An algorithm for QMC integration using low-discrepancy lattice sets

Vojtěch Franěk

Abstract. Many low-discrepancy sets are suitable for quasi-Monte Carlo integration. Skriganov showed that the intersections of suitable lattices with the unit cube have low discrepancy. We introduce an algorithm based on linear programming which scales any given lattice appropriately and computes its intersection with the unit cube. We compare the quality of numerical integration using these low-discrepancy lattice sets with approximations using other known (quasi-)Monte Carlo methods. The comparison is based on several numerical experiments, where we consider both the precision of the approximation and the speed of generating the sets. We conclude that up to dimensions about 15, low-discrepancy lattices yield fairly good results. In higher dimensions, our implementation of the computation of the intersection takes too long and ceases to be feasible.

Keywords: QMC integration, lattices, discrepancy

Classification: 65D30

1. Introduction

High-dimensional integrals are often numerically approximated using Monte Carlo or quasi-Monte Carlo methods. Some known error estimates, such as the Koksma-Hlawka inequality, and many numerical tests ([7], [13], [14], [22]) show that in many cases, the quasi-Monte Carlo methods using such sets outperform the Monte-Carlo method significantly. All these methods approximate the exact value[1] $\int_{I^d} f(x)\,\mathrm{d}x$ by the average $\frac{1}{N}\sum_{\vec{x}\in M} f(\vec{x})$, where M is a suitable N-point set. Well known examples are the Faure set, general (t,m,d)-nets and randomly scrambled modifications of these sets. (For more information, see e.g. [12], [15], [13], [14] or [11]). These constructions were found as examples of *low-discrepancy sets*, ie. sets with discrepancy[2] of the order $O(\log^{d-1} N)$ (the lowest discrepancy known so far). Skriganov in [18] proved the low-discrepancy property for a class of sets generated as the intersection of suitable lattices with I^d. This class contains the only low-discrepancy sets that have not yet been tested concerning numerical integration. One possible reason for this is the fact that computational problems with lattices in higher dimensions are often known to be hard (see e.g. Lovász [9]).

[1] In this paper, we consider integration only over the unit cube $I^d = [0,1]^d$.

[2] In this paper, we consider only the discrepancy for axis-parallel boxes (see [10]).

Here we suggest a method of generating such sets and we test its practical efficiency as well as the accuracy of quasi-Monte Carlo integration using these sets. We found out that our method ceases to be feasible in the dimension about 20. We constructed the sets up to the size $N = 10^5$, but reducing the size speeds up the construction only slightly, and one can expect that the number of generated points N does not affect the total computing time too much. As for the quality of the approximation, our lattice sets are about twice as good as the Monte-Carlo method (in lower dimensions even better). In our experiments, however, they cannot compete with some others, especially linear scrambled Faure sets.

The method based on Skriganov's results may sound familiar to the reader who knows about QMC methods using lattice rules. However, the lattice rules described in the literature ([19], [18], [12]) are based on lattices that include the integer lattice $\mathbb{Z}^d$ (and the corresponding node sets are easy to compute). That is not the case for the lattices considered here. The computation of the intersection of these lattices with the unit cube is substantially more difficult and we address it in this paper. We did not perform any comparison of our method with lattice rules but this would surely deserve some attention in a future work.

2. Lattices

As was mentioned in the introduction, we will generate point sets using low-discrepancy lattices[3]. A *lattice* L with basis $B = \left\{\vec{b}_1, \vec{b}_2, \dots, \vec{b}_d\right\} \in \mathbb{R}^d$ is the set of all integer linear combinations of the basis:

$$L = L(B) = \left\{ \sum_{j=1}^{d} i_j \vec{b}_j : i_1, \dots, i_d \in \mathbb{Z} \right\}.$$

The *norm* of a lattice L is defined as $\mathrm{Nm}(L) = \inf_{\vec{x} \in L \setminus \{0\}} |x_1 x_2 \dots x_d|$, where $\vec{x} = (x_1, x_2, \dots, x_d)$. The *determinant* $\det(L)$ of a lattice L is the absolute value of the determinant of the $d \times d$ matrix having basis vectors as rows. Skriganov in [18] proved the following result:

Theorem 1. *If* $\mathrm{Nm}(L) > 0$ *and* $\det(L) = 1$, *then the lattice* $\frac{1}{t}L$ (*where* $t > 0$ *is a real parameter*) *has discrepancy* $D(\frac{1}{t}L) = O(\log^{d-1} t)$ *as* $t \to \infty$. (*If* $\det(L) \neq 1$, *we can rescale* L *suitably.*)

Skriganov also mentions a class of lattices with positive norm. They have basis vectors of the form

$$\vec{b}_1 = (1, 1, \dots, 1), \vec{b}_2 = (\alpha_1, \alpha_2, \dots, \alpha_d), \dots, \vec{b}_d = (\alpha_1^{d-1}, \alpha_2^{d-1}, \dots, \alpha_d^{d-1}),$$

[3]The discrepancy of a lattice L is understood to be the discrepancy of its intersection with the unit cube.

where $\alpha_1, \alpha_2, \ldots, \alpha_d$ are mutually different roots of a monic polynomial $p(x)$ of degree d which is irreducible over the rationals and has integer coefficients. One known approach from theory of numbers that produces such polynomials is described e.g. in the book [10]:

Theorem 2. *Let $p = 2md+1 \geq 5$ be prime for an integer m, let $\omega = e^{2\pi i/p}$, and let r be a primitive element modulo p; that is, that all the powers $r^0, r^1, \ldots, r^{p-2}$ are mutually different modulo p (in other words, r is a generator of the multiplicative group of the field $\mathbb{Z}/p\mathbb{Z}$). Then $\alpha_j = \sum_{k=0}^{2m-1} \omega^{r^{kd+j}}$ for $j = 1, \ldots, d$ are all distinct real roots of the polynomial $q(x) = \prod_{j=1}^{d}(x - \alpha_j)$ and this polynomial is irreducible over rationals, has integer coefficients, and is monic.*

In this work, we tested the lattices with exactly these bases (suitably rescaled). To check theoretical results of suitability of these low-discrepancy lattices for numerical integration, we compared them with lattices with bases consisting of uniform random vectors from the unit ball and uniform random unit vectors. Our task is to find, as efficiently as possible, a lattice whose intersection M with the unit cube consists of (almost exactly) n points, ie. $N = |M| \approx n$, where n is a given number.

2.1 Scaling of the basis.

Suppose we already have some basis (denoted by B_0), given by d vectors $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d \in \mathbb{R}^d$. Our goal is, by some scaling of these vectors, to make the intersection M of the unit cube and the lattice generated by this new basis contain (nearly) n points. If the intersection of a lattice L with the unit cube should contain about n points, then we expect that $\det(L) \approx \frac{1}{n}$. (The determinant of a lattice is the volume of the parellelepiped spanned by its basis vectors.) Thus, if one of the approaches described above yields the basis B_0, we will take the basis

$$B_1 = B_0 \sqrt[d]{\frac{1}{n \det(L_0)}}\,.$$

Practical experiments have shown that the lattice with the basis scaled in this way has almost always the desired number of $n \pm 1$ points in the unit cube. However, due to the discrete nature of the lattice, in some scarce cases the first scaling is not so precise and we are forced to rescale it again[4] :

$$B_i = B_{i-1} \sqrt[d]{\frac{N_{i-1}}{n}}, \quad \text{where} \quad N_{i-1} = |L(B_{i-1}) \cap I^d|.$$

[4] This was necessary in our work only (and we never needed more than three scalings), because we wanted to compare some methods and thus the number of points had to be always the same; in practical situations the first scaling is accurate enough.

2.2 The algorithm.

As soon as we have the (properly scaled) basis B, we want to find and count the points of the intersection $L(B)\cap I^d$. We have developed a recursive algorithm, based on cutting the cube. Our task is to find all the points of the lattice within the unit cube, formally

$$\vec{0} \leq \sum_{k=1}^{d} x_k \vec{b}_k \leq \vec{1}, \text{ where } \ x_1, x_2, \dots x_d \in \mathbb{Z}.$$

The algorithm uses linear programming to estimate the limit values of x_1 of these points. Then it takes the lowest integer between these bounds, fixes x_1 to this value and finds the limits for x_2 with respect to this fixed value. It continues recursively with fixing x_2 and so on, until we have all the coefficients fixed. (In this case, we have found a point $\sum_{k=1}^{d} x_k \vec{b}_k$ of the lattice within the unit cube.) If there are no more coefficients to fix, or there is no integer within the specified range, we simply relax the last fixed coefficient and increase it by one. If this new value is still within the allowed range, we can continue the same way as above with fixing the increased value and looking for the limits of the first not fixed coefficient. If the new value is above the allowed range, we go on with relaxing the remaining coefficients until we find an acceptable number. If there are no more integers within the range for x_1, the algorithm ends. This way we test all the possible integer combinations of basis vectors, which may produce a point within the unit cube. Because the range for any coefficient is bounded, the number of tested combinations is finite and the algorithm always terminates.

Input: The dimension d and a basis $B = (\vec{b}_1, \vec{b}_2, \dots, \vec{b}_d)$.

Output: The set $M = L(B) \cap I^d$.

Step 1 (Init):

The number of fixed coefficients: $f = 0$.

The integer combination (given by the fixed coefficients) of first f vectors: $\vec{p}_0 = \vec{0}$.

Step 2 (Find limits): Solve the linear programming problem given by the inequalities

$$\vec{0} \leq \vec{p}_f + \sum_{k=f+1}^{d} a_k \vec{b}_k \leq \vec{1}$$

(where $a_{f+1}, a_{f+2}, \dots, a_d$ are the unknown variables) and the objective function $g(a_{f+1}, a_{f+2}, \dots, a_d) = a_{f+1}$. Denote its minimum by g^{min} and maximum by g^{max}.

Put $x_{f+1}^{min} = \lceil g^{min} \rceil$ and $x_{f+1}^{max} = \lfloor g^{max} \rfloor$. These are the limits for integer coefficient x_{f+1} in the unit cube cut by the subspace

$$S_f = \left\{ \vec{p}_f + \sum_{k=f+1}^{d} a_k \vec{b}_k : a_{f+1}, \dots, a_d \in \mathbb{R} \right\}.$$

If $x_{f+1}^{min} > x_{f+1}^{max}$, then go to **Update** (in this case, the intersection of the subspace S_f with the unit cube contains no integer combination of basis vectors, in particular, the coefficient a_{f+1} cannot be an integer); else

Step 3 (Increase recursion depth): Add one to f and put $x_f = x_f^{min}$, which means we have set and fixed the coefficient x_f. Set $\vec{p}_f = \vec{p}_{f-1} + x_f \vec{b}_f$ and go to **Test**.

Step 4 (Update): While $x_f = x_f^{max}$ (the last coefficient fixed cannot be incremented any more) decrease f by one. If $f = 0$, then the construction of the set M is done and we can end the algorithm; else add one to x_f, update $\vec{p}_f = \sum_{k=1}^{f} x_k \vec{b}_k$ by adding $\vec{b}_f$ and perform the following step:

Step 5 (Test): If $f = d$, then the point $\vec{p}_f = \vec{p}_d = \sum_{k=1}^{d} x_k \vec{b}_k$ is a point in the unit cube as well as a point of the lattice $L(B)$. Thus, add p_f to M and go to **Update**; else go to **Find limits**.

2.2.1 Integer versus linear programming.

It is obvious that if we use linear programming, the space we explore is not the smallest possible, still. It can happen that the intersection of the flat S_f with the unit cube does not contain any lattice points (i.e. any integer combination of the basis vectors) but contains linear combinations of the basis with first f' integer coefficients ($d > f' > f$). We cut this flat S_f with some other flats and we never find any lattice point, for after fixing $f' < d$ coefficients we reveal the non-integer one (that is the case when $x_{f'+1}^{min} > x_{f'+1}^{max}$), and thus all the cutting of S_f was unnecessary.

We could improve this by using integer programming which would detect this situation immediately. But the linear programming problem can be solved in polynomial time while the integer programming is NP-hard and there are no efficient algorithms known for solving it. Moreover, in our experiments we met this situation in only about ten to twenty percent of all cases and the difference $f' - f$ was almost always equal to one. Thus, even if we had some efficient integer programming algorithm it would not bring any significant speedup.

2.2.2 Small improvements.

If we analyse the behavior of our algorithm, we will find out soon that it is preferable to have the basis vectors sorted descendingly by their length. Figure 1 illustrates that using the basis $B = (\vec{b}_1, \vec{b}_2)$ we explore only five flats, in this case five lines parallel to $\vec{b}_2$. On the other hand, if we take the basis $B' = (\vec{b}_2, \vec{b}_1)$, then there are ten lines parallel to $\vec{b}_1$ we must explore. The same is true in higher dimensions: the longer the i-th basis vector is, the fewer $(d-i)$-flats we explore at the i-th level of recursion. Because it is obviously easier to explore flats of lower dimension, it is now clear why the decreasing basis is preferable.

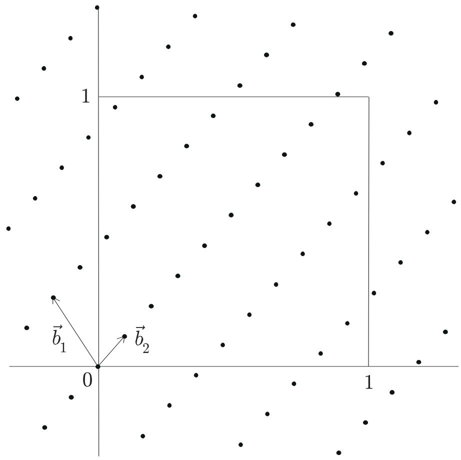

FIGURE 1. The advantage of decreasing basis.

The second improvement is at the level of implementation, though it speeds up the algorithm even more significantly. To find the extremes for x_d we do not need linear programming — it suffices to cut the line S_{d-1} according to the unit cube and we get the result in constant time.

The last amendment does not affect the speed of the computation but allows us to gain better both approximation and stability of the integration. The lattice defined so far always contained the origin, which is an exceptional point for many common functions and its presence tends to influence the approximation adversely. We cannot just omit it from M; this would corrupt the regularity of the lattice, which would result in a worse approximation. Instead, as in [2], we translate the whole lattice by a random vector $\vec{t}$. We thus work with the set $L = L(B) + \vec{t}$. We can also generate several sets M for one basis, and for the final value of the approximation of the integral we can take for example the median of results on these sets, which improves the stability of our computations.

2.3 Some known methods.

In this section, we mention some known methods of generating the set M, which we will use later for comparison. More on them can be found, for example, in the study [11].

Monte Carlo: The points are generated independently and uniformly at random.

Halton's sequence: This is the first known method that yields low-discrepancy sets.

Richtmyer's sequence: Let $2 = p_1 < p_2 < \ldots < p_d$ be the d smallest primes. Then the k-th component of i-th element of Richtmyer's sequence is

$$x_{i,k} = (i\sqrt{p_k}) \bmod 1.$$

It is not known whether the generated set has low-discrepancy. Nevertheless, it is very popular due to its easy implementation. Moreover, it proves to be quite a good choice, in the dimensions about 15 often even better than some other more sophisticated constructions.

Linearly Scrambled Faure: This method is based on the ideas of Owen [15] and Tezuka [23]. It takes Faure's sequence (see [4], [1]) to which it applies linear scrambling. All scrambled Faure's sequences have low discrepancy.

3. The experiments

3.1 The integrands.

For the purpose of testing the numerical integration using lattices, we have chosen the following five functions with various properties (continuous and discontinuous, smooth and non-smooth etc.):

GenzCont: Continuous function with discontinuous derivative according to Genz's set of testing functions [5]:

$$f_1(\vec{x}) = f_1(x_1, \ldots, x_d) = e^{-\sum_{k=1}^{d} c_k |x_k - w_k|},$$

where $c_k = \frac{2}{d}$ and $w_k = 0.4 + 0.4222\frac{k}{d}$ for every k.

GenzDiscont: Discontinuous function from the same Genz's set:

$$f_2(\vec{x}) = f_2(x_1, \ldots, x_d) = e^{-\sum_{k=1}^{d} c_k x_k} \quad \text{for} \quad x_1 \geq \mu_1 \quad \text{or} \quad x_2 \geq \mu_2.$$

This function is zero anywhere else. Again, we have chosen $c_k = \frac{2}{d}$ for every k, and $\mu_1 = 0.7$ and $\mu_2 = 0.3$.

L2NormTru: Let us consider the d-dimensional ball with radius $r = \sqrt{\frac{d}{6}}$ and center in $\vec{w} = (w_1, \ldots, w_d)$. Inside this ball, the function value is r. Outside, the function value is the distance of the argument from the center of the ball:

$$f_3(\vec{x}) = f_3(x_1, \ldots, x_d) = \max(r, \sqrt{s}),$$

and $s = \sum_{k=1}^{d}(x_k - w_k)^2$, where $w_k = 0.4 + 0.4222\frac{k}{d}$ for every k, again.

RandPoly: As a representative of smooth functions, we have chosen sparse polynomial in d variables with $5d$ random terms of degree 10 with random factors:

$$f_4(\vec{x}) = f_4(x_1, \dots, x_d) = \sum_{i=1}^{5d} a_i \prod_{k=1}^{10} x_{p_{i,k}},$$

where a_i is a random number from $(0, 1)$ for every i, and each $p_{i,k}$ is a random integer between 1 and d.

NiedAbs: Simple continuous non-smooth function from an example of Davis and Rabinowitz [3] (originally published by Roos and Arnold in [17]):

$$f_5(\vec{x}) = f_5(x_1, \dots, x_d) = \prod_{k=1}^{d} |4x_i - 2|.$$

3.2 Implementation notes.

The experiments ran on several computers under Unix and Linux, respectively. The program was written in C++. We have taken the code used in [11] to which we have added the algorithms described above, namely generating of the basis, its scaling, and constructing the set for numerical integration using lattices. As the pseudo-random number generator we used the algorithm `CombLS2` published in Tezuka's book [23].

Linear programming was done using the simplex algorithm, namely its implementation `LPsimp 2.6` (slightly rewritten to our needs), which was developed at the Operations Research Laboratory at the Seoul National University.

3.3 The results.

The following figures summarize the results of the experiments. We have decided for representation similar to the one in [11]. On every page, there are diagrams concerning one of the five integrands mentioned above. (Because f_1 and f_2 behaved similarly, we omitted the figure for f_1.) Five rows correspond to five dimensions and three columns correspond to three various desired cardinalities of the set M. Each of these diagrams contains seven columns corresponding to the seven approximating methods used:

`L`: lattices with low discrepancy according to Skriganov (the basis was generated as in Theorem 2),

`B`: lattices with basis consisting of random vectors from the unit ball,

`S`: lattices with basis consisting of random unit vectors,

`F`: linearly scrambled Faure (according to [11]),

`M`: the traditional Monte Carlo method,

`H`: the set given by Halton's sequence with omitting the first 100 elements, R: the set given by Richtmyer's sequence.

The long horizontal line shows the exact value of the integral. On the vertical axis, two other values are marked to illustrate the scale. The longer horizontal line in every column shows the value approximated by the corresponding method. In the first five cases, it is the median from thirty attempts, which are marked by the short horizontal lines. Because they are often too close to each other, they can be hard to distinguish. Nevertheless, one can make a sufficient idea about their distribution. Finally, the vertical line to the left of every one of the five non-deterministic method marks the standard deviation σ. More precisely, the line stretches from $\mu - \sigma$ to $\mu + \sigma$, where μ is the average.

If we compare the lattice-based methods with the other methods, we will observe the following facts. First, lattices cannot compete with Linear Scrambled Faure method, because this one beats the other methods almost every time. But as for the traditional Monte Carlo method, the situation is different here. We can see that lattices are very often superior to this method. Moreover, it seems (at least for lower dimensions), that this dominance becomes more clear with growing number of testing points, although this is less apparent in higher dimensions. The function f_5 was the hardest to integrate for all the methods. On the other hand, we obtained the best results for f_3.

If we try to compare the lattices generated by various types of bases, we will find out easily that the low-discrepancy bases really give the best results. Random bases from unit ball and bases consisting of random unit vectors are not so good and they both have approximately the same behavior. It seems that the former prevail somewhat, although the situation is exactly reversed for the function f_3.

All these observations can be expressed quantitatively in the following way. In lower dimensions, the Linear Scrambled Faure method is twice as good as the low-discrepancy lattices, which are twice as good as the lattices with random bases, which are twice as good as traditional Monte Carlo method. In higher dimensions, this ratio changes to $5 : 1.5 : 1 : 1$, save for f_3 as well as f_5, where all the methods behave more or less the same.

As for variance of the attempts attained by the randomized methods, we can say almost exactly the same as we just did for the final results (the medians). The worst performance of the Monte Carlo method is still more significant here. If we compare the standard deviations of these first five methods with the error of the last two deterministic methods, we can say that for f_2 (and f_1), Halton and Richtmyer are usually better than lattices but worse than Linear Scrambled Faure. They are approximately as good as low-discrepancy lattices for f_3 and sometimes even worse for the polynomial f_4. When integrating f_5, the deterministic methods stand out for small n in higher dimensions, primarily.

At last, let us say a few words about the time requirements of the methods. Because all the computations were done on differently effective computers, and often ran as processes with low priority, we were not able to do an exact timing,

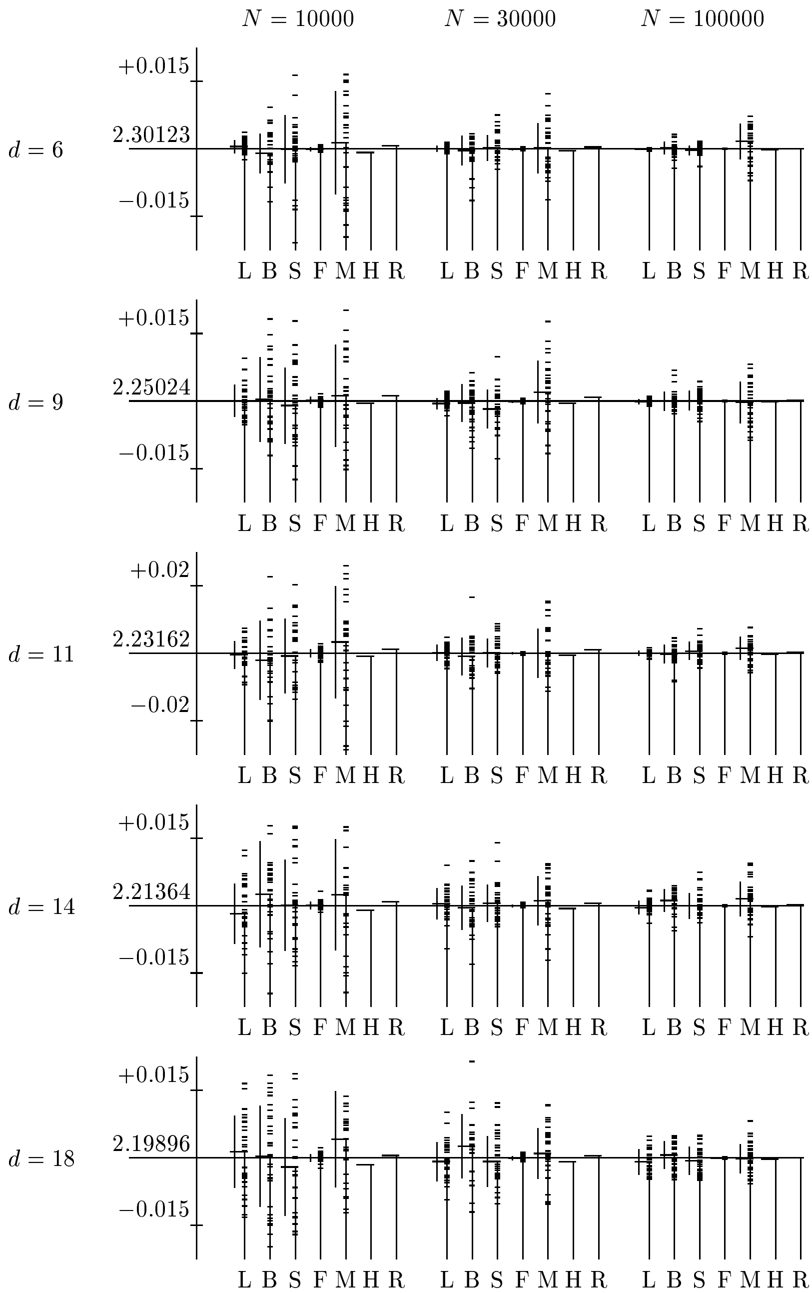

FIGURE 2. Numerical integration of the function f_2 (GenzDiscont).

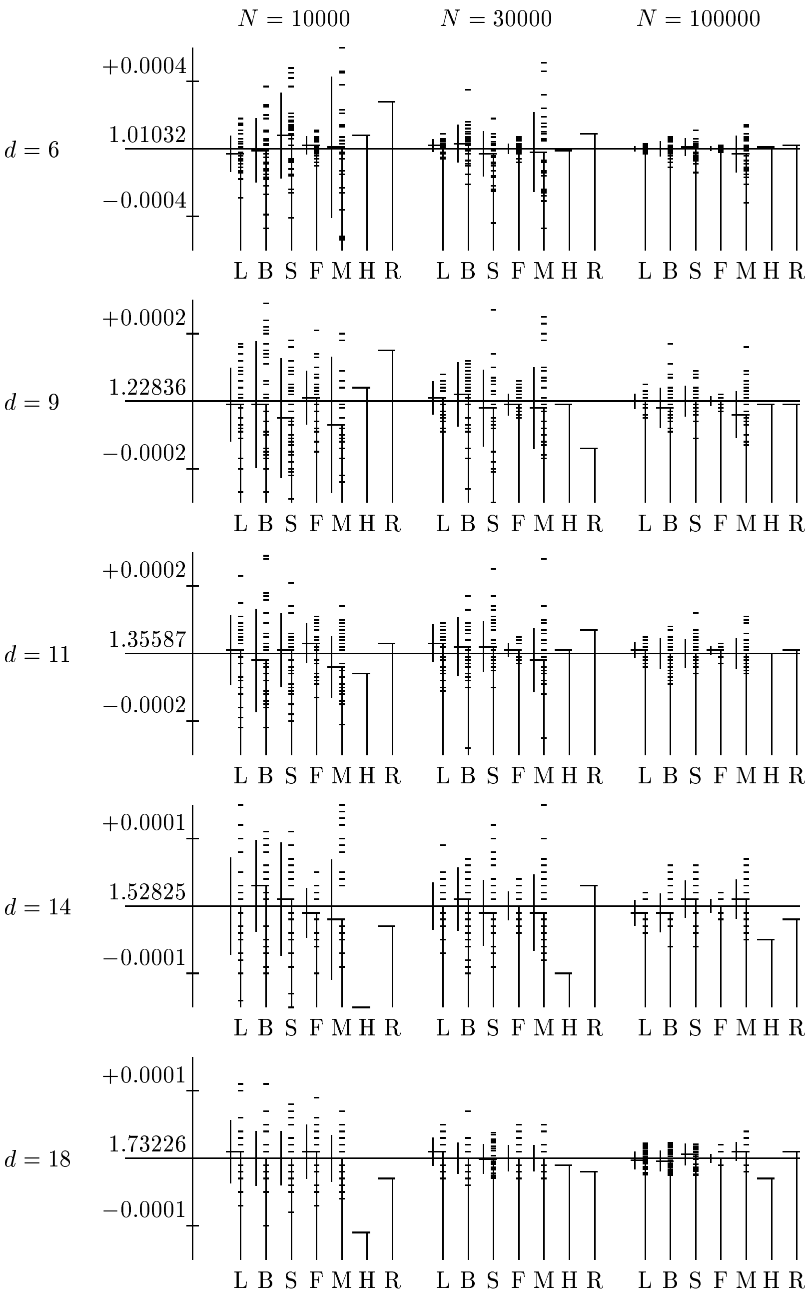

FIGURE 3. Numerical integration of the function f_3 (L2NormTru).

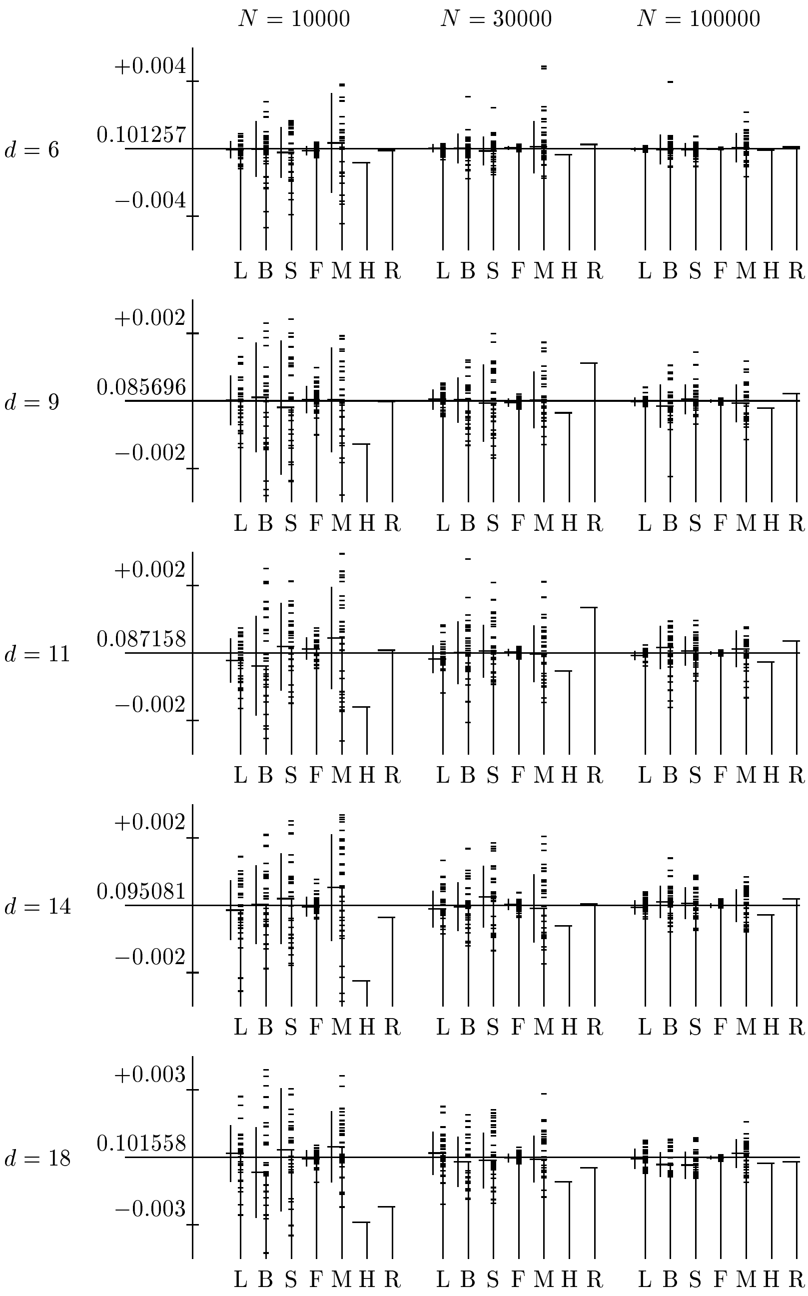

FIGURE 4. Numerical integration of the function f_4 (RandPoly).

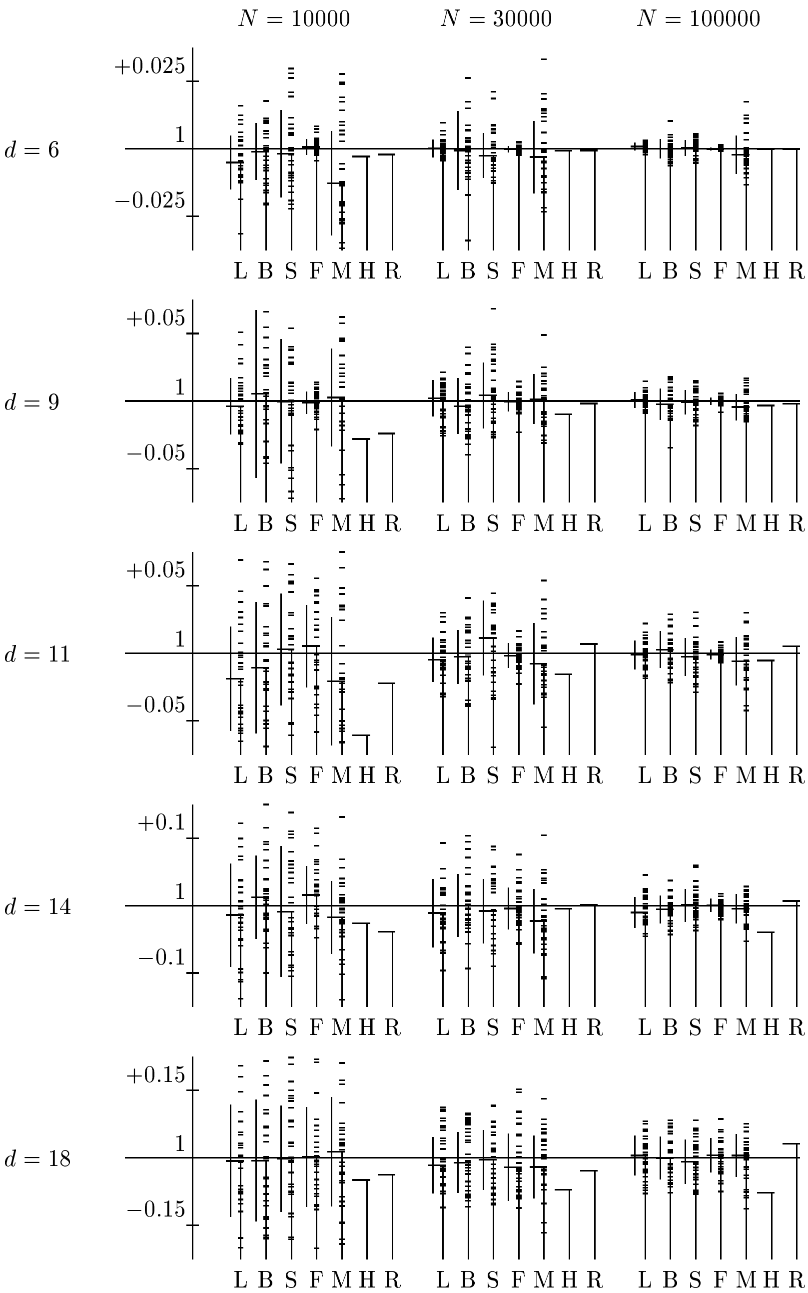

FIGURE 5. Numerical integration of the function f_5 (NiedAbs).

which could help us to make some really objective conclusions. But some reference attempts on a standalone machine suggested that in lower dimensions the lattices could be faster than the successful Linear Scrambled Faure method, but with growing d they slow down rapidly, for $d = 10$ they consume more or less the same time, and the higher dimension, the slower they are. The Monte Carlo, Halton and Richtmyer methods are always very fast, naturally.

The choice of 18 as the highest dimension considered was determined by the practical limitations given by numerical integration using lattices. For example, for $n = 10^5$ and $d = 15$ more than one million linear programming tasks need to be computed to obtain one result, for $d = 20$ even more than $5 \cdot 10^6$ which on the computer used took about one day. (Nevertheless, this machine was very old.) In dimension 18, one computation took, on the average, about two hours for $n = 10^5$ and about thirty minutes for $n = 10^4$. (Note that these timings are only approximate because of the technical limitations mentioned above.) It follows that it does not help too much to speed up the integration by reducing the set M.

4. Conclusion

Based on the known theory of lattices, we have developed an algorithm which generates low-discrepancy sets. We have used these sets for numerical integration with quasi-Monte Carlo method. The practical experiments suggest that our lattice-based method is a reasonable alternative to the other methods, especially in lower dimensions. The following topics deserve further examination.

First, we can try to integrate using lattices with bases different from those described here. Suitable and interesting candidates might be e.g. random orthogonal bases from the unit ball or randomly rotated orthonormal bases.

As for the speed of the computations performed, linear programming is the most time-consuming. We have used a very simple implementation of the simplex algorithm. The application of a more sophisticated method would surely bring some speedup. We can, for example, take advantage of the fact that the linear programming tasks are often very similar: the boundary conditions are always given by the unit cube and we often explore parallel subspaces, which means that we solve the same problem as before, differing only in the right-hand side etc.

Niederreiter in [12] presents some facts on *good lattice points* concerning integration of periodic functions. One should try to integrate such functions also using the method studied here. For the similarity to good lattice points, promising results might be expected as well. Another interesting functions might be those with variable effective dimension (see, e.g., Radovic et al. [16]).

As we have already mentioned in the introduction, some comparison of our method with lattice rules (see, e.g., [19], [20], [21], [8], [12], [6]) would be desirable, too.

Acknowledgment. I would like to thank Jiří Matoušek for his invaluable help with the design of the work and with preparation of this paper.

References

[1] Beck J., Chen W.L., *Irregularities of Distribution*, Cambridge University Press, Cambridge, 1987.

[2] Cranley R., Patterson T.N.L., *Randomization of number theoretic methods for multiple integration*, SIAM J. Numer. Anal. **13** (1976), no. 6, 909–914.

[3] Davis P.J., Rabinowitz P., *Methods of Numerical Integration*, 2nd edition, Academic Press Inc., Orlando FL, 1984.

[4] Faure H., *Discrepancy of sequences associated with a number system (in dimension s)*, Acta Arith. **41** (1982), no. 4, 337–351.

[5] Genz A., *Testing multidimensional integration routines*, in Tools, Methods and Languages for Scientific and Engineering Computation, B. Ford, J. C. Rault and F. Thomasset, Eds., North-Holland, Amsterdam, 1984.

[6] Hua L.K., Wang Y., *Applications of Number Theory to Numerical Analysis*, Springer, Berlin, 1981.

[7] Janse van Rensburg E.J., Torrie G.M., *Estimation of multidimensional integrals: is Monte Carlo the best method?*, J. Phys. A **26** (1993), 943–953.

[8] L'Ecuyer P., Lemieux C., *Variance reduction via lattice rules*, in Management Science **49-6** (2000), 1214–1235.

[9] Lovász L., *An algorithmic theory of numbers, graphs and convexity*, CBMS-NSF Regional Conference Series in Applied Mathematics 50, SIAM, Philadelphia, Pennsylvania, 1986.

[10] Matoušek J., *Geometric Discrepancy: An Illustrated Guide*, Springer, Berlin, 1999.

[11] Matoušek J., *On the L_2-discrepancy for anchored boxes*, J. Complexity **14** (1998), 527–556.

[12] Niederreiter H., *Random number generation and quasi-Monte Carlo methods*, CBMS-NSF Regional Conference Series in Applied Mathematics 63, SIAM, Philadelphia, Pennsylvania, 1992.

[13] Owen A.B., *Randomly permuted (t, m, s)-nets and (t, s)-sequences*, in Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, Harald Niederreiter and Peter Jau-Shyong Shiue, Eds., Springer, New York, 1995, pp. 299–317.

[14] Owen A.B., *Scrambled net variance for integrals of smooth functions*, Ann. Statist. **25** (1997), no. 4, 1541–1562.

[15] Owen A.B., *Monte-Carlo variance of scrambled net quadrature*, SIAM J. Numer. Anal. **34** (1997), no. 5, 1884–1910.

[16] Radovic I., Sobol' I.M., Tichy R.F., *Quasi-Monte Carlo methods for numerical integration: Comparison of different low-discrepancy sequences*, Monte Carlo Methods Appl. **2** (1996), 1–14.

[17] Roos P., Arnold L., *Numerische Experimente zur mehrdimensionalen Quadratur*, Österreich. Akad. Wiss. Math.-Natur. Kl. S.-B. II **172** (1963), 271–286.

[18] Skriganov M.M., *Constructions of uniform distributions in terms of geometry of numbers*, Algebra i Analiz **6** (1994), 200–230.

[19] Sloan I.H., Joe S., *Lattice Method for Multiple Integration*, Clarendon Press, Oxford University Press, New York, 1994.

[20] Sloan I.H., Kuo F.Y., Joe S., *Constructing randomly shifted lattice rules in weighted Sobolev spaces*, SIAM J. Numer. Anal. **40** (2002), 1650–1665.

[21] Sloan I.H., Kuo F.Y., Joe S., *On the step-by-step construction of quasi-Monte Carlo integration rules that achieve strong tractability error bounds in weighted Sobolev spaces*, Math. Comp. **71** (2002), 1609–1640.

[22] Spanier J., Maize E.H., *Quasi-random methods for estimating integrals using relatively small samples*, SIAM Review **36** (1994), no. 1, 18–44.
[23] Tezuka S., *Uniform Random Numbers. Theory and Practice*, Kluwer Academic Publishers, Dordrecht, 1995.

Charles University, Faculty of Mathematics and Physics, Department of Applied Mathematics, Malostranské nám. 25, 118 00 Prague 1, Czech Republic

E-mail: vojta@kam.mff.cuni.cz

(*Received* December 22, 2007, *revised* April 10, 2008)