A Parallel Quasi-Monte Carlo Method for Solving Systems of Linear Equations

Michael Mascagni[1], and Aneta Karaivanova[1,2]

[1] Department of Computer Science, Florida State University,
203 Love Building, Tallahassee, FL 32306-4530, **USA**,
mascagni@cs.fsu.edu,
URL: http://www.cs.fsu.edu/∼mascagni
[2] Central Laboratory for Parallel Processing, Bulgarian Academy of Sciences,
Acad. G. Bonchev St.,bl. 25 A, 1113, Sofia, Bulgaria,
aneta@csit.fsu.edu,
URL: http://copern.bas.bg/∼anet

Abstract. This paper presents a parallel quasi-Monte Carlo method for solving general sparse systems of linear algebraic equations. In our parallel implementation we use disjoint contiguous blocks of quasirandom numbers extracted from a given quasirandom sequence for each processor. In this case, the increased speed does not come at the cost of less thrust-worthy answers. Similar results have been reported in the quasi-Monte Carlo literature for parallel versions of computing extremal eigenvalues [8] and integrals [9]. But the problem considered here is more complicated - our algorithm not only uses an $s-$dimensional quasirandom sequence, but also its $k-$dimensional projections ($k = 1, 2, \ldots, s-1$) onto the coordinate axes. We also present numerical results. In these test examples of matrix equations, the martrices are sparse, randomly generated with condition numbers less than 100, so that each corresponding Neumann series is rapidly convergent. Thus we use quasirandom sequences with dimension less than 10.

1 Introduction

The need to solve systems of linear algebraic equations arises frequently in scientific and engineering applications, with the solution being useful either by itself or as an intermediate step in solving a larger problem. In practical problems, the order, n, may in many cases be large (100 - 1000) or very large (many tens or hundreds of thousands). The **cost** of a numerical procedure is clearly an important consideration — so too is the **accuracy** of the method.

Let us consider a system of linear algebraic equations

$$Ax = b, \tag{1}$$

⋆ Supported by the U.S. Army Research Office under Contract # DAAD19-01-1-0675

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2330, pp. 598–608, 2002.

where $A = \{a_{ij}\}_{i,j=1}^{n} \in I\!R^{n\times n}$ is a given matrix, and $b = (b_1, \ldots, b_n)^t \in I\!R^n$ is a given vector. It is well known (see, for example, [3, 6]) that the solution, x, $x \in R^n$, when it exists, can be found using

- *direct methods*, such as Gaussian elimination, and LU and Cholesky decomposition, taking $O(n^3)$ time;
- *stationary iterative methods*, such as the Jacobi, Gauss- Seidel, and various relaxation techniques, which reduce the system to the form

$$x = Lx + f, \tag{2}$$

  and then apply iterations as follows

$$x^{(0)} = f,\ x^{(k)} = Lx^{(k-1)} + f,\ , k = 1, 2, \ldots \tag{3}$$

  until desired accuracy is achieved; this takes $O(n^2)$ time per iteration.
- *Monte Carlo* methods (MC) use independent random walks to give an approximation to the truncated sum (3)

$$x^{(l)} = \sum_{k=0}^{l} L^k f,$$

  taking time $O(n)$ (to find n components of the solution) per random step.

Keeing in mind that the convergence rate of MC is $O(N^{-1/2})$, where N is the number of random walks, millions of random steps are typically needed to achieve acceptible accuracy. The description of the MC method used for linear systems can be found in [1], [5], [10]. Different improvements have been proposed, for example, including sequential MC techniques [6], resolvent-based MC methods [4], etc., and have been successfully implemented to reduce the number of random steps. In this paper we study the quasi-Monte Carlo (QMC) approach to solve linear systems with an emphasis on the parallel implementation of the corresponding algorithm. The use of quasirandom sequences improves the accuracy of the method and preserves its traditionally good parallel efficiency.

The paper is organized as follows: §2 gives the background - MC for linear systems and a brief description of the quasirandom sequences we use. §3 describes parallel strategies, §4 presents some numerical results and §5 presents conclusions and ideas for future work.

2 Background

2.1 Monte Carlo for linear systems - very briefly

We can solve problem (1) in the form (2) with the scheme (3) if the eigenvalues of L lie within the unit circle, [3]. Then the approximate solution is the truncated Neumann series:

$$x^{(k)} = f + Lf + L^2 f + \ldots + L^{(k-1)} f + L^k f, \;\; k > 0 \tag{4}$$

with a truncation error of $x^{(k)} - x = L^k(f - x)$.
We consider the MC numerical algorithm and its parallel realization for the following two problems:

(a) Evaluating the inner product

$$J(h) = (h, x) = \sum_{i=1}^{n} h_i x_i \tag{5}$$

of the unknown solution $x \in I\!R^n$ of the linear algebraic system (2) and a given vector $h = (h_1, \ldots, h_n)^t \in I\!R^n$.

(b) Finding one or more components of the solution vector. This is a special case of **(a)** with the vector h chosen to be $h = e(r) = (0, ..., 0, 1, 0, ..., 0)$ where the one is on the r-th place if we want to compute the r-th component.

To solve this problem via a MC method (MCM) (see, for example, [10]) one has to construct a random process with mean equal to the solution of the desired problem. Consider a Markov chain with n states:

$$k_0 \to k_1 \to \ldots \to k_i \to \ldots, \tag{6}$$

with $k_j = 1, 2, \ldots, n$, for $j = 1, 2, \ldots$, and rules for constructing: $P(k_0 = \alpha) = p_\alpha, P(k_j = \beta | k_{j-1} = \alpha) = p_{\alpha\beta}$ where p_α is the probability that the chain starts in state α and $p_{\alpha\beta}$ is the transition probability from state α to state β . Probabilities $p_{\alpha\beta}$ define a transition matrix P. The normalizing conditions are: $\sum_{\alpha=1}^{n} p_\alpha = 1$, $\sum_{\beta=1}^{n} p_{\alpha\beta} = 1$ for any $\alpha = 1, 2, ..., n,$, with $p_\alpha \geq 0$, $p_\alpha > 0$ if $h(\alpha) \neq 0$, $p_{\alpha\beta} \geq 0$, $p_{\alpha\beta} > 0$ if $a_{\alpha\beta} \neq 0$. Define the weights on this Markov chain:

$$W_j = \frac{a_{k_0 k_1} a_{k_1 k_2} \ldots a_{k_{j-1} k_j}}{p_{k_0 k_1} p_{k_1 k_2} \ldots a_{k_{j-1} k_j}} \tag{7}$$

or using the requrence $W_j = W_{j-1} \frac{a_{k_{j-1} k_j}}{p_{k_{j-1} k_j}}$, $W_0 = 1$.
The following random variable defined on the above described Markov chain

$$\Theta = \frac{h(k_0)}{p_{k_0}} \sum_{j=1}^{\infty} W_j f(k_j) \tag{8}$$

has the property

$$E[\Theta] = (h, f),$$

To compute $E[\Theta]$ we simulate N random walks (6), for each walk we compute the random variable Θ, (8), whose value on the sth walk is $[\Theta]_s$, and take the averaged value:

$$E[\Theta] \approx \frac{1}{N} \sum_{s=1}^{N} [\Theta]_s.$$

Each random walk is finite - we use a suitable stoping criterion to terminate the chain.

2.2 Quasirandom sequences

Consider an s-dimensional quasirandom sequence with elements $X_i = (x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(s)})$, $i = 1, 2, \ldots$, and a measure of its deviation from uniformity, the *Star discrepancy*:

$$D_N^* = D_N^*(x_1, \ldots, x_N) = \sup_{E \subset U^s} \left| \frac{\#\{x_n \in E\}}{N} - m(E) \right|,$$

where $U^s = [0,1)^s$. We are using the Sobol', Halton and Faure sequences, which can be very briefly defined as follows.

Let the representation of n, a natural number, in base b be

$$n = \ldots a_3(n) a_2(n) a_1(n), \qquad n > 0, n \in I\!R.$$

Then a one-dimensional quasirandom number sequence (the Van der Corput sequence) is defined as a radical inverse sequence

$$\phi_b(n) = \sum_{i=0}^{\infty} a_{i+1}(n) b^{-(i+1)}, \text{ where } n = \sum_{i=0}^{\infty} a_{i+1}(n) b^i,$$

and has star descrepancy $D_N^* = O\left(\frac{logN}{N}\right)$.

In our tests, we use the following multidimensional quasirandom number sequences:

Halton sequence:

$X_n = (\phi_{b_1}(n), \phi_{b_2}(n), \ldots, \phi_{b_s}(n))$, where the bases b_i are pairwise relatively prime.

Faure sequence:

$$x_n^{(k)} = \begin{cases} \sum_{i=0}^{\infty} a_{i+1}(n) q^{-(i+1)}, & k = 1 \\ \sum_{j=0}^{\infty} c_{j+1} q^{-(j+1)}, & k \geq 2 \end{cases},$$

where

$$c_j = \left[\sum_{i \geq j} (k-1)^{i-j} \frac{i!}{(i-j)!j!} a_i(n) \right] (mod\ q), \quad j \geq 1, \ q \text{ is a prime} (q \geq s \geq 2).$$

Sobol' sequence: $X_k \in \overline{\sigma}_i{}^{(k)}, k = 0, 1, 2, \ldots$, where $\overline{\sigma}_i{}^{(k)}$, $i \geq 1$ - set of permutations on every $2^k, k = 0, 1, 2, \ldots$ subsequent points of the Van der Corput sequence,

or in binary:

$$x_n^{(k)} = \bigoplus_{i \geq 0} a_{i+1}(n) v_i,$$

where v_i, $i = 1, \ldots, s$ is a set of direction numbers.

For the **Halton, Faure, Sobol'** sequences we have

$$D_N^* = O\left(\frac{log^s N}{N}\right).$$

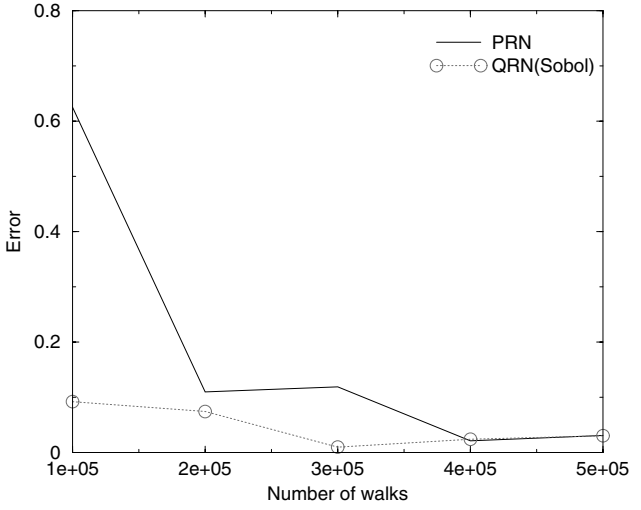

Fig. 1. Accuracy versus number of walks for computing (h, x), where x is the solution of a system with 2000 equations.

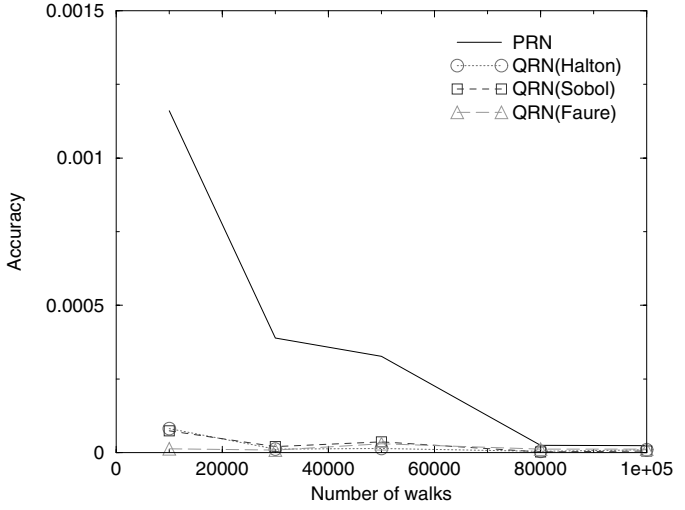

Fig. 2. Accuracy versus number of walks for computing one component, x_{64}, of the solution for a system with 1024 equations.

2.3 Convergence of the quasi-Monte Carlo Method

The MCM for computing the scalar product (h, x) consists of simulating N random walks (6), computing the value of the random variable Θ, (8), for each walk, and then average these values. In quasi-Monte Carlo we generate walks that are in fact not random, but have in some sense better distribution properties in the space of all walks on the matrix elements. The main difference in generating random and quasi-random walks is that we use l single pseudorandom numbers (PRNs) for a random walk of length l, but we use an l-dimensional sequence for the quasirandom walks of length l.
Computing the scalar product, $h^T A^i, f$ is equivalent to computing an $(i+1)$-dimensional integral, and we analyze it with bounds from numerical integration [8]. We do not know A^i explicitly, but we do know A, and we use quasirandom walks on the elements of the matrix to compute approximately $h^T A^i f$. Using an $(i+1)$-dimensional quasirandom sequence, for N walks, $[k_0, k_1, \ldots, k_i]_s$, $s = 1, \ldots, N$ we obtain the following error bound [8]:

$$\left| h^T A^i f - \frac{1}{N} \sum_{s=1}^{N} \left[\frac{g_{k_0}}{p_{k_0}} W_i f_{k_i} \right]_s \right| \leq C_1(A, h, f)\, D_N^*,$$

where W_i is defined in (7), and $[z]_s$ is the value of z on the s-th walk.
This gives us

$$\left| (h, x) - (h, x^{(k)}) \right| \leq C_2(A, h, f)\, k\, D_N^*.$$

Here D_N^* has order $O((log^k N)/N)$. Remember that the order of the mean square error for the analogous Monte Carlo method is $O(N^{-1/2})$. Figures 1 and 2 illustrate the accuracy versus the number of walks for computing the scalar product (h, x) (h is a given vector with 1 and 0, randomly chosen, and x is the solution of a system with 2000 equations), and for computing one component of the solution of a system with 1024 equations.

3 Parallel strategies

A well known advantage of the MCM is the efficiency by which it can be parallelized. Different processors generate independent random walks, and obtain their own MC estimates. These "individual" MC estimates are then combined to produce the final MC estimate. Such a scheme gives linear speed-up. However, certain variance reduction techniques usually require periodic communication between the processors; this results in more accurate estimates, at the expense of a loss in parallel efficiency. In our implementation this does not happen as we consider preparing the transition probabilities matrix as a preprocessing computation - which makes sense when we plan to solve the same system many times with different rigth-hand side vectors f.

In our parallel implementations we use disjoint contiguous blocks of quasirandom numbers extracted from a given quasirandom sequence for respective processors,

[9]. In this case, the increased speed does not come at the cost of less thrustworthy answers. We have previously used this parallelization technique for the eigenvalue problem, [8], but the current problem is more complicated. In the eigenvalue problem we need to compute only $h^T A^k h$ using a $k-$dimensional quasirandom sequence, while here we must compute $\sum_{i=1}^{s} h^T A^i f$ using an s-dimensional sequence and its k-dimensional projections for $k = 1, 2, \ldots, s$.

We solved linear systems with general very sparse matrices stored in "sparse row-wise format". This scheme requires 1 real and 2 integer arrays per matrix and has proved to be very convenient for several important operations such as the addition, multiplication, transposition and permutation of sparse matrices. It is also suitable for deterministic, direct and iterative, solution methods. This scheme permits us to store the entire matrix on each processor, and thus, each processor can generate the random walks independently of the other processors.

4 Numerical examples

We performed parallel computations to empirically examine the parallel efficiency of the quasi-MCM. The parallel numerical tests were performed on a Compaq Alpha parallel cluster with 8 DS10 processors each running at 466 megahertz and using MPI to provide the parallel calls.

We have carried out two types of numerical experiments. First, we considered the case when one only component of the solution vector is desired. In this case each processor generates N/p independent walks. At the end, the host processor collects the results of all realizations and computes the desired value. The computational time does not include the time for the initial loading of the matrix because we imagine our problem as a part of larger problem (for example, solving the same matrix equation for diffrent right-hand-side vectors) and assume that every processor independently obtains the matrix. Second, we consider computing the inner product (h, x), where h is a given vector, and x is the unknown solution vector.

During a single random step we use an l-dimensional point of the chosen quasirandom sequence and its k-dimensional projections ($k = 1, \ldots, l-1$) onto the coordinate axes. Thus we compute all iterations of $h^T A^k f$ ($1 \leq k \leq l$) using a single l-dimensional quasirandom sequence. The number of iterations needed can be determined using suitable stoping criterion: for example, checking how "close" two consecuent iterations $h^T A^{k-1} f$ and $h^T A^k f$ are, or by using a fixed number, l, on the basis of *a priori* analysis of the Neumann series convergence. In our numerical tests we use a fixed l - we have a random number of iterations per random step but the first l iterations (which are the most important) are quasirandom, and the rest are pseudorandom.

In all cases the test matrices are sparse and are stored in "sparse row-wise-format". We show the results for two matrix equations at size 1024 and 2000. The average number of non-zero elements per matrix row is $d = 57$ for $n = 1024$

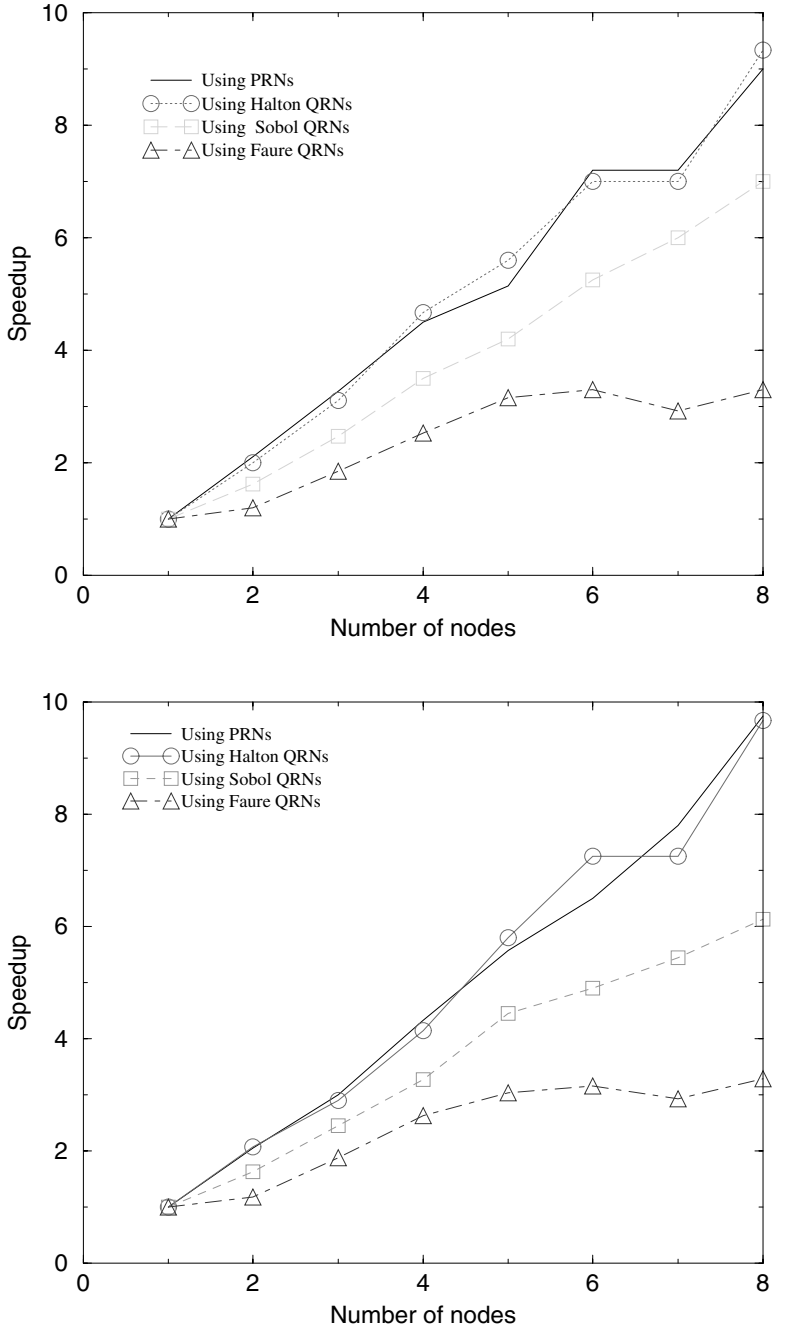

Fig. 3. Speedup when solving linear system with 1024 and 2000 equations using PRNs, Halton, Soboĺ and Faure sequences.

and $d = 56$ for $n = 2000$. For illustration, we present the results for finding one component of the solution using PRNs and Halton, Sobol' and Faure quasirandom sequences. The computed value, the run times and speedup are given in Tables 1 and 2. The use of quasirandom sequences (all kinds) improves the convergence rate of the method. For example, the exact value of the 54th component of the solution of a linear system with 2000 equations is 1.000, while we computed 1.0000008 (using Halton), 0.999997 (using Sobo/'l), 0.999993 (using Faure), while the average computed value using PRNs is 0.999950. This means that the error using the Halton sequence is 250 times smaller than the error using PRNs (Sobol' gives an error 17 times less). Moreover, the quasi-MC realization with Halton gives the smallest runing times - much smaller than a single PRN run. The results for average run time (single run) and efficiency are given in the Tables 1 and 2, the graphs for parallel efficiency are shown on Figures 1 and 2.

The results confirm that the high parallel efficiency of Monte Carlo methods is preserved with QRNs and our technique for this problem.

Table 1. Sparse system with 1024 equations: MPI times, parallel efficiency and the estimated value of one component of the solution using PRNs, Halton, Sobol' and Faure QRNs.

	1*pr.*	2*pr.*	3*pr.*	4*pr.*	5*pr.*	6*pr.*	7*pr.*	8*pr.*
MCM_{pseudo}								
Time(s)	36	17	11	8	7	5	5	4
Efficiency		1.05	1.09	1.125	1.02	1.2	1.05	1.125
Appr.x(54)	1.000031	1.000055	.999963	.999966	1.000024	1.000029	.999979	.999990
QMC_{Halton}								
Time(s)	28	14	9	6	5	4	4	3
Efficiency		1	1.03	1.16	1.12	1.16	1	1.16
Appr.x(54)	0.999981	0.999981	0.999981	0.999981	0.999981	0.999981	0.999981	0.999981
QMC_{Sobol}								
Time(s)	42	25	17	12	10	8	7	6
Efficiency		0.84	0.82	0.875	0.84	0.875	0.857	0.875
Appr.x(54)	0.999983	0.999983	0.999983	0.999983	0.999983	0.999983	0.999983	0.999983
QMC_{Faure}								
Time(s)	76	63	41	30	24	23	26	23
Efficiency		0.6	0.62	0.63	0.63	0.55	0.42	0.41
Appr.x(54)	0.999919	0.999919	0.999919	0.999919	0.999919	0.999919	0.999919	0.999919

Table 2. Sparse system with 2000 equations: MPI times, parallel efficiency and the estimated value of one component of the solution using PRNs, Halton, Sobol and Faure QRNs.

	$1pr.$	$2pr.$	$3pr.$	$4pr.$	$5pr.$	$6pr.$	$7pr.$	$8pr.$
MCM_{pseudo}								
Time(s)	39	19	13	9	7	6	5	4
Efficiency		1.02	1	1.08	1.11	1.08	1.11	1.21
x(54)	1.000077	1.000008	.999951	.999838	.999999	.999917	1.000044	.999802
QMC_{Halton}								
Time(s)	29	14	10	7	5	4	4	3
Efficiency		1.03	0.97	1.03	1.16	1.21	1.03	1.21
x(54)	1.0000008	1.0000008	1.0000008	1.0000008	1.0000008	1.0000008	1.0000008	1.0000008
QMC_{Sobol}								
Time(s)	49	30	20	15	11	10	9	8
Efficiency		0.82	0.82	0.82	0.89	0.82	0.78	0.76
x(54)	0.999997	0.999997	0.999997	0.999997	0.999997	0.999997	0.999997	
QMC_{Faure}								
Time(s)	79	67	42	30	26	25	27	24
Efficiency		0.59	0.63	0.66	0.61	0.53	0.42	0.41
x(54)	0.999993	0.999993	0.999993	0.999993	0.999993	0.999993	0.999993	0.999993

5 Future work

We plan to study this quasi-MCM for matrix equations with more slowly convergent Neumann series solution, using random walks with longer (and different for different walks) length. In this case, the dimension of of quasirandom sequence is random (depends on the used stoping criterion). Our preliminary numerical tests for solving such problems showed the effectivness of randomized quasirandom sequences, but we have yet to study the parallel behavior of this method in this case.

References

1. J. H. Curtiss, Monte Carlo methods for the iteration of linear operators, *Journal of Mathematical Physics*, **32**, 1954, pp. 209-323.
2. B. Fox, *Strategies for quasi-Monte Carlo*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1999.
3. G. H. Golub, C.F. Van Loon, *Matrix computations*, **The Johns Hopkins Univ. Press**, Baltimore, 1996.
4. Dimov I., V. Alexandrov, A. Karaivanova, Resolvent Monte Carlo Methods for Linear Algebra Problems, *Mathematics and Computers in Simulations*, Vol. **. 55**, 2001, pp. 25-36.

5. J.M. Hammersley, D.C. Handscomb, *Monte Carlo methods*, **John Wiley & Sons, inc.**, New York, London, Sydney, Methuen, 1964.
6. J.H. Halton, Sequential Monte Carlo Techniques for the Solution of Linear Systems, *SIAM Journal of Scientific Computing*, Vol.**9**, pp. 213-257, 1994.
7. Mascagni M., A. Karaivanova, Matrix Computations Using Quasirandom Sequences, *Lecture Notes in Computer Science*, (Wulkov, Yalamov, Wasniewsky Eds.), Vol.**1988**, Springer, 2001, pp. 552-559.
8. Mascagni M., A. Karaivanova, A Parallel Quasi-Monte Carlo Method for Computing Extremal Eigenvalues, to appear in: *Lecture Notes in Statistics*, Springer.
9. Schmid W. Ch., A. Uhl, Parallel quasi-Monte Carlo integration using (t, s)-sequences, In: *Proceedings of ACPC'99* (P. Zinterhof et al., eds.), Lecture Notes in Computer Science, **1557**, Springer, 96-106.
10. Soboĺ, I. M., *Monte Carlo numerical methods*, Nauka, Moscow, 1973 (in Russian).