ISCTE IUL
REPOSITÓRIO
INSTITUTO UNIVERSITÁRIO DE LISBOA

Repositório ISCTE-IUL

Deposited in *Repositório ISCTE-IUL*:
2020-04-20

Deposited version:
Post-print

Peer-review status of attached file:
Peer-reviewed

Citation for published item:
Acebron, J. A. (2020). A probabilistic linear solver based on a multilevel Monte Carlo Method. Journal of Scientific Computing. 82 (3)

Further information on publisher's website:
10.1007/s10915-020-01168-2

Publisher's copyright statement:
This is the peer reviewed version of the following article: Acebron, J. A. (2020). A probabilistic linear solver based on a multilevel Monte Carlo Method. Journal of Scientific Computing. 82 (3), which has been published in final form at https://dx.doi.org/10.1007/s10915-020-01168-2. This article may be used for non-commercial purposes in accordance with the Publisher's Terms and Conditions for self-archiving.

Use policy

Creative Commons CC BY 4.0
The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in the Repository
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Serviços de Informação e Documentação, Instituto Universitário de Lisboa (ISCTE-IUL)
Av. das Forças Armadas, Edifício II, 1649-026 Lisboa Portugal
Phone: +(351) 217 903 024 | e-mail: administrador.repositorio@iscte-iul.pt
https://repositorio.iscte-iul.pt

A probabilistic linear solver based on a multilevel Monte Carlo method

Juan A. Acebrón[1,2],

[1] Dept. Information Science and Technology, ISCTE-University Institute of Lisbon, Portugal

[2] INESC-ID,Instituto Superior Técnico, Universidade de Lisboa, Portugal

*E-mail address:*juan.acebron@iscte-iul.pt

February 23, 2020

Abstract

We describe a new Monte Carlo method based on a multilevel method for computing the action of the resolvent matrix over a vector. The method is based on the numerical evaluation of the Laplace transform of the matrix exponential, which is computed efficiently using a multilevel Monte Carlo method. Essentially, it requires generating suitable random paths which evolve through the indices of the matrix according to the probability law of a continuous-time Markov chain governed by the associated Laplacian matrix. The convergence of the proposed multilevel method has been discussed, and several numerical examples were run to test the performance of the algorithm. These examples concern the computation of some metrics of interest in the analysis of complex networks, and the numerical solution of a boundary-value problem for an elliptic partial differential equation. In addition, the algorithm was conveniently parallelized, and the scalability analyzed and compared with the results of other existing Monte Carlo method for solving linear algebra systems.

Keywords— Multilevel, Monte Carlo method, network analysis, parallel algorithms, high performance computing

1 Introduction

Among the many numerical methods proposed in the literature for solving linear algebra problems, the probabilistic methods were often perceived within the linear algebra community more as a curiosity than a serious alternative to the state-of-the-art deterministic methods. Even though it is broadly accepted that they offer interesting features from a computational point of view, such as being easily parallelizable, fault-tolerant, and more suited to heterogeneous architectures (features of paramount importance in view of the current high performance computers). However, the truth is that they also exhibit some significant weakness, such as a very slow convergence to the solution. This made the underlying algorithms highly demanding computationally, especially when dealing with high accuracy solutions. In addition, they require a continuous close monitoring by the user in order to control the numerical errors, being therefore often necessary to repeat several times the simulations before accepting the solution within the accuracy prescribed by the user. Rather the deterministic methods, such as the iterative methods, are basically governed by an automatic procedure, being often only necessary to impose initially a certain tolerance that should be reached by the algorithm before stopping safely the execution. This tolerance should be enough (with the exception of some pathological problems) to guarantee the convergence of the solution to the prescribed accuracy. Some of the aforementioned issues affecting the probabilistic methods have been progressively improved over the years, although it may seem that the advance is often relatively modest, specially when compared with the advance experimented by the counterpart deterministic methods.

Historically, the idea of using probabilistic methods based on Monte Carlo simulations for solving linear algebra problems goes back to the pioneering work of von Neumann and Ulam during the 1940's [18]. Although initially the method was proposed merely for computing the inverse of a matrix, it was later generalized for solving linear algebra problems in a series of seminal papers, see [13, 14] e.g., and [12, 30] for further references. Briefly the underlying idea consists in generating a discrete Markov chain which evolves by random paths through the different indices of the matrix. Mathematically, the method can be seen in a way as a Monte Carlo sampling of the Neumann series of the matrix. The convergence of the method was rigorously established in [27], and improved further more recently (see for instance [15], and [8] just to cite a few references). More specifically, in [16, 8] an important step forward has been done in the applicability of the probabilistic methods for solving more realistic problems. The method called Monte Carlo synthetic acceleration method is in fact a kind of hybrid scheme which combines the Richardson iterative method along with a Monte Carlo method. Essentially the role played by the Monte Carlo method consists in accelerating the convergence of the underlying iterative method, and has been shown to be competitive enough for a class a problems, comparing even with perhaps one of the most widely used iterative method such as the GMRES method.

Another related area of application of the probabilistic methods where some significant progress has recently made is in the field of matrix functions [24, 25]. In fact, in the specific case of the action of a matrix exponential over a vector, it was proposed in [3, 4] a probabilistic method based on a multilevel Monte Carlo method [21], which as an important feature improves notably the typical slow convergence rate of any Monte Carlo method. The multilevel method has become in fact a widely used method for accelerating stochastic simulations in general (see the excellent review in [20] e.g., and [5] for a specific application to Markov Chains), and in particular for the matrix exponential has been shown in [4] that can be even competitive against the classical deterministic methods based on Krylov subspace methods. Specifically, for large scale problems and extremely large number of processors, the multilevel method clearly outperforms the deterministic method for solving problems consisting in large matrices, not only in terms of computational time, but also in terms of memory requirements. Another remarkable feature of the multilevel method is precisely to be a method that gives rise to automatic algorithms in the aforementioned sense. In fact, typically the only interaction of the user with the algorithm consists merely to set up initially the prescribed accuracy of the solution, and subsequently the algorithm is capable of reaching autonomously the desired goal.

The aim of this paper is precisely to apply such a multilevel method for the problem of computing the action of a resolvent matrix over a vector. This is done in practice exploiting the well-known connection existing between the resolvent matrix with the matrix exponential through the Laplace transform. The multilevel method derived in [4] for the case of the matrix exponential is conveniently adapted for this specific problem, as well as the convergence of the resulting method and computational cost of the underlying algorithm conveniently analyzed. Moreover, to test the performance of this new algorithm several numerical examples were run. Those concern the computation of the so-called Katz centrality, which describes some important features in complex networks, and the numerical solution of a linear system coming from the discretization of a boundary-value problems for an elliptic partial differential equation. Finally, another noteworthy contribution of the paper was to parallelize the resulting algorithm, and compare the scalability of the algorithm, when running both algorithms in a multicore architecture, with other available Monte Carlo methods. A key result from this comparison is the autonomous operation of our approach, that, unlike other available Monte Carlo methods [8], the proposed method does not require parameter tuning for optimal scaling.

The paper is organized as follows. The probabilistic representation of the vector solution is presented in Section 2 along with the description of the multilevel method for the resolvent of a matrix. Section 3 describes the implementation of the multilevel algorithm. The analysis of the numerical errors and convergence of the method is presented in Section 4, while Section 5 is devoted to the analysis of the computational cost of the algorithm. In addition, in Section 6 the algorithm is tested running a few numerical examples, and the parallel performance of the method is compared with the performance of a classical Monte Carlo method. Finally, this work is closed summarizing the high points of the paper and discussing potential directions for future research.

2 Mathematical description of the probabilistic method

In order to apply a multilevel method to any Monte Carlo method, it is first required to have a probabilistic representation of the solution. To this purpose, next we describe the probabilistic method adopted for representing the action of the resolvent of a matrix over a vector. Let $A = \{a_{ij}\}_{i,j=1}^{n}$ a given sparse n-by-n matrix, and v and x an n-dimensional vectors. Using the Laplace transform, it holds that

$$x = (s\,\mathbb{1} - A)^{-1}\,v = \left(\int_0^\infty dt\, e^{-st}\, e^{t\,A} \right)\, v, \tag{1}$$

provided $\|A\|_2 < s$. The integral above can be discretized using a suitable Newton-Cotes numerical quadrature, and after approximating the improper integral by a finite one we obtain

$$x \approx \left(\sum_{i=1}^{N} \omega_i e^{-st_i}\, e^{t_i\,A} \right)\, v = [\sum_{i=1}^{N} \omega_i \prod_{j=1}^{i-1} \left(e^{-s\Delta t}\, e^{\Delta t\,A} \right)]\, v, \tag{2}$$

where $t_i = (i-1)\Delta t$, $i = 1, \ldots, N$, $\Delta t = T/N$ the corresponding discretization step, and ω_i the suitable weights corresponding to the chosen quadrature rules. Both, the integral discretization as well as the truncation of the improper integral by replacing the infinite limit by a finite one, T, introduces two source of errors which they will be analyzed in Section 4.

Concerning the action of the exponential matrix $e^{\Delta t\,A}$ over the vector v, this can be computed probabilistically resorting to the representation introduced in [3], and it was conveniently generalized here to deal with more general classes of matrices. Essentially the main idea consists in decomposing the matrix A as $D - U$. D is a diagonal matrix with entries $d_{ij} = 0\ \forall i \neq j$, $d_{ii} = d_i = a_{ii} + L_{ii}, i = 1, \ldots, n$, and the matrix U with entries u_{ij} is given by

$$u_{ij} = \begin{cases} L_{ii}, & \text{if } i = j \\ (-1)^{\sigma_{ij}}\, L_{ij}, & \text{otherwise} \end{cases} \tag{3}$$

where $\sigma = \{\sigma_{ij}\}$ is a binary matrix with entries taking the value 1 when $a_{ij} < 0$, and 0 otherwise. Here $L = (L_{ij})$ denotes a generalized Laplacian matrix, defined in the broad sense as a matrix with nonpositive off-diagonal entries $L_{ij} = -|a_{ij}|$, and zero row sums, that is $L_{ii} = -\sum_{j\neq i} L_{ij}$. Note that this does not constitute any restriction in the class of matrices amenable to be represented probabilistically. Quite the contrary one can see that any arbitrary matrix can be straightforward decomposed in such a way. Such decomposition allows in practice to approximate the action of the matrix exponential over the vector using a suitable splitting method. In [3] it was used the Strang method in view of being of sufficiently high order, and even more important for not introducing any additional computational cost when compared with the lower order Lie splitting method.

As it happens for the probabilistic representation described in [6], for this new representation it can be used as well for both, computing a single entry of the vector solution, or the full vector solution. In the following for simplicity we described merely the probabilistic representation for computing a single entry of the vector solution, being the derivation of the other straightforward (see [4],e.g.). Therefore, the probabilistic representation for computing a single entry of the action of the resolvent matrix over the vector reads as follows

$$\bar{x}_i = \mathbf{E}[\sum_{k=1}^{N} \bar{\omega}_k \left(\prod_{j=1}^{k} \eta_j\right)], \tag{4}$$

where $\eta_j = \phi_j \, e^{\Delta t \, \bar{d}_{i_j}}$, $j = 1, \ldots, N-1$, $\eta_N = \phi_N \, e^{\Delta t \, \bar{d}_{i_N}/2} \, v_{i_N}$, $\bar{\omega}_k = e^{\Delta t \, \bar{d}_i/2} \omega_k$, and $\bar{d}_i = d_i - s$. Concerning ϕ_k, this corresponds to a two-point random variable taking values -1 and 1 with a probability related to the matrix σ. $Q = -L$ is the infinitesimal generator of a continuous-time Markov chain on the set $S = \{1, 2, \cdots, n\}$, being the matrix transition probability $P = (p_{ij})$ the solution of the Kolmogorov's backward equations,

$$P'(t) = Q \, P(t), \quad P(0) = \mathbb{1} \qquad (t \geq 0). \tag{5}$$

For each starting point i, the variables i_k, $k = 1, \ldots, N$, correspond to a sequence of N discrete random variables with outcomes on S, and probabilities $p_{i_{k-1} \, i_k}(t)$, $k = 2, \ldots, N$, and $p_{i \, i_1}(t)$ for $k = 1$, defined by $P(\Delta t)$ for each k. Note that $\bar{x}$ corresponds in fact to an approximation of the true solution x, being the true solution recovered in the limit $N \to \infty$ and $T \to \infty$. This representation can be interpreted intuitively as follows: Lets generate a random path which starts at the chosen entry i of the vector $\bar{x}$. This evolves according to a continuous-time Markov chain governed by the generator Q, moving therefore randomly from i to any possible state on S. We evaluate the N functions η_k using the random values obtained along the process, and accumulated the number obtained after multiplying all previous functions η_k weighted by the corresponding quadrature weight $\bar{\omega}_k$. Finally the solution is obtained through a suitable expected value.

2.1 A multilevel method for computing the resolvent matrix

As it was described in the literature (see the excellent survey in [20], and references therein), essentially the idea behind the so-called geometric multilevel Monte Carlo method (MLMC for short) consists in approximating the finest solution P_L obtained to the level of discretization L using a sequence of coarser approximations obtained at previous levels l, from l_0 to $L-1$. This in practice entails accelerating the Monte Carlo simulations, since for a fixed given accuracy, the method generates more samples for the coarsest level with low computational cost, and less samples for the higher levels with higher computational cost. This method can be conveniently adapted to this specific problem, assuming in particular that the different levels of discretization correspond to the value of Δt_l, being now $\Delta t_l = T/N_l$, with $N_l = 2^l$. Moreover, concerning the minimum level of discretization l_0, as it was already pointed out in [4], it should be chosen specifically different from zero. This is because the computational cost turns out to be almost independent of the level when simulating the coarsest level. The multilevel method can be expressed in mathematical form through the following telescoping series,

$$\bar{x}^L = \mathbf{E}[P_L] = \mathbf{E}[P_{l_0}] + \sum_{l=l_0+1}^{L} \mathbf{E}[P_l - P_{l-1}], \tag{6}$$

where $P_l = \sum_{k=1}^{N} \bar{\omega}_k \left(\prod_{j=1}^{k} \eta_j\right)$, and $\eta_j = \phi_j \, e^{\Delta t_l \, \bar{d}_{i_j}}, j = 1, \ldots, N-1, \eta_N = \phi_N \, e^{\Delta t_l \, \bar{d}_{i_N}/2} \, v_{i_N}$ Since this series should be truncated for computational purpose, this entails a truncation error which is proportional to $\mathbf{E}[P_L - P_{L-1}]$. For a numerical purpose, when a finite independent sample of sizes $M_l, l = l_0, \ldots, L$ is used, Eq. (6) can be approximated by the following estimator, which is the empirical mean,

$$\bar{x}^L \approx \frac{1}{M_0} \sum_{m=1}^{M_0} P_{l_0}^{(m)} + \sum_{l=l_0+1}^{L} \frac{1}{M_l} \sum_{m=1}^{M_l} (P_l^{(m)} - P_{l-1}^{(m)}). \tag{7}$$

Since an empirical mean is used, one has to generate M_l independent samples $\{i_1^{(m)}, i_2^{(m)}, \ldots, i_N^{(m)}\}$ of the set of variables $\{i_1, i_2, \ldots, i_N\}$, and to evaluate the previously defined quantity P_l in Eq. (6), denoted hereafter as $P_l^{(m)}$. The specific values taken by the set of random variables $\{i_1^{(m)}, i_2^{(m)}, \ldots, i_N^{(m)}\}$ are determined by the transition probabilities P of the continuous-time Markov chain in Eq. (5). The detailed procedure followed so far to generate the random variables are described in Sec. 3.

It becomes crucial here to remark that the samples used for computing the approximation at level l should be reused for computing the level $l-1$. This is because the underlying correlation appearing between the two consecutive levels belonging to the same sample, works by reducing the overall variance. In fact, as it was pointed out in [21, 23] the multilevel Monte Carlo method can be seen as a recursive application of the control variate technique [22], frequently used in applied statistics for variance reduction. For the specific problem of Monte Carlo path simulation in Finance, the correlation appears when the samples used correspond to same discretized Brownian path generated using different time stepsizes, and for more general applications and examples, see [20] e.g.. In our particular problem, the procedure followed to reuse the paths are described in Sec. 3. As a consequence, the multilevel method is capable of reducing the computational cost by choosing conveniently an optimal sample size M_l, and this is done by keeping fixed the overall variance within a prescribed accuracy ε^2. The detailed procedure, consisting mostly in a minimization process, to find the optimal sample size is explained in detail in [20]. Here the main results are summarized only for the sake of completeness. The optimal sample size M_l for an accuracy ε turns out to be

$$M_l = \frac{1}{\varepsilon^2} \sqrt{\frac{V_l}{C_l}} \sum_{l=l_0}^{L} V_l C_l, \tag{8}$$

where C_l, m_l and V_l are the computational cost, mean and the variance for each level l, respectively. The overall computational cost and variance is calculated as follows

$$C_T = \sum_{l=l_0}^{L} M_l C_l, \qquad V_T = \sum_{l=l_0}^{L} \frac{V_l}{M_l}, \tag{9}$$

3 The multilevel algorithm

Before explaining the detailed algorithm followed to implement in practice the multilevel method described above, lets describe first the algorithm used to generate the random paths, and second the procedure followed to reuse the paths generated for a higher level of discretization l to a lower one $l-1$. Essentially the algorithm to generate the paths was introduced first in [4] for the specific problem of computing the action of a matrix exponential over a vector, and here it is summarized for the purpose of illustration. The random paths are generated in practice as it is done for a typical continuous-time Markov chain. More specifically, if $p_{ij}(t)$ represents the transition probability matrix governing the transition between the state i and j, we can use the Kolmogorov's backward equation in Eq. (5) in integral form,

$$p_{ij}(t) = \delta_{ij}\, e^{-L_{ii}\, t} + \sum_{j\neq i} \int_0^t ds\, L_{ii}\, e^{-L_{ii}\, s} k_{ij} p_{ij}(t-s), \tag{10}$$

where $k_{ij} = |L_{ij}|/L_{ii}$, to simulate a random path. In practice the random paths are given by the transitions among the different states on $S = \{1, 2, \cdots, n\}$, being those states the values taken by the set of random variables $\{i_1, i_2, \ldots, i_N\}$. Therefore, an algorithm to generate a path may work as follows: Generate a first random time S_0 obeying the exponential density function
$p(S_0) = L_{ii}\, e^{-L_{ii} S_0}$; Then, depending on whether $S_0 < t$ or not, two different alternatives are taken; If $S_0 > t$, the algorithm is stopped, and no jump from the state i to a different state is taken; If, on the contrary, $S_0 < t$, then the state i jumps to a different state j, which is chosen randomly according to the probability function k_{ij}, and a new second random number exponentially distributed S_1 governed now by the density function $p(S_1) = L_{jj}\, e^{-L_{jj} S_1}$ is generated; If $S_1 < (t - S_0)$ the algorithm proceeds repeating the same elementary rules, otherwise eventually it is stopped.

Algorithm 1 Multilevel Monte Carlo (MLMC) algorithm.

```
INPUT: $L = l_0$,$M = M_0$,$i$, $N$,$\varepsilon$
while $error \geq \varepsilon$ do
    Call MLMCL($i, \Delta t_l, N, M_0$) for fast estimating $m_l$ and $V_l$ for $l = l_0, \ldots, L$
    Compute the optimal number of samples $M_l$ for $l = l_0, \ldots, L$
    Call MLMCL($i, \Delta t_l, N, M_l - M_0$) for further improvement for $l = l_0, \ldots, L$
    if $error \leq \varepsilon$ then EXIT
    else
        Increase number of levels, $L = L + 1$
    end if
end while
```

To illustrate graphically how the paths generated from a higher level are reused for the lower one, in Fig. 1 a sketch diagram for the case of $l = 2$ is shown. Here they are plotted the four different scenarios that may occur when generating random paths starting at the same point i. Then, from Eq. (4), the possible outcomes of the two random variables may induce two transitions to any of the rows of a given matrix during the two time steps of size Δt_2. Rather only the last one should be used for determining the paths corresponding to the previous level $l = 1$. More specifically, the set of the four figures describe the following scenarios: a) Transitions occur at the first and at the second time step; b) Transition only at the first time step; c) Transition only at the second time step, and d) no transition at all. Note that the last scenario contributes with zero to the term $\mathbf{E}[P_2 - P_1]$ in (6).

The pseudocode implementing the described multilevel method is shown in Algorithm 1, which consists in practice in the general setting for any implementation of the method particularized for this specific problem choosing a procedure to compute the mean m_l and the second moment m_{2l} for any of the levels l. Note that both quantities

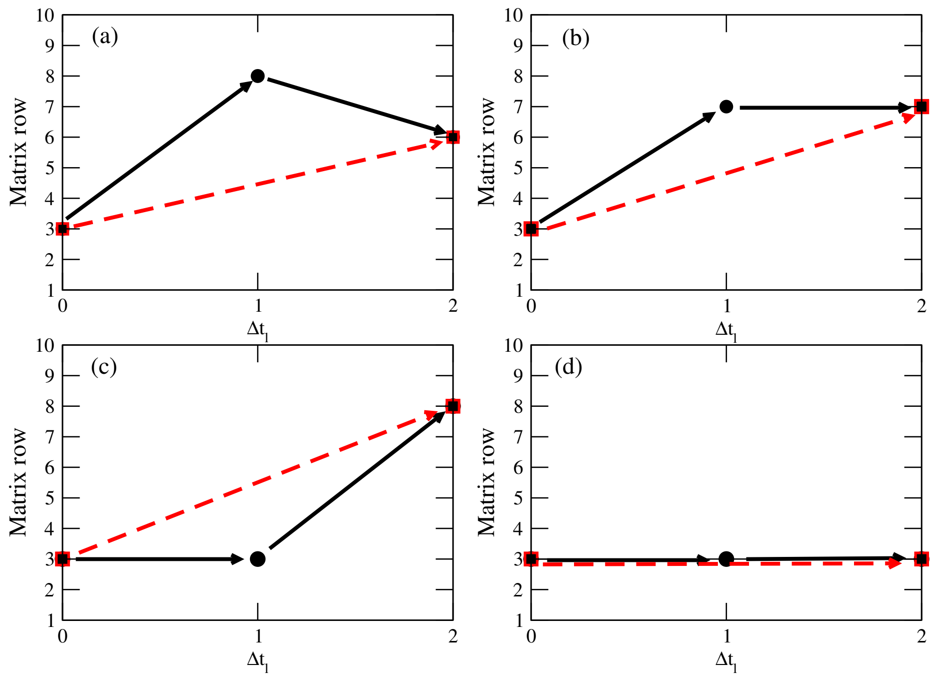

Figure 1: Sketch diagram showing the four possible sampled paths obtained for level $l = 2$, and for a matrix of size $N = 10$. The solid line corresponds to a random path obtained for a level number l, and the dotted line with $l - 1$.

are needed to compute the variance V_l, which is given by $V_l = m_{2l}/M_l - m_l^2$. This procedure is described in Algorithm 2.

4 Numerical errors

In this Section the different source of errors of the Monte Carlo method for computing Eq. (4) are discussed. Note that the error ε made in computing the vector solution at a single point i can be decomposed as follows

$$\varepsilon = x_i - \frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{N}\bar{\omega}_k\left(\prod_{j=1}^{k}\eta_j^{(m)}\right)v = \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4, \tag{11}$$

Algorithm 2 Procedure to compute a single entry i of the vector solution $\bar{x}_i$.

```
procedure MLMCL($i, \Delta t_l, N, M$)
    for $l = 1, M$ do
        $\eta_1 = 1$,$\eta_2 = 1$,$j = i$
        for $n = 1, \ldots, N$ do
            $\eta_2 = \eta_2 e^{\bar{d}_j \Delta t_l / 2}$
            if $n\, mod\, 2 \neq 0$ then
                $\eta_1 = \eta_1 e^{\bar{d}_j \Delta t_l}$
            end if
            generate($\tau$)
            while $\tau < \Delta t_l$ do
                $k = j$
                generate($S$), generate($j$)
                $\tau = \tau + S$
                $\eta_2 = (-1)^{\sigma_{kj}} \eta_2$
                $\eta_1 = (-1)^{\sigma_{kj}} \eta_1$
            end while
            $\eta_2 = \eta_2 e^{\bar{d}_j \Delta t_l / 2}$
            if $n\, mod\, 2 = 0$ then
                $\eta_1 = \eta_1 e^{\bar{d}_j \Delta t_l}$
            end if
            $integ_1 = integ_1 + \omega_n \eta_1$
            $integ_2 = integ_2 + \omega_n \eta_2$
        end for
        $m_l = m_l + [v_j(integ_2 - integ_1)]/M$
        $m_{2l} = m_{2l} + [v_j(integ_2 - integ_1)]^2/M$
    end for
    $V_l = m2l/M - m_l^2$
    return $(m_l, V_l)$
end procedure
```

where $\eta_k^{(m)}$ corresponds to the mth sample of the random variable η_k defined in (4), and

$$\begin{aligned}
\varepsilon_1 &= x_i - \left(\int_0^T dt\, e^{-st}\, e^{t\,A}\right) v, && (12)\\
\varepsilon_2 &= \left(\int_0^T dt\, e^{-st}\, e^{t\,A}\right) v - \left(\sum_{k=1}^{N} \omega_k e^{-s\,t_k}\, e^{t_k\,A}\right) v, && (13)\\
\varepsilon_3 &= \left(\sum_{k=1}^{N} \omega_k e^{-st_k}\, e^{t_k\,A}\right) v - \left(\sum_{k=1}^{N} \omega_k\, [e^{\Delta t\bar{D}/2} e^{-\Delta t\, U} e^{\Delta t \bar{D}/2}]^{k-1}\right) v, && (14)\\
\varepsilon_4 &= \left(\sum_{k=1}^{N} \omega_k\, [e^{\Delta t\bar{D}/2} e^{-\Delta t\, U} e^{\Delta t \bar{D}/2}]^{k-1}\right) v - \frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{N} \bar{\omega}_k \left(\prod_{j=1}^{k} \eta_j^{(m)}\right) v. && (15)
\end{aligned}$$

Here the matrix $\bar{D} = D - s\,\mathbb{1}$. Lets analyze then these four different errors separately. The first error ε_1 is due to the truncation of the improper integral in Eq. (1) by a finite one, where the unbounded domain of integration has been replaced by a finite one, replacing conveniently the infinite limit by a finite one, T. Such an error can be further evaluated and is given by

$$\varepsilon_1 = \left(\int_T^\infty dt\, e^{-st}\, e^{A\,t}\right) v. \tag{16}$$

Note that the integral can be computed analytically, and yields

$$\varepsilon_1 = B^{-1} e^{-B\,T}\, v, \tag{17}$$

where $B = s\,\mathbb{1} - A$. Moreover, it turns out that

$$\|\varepsilon_1\|_2 \leq \left\|B^{-1}\right\|_2 \left\|e^{-B\,T}\right\|_2, \tag{18}$$

and since we assume $\|A\|_2 < s$, it holds that

$$\|\varepsilon_1\|_2 \leq \left\|B^{-1}\right\|_2 e^{-\lambda_{min}(B)T}. \tag{19}$$

Here $\lambda_{min}(B)$ denotes the smallest eigenvalue of B, which corresponds in practice to $s - \lambda_{max}(A)$, being, $\lambda_{max}(A)$ the largest eigenvalue of A. Hence

$$\|\varepsilon_1\|_2 \leq \frac{1}{s - \lambda_{max}(A))} e^{-[s-\lambda_{max}(A)]T}. \tag{20}$$

The formula above can be used to find the minimum value, T_c, such as the error is less than the prescribed accuracy ε, and is given by

$$T \geq T_c \equiv \frac{1}{\lambda_{max}(A) - s} \ln\left[(s - \lambda_{max}(A))\varepsilon\right]. \tag{21}$$

To use in practice such formula the largest eigenvalue of the matrix A should be known. In general, especially for large matrices, this could be a formidable task in itself, being required in general to use suitable available approximations. For the specific case of complex networks, there are indeed some useful approximations (see [9] e.g), and they have used in Fig. 17 to verify this estimation. In fact, in Fig. 17 it is plotted the result corresponding to the absolute error made when truncating the improper integral as function of the finite limit T. In this example the solution corresponds to the Katz centrality of a given node for two different complex networks of the same size. The theoretical solution, which is needed to compute the error, was obtained using Matlab, being Eq. (17) computed using a high accuracy numerical method. For this

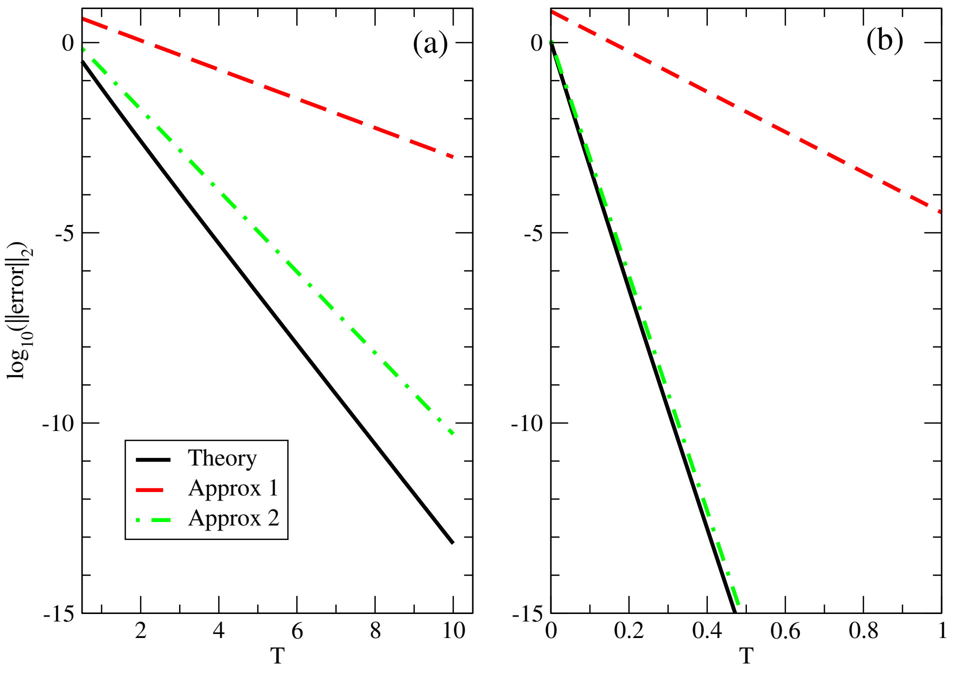

Figure 2: Absolute numerical error in log-scale made when truncating the improper integral in Eq. (1) using a finite limit, T. The solution corresponds to the Katz centrality of two complex networks: (a) a small-world network, and (b) a scale-free network, both of size of $1,000$ nodes. The dashed (Approx 1) and dot-dashed (Approx 2) lines correspond to the solution obtained using the approximation of the largest eigenvalue as d_{max}, and $max(d_{avg}, \sqrt{d_{max}})$, respectively. The value of the variable s was chosen to be $d_{max}/0.85$.

specific example we have used two different approximations for the largest eigenvalue of the adjacency matrix A. One assumes the largest eigenvalue equals to the maximum degree of the network, d_{max} (Approx1), while the other one (much more accurate), assumes the value to be $max(d_{avg}, \sqrt{d_{max}})$ (Approx2), where d_{avg} denotes the average degree of the network, $d_{avg} = \frac{1}{n}\sum_{i=1}^{n} d_i$. Note how the latter approximation fits in fact much better with the theoretical error curve plotted in Fig. 2.

Concerning the second error ε_2, this appears when the definite integral is approximated numerically using a suitable quadrature. Hence, the order of the error is merely the order of the error of the chosen quadrature rules, and therefore to minimize this error it would be advisable to use higher order methods However, as it will be shown later, it turns out that the order of the third error ε_3 is proved to be of order $O(\Delta t^2)$. Then, it becomes useless at this point to implement any higher order method for this approximation, since the order of the error ε_3 becomes dominant. Consequently, for the algorithm proposed it was used a simple trapezoidal quadrature, which is well known to be of order $O(\Delta t^2)$. Therefore, the quadrature weights used are given by $\omega_i = \Delta t, i = 2, \ldots, N-1$, and $\omega_1 = \omega_N = \Delta t/2$.

The analysis of the remaining errors, ε_3 and ε_4, coincides exactly with the analysis done in [3], but for the sake of completeness it is summarized here. The third error ε_3 is due merely to the splitting procedure, as a result of decomposing the matrix A as $D-U$, and the error turns out to be of order $O(\Delta t)$ or $O(\Delta t^2)$ [1], depending on whether the Lie or the Strang splitting is used. As mentioned already in Sec. 2, since the matrix D is diagonal matrix, it can be computed almost without any computational cost, being therefore much more convenient to adopt the Strang splitting to this purpose. The fourth error, ε_4 , is the pure Monte Carlo statistical error, and known to be of order $O(M^{-1/2})$. In fact, it is well known that the arithmetic mean appearing in (11) provides the best unbiased estimator for the expected value in (4). In practice, one should simulate on the computer the random variables, based on generating random numbers. By doing so, the error made in replacing the expected value with the mean over a finite size sample is statistical in nature. More precisely, ε_4 turns out to be, for a large M value, approximately a random Gaussian variable with standard deviation proportional to $M^{-1/2}$, i.e.,

$$\varepsilon_2 \approx \frac{\sigma\nu}{M^{1/2}}, \tag{22}$$

where σ denotes the square root of the variance, and ν is a standard normal (i.e., $N(0,1)$) random variable. All this clearly shows that the proposed Monte Carlo method could in principle have a poor numerical performance, and also that the error is merely statistical, so it can only be bounded by some quantity with a certain degree of confidence. However, there already exists many available statistical techniques, such as variance reduction, and quasi-random numbers [28], that can be used, in practice, to improve greatly the order of the global error, and consequently the overall performance of the algorithm.

5 Computational complexity of the multilevel algorithm

To properly estimate the computational complexity of the multilevel algorithm, it is required to establish first the convergence rate of both, the mean $m_l = |E[P_l - P]|$ and variance $V_l = V[P_l - P_{l-1}]$, as a function of the corresponding level l. In [4] this was done for the particular problem of computing the action of a matrix exponential over a vector. Note that the method proposed here required to compute the action of a matrix exponential over a vector, and therefore the results found there can be applied here straightforwardly. More specifically in [4] it was proved that the mean $|E[P_l - P]|$ turns out to be of order $\mathcal{O}(\Delta t_l^2)$. This is because the splitting method used for the computing the matrix exponential was the Strang splitting. Besides computing the matrix exponential, recall that it is required as well to approximate numerically the definite integral in Eq. (1). As was mentioned above this is done through a suitable trapezoidal numerical quadrature. However, this procedure does not modify the order of convergence, and this can be seen readily through the following Lemma.

Lemma 1 *Let consider $I = \sum_{i=1}^{N} \omega_i f_i$ the discretization of a given definite integral obtained by means of a trapezoidal numerical quadrature, where ω_i are the corresponding quadrature weights, and f_i the values of a given function evaluated at equally spaced points within the integration domain. Assume that the values $f_i, i = 1, \ldots N$ are known with an error of order $\mathcal{O}(\Delta t^3)$. Then it holds that the error of I is of order $\mathcal{O}(\Delta t^2)$.*

Proof. If every term of the sum is of order $\mathcal{O}(\Delta t^3)$, the sum of all of them turns out to be of order of $\mathcal{O}(N^2 \Delta t^3)$, and since $N = T/\Delta t$ the order is reduced in a factor of

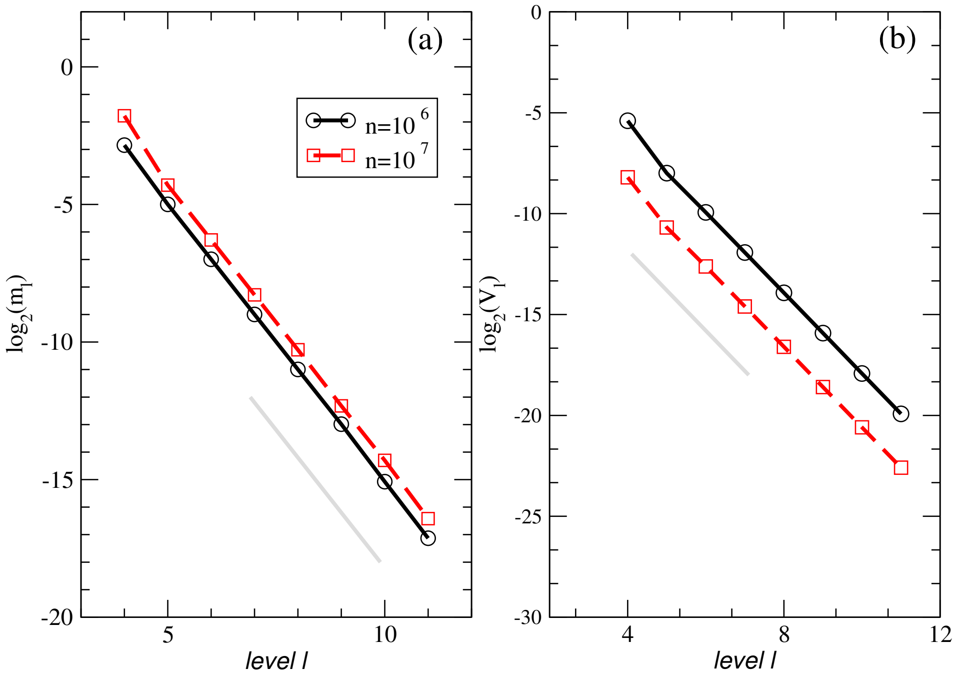

Figure 3: (a) Mean and (b) variance of $P_l - P_{l-1}$ in log_2 scale versus the level number l obtained numerically. The adjacency matrix corresponds to a small-world network of different size n. The brown line denotes to an ancillary function of slope -2.

two. However, after multiplying by the trapezoidal weights, which are proportional to Δt, the order of the error becomes finally $\mathcal{O}(\Delta t^2)$. □

Concerning the convergence rate of the variance $V[P_l - P_{l-1}]$, the same result already proved for the mean holds. Since the convergence rate of the variance for computing the action of the matrix exponential over a vector was proved to be $\mathcal{O}(\Delta t_l^2)$ in [4], the convergence rate for this specific problem turns out to be similarly of order $\mathcal{O}(\Delta t_l^2)$.

In Fig. 3, the mean $E[P_l - P_{l-1}]|$ (a) and $V[P_l - P_{l-1}]$ (b) are shown as a function of the level l. The matrices correspond to the adjacency matrices of a small-world network of two different size. Note that the obtained numerical convergence rate fully agrees with the theoretical estimation.

To determine theoretically the computational complexity of the multilevel algorithm, it is needed also an estimation of the computational time of the Monte Carlo method. We have seen that the numerical method requires to compute in practice the action of the matrix exponential over a vector, and note that this should be done a few times, namely as many as the number of discretized points of the numerical quadrature. Therefore, one might wrongly conclude that the overall computational

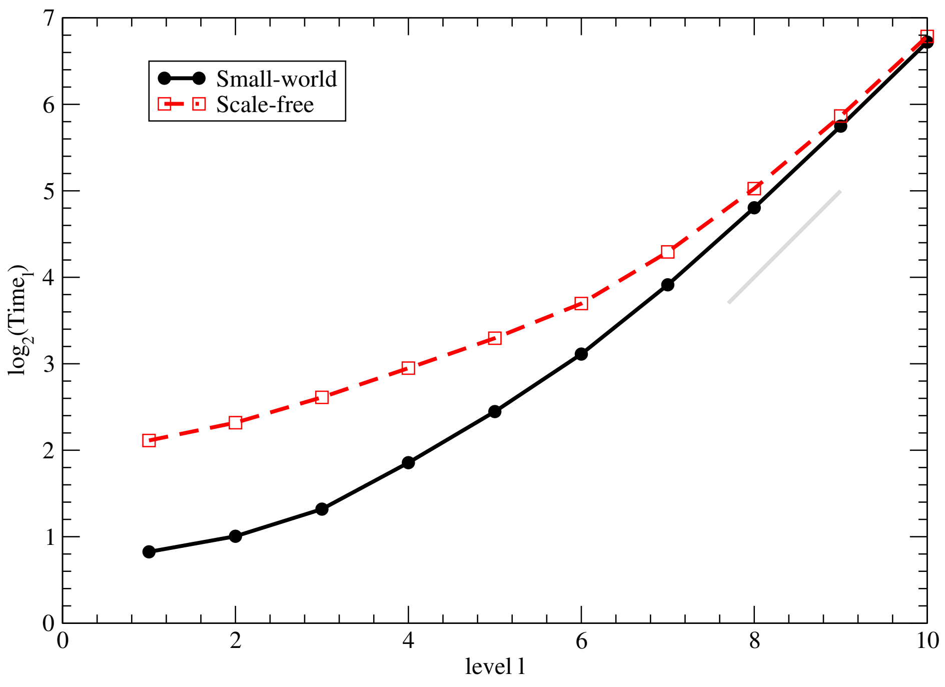

Figure 4: Computational time in log_2 scale versus the level l for adjacency matrices corresponding to two different complex networks of size $n = 10^6$. The brown line corresponds to an ancillary function of slope 1.

time depends on such a number. However, it turns out that the matrix exponential required for any time step is computed using the matrix exponential obtained for all the previous time steps, as it can been seen from Eq. (2). In practice this means that is enough to compute once the matrix exponential (and only for the finite limit T_c), provided that all the values of the matrix exponential obtained for intermediate times are conveniently saved. Therefore, the computational time of the algorithm is merely due to the computational time required to compute the action of a matrix exponential over a vector evaluated exclusively at T_c. This time was already estimated in [3]. In Fig. 4 the results corresponding to the CPU time spent by the Monte Carlo algorithm when computing the Katz centrality of two different networks characterized by different values of d_{avg} are plotted. Note that for Δt_l sufficiently large (or equivalently l sufficiently small) the computational time tends to a constant value, while for smaller values the computational time scales as $1/\Delta t_l$. Using the theoretical estimation in [3], it can be estimated that the minimum value of the level l_0 to obtain a better performance of the MLMC algorithm is given by

$$l_0 \gg l_c \equiv log_2(T_c d_{avg}). \tag{23}$$

Known the convergence rates of the mean, variance, and computational time, the

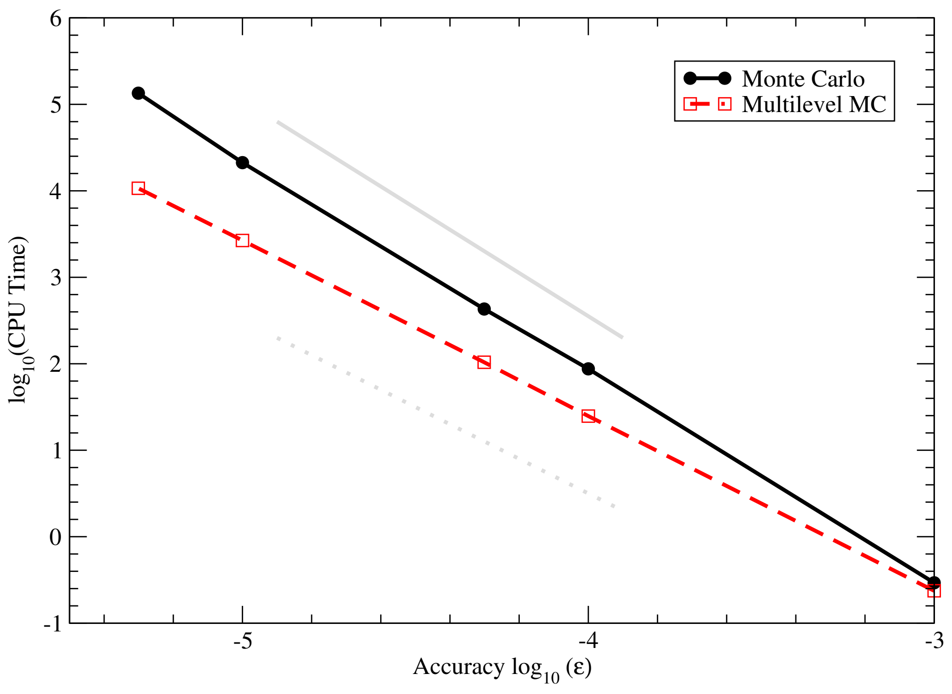

Figure 5: Computational time as function of the prescribed accuracy ε, both in log_{10} scale. The brown solid line corresponds to an ancillary function of slope $-5/2$, while the dotted line to a function of a slope -2. The results correspond to the Katz centrality for a single node of a small-world network of size $n = 10^6$

theorem in [20] can be applied, and as a result it can be established that the computational complexity of the proposed MLMC algorithm for computing the resolvent of a matrix should be of order $\mathcal{O}(\varepsilon^{-2})$. This contrasts favorably with the complexity of the Monte Carlo method, which can be concluded to be of order $\mathcal{O}(\varepsilon^{-5/2})$. In fact, the action of the matrix exponential over a vector using the Monte Carlo method was estimated in [3] as being of order $\mathcal{O}(\varepsilon^{-5/2})$. Recall that the algorithm for computing the resolvent matrix requires computing such a matrix exponential, and to avoid calculating such matrix exponential several times for each time steps, we can use the same procedure explained already for the MLMC method. Consequently we can conclude than the computational time of the Monte Carlo method for this problem remains of order $\mathcal{O}(\varepsilon^{-5/2})$. In Fig. 5 the results corresponding to the computational time spent to compute the Katz centrality of a single node of a small-world network of size $n = 10^6$ is plotted as a function of the chosen prescribed accuracy ε for both, the MLMC method and the Monte Carlo method. Note the perfect agreement with the theoretical estimates, and the performance notably superior to the Monte Carlo method for lower accuracy values.

6 Numerical examples and computational-related issues

6.1 Numerical examples

In this section the results corresponding to several numerical examples are shown. The chosen examples consist in both, the numerical computation of the aforementioned metric Katz centrality in some synthetic complex networks and the numerical solution of some elliptic boundary-value problems with Dirichlet boundary conditions. More specifically, these are the Poisson equation discretized by means of the finite difference method, and a reaction-diffusion equation in an arbitrary domain discretized by the finite element method. In all of them the solution is computed exclusively at a single node. Concerning the synthetic networks, these consist of small-world and scale-free networks that can be generated easily for an arbitrary size using the functions *smallw*, and *pref*, respectively, freely available through the toolbox CONTEST [11] for Matlab.

To compare the performance of the algorithm with others available in the literature, we implemented a different available Monte Carlo method. This method was first proposed in [6], and in the following to distinguish from our Monte Carlo method it will be termed the *classical Monte Carlo method*. Basically this method depends on two free parameters to be fixed by the user according to the accuracy desired for the solution. These are first, the so-called history length, which basically is related with the number of random jumps allows to occur within the matrix before stopping the algorithm, and second, the number of random walks picked up from a finite sample. Both discretized parameters are involved in different source of errors. Essentially they are the equivalent of the bias error, and statistical error appearing as source of errors for the MLMC method, but they are treated by the algorithm in a totally different way. In fact, while the classical Monte Carlo algorithm considers both separately, the MLMC algorithm works with both parameters in a unified way within the goal to reach the prescribed accuracy for the solution. The algorithm proposed in [6] is indeed adaptative, being capable of finding automatically the optimal solution within a prescribed accuracy. However, note that this is done considering only the statistical error, but seems to fail to find the optimal history length satisfying such an accuracy. Concerning the statistical error this is done basically by increasing progressively the number of random walks simulated until a certain condition (related basically with the variance) is satisfied. On the other hand, the history length, which controls the bias error, this should be fixed independently according to another stopping criterion of the algorithm. This truncation criterion to stop running the random walks was proposed in [16], and consists in imposing a certain relative weight, which acts basically as a cutoff threshold stopping automatically the random walk whenever the underlying random variable overcomes such a threshold. However, it becomes not trivial to relate the value of this threshold with the statistical error, and with the prescribed accuracy of the solution desired by the user. In practice what happens is that the user may have to run several times the algorithm, changing accordingly such a threshold, until the desired solution within the prescribed accuracy is finally reached. Rather the multilevel algorithm by construction is a truly automatic algorithm, meaning that the only intervention of the user consists in setting initially the accuracy desired for the solution, leaving the algorithm to find alone both the optimal sample size and history length. Moreover, this offers a further computational advantage for the parallelization point of view, as it will be discussed below.

All Monte Carlo codes were implemented in Fortran 90 and the simulations run in a multi-core architecture consisting on a computer equipped with an AMD Ryzen 1800X Octa-core at 3.6 GHz with 32 GB of RAM.

Due to the random nature of any of the Monte Carlo algorithms, it is worth observing that the measured computational time can vary from simulation to simulation.

To mitigate such a variability of the computational times, the CPU times shown in the tables below correspond to an average CPU time obtained repeating the simulations with 10 different initial random seeds of the pseudorandom generator.

Example A: Complex networks

As it was mentioned above, in the case of complex networks the evaluation of the resolvent matrix over a vector is related with the so-called Katz centrality of the network. In fact, the Katz centrality of a node i of a network is defined mathematically [7, 26] as

$$K_i(\alpha) = [(\mathbb{1} - \alpha A)^{-1}\mathbf{1}]_i, \tag{24}$$

where A is the adjacency matrix of the network, $\mathbf{1}$ a vector of ones, an α, with $0 < \alpha < 1/\lambda_{max}(A)$, is a attenuation factor suitable chosen. Intuitively, the Katz centrality is a metric of the network that measures the relative degree of influence of a node within the network, weighting conveniently the importance of the connection between the node i and distant neighbors by an attenuation factor α.

In Tables 1, and 2 the CPU time spent to compute the Katz centrality of a small-world network and scale-free network using the Monte Carlo method, the MLMC method, and the classical Monte Carlo method in [6] is shown. This has been done for different network sizes, being the accuracy kept fixed to $\varepsilon = 3\times10^{-5}$ for all simulations. Note that the CPU time is almost independent of the network size for any of the Monte Carlo methods. This is due to the fact that the Monte Carlo method essentially is based on local computations, rather than the classical deterministic methods which typically require as part of the algorithm multiplying a matrix by vector or even worse a matrix by a matrix. This gives rise in practice to an implicit dependence on the size of the problem as it can be seen in [3].

Concerning the results, it is worth observing the notably performance of both, the MLMC method and the classical Monte Carlo, compared with the Monte Carlo method, and a similar performance between the MLMC method and the classical method.

Table 1: CPU time spent for computing the Katz centrality of a small-world network using the Monte Carlo method (MC), the multilevel Monte Carlo method ($MLMC$), and the classical Monte Carlo method (MC_c). The accuracy ε of the solution was kept fixed to 3×10^{-5}.

Network size	CPU Time MC (s)	CPU Time MLMC (s)	CPU Time MC_c (s)
10^5	$8,534$	$2,054$	$1,848$
10^6	$8,575$	$2,063$	$1,869$
10^7	$8,601$	$2,141$	$1,980$
10^8	$8,620$	$2,250$	$2,034$

Table 2: CPU time spent for computing the Katz centrality of a scale-free network using the Monte Carlo method (MC), the multilevel Monte Carlo method ($MLMC$), and the classical Monte Carlo method (MC_c). The accuracy ε of the solution was kept fixed to 3×10^{-5}.

Network size	CPU Time MC (s)	CPU Time MLMC (s)	CPU Time MC_c (s)
10^5	$8,410$	$2,323$	$1,819$
10^6	$8,489$	$2,441$	$1,951$
10^7	$8,537$	$2,583$	$2,134$
10^8	$8,663$	$2,635$	$2,356$

Example B: Partial differential equations

• *Example 1.* This example concerns the numerical solution of a Dirichlet boundary value problem consisting in the 2D Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \quad = \quad f, \quad \text{in } \Omega = [-1,1]^2, \tag{25}$$

and solved on a square domain Ω with zero Dirichlet boundary condition, $u(x,y)|_{\partial\Omega} = 0$. The domain is conveniently discretized using a computational grid with discretization parameters $\Delta x = \Delta y = \Delta z = 2/n_x$, and the operator by a finite difference scheme using the standard 5−point stencil [29]. Therefore, the discretized problem to be solved is $A\,u = \bar{f}$, being A the well-known block tridiagonal matrix corresponding to the discrete Laplace operator with zero Dirichlet boundary conditions, and $\bar{f}$ the source term evaluated at the grid nodes.

The Monte Carlo method can be applied readily to solve such a linear problem, but first to guarantee the convergence of the method, it is necessary to use a suitable preconditioner as it was pointed out in [6], and second to rewrite the solution of the problem as follows

$$u = (\mathbb{1} - H)^{-1} P^{-1} \bar{f}. \tag{26}$$

Here P denotes the preconditioner, and $H = \mathbb{1} - P^{-1}A$. Specifically for this problem has been used a left Jacobi preconditioner. Comparing Eq. (1) with Eq. (26), note that Eq. (26) corresponds indeed to the action of the resolvent of the matrix H (setting $s = 1$) over the vector $P^{-1}\bar{f}$, and therefore both, the Monte Carlo method and the MLMC method can be applied indeed for solving this problem.

Table 3 shows the computational time spent by the three Monte Carlo algorithms for different grid sizes. As in the previous example, the same conclusions can be drawn. In fact, both, the MLMC method and the classical Monte Carlo method, exhibit a similar performance, being notably superior than the performance of the Monte Carlo method, and furthermore this holds for any size of the problem.

Finally, it is worth observing in Table 3 that the computational time spent by all three methods scales almost linearly with the matrix size. In fact, when the grid size is scaled by a factor of 4, or equivalently the matrix size by a factor of 16, the computational time increases approximately by a factor equal to 16, since $n = n_x^2$. This can be explained as follows: The largest eigenvalue of the discrete 2D Laplace operator with zero Dirichlet boundary condition is known to be

$$\lambda_{max}(A) = 4 - 8\sin^2(\frac{\pi}{2n_x + 2}), \tag{27}$$

therefore $\lambda_{max}(H) = 1 - 2\sin^2(\frac{\pi}{2n_x+2})$. From the definition of T_c in Eq. (21), asymptotically for large n_x, it holds that $T_c \sim n_x^2 \ln(n_x)$, and therefore from Eq.(23) it follows that $l_c \sim \log_2(n_x^2 \ln n_x)$. Note that the computational time of the MLMC method depends on the value of the minimum level l_c, being $T_{MLMC} > T_{CPU}(l_c)$. Using the theoretical estimation in [3] asymptotically for sufficiently small Δt_l, it is known that $T_{CPU}(l) \sim 2^l$, then it holds that $T_{CPU}(l_c) \sim n \ln(n)$.

Table 3: CPU time spent for computing the solution of the 2D Poisson equation at a single point using the Monte Carlo method (MC), the multilevel Monte Carlo method ($MLMC$), and the classical Monte Carlo method (MC_c). The accuracy ε of the solution was kept fixed to 10^{-3}, and the chosen point $i = n/2$.

n_x	Matrix size	CPU Time MC (s)	CPU Time MLMC (s)	CPU Time MC_c (s)
256	65, 536	35	21	16
1, 024	1, 048, 576	596	322	277
4, 096	16, 777, 216	10, 065	5, 207	4, 811

• *Example 2.* This second example consists in the numerical solution of the underlying linear algebra problem corresponding to the discretization of a 2D reaction-diffusion equation given by

$$-\nabla(\beta(\mathbf{x})\nabla u) + u = 1, \quad \mathbf{x} \in \Omega, \quad u(\mathbf{x})|_{\partial\Omega} = 0, \tag{28}$$

where $\Omega \subset \mathbb{R}^2$, $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$, and $\beta(x, y)$ corresponds to the diffusion coefficient characterized by a 2×2 positive-definite matrix $\beta = \{\beta_{ij}\}_{i,j=1}^{2}$. In particular, for this example this matrix has been chosen to be diagonal with entries $\beta_{11} = 1 + y^2/\alpha^2(\alpha^2 - x^2/\alpha^2)$, and $\beta_{22} = 1 + x^4/\alpha^4(\alpha^2 - y^2/\alpha^2)$. The domain consists in an arbitrary geometry, which is plotted in Fig. 6, and the Dirichlet boundary data was chosen to be 0 at both, the inner and outer circle with radius 0.25α and α, respectively. The size of the domain can be conveniently increased by simply rescaling both circles using a single scale parameter α. To generate the computational mesh, and obtaining the corresponding FEM stiffness matrix and right-hand side vector, the scientific software *COMSOL* [10] was used, setting specifically linear elements as the discretization basis. Different values of the corresponding maximum element size h_{max} used when meshing the geometry was chosen to test the algorithms for different matrix sizes.

Similarly to the previous example, to ensure the convergence of the Monte Carlo method it is required to use suitable preconditioners. Also in this example it has been used a left Jacobi preconditioner as it was described in Eq. (26). However, note that for this more involved problem, the maximum eigenvalue of the matrix is not theoretically known, being therefore necessary to resort to suitable approximations in order to apply Eq. (21) and thus obtaining the minimum value, T_c. For this purpose it has been used the Gershgorin circle theorem to find a reasonable bound for the maximum eigenvalue of the matrix.

The computational time spent for computing the solution at a single point inside the domain is shown in Table 4, and the same conclusions hold as in the previous example.

Table 4: CPU time spent for computing the solution of a 2D reaction-diffusion equation at a single point using the Monte Carlo method (MC), the multilevel Monte Carlo method ($MLMC$), and the classical Monte Carlo method (MC_c). The accuracy ε of the solution was kept fixed to 10^{-3}, the scale parameter α to 10, and the chosen point $\mathbf{x} = (0, 0)$.

h_{max}	Matrix size	CPU Time MC (s)	CPU Time MLMC (s)	CPU Time MC_c (s)
0.1	$48,314$	7.8	4.5	2.7
0.03	$513,455$	52	33	29
0.02	$1,1143,744$	202	110	91

6.2 Computational-related issues

Parallel performance. To analyze the scalability of the Monte Carlo methods, the algorithms were conveniently parallelized using OpenMP. This allows to exploit fully the multi-core architecture of the available server. In Table 5 the computational time required to compute the Katz centrality of a small-world network of size $n = 10^7$ is shown as function of the number of cores. Note that both the Monte Carlo method, and MLMC method exhibit a remarkable scalability being almost close to the ideal one. This is not surprising, since as it was theoretically anticipated before, any Monte Carlo method requires to compute independent simulations which can be trivially split in so many independent tasks as the available cores of the server, being therefore the intercommunication overhead of the parallelized algorithm almost negligible.

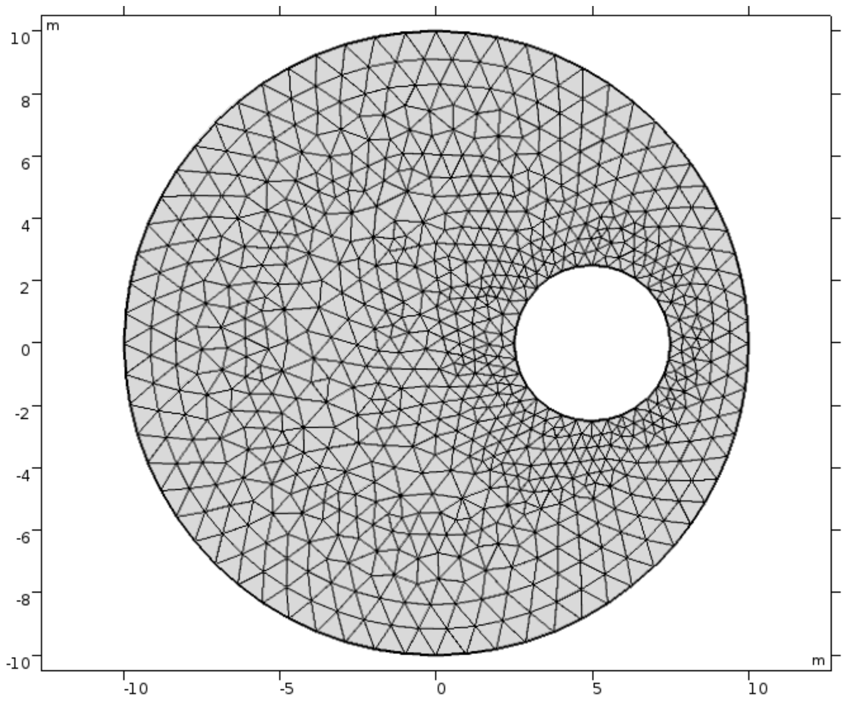

Figure 6: Computational mesh describing the domain used for solving the 2D reaction-diffusion equation.

For the classical Monte Carlo method, as was explained above it is based on a adaptive algorithm, which requires to fix a given parameter before running in parallel . It consists namely on the number of random paths N_0 chosen to be increased for each iteration of the algorithm, and it turns out that this parameter is fundamental to improve the scalability of the algorithm. In fact, this parameter affects indeed the total iterations of the algorithm, and moreover has a strong impact in the intercommunication overhead of the algorithm, and consequently in the scalability and parallel performance. In fact, as it can be seen in Table 5, for a fixed number of cores increasing N_0 has always a positive impact on the scalability improving accordingly the speed-up of the algorithm. This is because in practice it reduces the total iterations needed to achieve convergence. However note that this may increase unnecessarily the overall CPU time, since it could happen that the total number of random paths simulated exceeds unnecessarily the random paths needed to attain the prescribed accuracy.

This would never happen with the MLMC method, because the algorithm exploits the mathematical relation in Eq. (8). Therefore the MLMC algorithm is capable of computing precisely the number of random paths needed to attain the prescribed accuracy, provided the variance and computational cost for each level is known. Since in practice these values have to be computed numerically as well, further iterations may be needed to improve the accuracy of such values. However, it is important to remark that typically only a few number of iterations are required after all. This indeed explains the remarkable scalability of the algorithm observed in Table 5.

Simultaneous computing of both, Katz centrality and total subgraph communicability, of a complex network Another important feature of the MLMC method compared with the classical Monte Carlo is the fact that without any additional computational cost it allows to compute other useful metrics, such as the total

Table 5: Elapsed time spent for computing the Katz centrality of a small-world network as a function of the number of cores. The size of the matrix is $n = 10^7$, and the accuracy ε was kept fixed to 3×10^{-5}. The simulations were run on a octa-core server.

Cores	Time MC (s)	Time $MLMC$ (s)	Time MC_c (s)		
			$\mathbf{N_0 = 100}$	$\mathbf{N_0 = 1000}$	$\mathbf{N_0 = 10000}$
1	$8,601$	$2,141$	$1,819$	$1,617$	$1,543$
2	$4,361$	$1,066$	$1,362$	980	960
4	$2,185$	527	986	544	497
8	$1,076$	267	$1,276$	583	395

subgraph communicability of a node [7], and this could be done simultaneously. This metric is defined as

$$C_i(A) = (e^{\beta A}\mathbf{1})_i, \tag{29}$$

and measures the importance of the node i, weighting now the walks of length k by a penalty factor of value $\beta^k/k!$. Concerning β, it is typically interpreted as an effective "temperature" of the network (see [17], e.g.). Note from Eq. (2) that both, the Monte Carlo method and the MLMC method, compute automatically ancillary quantities such as

$$\xi_i^j = (e^{-j\,s\,\Delta t}\,e^{j\,\Delta t\,A}\,v)_i, \quad j = 1, \ldots, N, \tag{30}$$

as part of the algorithm. These quantities are trivially related with the total subgraph communicability of a given node for a discrete set of effective temperatures as follows

$$C_i(A) = e^{j\,s\,\Delta t}\xi_i^j, \tag{31}$$

where $\beta_j = j\Delta t$. Although in practice they are not saved individually, because they are not indeed needed to compute the Katz centrality, being rather only effectively computed the sum of all of them, however a simple modification in the algorithm would allow in practice saving such quantities. Furthermore this might be done without almost any additional computational cost of the algorithm. In case of the MLMC method, a crucial point here is to ensure that the numerical error made when computing these ancillary quantities does not exceed the prescribed accuracy. In fact, this cannot be guaranteed by the method, because the proposed MLMC method was developed to compute exclusively the resolvent matrix within a prescribed accuracy, and not any of their ancillary quantities. Nevertheless, in all the examples analyzed so far the error made never exceeded the prescribed accuracy of the algorithm as it is shown in Fig. 7 for the particular case of a small-world network. The numerical error was computed here using the solution obtained using a Krylov-based method [19] of higher accuracy as it was the theoretical solution.

7 Conclusion

The goal of this paper was to propose a new probabilistic method based on the multilevel Monte Carlo method for computing the action of the resolvent matrix over a vector. More specifically, the idea was to compute such an action resorting to the numerical evaluation of the Laplace transform of the matrix exponential, because it is known that the action of a matrix exponential over a vector can be computed efficiently using a multilevel Monte Carlo method. In fact a method was introduced recently in the literature for such a purpose, and essentially it requires generating suitable random paths which evolve through the indices of the matrix according to the

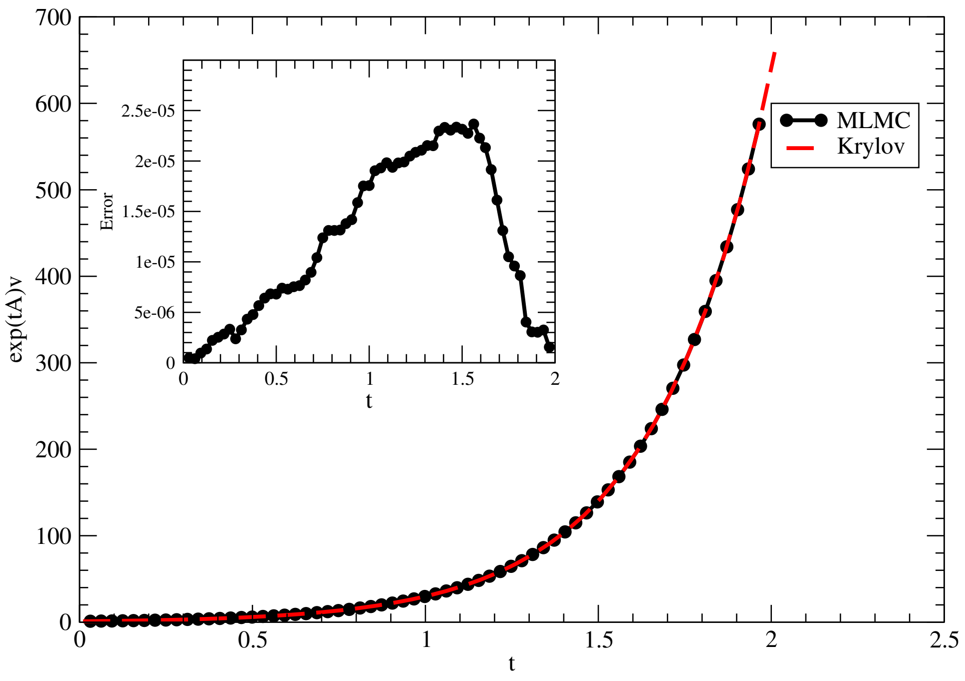

Figure 7: Total subgraph communicability of a single node $(n/2)$ for different values of the effective temperature t. The network is a small-world network of size $n = 10^7$. The solid line corresponds to the solution obtained using a Krylov-based method, while the dotted-line to the solution obtained simultaneously when computing the Katz centrality using a modified MLMC algorithm capable of saving the ancillary quantities. In the inset it is shown the relative error of the computed solution.

probability law of a continuous-time Markov chain governed by the associated generalized Laplacian matrix. The convergence of the proposed multilevel method has been conveniently analyzed in this paper, and several numerical examples were run to test the performance of the algorithm.

Since there is in the literature other well-established Monte Carlo method for solving linear algebra systems, we compared in this paper the results obtained using both methods. It is needless to say that being both methods based on the Monte Carlo method they share similar advantages from a computational point of view, such as the comparative ease of implementation in parallel, fault-tolerant, and in general they are well suited for heterogeneous architectures. However, we show here that the multilevel stands out especially in a feature that is apparently lacking in the classical method, which is the autonomous operation of the multilevel algorithm, in contrast with the classical Monte Carlo algorithm. In fact, for the multilevel method is enough, in general, to prescribe the desired accuracy of the solution, and the algorithm proceeds automatically in order to meet the requirements established for the solution. Rather the classical method requires typically a continued surveillance of the underlying errors by the user, being often necessary to repeat simulations in order to satisfy the requirements demanded for the solution in terms of accuracy.

This feature of the multilevel method is of paramount importance, and it was namely one of the main goals of this paper. In fact, it was not intended to show here that the multilevel method is more efficient than the classical Monte Carlo method, which is clearly not the case in view of the results obtained, but rather to draw the attention to this inherent feature of the multilevel method. Other than facilitating the interaction of the user with the algorithm, it has a positive impact in the scalability of the algorithm when parallelized conveniently, as is also shown in this paper. To this purpose both methods have been parallelized and some examples run in a multicore architecture. The results show in general a clearly better scalability of the multilevel method compared with the classical Monte Carlo.

Finally, a further advantage of the proposed method is discussed, and lies in the potential capability of the method to compute simultaneously two different metrics of a complex networks for about the same computational cost. These are the Katz centrality, and the total subgraph communicability of a network. This is discussed to close the paper, being left as a possible future work the analysis of the associated errors.

Acknowledgments

This work was supported by Fundação para a Ciência e a Tecnologia under Grant No. UIDB/50021/2020. The author gratefully thanks Paula Marques for her suggestions and assistance with this paper.

References

[1] T. Jahnke, and C. Lubich: Error bounds for exponential operator splittings. BIT,40 (2000) 735744.

[2] R. Merris: Laplacian matrices of graphs: A survey. Linear Algebra and Its Applications, 197 (1994) 143-176.

[3] J. A. Acebrón: A Monte Carlo method for computing the action of a matrix exponential on a vector. Appl. Math. Comput., 362 (2019) 124545.

[4] J. A. Acebrón, J.R. Herrero, and J. Monteiro: A highly parallel algorithm for computing the action of a matrix exponential on a vector based on a multilevel Monte Carlo method. Submitted (2019). https://arxiv.org/abs/1904.12754

[5] D.F Anderson, and D.J. Higham: Multilevel Monte Carlo for continuous time Markov chains, with applications in biochemical kinetics. Multiscale Model. Simul., 10 (2012) 146179.

[6] M. Benzi, E. Estrada, and C. Klymko: Ranking hubs and authorities using matrix functions. Linear Algebra and Its Applications,438 (2013) 2447-2474.

[7] M. Benzi, and C. Klymko: Total communicability as a centrality measure. J. Complex Networks, 1 (2013) 124149.

[8] M. Benzi, T.M. Evans, S.P. Hamilton, M.L. Pasini, and S.R. Slattery: Analysis of Monte Carlo accelerated iterative methods for sparse linear systems. Numerical Linear Algebra with Appl., 24 (2017).

[9] F. Chung, and L. Lu: Complex Graphs and Networks. American Mathematical Society, 2006.

[10] `http://www.comsol.com/`

[11] `http://www.maths.strath.ac.uk/research/groups/numerical_analysis/contest`

[12] I.T. Dimov: Monte Carlo Methods for Applied Scientists. World Scientific, 2008.

[13] I. T. Dimov, T. T. Dimov, and T. V. Gurov: A new iterative Monte Carlo Approach for Inverse Matrix Problem. J. Comput. Appl. Math., 92 (1998) 1535.

[14] I. T. Dimov, V.N. Alexandrov, and A. Karaivanova: Parallel resolvent Monte Carlo algorithms for linear algebra problems. Mathematics and Computers in Simulation, 55 (2001) 25-35.

[15] I. Dimov, S. Maire, and J.M. Sellier: A new Walk on Equations Monte Carlo method for solving systems of linear algebraic equations. Applied Mathematical Modelling, 39 (2015) 4494-4510.

[16] T.M. Evans, S.W. Mosher, S.R. Slattery, and S.P. Hamilton: A Monte Carlo synthetic-acceleration method for solving the thermal radiation diffusion equation. J. Comput. Phys., 258 (2014) 338-358.

[17] E. Estrada, N. Hatano, and M. Benzi: The physics of communicability in complex networks. Physics Reports 514 (2012) 89-119.

[18] G. Forsythe, and R. Leibler: Matrix inversion by a Monte Carlo method. Math. Tables Other Aids Comput., 4 (1950) 127129.

[19] `http://www.mathe.tu-freiberg.de/guettels/funmkryl/`

[20] M.B. Giles: Multilevel Monte Carlo methods. Acta Numerica, 24 (2015) 259-328.

[21] M.B. Giles: Multilevel Monte Carlo path simulation. Operations Research, 56 (2008) 607-617.

[22] P. Glasserman: Monte Carlo Methods in Financial Engineering. Springer, 2004.

[23] D. Higham: An introduction to multilevel Monte Carlo for option valuation. Int. J. Comput. Math.,92 (2015) 2347-2360.

[24] N.J. Higham, and A. H. Al-Mohy: Computing matrix functions. Acta Numerica, 19 (2010) 159208.

[25] N.J. Higham, and A. H. Al-Mohy: Functions of matrices: Theory and Computation. SIAM, 2008.

[26] L. Katz: A new status index derived from sociometric analysis. Psychometrika, 8 (1953) 39-43.

[27] H. Ji, M. Mascagni, and Y. Li: Convergence Analysis of Markov Chain Monte Carlo Linear Solvers Using Ulam–von Neumann Algorithm. SIAM J. Numer. Anal., 51 (2013) 21072122.

[28] M. Mascagni, and A. Karaivanova: A parallel Quasi-Monte Carlo method for solving systems of linear equations. International Conference on Computational Science, (2002) 598-608.

[29] R.M.M. Mattheij, S.W. Rienstra, and J.H.M. ten Thije Boonkkamp: Partial Differential Equations: Modeling, Analysis, Computation. SIAM monographs,2005.

[30] G. Ökten: Solving linear equations by Monte Carlo simulation. SIAM J. Sci. Comput. 27 (2005) 511531.