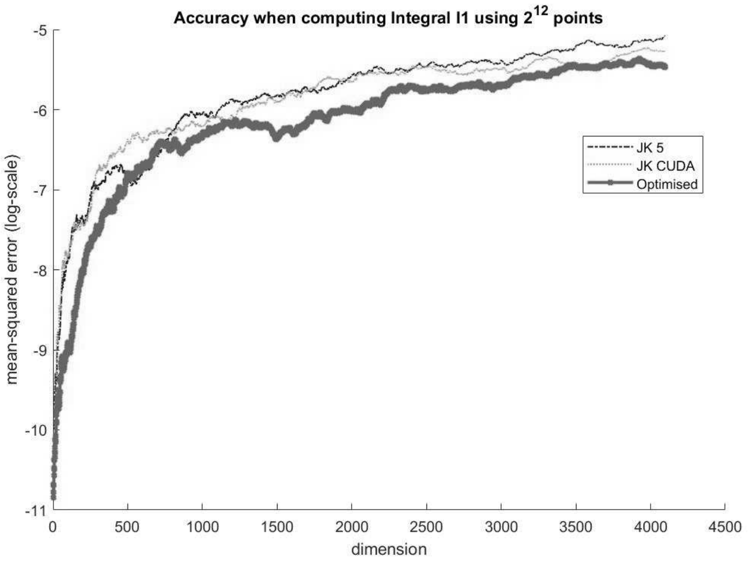

Fig. 2. Accuracy of computation of integral I_1 using 2^{12} points with different sets of direction numbers

In many practical problems the speed of generation is important and thus it is not possible to use the complex Owen scrambling. The simpler Matousek scrambling can be accomplished without loss of speed compared with not using any scrambling at all. That is why in the next figure we show a comparison of the same sets of direction numbers and integrals with only the Matousek scrambling applied.

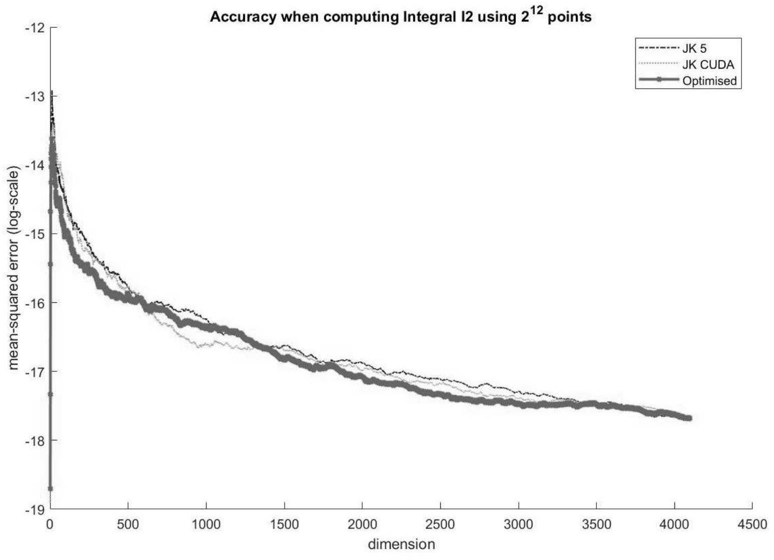

Fig. 3. Accuracy of computation of integral I_2 using 2^{12} points with different sets of direction numbers

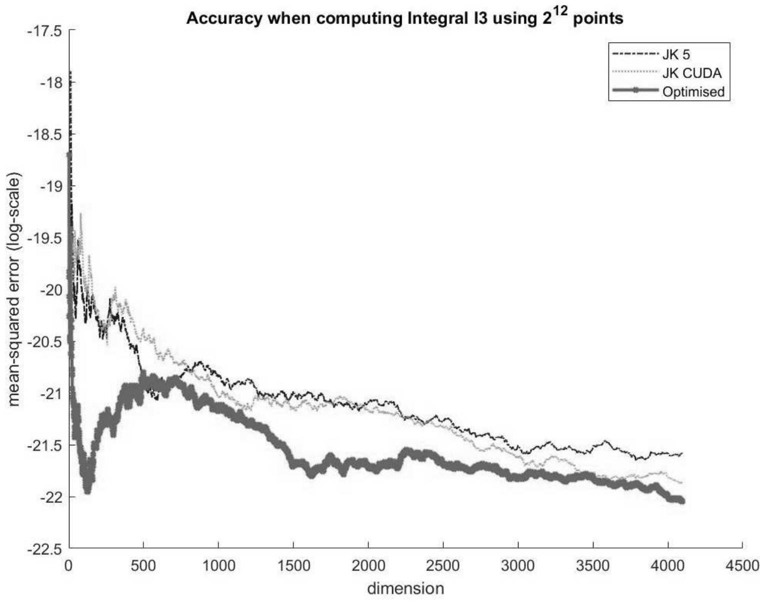

Fig. 4. Accuracy of computation of integral I_2 using 2^{12} points with different sets of direction numbers

One can see that the optimised direction numbers command significant advantage, in some cases several times (the mean squared error is shown in log-scale, in base 2).

8. Conclusions and directions for future work. We presented a flexible algorithm for obtaining direction numbers of the Sobol sequences, which achieves significantly better results on widely used benchmark problems and can be tuned to take into account properties of the sub-integral function that stem from the practical problems under considerations. Apart from obtaining a good fixed set of direction numbers with wide applicability, one can also use it to obtain such sets of numbers dynamically, at a pre-processing stage of the computation, even dynamically deciding which parameters to use for the optimisation algorithm.

There has been a significant interest in problems with function spaces that include weights in their definitions. Our algorithm allows for such weights to be taken into account and it is to be expected that the incorporation of weights and varying appropriately the parameter λ with the dimension will lead to even better practical results.

Acknowledgments. The author wants to express his gratitude to Sergei Kucherenko for the provided codes, including [4], and the invaluable discussions that considerably impacted the present work.

REFERENCES

[1] E. Atanassov, S. Ivanovska, A. Karaivanova. Optimization of the Direction Numbers of the Sobol Sequences. Studies in Computational Intelligence, 902 SCI, 2021, 145–154.

[2] NVIDIA CUDA Toolkit. `https://developer.nvidia.com/cuda-toolkit`. Last accessed 7 Feb 2021.

[3] A. Owen. Scrambling Sobol' and Niederreiter-Xing points. *Journal of Complexity*, **14** (1998), 466–489.

[4] Software for generating the Sobol sequences by BRODA. `https://www.broda.co.uk/software.html`. Last accessed 7 Feb 2021.

[5] IEEE Standard for Floating-Point Arithmetic. In: IEEE Std 754-2008 , 1–70, 29 Aug. 2008, doi: 10.1109/IEEESTD.2008.4610935.

[6] L. Kuipers, H. Niederreiter. Uniform Distribution of Sequences, New York, Dover Publications, 2006.

[7] N. M. Korobov. The approximate computation of multiple integrals. *Dokl. Akad. Nauk SSSR*, **124** (1959), 1207–1210 (in Russian).

[8] I. M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *Comput. Math. Math. Phys.*, **7** (1967), 86–112.

[9] I. M. Sobol', D. Asotsky, A. Kreinin, S. Kucherenko. Construction and Comparison of High-Dimensional Sobol' Generators. Wilmott, Nov 2012, 64–79.

[10] E. I. Atanassov. On the dyadic diaphony of the Sobol' sequences. *LNCS*, **2179** (2001), 133–140.

[11] S. Joe, F. Y. Kuo. Remark on Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Trans. Math. Softw.*, **29** (2003), 49–57.

[12] S. Joe, F. Y. Kuo. Constructing Sobol sequences with better two-dimensional projections. *SIAM J. Sci. Comput.*, **30**, (2008) 2635–2654.

[13] E. Kovaleva (under supervision by C. Lemieux) Genetic Algorithm for searching parameters of Sobol sequence, 2013, `https://uwaterloo.ca/computational-mathematics/sites/ca.computational-mathematics/files/uploads/files/kovaleva_project.pdf`

[14] I. Sobol', D. Asotsky, A. Kreinin, S. Kucherenko. Construction and comparison of high-dimensional Sobol generators. *Wilmott J.*, **56** (2011), 64–79.

[15] P. Zinterhof. Uber einige Abschatzungen bei der Approximation von Funktionen mit Gleichverteilungsmethoden, Sitzungsber. iisterr. Akad. Wiss. Math.-Naturwiss. Kl. II 185 (1976), 121–132.

[16] P. Zinterhof, H. Stegbuchner. Trigonometrische Approximation mit Gleichverteilungsmethoden, *Studia Sci. Math. Hungar.*, **13**, No 3–4 (1978), 273–289.

[17] S. Kucherenko, B. Feil, N. Shah, W. Mauntz. The identification of model effective dimensions using global sensitivity analysis. Reliability Engineering and System Safety, **96** (2011), 440–449.

[18] R. Liu, A. Owen. Estimating mean dimensionality of analysis of variance decompositions. *Journal of the American Statistical Association* **101**, No 474 (2006), 712–721.

Emanouil Atanassov
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Acad. G. Bonchev Str., Bl. 25A
1113 Sofia, Bulgaria
e-mail: emanouil@parallel.bas.bg

ДЕТЕРМИНИСТИЧЕН АЛГОРИТЪМ ЗА ОПТИМИЗАЦИЯ НА НАПРАВЛЯВАЩИТЕ ЧИСЛА НА РЕДИЦИТЕ НА СОБОЛ

Емануил Атанасов

Квази-Монте Карло (QMC) методите използват *детерминирани редици*, обикновено с *нисък дискрепанс*, за да постигнат подобрен ред на сходимост в сравнение с Монте Карло методите, решаващи същия проблем. Въпреки че тази подобрена сходимост зависи от гладкостта на разглежданите функции и понякога не може да се оцени теоретично, на практика са получени много добри резултати в редица приложни области, например във финансовата математика. Повечето от квази-Монте Карло методите могат да се използват с всяка редица с *добро разпределение* в съответното многомерно пространство. Поради това са разработени и изследвани много фамилии от редици с нисък дискрепанс. Редиците на Собол са не само едни от най-старите, но също са и най-популярни в практически приложения. Тъй като те имат много свободни параметри, желателно е да се разработят методи за получаване на добри набори подходящи параметри за широк клас от проблеми. В тази работа се предлага детерминистичен метод за оптимизиране на направляващите числа на редиците на Собол, който е достатъчно гъвкав за да отчете гладкостта на потенциалните подинтегрални функции. Демонстрираме как чрез прилагане на този алгоритъм на съвременните графични процесори е възможно да се получат добри набори от параметри за разумно време. Числените тестове показват значително предимство по отношение на наблюдаваната точност, което прави съответните квази-Монте Карло методи още по-конкурентни.