Low-discrepancy Lattice Sets and QMC Integration

Vojtěch Franěk

Department of Applied Mathematics

Charles University Prague

Czech Republic

vojta@kam.ms.mff.cuni.cz

Postal address:

Vojtěch Franěk

Pekárenská 2

141 00 Praha 4

Czech Republic

Abstract

Many low-discrepancy sets are suitable for quasi-Monte Carlo integration. Skriganov showed that the intersections of suitable lattices with unit cube have low discrepancy. We introduce adaptive algorithm based on linear programming which scales any given lattice appropriately and computes its intersection with 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 of the sets. We conclude that up to dimensions about 15, low-discrepancy lattices yield fairly good results. In higher dimensions, the computation of the intersection takes too long and ceases to be feasible.

Keywords: QMC integration, lattices, discrepancy

1 Introduction

Higher-dimensional integrals are often numerically approximated using Monte Carlo or quasi-Monte Carlo methods; the exact value[1]

$$\int_{I^d} f(x)\,\mathrm{d}x.$$

is approximated by the average

$$\frac{1}{N}\sum_{\vec{x}\in M} f(\vec{x}),$$

where M is a suitable N-point set. In the Monte Carlo method, this set is generated independently and uniformly at random, while in quasi-Monte Carlo methods it is constructed using some semi-random or even purely determinisic rule. Well known examples are the Faure set, (t, m, d)-nets and randomly scrambled modifications of these sets. (For more information, see [9], [12], [10], [11] or [8]). These constructions were found as examples of sets with low discrepancy. Some known error estimates, such as the Koksma-Hlawka inequality, and many numerical tests ([5], [10], [11], [15]) show that in many cases, the quasi-Monte Carlo methods using such sets outperform the Monte-Carlo method significantly.

Skriganov in [14] proved very strong upper bounds on the discrepancy of sets generated as the intersection of suitable lattices with I^d. We are aware of no tests of these sets concerning numerical integration. In particular, it is not obvious whether they can be generated efficiently, since computational

[1] In this paper, we consider integration only over the unit cube, that is over the set $I^d = [0,1]^d$. Integration over this region is one of the most important and most common.

problems with lattices in higher dimensions are often known to be hard (see e. g. Lovász [6]). Here we suggest one method of generating such sets and we test its practical efficiency as well as the accuracy of quasi-Monte Carlo integration using these sets: in the dimensions where the generation proved feasible, we test the error of numerical integration for some test functions and we compare the results with some more traditional quasi-Monte Carlo methods.

We found 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 doesn't affect the total computing time too much. As for the quality of the approximation, our lattice sets are about twice as good as the crude Monte-Carlo method (in lower dimensions even better). In our experiments, however, they cannot compete with some others, especially linear scrambled Faure sets. Still, there are some special classes of functions (e. g. periodic functions, which we have not tested), where our lattices may have far better behavior.[2]

2 Lattices

As was mentioned in the introduction, we will generate point sets using lattices, particularly lattices with low discrepancy. Discrepancy can be shortly characterized as the measure of non-uniformity of distribution. It has proved

[2]This conjecture is motivated by the results (presented by Niederreiter) on *good lattice points*, which are in many aspects similar to our lattices.

to be a good measure of sutability of a set for numerical integration. Here we briefly recall the dicrepancy for axis-parallel boxes, which we use throughout this paper.

First, let us consider an axis-parallel box $R = [a_1, b_1) \times [a_2, b_2) \times \ldots \times [a_d, b_d) \subseteq I^d$. We denote its volume by $\mathrm{vol}(R)$. If all the N points of the set M are uniformly distributed in the unit cube, one can expect that the intersection $M \cap R$ contains about $N \cdot \mathrm{vol}(R)$ points. The discrepancy of R is therefore defined as

$$D(M, R) = N \cdot \mathrm{vol}(R) - |M \cap R|.$$

Let $\mathcal{R}$ denote the set of all axis-parallel boxes in the unit cube I^d. Then the number

$$D(M) = \sup_{R \in \mathcal{R}} |D(M, R)|$$

is called the *discrepancy* of M for axis-parallel boxes.

A *lattice* L in $\mathbb{R}^d$ is the set of all integer linear combinations of a basis of the space $\mathbb{R}^d$:

$$L = L(B) = L\left(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d\right) = \left\{ \sum_{j=1}^{d} i_j \vec{b}_j : i_1, \ldots, i_d \in \mathbb{Z} \right\}.$$

The set of linearly independent vectors $B = \left\{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d \in \mathbb{R}^d\right\}$ is called a *basis* of the lattice L. One lattice has many different bases.

The *norm* of a lattice L is defined as

$$\mathrm{Nm}(L) = \inf_{\vec{x} \in L \setminus \{0\}} |x_1 x_2 \ldots x_d|,$$

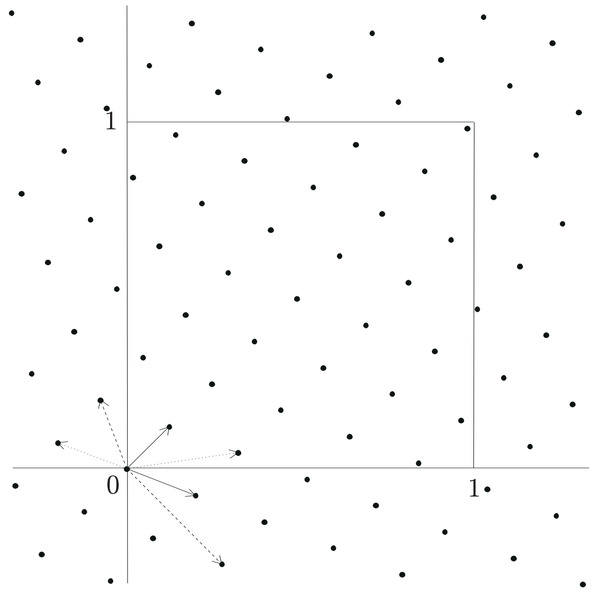

Figure 1: Three different bases of one lattice ($d = 2$).

where $\vec{x} = (x_1, x_2, \ldots, x_d)$. 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 [14] 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 any real number) has discrepancy*[3] $D(\frac{1}{t}L) = O(\log^{d-1} t)$. *(If* $\det(L) \neq 1$, *we can rescale* L *suitably.)*

Skriganov also proposes one class of lattices with positive norm. They have the basis vectors of the form

$$\vec{b}_1 = (1, 1, \ldots, 1), \vec{b}_2 = (\alpha_1, \alpha_2, \ldots, \alpha_d), \vec{b}_3 = (\alpha_1^2, \alpha_2^2, \ldots, \alpha_d^2), \ldots,$$

$$\ldots, \vec{b}_i = (\alpha_1^{i-1}, \alpha_2^{i-1}, \ldots, \alpha_d^{i-1}), \ldots, \vec{b}_d = (\alpha_1^{d-1}, \alpha_2^{d-1}, \ldots, \alpha_d^{d-1}),$$

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 [7]:

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 (obviously) monic.*

[3] If we talk about the discrepancy of a lattice L, we always mean the disrepancy of its intersection with unit cube.

In our work, we tested the lattices with exactly these bases (suitably rescaled). To check the theoretical results of suitability of these low-discrepancy lattices for numerical integration, we compared them with the lattices with following bases:

Random vectors from unit ball $B^d = \left\{ \vec{b} \in \mathbb{R}^d : \left\| \vec{b} \right\| \leq 1 \right\}$.[4] We can obtain a vector of the basis if we generate random point $\vec{b} \in [-1, 1]^d$ over and over again, until it is an element of unit ball, ie. until $\left\| \vec{b} \right\| \leq 1$. Thus we ensure that the basis vectors have uniform random distribution in the ball B^d as desired. Some troubles can arise in higher dimensions where most vectors do not fulfil the condition $\left\| \vec{b} \right\| \leq 1$. But for the small dimensions we work with (up to $d = 20$) the generation of the basis still does not take too much time and we can be satisfied with this algorithm.

Random unit vectors, i. e. $\left\| \vec{b}_1 \right\| = \left\| \vec{b}_2 \right\| = \ldots = \left\| \vec{b}_d \right\| = 1$.[4] Obviously, we can proceed in the same way as we did in the preceding case with the only difference that we normalize all the vectors that met our condition.

Our task is to find, as efficiently as possible, a lattice whose intersection M with 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

[4] Of course, we cannot take arbitrary random vectors. To form a basis, they must fulfil the condition of linear independency. But if we use a good pseudo-random number generator, the resulting vectors are independent with probability near 1.

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 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 yielded the basis B_0, we first 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[5]
:

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

In our computations, we never needed more than three scalings.

2.2 The Algorithm

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

[5]This was necessary in our work only, 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.

the lattice within the unit cube, formally

$$\vec{0} \leq \sum_{k=1}^{d} x_k \vec{b}_k \leq \vec{1},$$

where every x_i is an integer.

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, till 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 coefficient 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.

To be more precise, we describe the algorithm formally:

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

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 $a_{f+1}, a_{f+2}, \ldots, a_d$ are the unknown variables) and the objective function $g(a_{f+1}, a_{f+2}, \ldots, a_d) = a_{f+1}$. 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

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

If $x_{f+1}^{min} > x_{f+1}^{max}$, then go to **Update** (in this case, the intersection of the subspace S_f with 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 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 still is not the smallest possible. 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 others 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 is 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 Some small improvements

If we analyse the behavior of our algorithm, we soon find out that it is preferable to have the basis vectors sorted descendingly by their length. Figure 2 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,

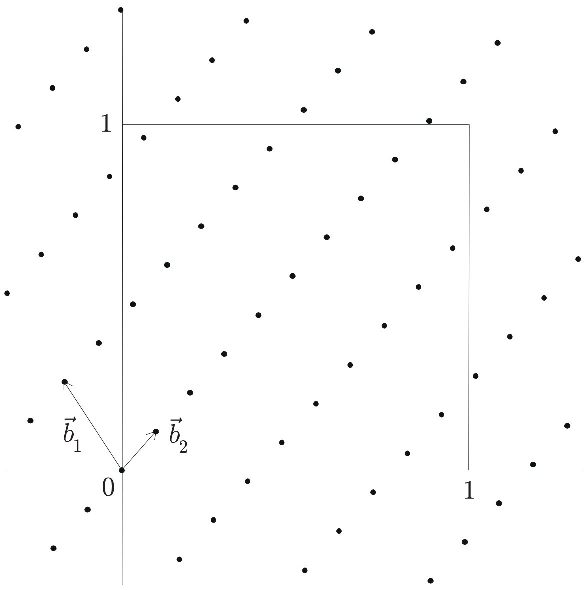

Figure 2: The advantage of decreasing basis.

the less $(d-i)$-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 the preferable one.

The second improvement is just at the level of implementation, though it speeds up the algorithm even more significantly than the preceding sorting of the basis. It suffices to realize that finding the extremes for x_d has by no means to be done with linear programming; it is much easier just to cut the line S_{d-1} according to the unit cube. We get the same 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. Because this is a significant point for many common functions and the behavior of the function in its neighborhood can be very different from the course in the other parts of the unit cube, it tends to influence the approximation adversely. It is better not to put it in the set M. (But we cannot just to omit it; this would corrupt the regularity of the lattice, which would result in a worse approximation.)

We do this by a little trick: we translate the whole lattice by some vector $\vec{t}$. We reformulate the lattice definition so that it is given not only by its basis, but with the location of its origin as well:

$$L = L(B,\vec{t}) = \left\{ \vec{t} + \sum_{j=1}^{d} i_j \vec{b}_j : i_1, i_2, \ldots, i_d \in \mathbb{Z} \right\}, \vec{t} \in \mathbb{R}^d,$$

where $B = \vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d \in \mathbb{R}^d$ is the basis of $\mathbb{R}^d$. The original lattice can be expressed as $L(B,\vec{0})$.[6] If we take for $\vec{t}$ some random quantity (in our

[6] Note that this trick has the same both motivation and nature as omitting the beginning of Halton's sequence in the next paragraph.

implementation it was a pseudorandom vector from the unit cube), we bring some indeterminism even to the lattices which were fixed so far, namely to the lattices with the basis given deterministically. Thus, we can generate more mutually independent sets M given by one basis, and for the final value of the approximation of the integral we 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.[7]

Monte Carlo: The points of the set M are generated purely at random. To make this method more stable, we use more (about 30) independently generated sets and the final value is taken as the median of the results obtained using these sets.[8]

Halton's sequence: Let $2 = p_1 < p_2 < \ldots < p_d$ be the d smallest primes. For an integer i and some index $k \in 1, \ldots, d$, let $i = (c_m c_{m-1} \ldots c_1)_{p_k}$ be the representation of i in the radix p_k:

$$i = \sum_{j=1}^{m} c_j p_k^{j-1}.$$

Then the k-th component of the i-th element of Halton's sequence is

$$x_{i,k} = (0.c_1 c_2 \ldots c_m)_{p_k} = \sum_{j=1}^{m} c_j p_k^{-j}.$$

[7]More can be found, for example, in the study [8].

[8]In fact, this would prefer the stochastic methods to the strictly deterministic ones. To see how reliable the results are, one should look rather at the standard deviation than at the median.

It is known that if the set M is created by the first N elements of Halton's sequence, then it has low discrepancy and thus is suitable for numerical integration. The first several points can cause some distortion due to their special placement (for example, the element $\vec{x}_1$ lies near the origin). This can be succesfully avoided by omitting several (say 100) of the first elements of the sequence:

$$M = \vec{x}_{101}, \vec{x}_{102}, \ldots, \vec{x}_{100+N}.$$

Richtmyer's sequence: Again, 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 it has low discrepancy. Nevertheless, it is very popular due to its really 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.

The fourth and the last method described here is called *Linear Scrambled Faure*. This method is based on the ideas of Owen [12] and Tezuka [16], and it is described more thoroughly in [8]. It takes Faure's sequence (see [3], [1]) to which it applies linear scrambling. Both Faure's and 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 following five functions with various properties (continuous and discontinuous, smooth and non-smooth etc.):

GenzCont: Continuous function with discontinuous derivation according to Genz's set of testing functions [4]:

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

We have chosen the constants in exponent as follows: $c_k = \frac{2}{d}$ and $w_k = 0.4 + 0.4222\frac{k}{d}$ for every k.

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

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

This function is zero anywhere else (for $x_1 < \mu_1$ and $x_2 < \mu_2$). 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. The radius is chosen so that the volume of the ball is comparable to the volume of the unit cube.

This function is directionally homogeneous, ie. it is not related to the coordinate system in any way.

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:

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

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: It is a simple continuous non-smooth function from an example of Davis and Rabinowitz [2], which was originally published by Roos and Arnold [13]:

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

3.2 Other Parameters of the Tests

All the experiments with lattices described in the following were executed with decreasing basis and with the origin of the lattice randomly shifted within the unit cube. We used the adaptive algorithm with linear programming.

We compared our lattice-based method with the traditional Monte Carlo method and with the easy methods according to Richtmyer and to Halton.

Among the more sophisticated methods, we used for comparison the method Linear Scrambled Faure which produced very good and stable results. All the final values are medians of thirty attempts (except for Halton's and Richtmyer's methods, of course: they are strictly deterministic and thus done only once). To ensure some stability of the computations, we approximated using the set M containing 10^5 points. For comparison, we have also the results for 10^4 and $3 \cdot 10^4$ points. Note that the sets generated by the lattices did not have exactly these numbers of points. We allowed the number varying at most 0.1% of these values. For our purposes, this measurement affects the resutls negligibly, but it speeds up the computation cosiderably, because the first approximation when scaling the basis was always within this tolerance.

Because very good numerical integration methods are known for small dimensions, which our lattices cannot compete with, we were interested in results in higher dimensions. As the representatives, we have chosen the dimensions 6, 9, 11, 14 and 18.

3.3 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 [8] 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 [16].

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. (Detailed information about this software can be found at the web pages of the laboratory: `http://orlab.snu.ac.kr/software/or1.html`.)

3.4 The Results

The following figures summarize the results of the experiments. We have decided for representation similar to the one in [8]. On every page, there are diagrams concerning one of the five integrands mentioned above. Five rows correspond to five dimensions and three columns correspond to three various desired cardinalities of the set M. Each of these diagrams contains of 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:` Linear Scrambled Faure (according to [8])

`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. It is the cost for retaining the scale the same in the whole row. 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.

The figures clearly illustrate two main trends: the quality of the approximation of the integral using lattices grows with the number of points in the set M and decreases with growing dimension. (The only significant exception is the function f_3 where the approximation improves with growing dimension. This abnormality can be easily explained by the fact that the position of the ball changes with growing dimension so that the numerical integration becomes easier.) Naturally, this could be expected and the other methods behave similarly.

If we compare the lattice-based methods with the other methods, we observe several more interesting and not so obvious facts. First, lattices cannot compete with Linear Scrambled Faure method, because this one beats the other methods almost everytime. But as for the traditional Monte Carlo method, the situation is different here. We can see that lattices are

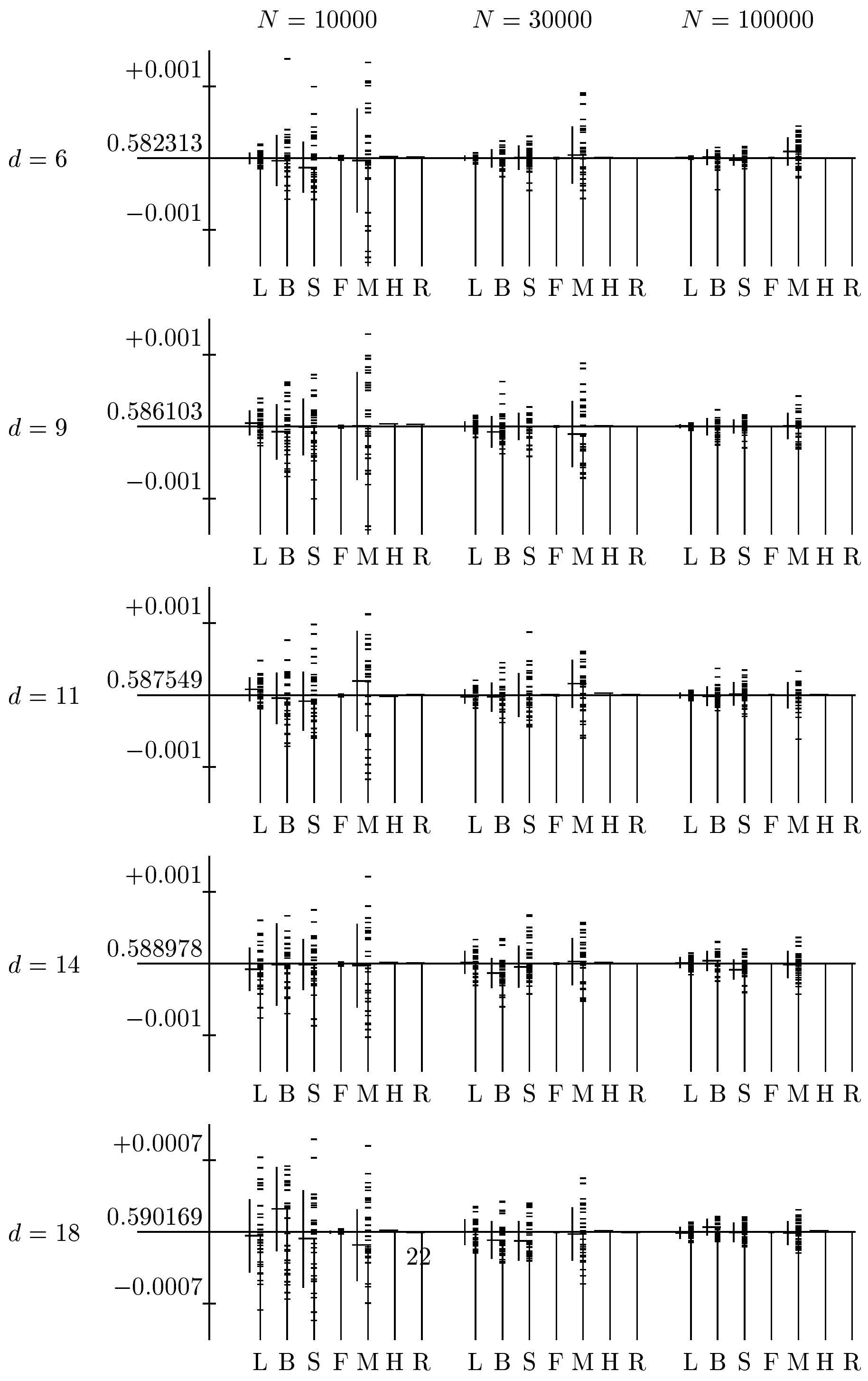

Figure 3: Numerical integration of the function f_1 (GenzCont).

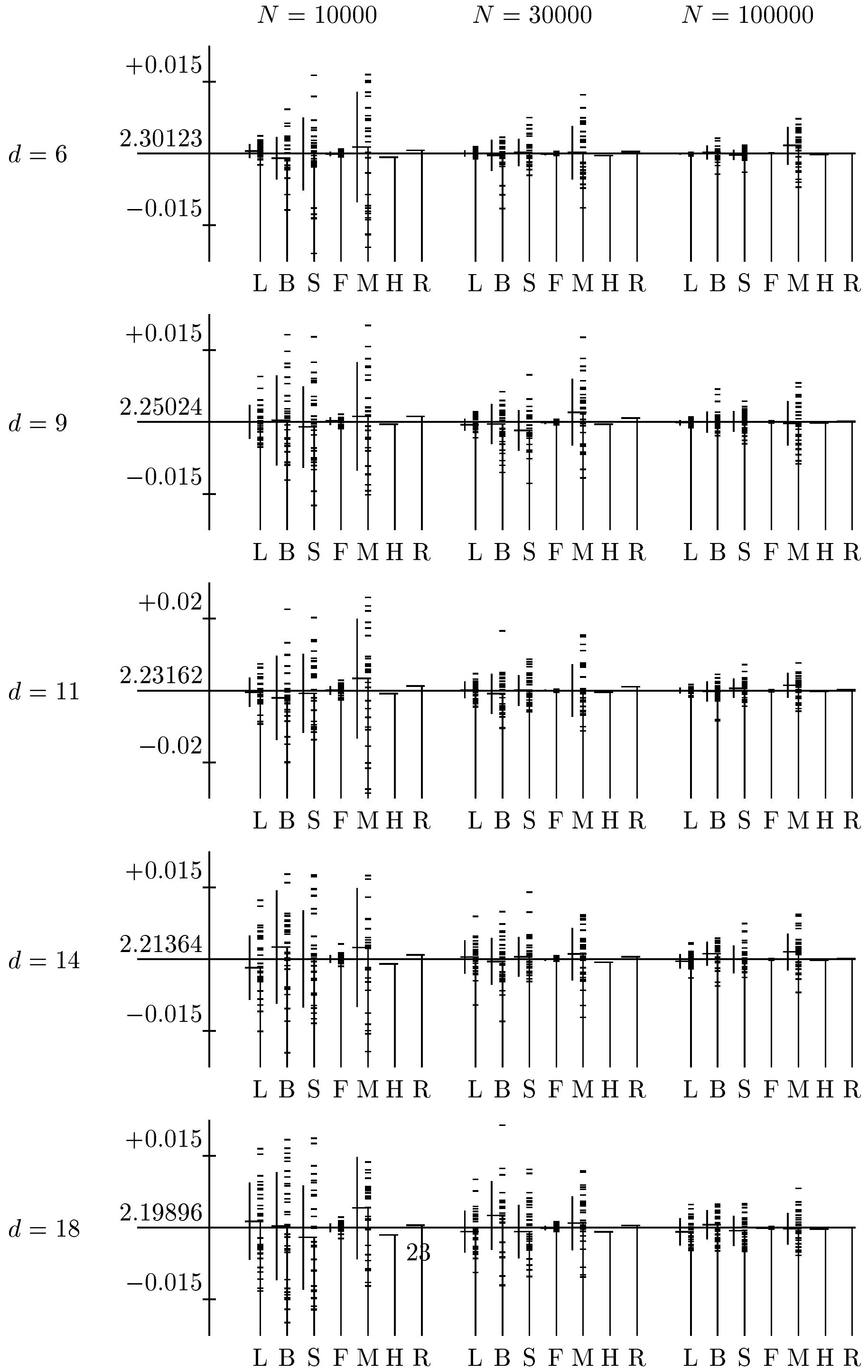

Figure 4: Numerical integration of the function f_2 (GenzDiscont).

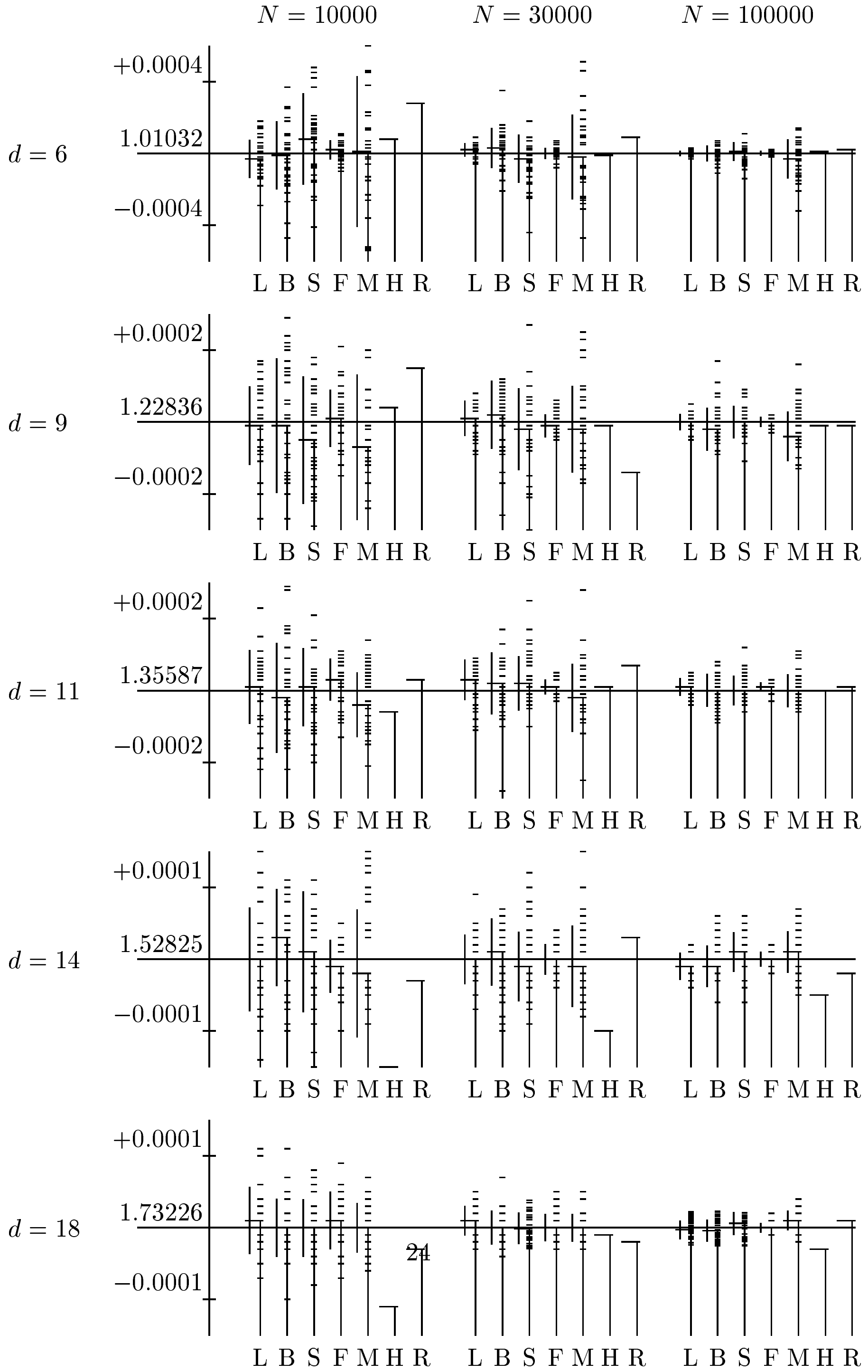

Figure 5: Numerical integration of the function f_3 (L2NormTru).

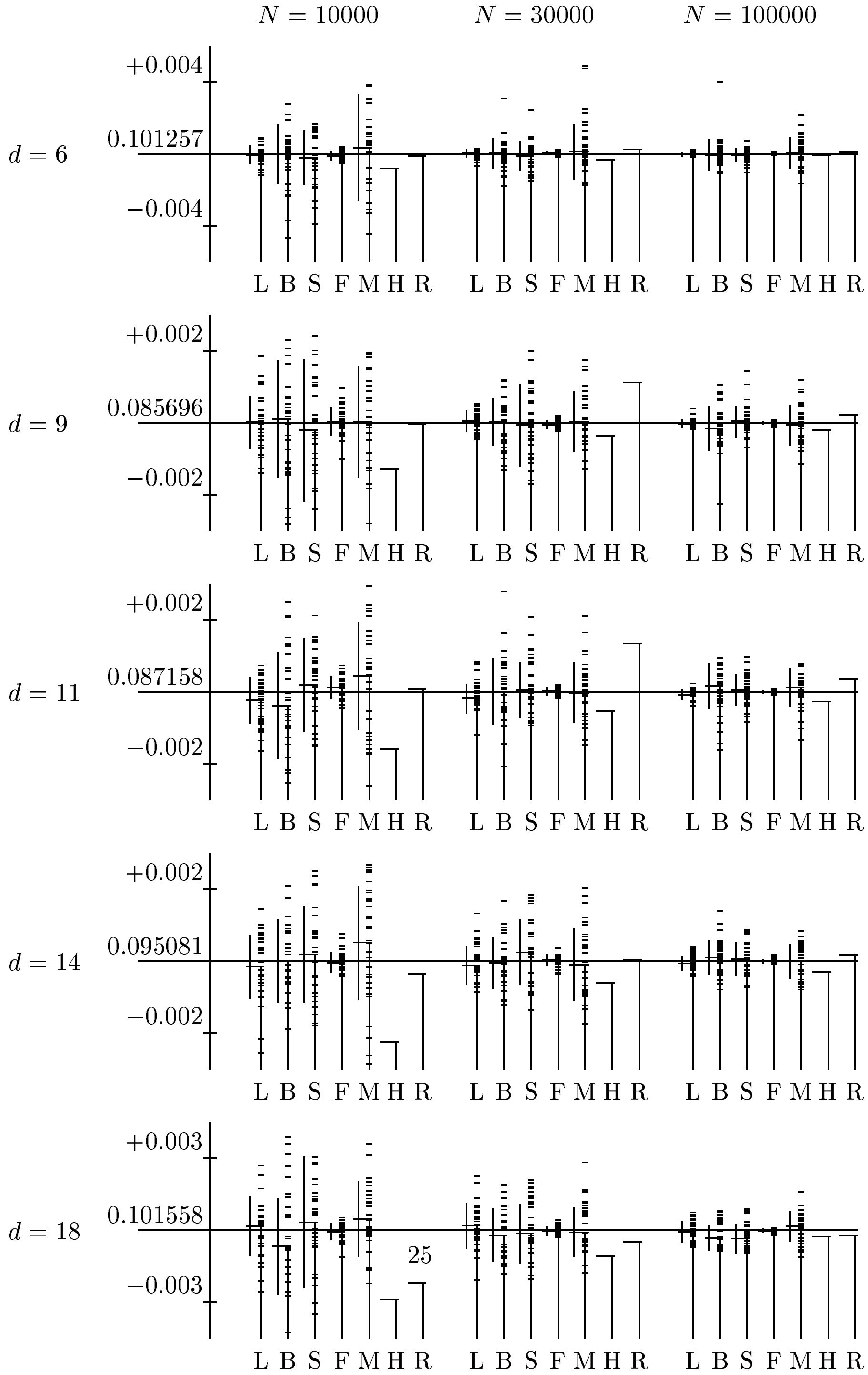

Figure 6: Numerical integration of the function f_4 (RandPoly).

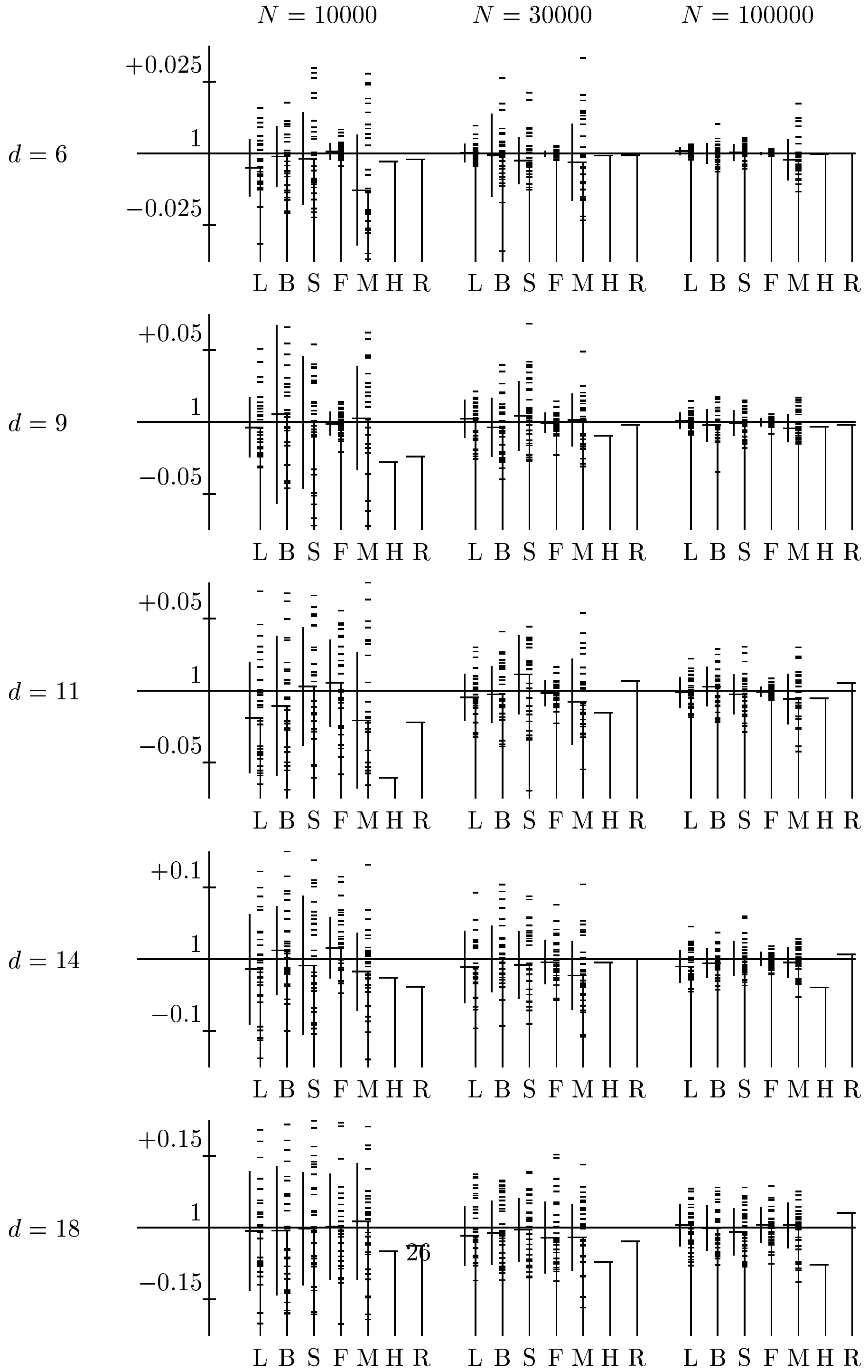

Figure 7: Numerical integration of the function f_5 (NiedAbs).

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 when integrating f_3.

If we try to compare the lattices generated by various types of basis, we easily find 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 prevails somewhat, although the situation is exactly reversed for the function f_3.

All these observations can be quantitatively expressed 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_1 and f_2 are Halton and Richtmyer 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 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, 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 succesful 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. 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 (and so lose precision).

4 Conclusion

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

First, we can try to integrate using lattices with bases different from those described here. Suitable and interesting candidates might surely 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-consumpting. We have used a very simple implementation of the simplex algorithm. The application of some more sophisticated method would bring some speedup surely. 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.

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

References

[1] Beck J., Chen W. L. (1987): *Irregularities of Distribution.* Cambridge Univ. Press.

[2] Davis P. J., Rabinowitz P. (1984): *Methods of Numerical Integration, 2nd Ed.* Academic Press Inc., Orlando FL.

[3] Faure H. (1982): Discrepancy of sequences associated with a number system (in dimension s). *Acta Arith.* 41, number 4, pp. 337–351.

[4] Genz A. (1984): Testing Multidimensional Integration Routines. *Tools, Methods and Languages for Scientific and Engineering Computation.* B. Ford, J. C. Rault and F. Thomasset, eds., North-Holland, Amsterdam.

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

[6] Lovász L. (1986): *An Algorithmic Theory of Numbers, Graphs and Convexity.* CBMS-NSF Regional Conference Series in Applied Mathematics 50, SIAM, Philadelphia, Pennsylvania.

[7] Matoušek J. (1999): *Geometric discrepancy: An Illustrated Guide.* Springer-Verlag, Berlin etc.

[8] Matoušek J. (1998): On the L2-Discrepancy for Anchored Boxes. *Journal of Complexity* 14, pp. 527–556.

[9] Niederreiter H. (1992): *Random Number Generation and Quasi-Monte Carlo Methods.* CBMS-NSF Regional Conference Series in Applied Mathematics 63, SIAM, Philadelphia, Pennsylvania.

[10] Owen A. B. (1995): 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 Editors, pp. 299–317. Springer-Verlag, New York.

[11] Owen A. B. (1997): Scrambled Net Variance for Integrals of Smooth Functions. *Annals of Statistics* volume 25, number 4, pp. 1541–1562.

[12] Owen A. B. (1997): Monte-Carlo variance of scrambled net quadrature. *SIAM J. Numer. Anal.* 34, number 5, pp. 1884–1910.

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

[14] Skriganov M. M. (1994): Construction of uniform distributions in terms of geometry of numbers. *Algebra i Analiz* 6, pp. 200–23.

[15] Spanier J., Maize E. H. (1994): Quasi-Random Methods for Estimating Integrals Using Relatively Small Samples. *SIAM Review* Vol. 36, No. 1, pp. 18–44.

[16] Tezuka S. (1995): *Uniform Random Numbers. Theory and Practice.* Kluwer Academic, Dordrecht.