A highly parallel algorithm for computing the action of a matrix exponential on a vector based on a multilevel Monte Carlo method

Juan A. Acebrón [a,b], José R. Herrero [c], and José Monteiro [b]

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

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

[c] *Dept. d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, Spain*

Abstract

A novel algorithm for computing the action of a matrix exponential over a vector is proposed. The algorithm is based on a multilevel Monte Carlo method, and the vector solution is computed probabilistically generating suitable random paths which evolve through the indices of the matrix according to a suitable probability law. The computational complexity is proved in this paper to be significantly better than the classical Monte Carlo method, which allows the computation of much more accurate solutions. Furthermore, the positive features of the algorithm in terms of parallelism were exploited in practice to develop a highly scalable implementation capable of solving some test problems very efficiently using high performance supercomputers equipped with a large number of cores. For the specific case of shared memory architectures the performance of the algorithm was compared with the results obtained using an available Krylov-based algorithm, outperforming the latter in all benchmarks analyzed so far.

Key words: Multilevel, exponential integrators, Monte Carlo method, matrix functions, network analysis, parallel algorithms, high performance computing
PACS: 65C05, 65C20, 65N55, 65M75, 65Y20

Email addresses: juan.acebron@iscte-iul.pt (Juan A. Acebrón), josepr@ac.upc.edu (José R. Herrero), jcm@inesc-id.pt (José Monteiro).

1 Introduction

In contrast to the numerical methods for solving linear algebra problems, the development of methods for evaluating function of matrices has been in general much less explored. This can be explained partially due to the underlying mathematical complexity of evaluating the function, but also under the computational point of view, because the algorithms developed so far tend to be less efficient and in general more difficult to be parallelized. In addition, related to the first issue, an added difficulty appears in estimating the associated error of the numerical method, which is well understood for solving iteratively linear algebra problems, but becomes a rather cumbersome process for functions of matrices. This is even worse in the case of the matrix exponential, due to the lack of a clear and consensually agreed notion of the residual of the iterative method, see for instance [13].

In particular the second issue represents indeed a serious drawback, since it is preventing in practice to deal with large scale problems appearing in science and engineering. Nowadays there are a plethora of applications described by mathematical models which require evaluating some type of function of matrices in order to be solved numerically. For the specific case of the matrix exponential, we can find applications in fields as diverse as circuit simulations [41]; power grid simulations [35,42]; nuclear reaction simulations [34]; analysis of transient solutions in Markov chains [36]; numerical solution of partial differential equations (PDEs) [30]; and analysis of complex networks [10], to cite just a few examples. More specifically, in the field of partial differential equations, numerically solving a boundary-value PDE problem by means of the method of lines requires in practice to compute the action of a matrix exponential using therefore exponential integrators [28]. On the other hand, in network analysis, determining some relevant metrics of the network, such as for instance the total communicability which characterizes the importance of the nodes inside the network, entails computing the exponential of the adjacency matrix of the network.

For the specific problem of computing the action of the matrix exponential over a vector several classes of numerical methods have been proposed in the literature in the last decades (see the excellent review in [26], and references therein). Probably the most analyzed and disseminated methods are those based on Krylov-based subspace methods, which use in practice a basis of a subspace constructed using the Arnoldi process, and compute the exponential of the projected matrix (typically much smaller) by using standard matrix exponential techniques [27].

An alternative to the aforementioned deterministic methods does exist, and consists in using probabilistic methods based on Monte Carlo (MC) simula-

tions. Although much less known than the former methods, the Monte Carlo methods specifically used for solving linear algebra problems have been discussed in the literature in various forms along the years. In fact, it was the seminal paper by von Neumann and Ulam during the 40's [22] that gives rise to an entire new field, and from there a multitude of relevant results, and substantial improvements of the original algorithm have appeared in the literature during the last years, see e.g. [17] and [16] for further references. Essentially the main goal is to generate a discrete Markov chain whose underlying random paths evolve through the different indices of the matrix. The method can be understood formally as a procedure consisting in a Monte Carlo sampling of the Neumann series of the inverse of the matrix. The convergence of the method was rigorously established in [29], and improved further more recently (see for instance [12], and [19] just to cite a few references).

Generalizing the method for dealing with some functions of matrices, such as the matrix exponential, was only recently accomplished in [7]. The method is based on generating random paths, which evolve through the indices of the matrix, governed now by a suitable continuous-time Markov chain. The vector solution is computed probabilistically by averaging over a suitable multiplicative functional.

The main advantages of the probabilistic methods, as it was already stated in the literature, are mainly due to its privileged computational features, such as simplicity to code and parallelize. This in practice allows us to develop parallel codes with extremely low communication overhead among processors, having a positive impact in parallel features such as scalability and fault-tolerance. Furthermore, there is also another distinguishing aspect of the method, which is the capability of computing the solution of the problem at specific chosen points, without the need for solving globally the entire problem. This remarkable feature has been explored for efficiently solving continuous problems such as boundary-value problems for PDEs in [3,5,6], offering significant advantages in dealing with some specific applications found in science and engineering.

Yet an important disadvantage of any Monte Carlo method is the slow convergence rate to the solution of the numerical method [20], being in general of order $\mathcal{O}(N^{-1/2})$, where N denotes the sample size. Nevertheless, there already exist a few statistical techniques, such as variance reduction, multilevel Monte Carlo (MLMC), and quasi-random numbers, which have been proposed to mitigate in practice such a poor performance, improving the order of the global error, and consequently the overall performance of the algorithm. Among all the aforementioned methods, the multilevel method clearly stands out, and currently it has become in fact the preferred method to speed up the convergence of a variety of stochastic simulations, with a remarkable impact on a wide spectrum of applications. An excellent review has been recently published in [24] describing in detail the method as well as a variety of applications where

it was successfully applied (see also [8] for more details specifically related with the topic of this paper).

One of the main contributions of this paper is precisely to develop a multilevel method for the problem of computing the action of a matrix exponential over a vector. This is done by conveniently adapting the probabilistic representation of the solution derived in [7] to the multilevel framework. In addition, the convergence of the method is analyzed, as well as the computational cost estimated. The second important contribution was to parallelize the resulting algorithm, and finally run successfully several relevant benchmarks for an extremely large number of processors using high performance supercomputers belonging to the top-performance supercomputers in the world (according to the well-known *TOP500* list [40]).

The outline of the paper is as follows. Briefly, the mathematical description of the probabilistic method is summarized in Sec. 2, and the problem is mathematically formalized according to the multilevel framework. In Sec. 3, the developed algorithm is described through the corresponding pseudocodes. Sec. 4 is devoted to the analysis of both, the algorithm complexity, and the numerical errors of the method. Finally in Sec. 5 several benchmarks are run to assess the performance and scalability of the method, and whenever available, a comparison with the performance obtained by the classical Krylov-based method is done. In closing, we highlight the main results and suggest further directions for future research.

2 Mathematical description of the probabilistic method and multilevel Monte Carlo method

In order to implement any multilevel Monte Carlo method it is mandatory to have a probabilistic representation of the solution. Thus, we describe next the probabilistic method used so far to compute the action of a matrix exponential over a vector.

2.1 A probabilistic method

A probabilistic representation for the action of a matrix exponential over a vector was introduced in [7] for dealing exclusively with adjacency matrices of undirected graphs. However, in the following we show that this representation can be straightforwardly generalized for dealing with arbitrary matrices.

Consider $A = (a_{ij})_{i,j=1,\dots,n}$ a general n-by-n matrix, $\mathbf{u}$ a given n-dimensional

vector, and $\mathbf{x}$ an n-dimensional vector. This vector corresponds to the vector solution after computing the action of a matrix exponential over the vector u, that is $\mathbf{x} = e^{\beta A}\,\mathbf{u}$. Here the parameter β is a constant, typically interpreted as the time variable in partial differential equations, or an effective *"temperature"* of the network in problems related with complex networks (see [21], e.g.).

Let us define a diagonal matrix D, represented hereafter as a vector $\mathbf{d}$, with entries $d_{ij} = 0\ \forall i \neq j$, $d_{ii} = d_i = a_{ii} + l_{ii}, i = 1, \ldots, n$, and a matrix T with entries t_{ij} given by

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

where σ_{ij} is a binary matrix with entries taking the value 1 when $a_{ij} < 0$, and 0 otherwise. Here $L = (l_{ij})$ denotes the 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 $l_{ii} = -\sum_{j \neq i} l_{ij}$. Then, it holds that $A = D - T$. Note that our definition differs from the classical one $A = D - L$ addressed to adjacency matrices [2]. Instead, in this paper matrix A can be any matrix. This is possible due to two changes. First, our diagonal matrix D is not a degree matrix since the diagonal term a_{ii} in the original matrix is added to the degree of the row (stored in l_{ii}). Second, we replace matrix L with matrix T, which takes into account that matrix A can have both positive and negative values unlike an adjacency matrix. Thus, 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.

Finding a probabilistic representation for this problem requires in practice [7] to use a splitting method for approximating the action of the matrix exponential over the vector $\mathbf{u}$ as follows,

$$\bar{\mathbf{x}} = \left(e^{\Delta t D/2} e^{-\Delta t T} e^{\Delta t D/2}\right)^N \mathbf{u}, \tag{2}$$

where $\Delta t = \beta/N$, which in the following and for convenience it will be termed as the time step. Note that $\bar{\mathbf{x}}$ corresponds to an approximation of the true solution $\mathbf{x}$. In fact, this corresponds to the Strang splitting method, and therefore leads to an error, which after one time step is known [1] to be of order $\mathcal{O}(\Delta t^3)$ locally, and of order $\mathcal{O}(\Delta t^2)$ globally. Therefore, the true solution is recovered in the limit $N \to \infty$.

In [7] a probabilistic representation for the particular problem consisting of adjacency matrices was derived. Here such a representation is generalized to deal with more general matrices. This is done resorting first to the following Lemma which provides a way to represent probabilistically the vector $e^{-\Delta t T} e^{\Delta t D/2}\,\mathbf{u}$.

Lemma 1 *Let $\{X_t : t \geq 0\}$ be a stochastic process with finite state space*

$\Omega = \{1, 2, \cdots, n\}$ *corresponding to a continuous-time Markov chain generated by the infinitesimal generator* $Q = -(l_{ij})$*, and final state* $X_0 = i$*. Then, any entry* i *of the vector*

$$\mathbf{y} = e^{-\Delta t T} e^{\Delta t D/2}\, \mathbf{u}, \tag{3}$$

can be represented probabilistically as $y_i = \mathbb{E}[\eta]$*, with* $\eta = \phi\, e^{\Delta t\, d_{X_{\Delta t}}/2}\, u_{X_{\Delta t}}$*, and* $\mathbb{E}[\eta]$ *its expected value. Here* ϕ *is a multiplicative random variable defined as* $\phi = \prod_{k=1}^{K-1} (-1)^{\sigma_{X_{S_{k-1}} X_{S_k}}}$ *if* $K \geq 2$*, and* $\phi = 1$ *otherwise, where* $\{S_k : k = 0, 1, \cdots\}$ *are random times defined as* $S_0 = 0$*,* $S_1 = inf\{t > 0 : X_t \neq X_0\}$*,* $S_k = inf\{t > S_{k-1} : X_t \neq X_{S_{k-1}}\}, k > 1$*, and* $K = inf\{k \geq 0 : S_k > \Delta t\}$*.*

Proof. Since D is a diagonal matrix, then y_i can be computed as follows

$$y_i = \sum_{j=1}^{n} (e^{-\Delta t T})_{ij} w_j, \tag{4}$$

where $w_j = e^{\Delta t\, d_j/2}\, u_j$. Note that the matrix $F = (f_{ij}) = (e^{-\Delta t T})_{ij}$ is the solution of the system of differential equations

$$\frac{df_{ij}}{dt} = \sum_{k=1}^{n} t_{ik} f_{kj}, \quad f_{ij}(0) = \delta_{ij}, \tag{5}$$

evaluated at time Δt, or expressed in matrix notation as

$$\frac{dF}{dt} = T\, F, \quad F(0) = \mathbb{1} \qquad (t \geq 0). \tag{6}$$

Here δ_{ij} denotes the Kronecker delta function. Using the definition of matrix T in Eq. (1), this system can be rewritten as the following system of integral equations

$$f_{ij}(t) = \delta_{ij}\, e^{-l_{ii} t} + \sum_{k \neq i} \int_0^t ds\, l_{ii}\, e^{-l_{ii}\, s}\, (-1)^{\sigma_{ik}}\, \mu_{ik}\, f_{kj}(t - s), \tag{7}$$

where $\mu_{ik} = |l_{ik}|/l_{ii}$. Note that when $\sigma_{ik} = 0$, which corresponds to matrices characterized by having all positive entries, $a_{ik} > 0$, these equations reduce to the corresponding equations for the transition probabilities of the continuous-time Markov chain solution of the Kolmogorov's backward equations,

$$\frac{dP}{dt} = Q\, P, \quad P(0) = \mathbb{1}, \tag{8}$$

for the matrix transition probability $P(t) = (p_{ij}) = \mathbb{P}(X_0 = j | X_t = i)$ [9], and infinitesimal generator $Q = -L$. For general matrices with entries of arbitrary sign this does not hold, however for this case, an alternative probabilistically representation of the solution can be established. This has been done by adapting conveniently the formalism introduced in [4] in the framework of parabolic partial differential equations for this specific matrix problem.

Let S be a continuous random variable defined on a suitable probability space, and governed by the exponential density function $p(s) = \frac{d}{ds}\mathbb{P}[s < S] = l_{ii}\, e^{-l_{ii}\, s}$, and k a discrete random variable that takes values on $\Omega = \{1, 2, \cdots, n\}$ with probability μ_{ik}. Note that $\sum_{k \neq i} \mu_{ik} = 1$ by definition of the Laplacian matrix. Then, the following probabilistic representation for $\xi_{ij} = f_{ij}(t) w_j$ is obtained

$$\xi_{ij}(t) = \delta_{ij}\, w_j\, \mathbb{P}\left[S > t\right] + \mathbb{E}\left[(-1)^{\sigma_{ik}}\, \mu_{ik}\, \xi_{kj}(t - S) \mathbb{1}_{[S \leq t]}\right]. \tag{9}$$

Here $\mathbb{E}$ is the expected value with respect to the joint distribution function of the random variables S and k, and $\mathbb{1}_A$ denotes the indicator function, being 1 or 0 depending on whether the event A occurs. Note that this system of equations is an implicit coupled system in which to evaluate ξ_{ij} for a given time t is required to evaluate ξ_{kj} at previous instants of time. This can be readily solved resorting to Picard iteration for $\xi_{ij}(t)$ as it was proposed in [4], thus obtaining a Picard series that it can be probabilistically sampled according to the following recursive algorithm:

(1) Generate a first random time S_0 obeying the exponential density function $l_{ii}\, e^{-l_{ii}\, s}$;
(2) Then, depending on whether $S_0 < t$ or not, two different alternatives are taken;
(3) If $S_0 > t$, the algorithm stops, and no jump from the state i to a different state is taken;
(4) If, on the contrary, $S_0 < t$, then the state i jumps to a different state k according to the probability function μ_{ik}, and a new second random number exponentially distributed S_1 is generated. Furthermore, the sign of the entry a_{ik} is taken into account by updating conveniently the value of the random variable ϕ;
(5) If $S_1 < (t - S_0)$ the algorithm proceeds repeating the same elementary rules, otherwise it stops.

Such a procedure generates a random path, which evolves backward in time from the state i at $t = \Delta t$ to a final state for $t = 0$, jumping randomly from i to any state on Ω governed by an exponential random time distribution with rate parameter l_{ii}, and transition probabilities between states μ_{ik}. Note that, mathematically, this corresponds to a realization of the stochastic process consisting in a continuous-time Markov chain having the infinitesimal generator $-L$. In addition, when generating the random path, and every time a jump is taken, the multiplicative random variable ϕ, which has been initialized to 1, is updated multiplying the previous obtained value by the number $(-1)^{\sigma_{ik}}$.

Intuitively, the role played by the variable ϕ in the probabilistic representation is collecting the signs of all entries a_{ij} of the matrix A multiplying conveniently its values. Here i and j correspond to the different states the random path has visited when jumping randomly through the matrix.

Therefore, from Eq. (4) and using the probabilistic representation for ξ_{ij} in Eq. (9), a probabilistic representation for y_i can be written as follows

$$y_i = \mathbb{E}[\phi\, w_{X_{\Delta t}}]. \tag{10}$$

Here $\{X_t\}$ is a random process with state space $\Omega = \{1, 2, \cdots, n\}$, whose paths corresponds to a continuous-time Markov chain generated by the infinitesimal generator $-L$, matrix transition probability $P(t)$, and with initial state $X_{\Delta t} = i$. □

It is worth to remark here the following fact concerning the matrix transition probability. Unlike what happens for the probabilistic methods developed so far for solving linear systems (see [10] e.g), where the choice for the transition probability of the corresponding Markov chain is not uniquely determined, this does not hold anymore for the proposed method. In fact, note that the transition probability of the corresponding continuous-time Markov chain $P(t)$, which is the solution of the Kolmogorov's backward equations in Eq. (8), is unique, being the solution of the Kolmogorov's backward equations in Eq. (8). Moreover, the infinitesimal generator of a continuous-time Markov chain is also unique, since by construction of the method the original matrix A has to be decomposed as $A = D - T$, and it is mandatory that D should be a diagonal matrix, and T a matrix related with the Laplacian matrix L, as it was defined in Eq. (1).

This probabilistic representation for the vector $\mathbf{y}$ can be used straightforwardly to derive the probabilistic representation for computing a single entry i of the vector solution $\bar{\mathbf{x}}$. In fact, from Eq. (2) the vector $\bar{\mathbf{x}}$ can be obtained applying the following recursive procedure

$$\begin{aligned}
&\mathbf{y}^{(1)} = e^{-\Delta t T} e^{\Delta t D/2}\, \mathbf{u},\\
&\mathbf{y}^{(k)} = e^{-\Delta t T} e^{\Delta t D}\, \mathbf{y}^{(k-1)}, \quad k = 2, \ldots, N.\\
&\bar{\mathbf{x}}_i = e^{\Delta t D/2} \mathbf{y}^{(N)},
\end{aligned}$$

Then, applying the Lemma 1 to every partial vector $\mathbf{y}^{(k)}$ we can derive the probabilistic representation, and is given by

$$\bar{x}_i = e^{\Delta t\, d_i/2} \mathbb{E}[\prod_{k=1}^{N} \eta_k], \tag{11}$$

where $\eta_k = \phi_k\, e^{\Delta t\, d_{i_k}}$, $k = 1, \ldots, N-1$, and $\eta_N = \phi_N\, e^{\Delta t\, d_{i_N}/2}\, u_{i_N}$. The i_k, $k = 1, \ldots, N$, is a sequence of N discrete random variables with outcomes on $\Omega = \{1, 2, \cdots, n\}$. The probabilities $p_{i_{k-1}\, i_k}(t)$, $k = 2, \ldots, N$, and $p_{i\, i_1}(t)$ for $k = 1$, correspond to the transition probabilities of a continuous-time Markov chain generated by the infinitesimal generator $Q = -L$ and evaluated at time Δt for each k. ϕ_k was defined already in the Lemma 1 , and in practice consists

in a two-point random variable taking values -1 and 1 according to the matrix σ_{ij}, and with a probability of occurrence governed by the transition probability P of the Markov chain. A neat picture of this probabilistic representation can be described as follows: A random path starting at the chosen entry i is generated according to the continuous-time Markov chain governed by the generator Q, and evolves in time jumping randomly from i at time t to any state on Ω. Along this process, N functions η_k are evaluated, being the solution obtained through an expected value of the multiplicative functional $\prod_{k=1}^{N} \eta_k$.

Notice that such a representation allows in practice to compute a single entry i of the vector solution, but can be conveniently modified to represent as well the full vector solution $\bar{\mathbf{x}}$. For the specific problem of solving linear systems this has been done resorting to the so-called adjoint method [10]. In our problem the procedure to follow is similar. Essentially the representation requires generating random paths, which start now at a randomly chosen state j and time $t = 0$ according to a suitable distribution $p_j^{(0)} = \mathbb{P}(X_0 = j)$, and evolve forward in time governed by a continuous-time Markov chain having now $Q^\intercal$ as the infinitesimal generator. This is partially proved by the following Lemma.

Lemma 2 *Let $\{X_t : t \geq 0\}$ be a stochastic process with finite state space $\Omega = \{1, 2, \cdots, n\}$ corresponding to a continuous-time Markov chain generated by the infinitesimal generator $Q^\intercal$, final state $X_{\Delta t}$, and initial distribution $X_0 = j$ with j randomly chosen from a given probability function $p_j^{(0)}$. Then, the vector*

$$\mathbf{y} = e^{-\Delta t T} e^{\Delta t D/2} \mathbf{u}, \tag{12}$$

can be obtained probabilistically as $y_{X_{\Delta t}} = \mathbb{E}[\eta]$, where $\eta = \phi\, e^{\Delta t\, d_{X_0}/2}\, u_{X_0}/p_{X_0}^{(0)}$ and $\mathbb{E}[\eta]$ is the expected value. Here ϕ is the multiplicative random variable defined in Lemma 1.

Proof. The proof of this Lemma is similar to Lemma 1, being the more significant difference the direction of time when generating the random paths. Essentially, it can be seen that both probabilistic representations are connected through the Bayes' theorem. In fact, applying the theorem to the transition probability $P = (p_{ij})$ in Eq. (8), we obtain

$$p_{ij} = \mathbb{P}(X_t = i | X_0 = j) \frac{\mathbb{P}(X_0 = j)}{\mathbb{P}(X_t = i)} = p_{ji} \frac{p_j^{(0)}}{p_i} = (P^\intercal)_{ij} \frac{p_j^{(0)}}{p_i}, \tag{13}$$

where $p_i = \mathbb{P}(X_t = i)$. Note that the matrix transition probability $(P^\intercal)(t)$ is the solution of the transposed equation of Eq. (8). This equation is the Kolmogorov's forward equation, and therefore the solution corresponds to a continuous-time Markov chain with generator $Q^\intercal$. Another important difference is that the initial distribution for X_0 is not a prescribed state anymore, being instead randomly chosen according to the distribution function $p_j^{(0)}$.

Concerning the final state, the random path X_t could end at any state on Ω at time Δt. If the state i is reached at time Δt for a given trial, then this path will contribute to the ith component of the vector $\mathbf{y}$ with the value η. Therefore, in practice the vector $\mathbf{y}$ is obtained probabilistically as a whole, and not componentwise as in Lemma 1, and is given by $y_{X_{\Delta t}} = \mathbb{E}[\eta]$. $\square$

Notice that the choice of the distribution function $p_j^{(0)}$ is not unique, and clearly the choice of the function may have a direct impact on the variance, and, in turn, on the performance of the algorithm. The more reasonable choice seems to be choosing $p_j^{(0)}$ proportional to $|u_j|$, since this resembles the well known importance sampling method for variance reduction, where the sampling is done according to the importance of the data. In fact, in this work we chose the probability function to be $p_j^{(0)} = |u_j|/u$, with $u = \sum_{j=1}^{n} |u_j|$. But obviously there are many other possible choices, some of them could even provide better performance results than the results already shown in Sec. 5. Therefore it is worth investigating more carefully this issue in future works, searching for an optimal probability function given the specific input vector $\mathbf{u}$. Moreover, note that when the probability function $p_j^{(0)}$ is used, the function η in Lemma 2 should be redefined as $\eta = u\,\phi\, e^{\Delta t\, d_{X_0}/2}\, \mathrm{sgn}(u_{X_0})$ to account for possible arbitrary signs in the coefficients of the vector $\mathbf{u}$.

Concerning the probabilistic representation for the full vector $\bar{\mathbf{x}}$, this can be derived using the same procedure as in (11). The contribution to every entry i of the vector is then mathematically formalized through the following representation:

$$\hat{x}_i = e^{\Delta t\, d_i/2}\, u\, \mathbb{E}[\prod_{k=1}^{N} \eta_k], \tag{14}$$

where $\eta_k = \phi_k\, e^{\Delta t\, d_{i_k}}$, $k = 1, \ldots, N-1$, and $\eta_N = \phi_N\, e^{\Delta t\, d_{i_N}/2}\, u_{i_N}$. Concerning the expected value now is taken with respect also to a random variable j on Ω governed by the probability function $p_j^{(0)}$.

In order to adapt this representation to the multilevel Monte Carlo framework, it is convenient to use the typical notation used so far in the literature. This entails rewriting the probabilistic representation for computing a single entry i of the vector solution as

$$\bar{x}_i = \mathbb{E}[P], \quad P = \prod_{j=1}^{N/2} \eta_j, \tag{15}$$

where

$$\begin{aligned} \eta_j &= \bar{\phi}_j\, e^{\Delta t(d_{i_k}/2 + d_{i_{k+1}} + d_{i_{k+2}}/2)}, \qquad j = 1, \ldots, N/2-1, \\ \eta_j &= \bar{\phi}_j\, e^{\Delta t(d_{i_k}/2 + d_{i_{k+1}} + d_{i_{k+2}}/2)}\, u_{k+2}, \; j = N/2. \end{aligned} \tag{16}$$

Here $k = 2j - 1$, $i = i_1$, and $\bar{\phi}_j = \phi_k \phi_{k+1}$. Note that this representation can be obtained by simply expanding Eq. (11),

$$\bar{x}_i = e^{\Delta t\, d_{i_1}/2} \mathbb{E}[\phi_1\, e^{\Delta t\, d_{i_1}}\, \phi_2\, e^{\Delta t\, d_{i_2}} \cdots \phi_N\, e^{\Delta t\, d_{i_N}/2}\, u_{i_N}], \tag{17}$$

then renaming the random variables as $i_1 \to i_2, i_2 \to i_3, \ldots, i_N \to i_{N+1}, i \to i_1$, and finally rearranging the expression in groups of terms as follows,

$$\begin{aligned} \bar{x}_i = \mathbb{E}\Big[\phi_1\phi_2\, e^{\Delta t(d_{i_1}/2+d_{i_2}+d_{i_3}/2)}\phi_3\phi_4\, e^{\Delta t(d_{i_3}/2+d_{i_4}+d_{i_5}/2)} \cdots \\ \times \phi_{N-1}\phi_N\, e^{\Delta t(d_{i_{N-1}}/2+d_{i_N}+d_{i_{N+1}}/2)}\, u_{i_{N+1}}\Big]. \end{aligned} \tag{18}$$

Note that this rearranging of terms reduces in practice by a half the upper limit in the product of Eq. (11).

It is worth observing here that we have to deal with two sources of error when implementing in practice the probabilistic method, that is the statistical error coming from the use of a finite sample size for estimating the expected value, and the error due to the splitting method. In fact this error can be considered as being the equivalent to the truncation error appearing in discretizing differential equations, and in the following it will be termed as truncation error.

Concerning the statistical error, and to ensure the convergence to the mean of the corresponding estimator used in our Monte Carlo simulations, it is mandatory to guarantee the finiteness of its variance, and therefore the finiteness of both moments, $\mathbb{E}[\prod_{j=1}^{N/2} \eta_j]$, and $\mathbb{E}[\prod_{j=1}^{N/2} (\eta_j)^2]$. However, in practice it is only needed to prove the finiteness of the second moment, since the finiteness of the first one follows from the finiteness of the second one. From the definition of η_j in Eq. (15), and considering the worst case scenario in which d_{i_k} is the maximum positive value of $\mathbf{d}$, say d_{max}, for all k, then it holds that

$$\prod_{j=1}^{N/2} (\eta_j)^2 = e^{2N\, \Delta t\, d_{max}}\, u_{i_{N+1}}^2 = e^{2\beta\, d_{max}}\, u_{i_{N+1}}^2 < \infty. \tag{19}$$

From here it follows the finiteness of the random variable $\prod_{j=1}^{N/2} (\eta_j)^2$ for any other possible scenario, and consequently this guarantees the convergence of the estimator for any matrix A.

2.2 An alternative probabilistic method

We discuss other possible probabilistic method to compute the action of a matrix exponential over a vector can be derived as follows. Let $\mathbf{x}(t)$ be the

solution of the system of differential equations,

$$\frac{d\mathbf{x}}{dt} = A\,\mathbf{x}, \quad \mathbf{x}(0) = \mathbf{u} \qquad (t \geq 0). \tag{20}$$

Such a system of differential equations can be rewritten in an integral form, and is given by

$$x_i(t) = e^{d_i t}\, e^{-l_{ii} t}\, u_i + \sum_{k \neq i} \int_0^t ds\, l_{ii}\, e^{d_i s}\, e^{-l_{ii}\, s}\, (-1)^{\sigma_{ik}}\, \mu_{ik}\, x_k(t-s). \tag{21}$$

The solution can be obtained recursively, replacing the solution $x_k(t-s)$ on the right-hand side with the solution $x_i(t)$, obtaining in such way an expansion in terms of multiple exponential random times, S_i. This procedure was done for the previous proposed probabilistic representation in Eq. (7), and can be applied here straightforwardly. Note, however, that the procedure now is much more involved since now the integral term contains, along with the solution itself, the time-dependent coefficient, $e^{d_i s}$. A probabilistic representation can be written as follows

$$x_i(t) = e^{d_i t}\, u_i\, \mathbb{P}\left[S > t\right] + \mathbb{E}\left[e^{d_i S}\, (-1)^{\sigma_{ik}}\, \mu_{ik}\, x_k(t-S) \mathbb{1}_{[S \leq t]}\right], \tag{22}$$

where the random variables S and k are the same variables already defined in Eq. (9). Similarly to the previous probabilistic representation, this implicit equation can be solved resorting to Picard iteration for x_i, and then sampled probabilistically using the same recursive algorithm proposed in Sec. 2.1.

Note that the main advantage of this alternative probabilistic method is the fact that the solution $x_i(t)$ can now be obtained directly at time t without the need of discretizing the time variable as required for the probabilistic representation proposed in Sec. 2.1. As a result, the corresponding numerical method is free of any truncation error due to the time discretization, remaining exclusively the statistical error as the unique source of error of the numerical method. However, some caution should be paid since other important disadvantages compared with the previous probabilistic representation may arise, and therefore it is required to be carefully investigated elsewhere.

A major concern is the fact that the coefficient $e^{d_i S}$ multiplying the solution $x_k(t-S)$ in Eq. (22) might be much greater than 1, therefore being the convergence of the numerical procedure not guaranteed. In fact, the series obtained by expanding the implicit equation could be divergent, and in general cannot be summed simply by a sequence of partial sums. A similar behavior has been already described in [5]. Moreover, the presence of such a coefficient in the probabilistic method may increase the variance of the underlying Monte Carlo algorithm, thus increasing the statistical error of the solution, and consequently degrading the computational performance of the algorithm.

Another important disadvantage of this approach lies in the fact that this probabilistic method is able to compute the solution exclusively at time t in a single evaluation, while the probabilistic method proposed in Sec. 2.1 yields in a single evaluation the values of the matrix exponential at N intermediate times. This will be explained carefully later, and has been explored specifically in Sec. 5 for solving efficiently boundary-value problems with time-dependent boundary data as it is shown in the example in Eq. (44).

Finally, we will show in the following that the multilevel Monte Carlo method based on the probabilistic representation proposed in Sec. 2.1 stands out especially in a feature that is apparently lacking for any other probabilistic methods, which is the autonomous operation of any multilevel algorithm. In fact, for the multilevel method it 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, this alternative probabilistic method may require a continued surveillance by the user of the underlying statistical errors, being often necessary to repeat simulations a few times in order to satisfy the requirements demanded for the solution in terms of accuracy.

2.3 The multilevel Monte Carlo method

The multilevel Monte Carlo method we have developed is essentially based on the well-known method many times described in the literature. In the following we introduce briefly the ideas underlying the method for those readers not familiar with the topic. For further details see the excellent survey in [24], and references therein.

Essentially, the goal of the geometric multilevel Monte Carlo method 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 $L-1$. In our specific problem this corresponds to different levels of discretization according to the value of Δt, being now $\Delta t_l = \beta/N_l$, with $N_l = 2^l$. The minimum and initial level l_0 is chosen typically to be the entire interval, that is $\Delta t_0 = \beta$. However this is not theoretically required, and for this specific problem we show in Sec. 4 that it is best not to do so. This is because the computational cost tends to be independent of the level when simulating for the coarsest level of simulation. Therefore, in the following we assume that the minimum level to be chosen is l_0. The multilevel method can be formalized mathematically through the following telescoping series,

$$\bar{x}^L = \mathbb{E}[P_L] = \mathbf{E}[P_{l_0}] + \sum_{l=l_0+1}^{L} m_l, \tag{23}$$

where $m_l = \mathbb{E}[P_l - P_{l-1}]$, $P_l = \prod_{j=1}^{N/2} \eta_j^{(l)}$, and

$$\begin{aligned}
\eta_j^{(l)} &= \bar{\phi}_j^{(l)}\, e^{\Delta t_l (d_{i_k}/2 + d_{i_{k+1}} + d_{i_{k+2}}/2)},\ j = 1, \ldots, N/2 - 1, \\
\eta_j^{(l)} &= \bar{\phi}_j^{(l)}\, e^{\Delta t_l (d_{i_k}/2 + d_{i_{k+1}} + d_{i_{k+2}}/2)}\, u_k,\ j = N/2,
\end{aligned} \tag{24}$$

with $k = 2j-1$, $i_1 = i$, and the superscript l denotes the corresponding level of discretization. Note that this induces a truncation error which is proportional to $\mathbf{E}[P_L - P_{L-1}]$. Numerically, when a finite sample of sizes $M_l, l = l_0, \ldots, L$ is used, Eq. (23) can be approximated by the following estimator

$$\bar{x}^L \approx \frac{1}{M_0} \sum_{i=1}^{M_0} P_{l_0}^{(i)} + \sum_{l=l_0+1}^{L} \frac{1}{M_l} \sum_{i=1}^{M_l} (P_l^{(i)} - P_{l-1}^{(i)}). \tag{25}$$

It is worth observing that the samples used for computing the approximation at level l are reused for computing the level $l-1$ adapting them conveniently for such a coarse level. In fact, the underlying correlation appearing between the two consecutive levels belonging to the same sample becomes essential in order to reduce the overall variance for the same computational cost. However, the final goal of the multilevel method is the opposite, that is, reducing the computational cost by choosing conveniently an optimal sample size M_l, keeping fixed the overall variance within a prescribed accuracy ε^2. After a suitable minimization process, the result as explained in [24] is given by

$$M_l = \frac{1}{\varepsilon^2} \sqrt{\frac{V_l}{C_l}} \sum_{j=l_0}^{L} V_j C_j, \tag{26}$$

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

$$C_T = \sum_{l=l_0}^{L} M_l C_l, \qquad V_T = \sum_{l=l_0}^{L} \frac{V_l}{M_l}, \tag{27}$$

3 The multilevel algorithm

To implement in practice the multilevel method for computing the action of the matrix exponential over a vector, it is first necessary to introduce a suitable algorithm capable of generating efficiently the random paths. Second, we need to describe the strategy followed to compute the difference between any two consecutive levels as appears in Eq. (25). This requires an efficient technique to reuse the paths obtained when simulating with a higher level l for the lower level at discretization $l-1$.

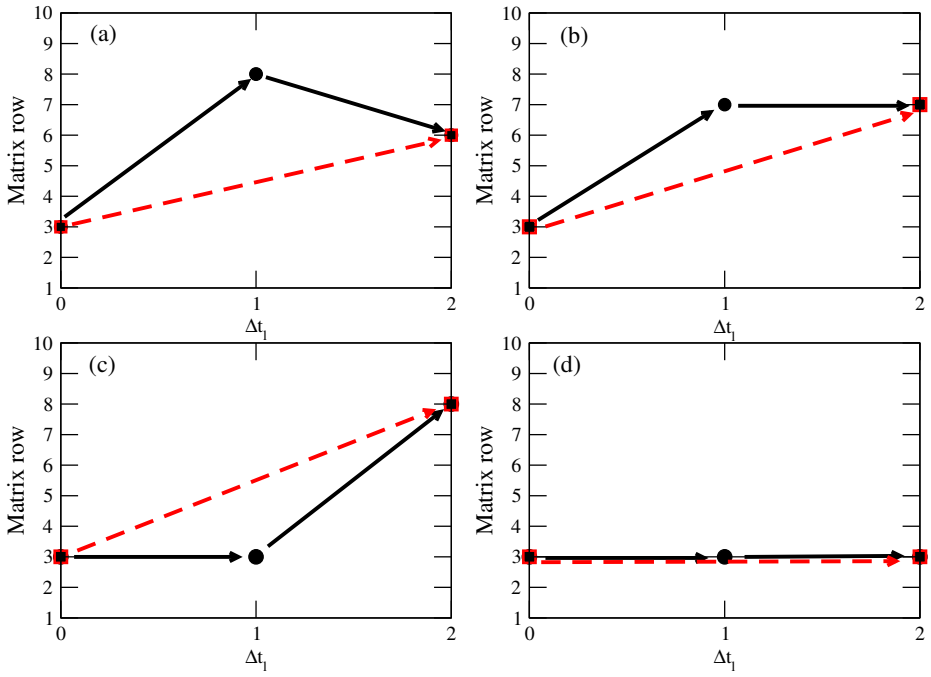

Fig. 1. Sketch diagram showing the four possible sampled paths obtained for level $l = 2$, and for a matrix of size $n = 10$. The solid line corresponds to a random path obtained for a level number l, and the dotted line with $l - 1$.

Concerning the first issue, it was already described in Sec. 2, and in Fig. 1 a sketch diagram for the case of $l = 2$ is shown.

This illustrates graphically how the second issue, related to the computation of the coarse level $l - 1$ using the higher level l, has been solved in practice. There we plot the four different scenarios that may occur when generating random paths (assuming we are interested in computing only a single entry i of the vector solution, and therefore forcing all random paths to start at the same state i). Thus, from Eq. (15), 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. But 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 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 $\mathbb{E}[P_2 - P_1]$ in (23).

In Algorithm 1, we describe a pseudocode corresponding to the implemen-

Algorithm 1. Multilevel Monte Carlo (MLMC) algorithm.
INPUT: $L = l_0 + 4$, $M = M_0$, i, N, ε, β
Call **MLMCL**$(i, \Delta t_l, N, M_0)$ for fast estimating m_l and V_l for $l = l_0, \ldots, L$
while *error* $\geq \varepsilon$ **do**
Compute the optimal number of samples M_l for $l = l_0, \ldots, L$
Call **MLMCL**$(i, \Delta t_l, N, M_l)$ 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

tation of the multilevel method. In fact, this consists in the general setting for any implementation of the method for a variety of problems. The distinguishing feature among them is the suitable procedure chosen to compute in practice any of the terms of the expansion in Eq. (23), as well as the associated variances. The pseudocode of the procedure for computing a single entry of the vector solution is described in Algorithm 2.

Although the multilevel method could be used to compute the full vector solution as well, the implementation is much more involved and the performance of the algorithm less efficient. This is because it will require in practice to save vectors instead of scalars for any of the levels in Eq. (23). This can be mitigated instead by computing a scalar function of the full vector solution, and since the complexity of the algorithm for computing the full vector solution by Monte Carlo is similar to that for obtaining the solution of a single entry, in principle the computation time of the multilevel method for the former case should be comparable. In fact, the pseudocode is similar (see Eq. (11) and Eq. (14)).

4 Convergence and Computational complexity of the multilevel algorithm

The computational complexity of any MLMC algorithm can be established properly resorting to Theorem 1 in [24]. However, it is mandatory to characterize previously the convergence of some important quantities such as the mean $|\mathbb{E}[P_l - P]|$ and variance $V[P_l - P_{l-1}]$, as well as the computational time of the Monte Carlo algorithm, as a function of the level l.

Concerning the scaling of the mean $|\mathbb{E}[P_l - P]|$ with the level l, it can be readily estimated as follows. Since $\mathbb{E}[P]$ corresponds to the theoretical solution, $\mathbf{x} = e^{\beta A}\,\mathbf{u}$, obtained probabilistically in practice when $N \to \infty$, $|\mathbb{E}[P_l - P]|$ corresponds in fact to the truncation error $|\mathbb{E}[P_l] - x]|$. Recall that this was

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

procedure MLMCL($i, \Delta t_l, N, M$)
  $m_l = 0$, $m2l = 0$
  for $l = 1, M$ **do**
    $\eta_1 = 1$, $\eta_2 = 1$, $j = i$
    for $n = 1, \ldots, N$ **do**
      $\eta_2 = \eta_2 e^{d_j \Delta t_l/2}$
      if $n \, mod \, 2 \neq 0$ **then**
        $\eta_1 = \eta_1 e^{d_j \Delta t_l}$
      end if
      generate τ exponentially distributed
      while $\tau < \Delta t_l$ **do**
        generate S exponentially distributed
        $k = j$
        generate j according to Eq.(7)
        $\tau = \tau + S$
        $\eta_2 = (-1)^{\sigma_{kj}} \eta_2$
        $\eta_1 = (-1)^{\sigma_{kj}} \eta_1$
      end while
      $\eta_2 = \eta_2 e^{d_j \Delta t_l/2}$
      if $n \, mod \, 2 = 0$ **then**
        $\eta_1 = \eta_1 e^{d_j \Delta t_l}$
      end if
    end for
    $m_l = m_l + [u_j(\eta_2 - \eta_1)]/M$
    $m2l = m2l + [u_j(\eta_2 - \eta_1)]^2/M$
  end for
  $V_l = m2l/M - m_l^2$
  return (m_l, V_l)
end procedure

considered previously as being due to the Strang splitting method. Therefore, the local error after one time step ε_S of this approximation is known [1] to be

$$\varepsilon_S = \Delta t_l^3 \left(\frac{1}{12}[D,[D,T]] - \frac{1}{24}[T,[T,D]]\right) u + \mathcal{O}(\Delta t_l^4), \tag{28}$$

and globally of order $\mathcal{O}(\Delta t_l^2)$, where $[\cdot,\cdot]$ denotes the commutator of the two matrices, defined as $[A,B] = A\,B - B\,A$. This is in agreement with Fig. 2(a), where the mean $|\mathbb{E}[P_l - P_{l-1}]|$ is plotted as a function of the level l for the example consisting in simulations of a small-world network of three different sizes.

Characterizing the variance $V[P_l - P_{l-1}]$ as a function of the level l turns out

to be a much more involved procedure. To start, it holds that

$$V[P_l - P_{l-1}] = \mathbb{E}[(P_l - P_{l-1})^2] - (\mathbb{E}[P_l - P_{l-1}])^2 \leq \mathbb{E}[(P_l - P_{l-1})^2]. \quad (29)$$

Hence, the problem can be reduced to the problem of estimating $\mathbb{E}[(P_l - P_{l-1})^2]$. For this purpose, and as a preliminary step, it will be estimated next a partial result regarding the random variable $\eta_2^{(l)}$ and $\eta_1^{(l-1)}$, and then the final result will be estimated accordingly. These random variables are obtained when generating paths for a single time step (when the level is $l-1$), and two consecutive time steps (when the level is l). The subscripts 2 and 1 denote two and one consecutive steps respectively. Therefore, we first establish the following Lemma.

Lemma 3 *Let j and k discrete random variables that take values on $\Omega = \{1, 2 \cdots, n\}$, with probability $p_{ij}(t)$ and $p_{jk}(t)$ given by the transition probabilities of a continuous-time Markov chain generated by the infinitesimal generator $Q = -(L)_{ij}$ and evaluated at time Δt_l. Then, it holds that*

$$\mathbb{E}[(\eta_2^{(l)} - \eta_1^{(l-1)})^2] = \mathcal{O}(\Delta t_l^3), \quad (30)$$

where $\eta_2^{(l)} = e^{\Delta t_l\, d_i/2} e^{\Delta t_l\, d_j} e^{\Delta t_l\, d_k/2}\, u_k$, and $\eta_1^{(l-1)} = e^{\Delta t_l\, d_i} e^{\Delta t_l\, d_k}\, u_k$, respectively.

Proof. Expanding $\eta_2^{(l)}$ and $\eta_1^{(l-1)}$ in powers of Δt_l yields

$$\mathbb{E}[(\eta_2^{(l)} - \eta_1^{(l-1)})^2] = \Delta t_l^2 \mathbb{E}[\xi^2] + \mathbb{E}[\mathcal{O}(\Delta t_l^3)], \quad (31)$$

where $\xi = (-d_i/2 + d_j - d_k/2)u_k$. The possible outcomes of the random variables j, and k can be one of the following four different cases: (a) $j \neq k \neq i$; (b) $j \neq i, k = j$; (c) $j = i, k \neq i$ and (d) $j = k = i$ (see Fig. 1 for illustration). To distinguish among them, consider one pair of binary variables (α_1, α_2), taking values $\{(0,0), (0,1), (1,0), (1,1)\}$ and corresponding to the cases a, b, c, and d, respectively. From the transition probabilities of the corresponding continuous-time Markov chain, the probability of obtaining each of them is given by

$$p_{\alpha_1\alpha_2} = [\alpha_1\, e^{\Delta t_l\, d_i} + (1-\alpha_1)(1 - e^{\Delta t_l\, d_i})][\alpha_2\, e^{\Delta t_l\, d_j} + (1-\alpha_2)(1 - e^{\Delta t_l\, d_j})] \quad (32)$$

It is worth observing that any multiple transitions within time Δt_l are not considered in this calculation, since it can be seen that they contribute to a higher order $\mathcal{O}(\Delta t_l^2)$ to the transition probability.

By expanding $p_{\alpha_1\alpha_2}$ in powers of Δt_l, we have

$$p_{\alpha_1\alpha_2} = \alpha_1\, \alpha_2 + C\, \Delta t_l + \mathcal{O}(\Delta t_l^2), \quad (33)$$

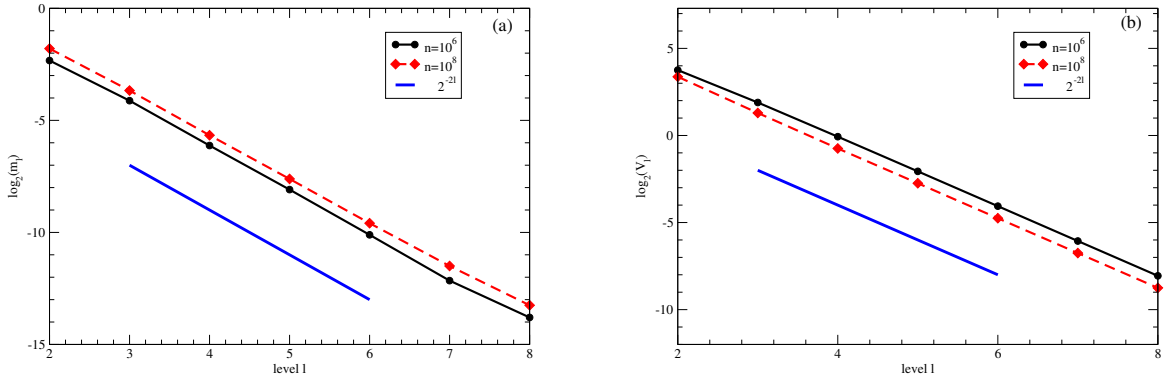

Fig. 2. (a) Mean m_l and (b) variance V_l of $P_l - P_{l-1}$ in log_2 scale versus the level number l obtained numerically. The adjacency matrix corresponds to a small-world network of different sizes n. The blue line denotes an ancillary function of slope -2.

where C depends merely on d_i, d_j, d_k. Note that for the case (d), which corresponds to $(\alpha_1, \alpha_2) = (1, 1)$, ξ turns out to be 0, therefore it follows that $\mathbb{E}[(\eta_2^{(l)} - \eta_1^{(l-1)})^2]$ is $\mathcal{O}(\Delta t_l^3)$ and the proof is complete. □

To find the global convergence rate, consider the following Lemma.

Lemma 4 *Assume i_k, with $k = 1, \cdots, N$, are N discrete random variables taking values on $\Omega = \{1, 2 \cdots, n\}$, with probability $p_{i_k\, i_{k+1}}(t)$ given by the transition probabilities of a continuous-time Markov chain generated by the infinitesimal generator $Q = -(L)_{ij}$ and evaluated at time Δt_l. Then, it holds that*

$$\mathbb{E}[(P_l - P_{l-1})^2] = \mathcal{O}(\Delta t_l^2) \tag{34}$$

Proof. Expanding $\eta_j^{(l)}$ and $\eta_j^{(l-1)}$ in Eq. (24) in powers of Δt_l, we have

$$\mathbb{E}[(P_l - P_{l-1})^2] = \Delta t_l^2 \mathbb{E}[(\sum_{j=1}^{N_l/2} \xi_j)^2] + \mathcal{O}(\Delta t_l^3), \tag{35}$$

where $\xi_j = (d_{i_k}/2 + d_{i_{k+1}} + d_{i_{k+2}}/2) - (d_{i_k} + d_{i_{k+2}})$, and $k = 2j - 1$. We can estimate the probability of occurrence of the event q characterized by the value $\sum_{j=1}^{N_l/2} \xi_j = 0$. This event may occur when zero transitions took place during time t. Recall that we are ignoring here, for the same reason pointed out in Lemma 4, the contribution to the probability of multiple transitions leading to a final state equal to the initial one. Therefore, assuming the initial state to be i, the probability of occurrence of the event q is given by

$$\mathbb{P}(q) = \prod_{k=1}^{N_l} e^{-\Delta t_l\, l_{ii}} = e^{-N_l \Delta t_l\, l_{ii}}, \tag{36}$$

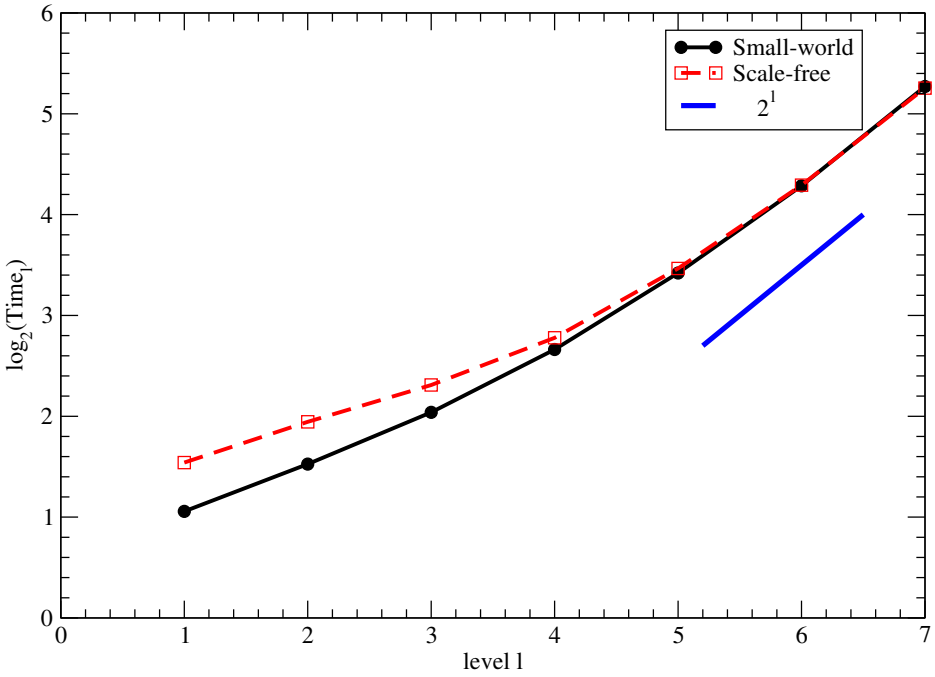

Fig. 3. Computational time in log_2 scale versus the level l for adjacency matrices corresponding to two different complex networks of size $n = 10^6$. The blue line corresponds to an ancillary function of slope 1.

which is independent of Δt_l, since $N_l = \beta/\Delta t_l$, and moreover for $\beta \neq 0$ strictly different from 1. Therefore, when $\Delta t_l \to 0$ we have a non-zero probability of obtaining non-zero values for $\sum_{j=1}^{N_l/2} \xi_j$. Then, $\mathbb{E}[(\sum_{j=1}^{N_l/2} \xi_j)^2]$ cannot be zero, and consequently, from Eq. (35) it follows that $\mathbb{E}[(P_l - P_{l-1})^2]$ should be of order $\mathcal{O}(\Delta t_l^2)$. □

In Fig. 2(b), $V[P_l - P_{l-1}]$ is shown as a function of the level l. The adjacency matrices correspond to a small-world network of three different sizes. Note that the obtained numerical convergence rate fully agrees with the theoretical estimation.

The computational time of the Monte Carlo algorithm was already estimated in [7], and it is given by

$$T_{CPU} = \alpha_{in}\beta\bar{d}\,M + \alpha_{out}\frac{\beta}{\Delta t_l}\,M. \tag{37}$$

Here $\bar{d}$ is $\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$, while α_{in} and α_{out} are suitable proportionality constants. In Fig. 3, the results corresponding to the CPU time spent by the Monte

Carlo algorithm when computing the total communicability of two different networks characterized by different values of $\bar{d}$ is shown. The results are in agreement with the theoretical estimation in Eq. (37). In particular, 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 $1/\Delta t_l$. This also explains what was mentioned previously in Sec. 2, which is that the initial level l_0 of the multilevel method could be different from zero to obtain a better performance of the MLMC algorithm. In fact, depending on the value of Δt_l and consequently on the level l, two different working regimes can be observed, and only for the regime characterized by a value of Δt_l sufficiently small, the computational time asymptotically increases with l. Specifically this occurs when the contribution to the computational time of the second term in Eq. (37) is much larger than the first term. Assuming that the value of the proportionality constants α_{in}, α_{out} are similar, we can readily estimate the minimum value of the level needed for this purpose, and is given by

$$l_0 \gg log_2(\beta \bar{d}). \tag{38}$$

However, in general both constants α_{in}, and α_{out} are not only different, but also difficult to be theoretically estimated. From numerical simulations, however, a more practical lower bound has been found and is given by

$$l_0 = log_2(2\beta d_{max}), \tag{39}$$

where d_{max} corresponds to the maximum value of the diagonal matrix D. In Fig. 4 the computational time spent by the MLMC method for different values of the initial level l_0 is shown. Here the MLMC was applied to the problem of computing the total communicability [11] of two different networks, small-world and scale-free, of size $n = 10^6$. For the small-world network the maximum degree is 7, while for the scale-free network is 3763. The value of β was chosen to be 1 for the small-world network and $1/d_{max}$ for the scale-free network. The last one was chosen specifically to ensure the convergence of the method, as it was pointed out in [7]. Note that, for both networks, the computational time attains a minimum at a specific value of l_0, which is well approximated by Eq. (39).

In view of the convergence rates estimated above, we can apply the aforementioned Theorem 1 in [24] and conclude that the computational complexity of the proposed MLMC algorithm is of order $\mathcal{O}(\varepsilon^{-2})$. In practice this means that the error due to the splitting in Eq. (2) is totally canceled out from the algorithm, remaining only the computational cost inherent to any Monte Carlo method due to the statistical error. Rather, the complexity of the classical Monte Carlo algorithm proposed in [7] is of order of $\varepsilon^{-5/2}$. Indeed this can be readily proved as follows. Concerning the statistical error, the sample size M required to achieve a prescribed accuracy ε is given by $M = \varepsilon^{-2}$, while for the splitting error, being the method of order of Δt^2, the time step required

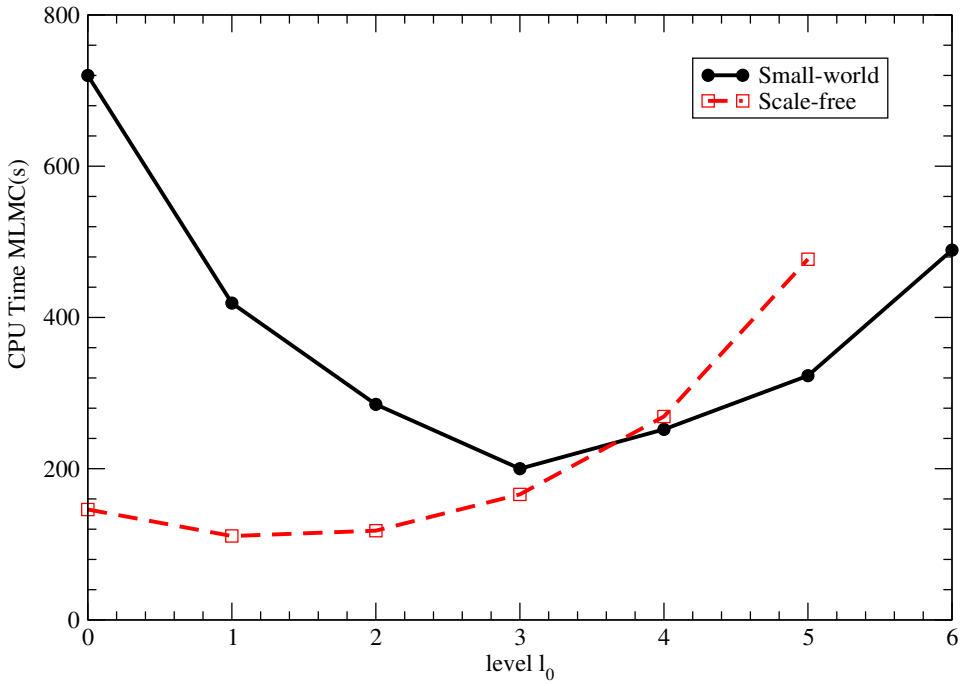

Fig. 4. Computational time of the MLMC method for different values of the initial level l_0. The matrices correspond to the adjacency matrices of two different complex networks of size $n = 10^6$. The accuracy for the small-world network is $\varepsilon = 6.25 \times 10^{-4}$, while for the scale-free network is $\varepsilon = 2 \times 10^{-7}$.

for a given ε is $\Delta t = \varepsilon^{1/2}$. Therefore, the computational complexity, which depends on $M/\Delta t$, is given by $\mathcal{O}(\varepsilon^{-5/2})$. In Fig. 5 the results corresponding to the computational time spent to compute the communicability of a single node of a small-world network of size $n = 10^6$ are plotted as a function of a chosen prescribed accuracy ε for both, the multilevel Monte Carlo and the classical Monte Carlo method. Note the perfect agreement with the theoretical estimates, and the performance notably superior to the classical Monte Carlo method in [7] for lower accuracy values.

5 Performance evaluation

To illustrate the performance of the multilevel Monte Carlo method, in the following we show the results corresponding to several benchmarks conducted so far. They concern the numerical solution of a linear parabolic differential equation by means of an exponential integrator, as well as, the numerical

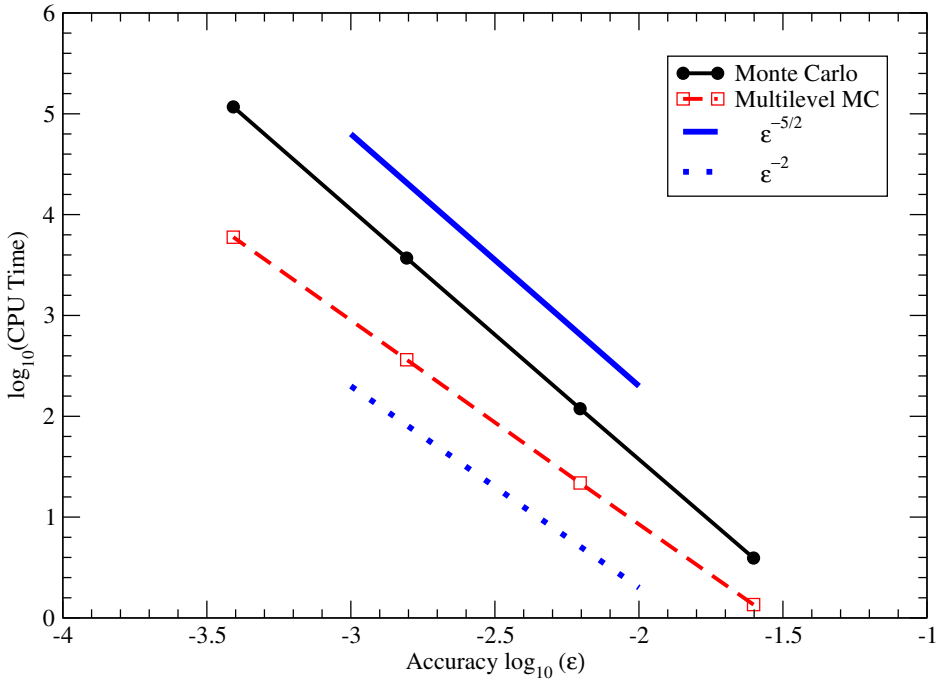

Fig. 5. Computational time as a function of the prescribed accuracy ε, both in log_{10} scale. The blue solid line corresponds to an ancillary function of slope $-5/2$, while the dotted line to a function of a slope -2. The results correspond to the communicability for a single node of a small-world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

computation of the total communicability metric in some complex synthetic networks.

In fact, among an important application of the multilevel algorithm for computing the action of a matrix exponential, we have the numerical solution of parabolic PDEs by means of the method of lines, using therefore an exponential integrator. When the method of lines [31] is applied to an initial parabolic PDE problem discretizing the spatial variable, a system of coupled ordinary differential equations, with time as the independent variable, is obtained. Finally, the system can be solved resorting to the computation of a matrix exponential which acts on the discretized initial value function. The method was here applied for solving the Dirichlet boundary-value problem for both, a 3D heat equation, and a 3D convection-diffusion equation. The former problem is given by

$$\frac{\partial u}{\partial t} = \nabla^2 u, \quad \text{in } \Omega = [-\delta, \delta]^3, t > 0, \tag{40}$$

with boundary- and initial-conditions

$$u(\mathbf{x},t)|_{\partial\Omega} = 0, \qquad u(\mathbf{x},0) = f(\mathbf{x}). \tag{41}$$

The approximated solution $\hat{u}(\mathbf{x},\mathbf{t})$ where $\mathbf{x} \in \mathbf{R}^3$, $\mathbf{x} = (x,y,z)$, after discretizing in space with grid spacing $\Delta x = \Delta y = \Delta z = 2\delta/n_x$, and using the standard $7-$point stencil finite difference approximation, can be written formally as

$$\hat{u}(\mathbf{x},t) = e^{\frac{n_x^2 t}{4\delta^2}\hat{L}}\,\hat{u}_0(\mathbf{x}), \tag{42}$$

where $\hat{L}$ denotes the corresponding discretized Laplacian operator.

This problem is specially relevant, because the eigenvalues of the matrix $A = n_x^2/4\delta^2\hat{L}$ can be analytically obtained [38]. In fact, the largest eigenvalue is known to be $\lambda_{max} = 6n_x^2/4\delta^2(1-\cos\frac{n_x\pi}{n_x+1})$, while the smallest one is $\lambda_{min} = 6n_x^2/4\delta^2(1-\cos\frac{\pi}{n_x+1})$. Therefore, we can compute readily the so-called stiffness ratio $r = \lambda_{max}/\lambda_{min}$ for this problem, and asymptotically for large values of n_x is given by $r \sim 4n_x^2/\pi^2$. Note that in view of the values of n_x used for the simulations in Tables 1 and 2, this problem represents indeed a suitable example to test the performance of the method for solving stiff problems.

Concerning the convection-diffusion equation, mathematically we have

$$\begin{aligned} &\frac{\partial u}{\partial t} = \nabla^2 u + \beta\cdot\nabla u, \qquad \mathbf{x}\in\Omega, t>0,\\ &u(\mathbf{x},t)|_{\partial\Omega} = g(\mathbf{x},\mathbf{t}),\\ &u(\mathbf{x},0) = f(\mathbf{x}), \end{aligned} \tag{43}$$

where β is the velocity field. After applying the standard Galerkin finite element method [43] to the discretized nodes $\mathbf{x}_i, i = 1,\ldots,n$, the following linear system of coupled first order ODEs is obtained

$$M\frac{d\mathbf{u}}{dt} = K\mathbf{u} + \mathbf{F}, \quad \mathbf{u}(0) = \mathbf{u}_0, \tag{44}$$

where $\mathbf{u} = (u(\mathbf{x}_1,t),\ldots,u(\mathbf{x}_n,t))$, M is the assembled mass matrix, K is the corresponding assembled stiffness matrix, and $\mathbf{F}$ is the load vector. Concerning the boundary data, these are included modifying as usual the matrices and the vector. For computational convenience, in the following the mass matrix was lumped [43], resulting in practice in a diagonal mass matrix.

Formally, the solution of the inhomogeneous system of ODEs (44) can be written in terms of a matrix exponential as follows

$$\mathbf{u}(\mathbf{x},t) = e^{-t\,M^{-1}K}\mathbf{u}_0 + \int_0^t ds\, e^{-s\,M^{-1}K}\mathbf{F}(t-s). \tag{45}$$

Note that for the particular case of having time-independent boundary data, the load vector becomes therefore constant, and the solution simplifies to

$$\mathbf{u}(\mathbf{x},t) = e^{-t\,M^{-1}K}\mathbf{u}_0 - K^{-1}M\left(e^{-t\,M^{-1}K} - \mathbb{1}\right)\mathbf{F}. \tag{46}$$

On the other hand, for arbitrary time-dependent boundary data, the integral in Eq. (45) can be computed resorting to suitable numerical quadratures. This procedure can be followed in any case to avoid evaluating the inverse of the matrix K in Eq. (46), which in general can be computationally costly. In fact, this was specifically used here for solving numerically the system of equations in (44). Since to compute the matrix exponential the error was estimated to be of order $\mathcal{O}(\Delta t^2)$ (see Sec. 4), to avoid lowering down this order, in the following we have implemented the Simpson quadrature rule, which is known to be of much higher order. More specifically, the solution $\hat{\mathbf{u}}(\mathbf{x},t)$ can be computed as follows

$$\begin{aligned}\hat{\mathbf{u}}(\mathbf{x},t) = e^{-t\,M^{-1}K}\mathbf{u}_0 + \Delta t\Bigg(&\mathbf{F}(t) + 2\sum_{j=1}^{N/2-1} e^{-t_j\,M^{-1}K}\mathbf{F}(t-t_j) \\ &+4\sum_{j=1}^{N/2} e^{-t_j\,M^{-1}K}\mathbf{F}(t-t_j) + e^{-t\,M^{-1}K}\mathbf{F}(0)\Bigg),\end{aligned} \tag{47}$$

where $t_j = j\Delta t$, $j = 1,\ldots,N$, and $\Delta t = t/N$. Note that we require to compute N independent matrix exponential evaluations at N different instants of time. However, it turns out that using the Algorithm 2, in practice only a single evaluation at the final time t is needed to compute. This is because when using the Monte Carlo method for computing the matrix exponential at time t, the information required to evaluate the matrix exponential at intermediate times have been also automatically generated by the algorithm. In fact, the random paths generated up to time t, which have been simulated advancing in time steps of size Δt, can be used directly to evaluate the matrix exponential over the vector $\mathbf{F}$ at time $j\Delta t$. Moreover, it should be stressed that this can be accomplished without any additional computational cost.

In practice this can be readily done modifying slightly the Algorithm, as it is shown in boldface in the new Algorithm 3. Here ω_n is a vector containing the suitable weights $(1,2,4,\ldots,2,4,1)$ corresponding to the Simpson quadrature rule.

Other important application of the matrix exponential consists in computing the total communicability of a network. By definition, the total communicability of a network [11] is given by

$$TC = (\mathbf{1},\, e^{A}\,\mathbf{1}), \tag{48}$$

where $\mathbf{1}$ is a vector of ones, and $(\cdot,\cdot)$ denotes the scalar product. In the fol-

Algorithm 3. Procedure to compute a single entry i of the vector solution $\hat{\mathbf{u}}(\mathbf{x}, t)$

procedure MLMCL-FEM$(i, \Delta t_l, N, M)$
  $m_l = 0$, $m2_l = 0$,$integ_1 = 0$,$integ_2 = 0$
  for $l = 1, M$ **do**
    $\eta_1 = 1$, $\eta_2 = 1$, $j = i$
    for $n = 1, \ldots, N$ **do**
      $\eta_2 = \eta_2 e^{d_j \Delta t_l/2}$
      if $n \, mod \, 2 \neq 0$ **then**
        $\eta_1 = \eta_1 e^{d_j \Delta t_l}$
      end if
      generate τ exponentially distributed
      while $\tau < \Delta t_l$ **do**
        $k = j$
        generate S exponentially distributed
        generate j according to Eq.(7)
        $\tau = \tau + S$
        $\eta_2 = (-1)^{\sigma_{kj}} \eta_2$
        $\eta_1 = (-1)^{\sigma_{kj}} \eta_1$
      end while
      $\eta_2 = \eta_2 e^{d_j \Delta t_l/2}$
      if $n \, mod \, 2 = 0$ **then**
        $\eta_1 = \eta_1 e^{d_j \Delta t_l}$
      end if
      $\mathbf{integ_1 = integ_1 + \omega_n \eta_1}$
      $\mathbf{integ_2 = integ_2 + \omega_n \eta_2}$
    end for
    $\mathbf{m_l = m_l + [u_j(\eta_2 - \eta_1)]/M + [F_j(integ_2 - integ_1)]/M}$
    $\mathbf{m2_l = m2_l + [u_j(\eta_2 - \eta_1)]^2/M + [F_j(integ_2 - integ_1)]^2/M}$
  end for
  $V_l = m2_l/M - m_l^2$
  return (m_l, V_l)
end procedure

lowing we analyze the total communicability for several networks consisting in generated synthetic networks of the type small-world and scale-free of arbitrary size. These networks have been generated in Matlab using the functions *smallw* and *pref*, respectively, both freely available through the toolbox CONTEST [15]. In contrast to the small-world network, the scale-free networks are characterized by the presence of hubs, which in practice entail a much larger largest eigenvalue than for the small-world networks. Then, since the value of this eigenvalue increases with the network size, and in order to keep constant the numerical error, it may be necessary for the MLMC method to increase strongly the number of required levels accordingly. To prevent such a computationally costly procedure, a reasonable alternative relies on computing a

generalization of the communicability, that is $e^{\beta A}$, where β is typically interpreted as an effective "temperature" of the network (see [21], e.g.). Essentially the idea that was exploited in [7] was to use the inverse of the largest eigenvalue as the value of the parameter β, which in practice will control the rapid growth of the norm of the matrix A with the size of the network. Different values of β could have a direct impact not only on the entries of the communicability vector, but also on the ranking of the nodes according to their communicability values. However, in practice this does not occur. Through the analysis of the intersection similarity of several networks [7] it was shown that the chosen value of β does not affect significantly the results, being in all cases the differences well below the typical error tolerances, and even becoming smaller for increasingly larger network sizes. Consequently, and to ensure fast convergence of the method, in the simulations below we have used $\beta = 1/\lambda_{max}$, where λ_{max} is the largest eigenvalue of A. However, finding the largest eigenvalue for large networks is itself computationally costly and, in the following, a faster alternative based on computing the maximum degree of the network, d_{max}, was used instead as an upper bound value.

5.1 Shared memory architecture

The simulations corresponding to the shared memory architecture were run on both a commodity server equipped with 12 cores and 32 GB of RAM, and the MareNostrum supercomputer using a single node with 48 cores. The MLMC algorithm has been implemented in OpenMP, and the *SPRNG* library [37] for the parallel random number generator. To compare the performance with other methods, as well as to control the numerical errors, the MATLAB toolbox *funm−kryl* freely available in [23] has been used. This method consists in the implementation of a Krylov subspace method with deflated restarting for matrix functions [27]. Note that Matlab was originally written in C/C++ and, specifically, operations involving matrix-vector multiplication or matrix-matrix multiplication show nowadays an optimal performance in the latest versions of Matlab, since they are exploiting very efficiently multithreading execution as well as SIMD units available in current microprocessors. Taking into account that the Krylov subspace method requires matrix-vector multiplications extensively, we assume the obtained performance of the Matlab code to be more than competitive with respect to the performance of a native code in C/C++. Moreover, our implemented OpenMP code was not optimized to ensure a fair comparison with Matlab. Finally, it is worth remarking that the choice for using Matlab for comparison and not a native code was essentially motivated by the lack of any parallel code freely available in C/C++.

Example A: Partial Differential equations.

The computational time spent by both, the MC and MLMC method, for solving the initial-boundary value problem for a 3D heat equation at a single point is shown in Tables 1 and 2. This has been done for different matrix sizes and number of cores running on the commodity server, and for about the same accuracy.

It is worth noting that in view of the probabilistic nature of any Monte Carlo-based algorithm, the measured computational times cannot be uniquely defined. For that reason the simulations have been repeated a few times, reporting therefore in the corresponding tables both, the mean value and the value of the two standard deviation (95% confidence interval) in parenthesis. However, this has been done exclusively for the simulations run using the commodity server, since for the other simulations run using supercomputers (MareNostrum and Marconi) the computational time was measured only once in a single simulation due to the limited CPU-time available during the course of this work.

Concerning the error, this was estimated using the aforementioned Krylov-based method by setting a very small value of the stopping-accuracy parameter, 10^{-16}, as well as the restart parameter to 40.

For comparison, the computational time spent by Matlab is also shown only for the smaller matrix size, since for the larger one Matlab simulations run out of memory. As it was pointed out in [7] this is mainly due to the memory demands of any Krylov-based algorithm. Instead, the Monte Carlo method is extremely efficient in terms of memory management, since it requires only to allocate in memory the input matrix.

Table 1
Elapsed time spent for computing the solution of the 3D heat equation at the single point $(0, 0, 0)$, and for time $t = 1$ as a function of the number of cores. The initial value function was $f(\mathbf{x}) = e^{-(x^2+y^2+z^2)}$. The accuracy was kept fixed to 5×10^{-4}. The length of the domain was $\delta = 4$ and the number of grid points was n_x^3, with $n_x = 256$.

Cores	**Time MC (s)**	**Time MLMC (s)**	**Time Matlab (s)**
1	231(1)	164(5)	428
4	66(3)	46(2)	306
8	33(1)	24(2)	327
12	22(1)	16(1)	334

For the solution of the convection-diffusion equation, the arbitrary complex geometry plotted in Fig. 6 was used as the domain, being the Dirichlet boundary data chosen to be $u = 0$ at the surface of the outer sphere, and $u = 1$ at the surface of the inner cylinder. The size of the domain can be conveniently increased by simply rescaling both, the sphere and cylinder, using a single

Table 2
Elapsed time spent for computing the solution of the 3D heat equation at the single point $(0, 0, 0)$, and for time $t = 1$ as a function of the number of cores. The accuracy was kept fixed to 5×10^{-4}. The length of the domain was $\delta = 4$ and the number of grid points was n_x^3, with $n_x = 512$.

Cores	**Time MC (s)**	**Time MLMC (s)**
1	1120(1)	786(10)
4	316(5)	227(6)
8	164(2)	117(6)
12	110(2)	78(2)

scale parameter *scale*. To generate the computational mesh, and obtaining the corresponding FEM matrices and vector, the scientific software *COMSOL* [14] was used, choosing specifically linear elements at the discretization setting. Concerning the element size used when meshing the geometry, it was kept fixed to be 0.8 and 0.14 for the maximum and minimum size, respectively.

It is worth to remark here that this is the more general example, consisting in an unsymmetric matrix with entries of arbitrary sign.

To gain some insight about the properties of the underlying matrix used in this example, it can be useful to analyze the largest and smallest eigenvalue of the matrix, as it was done for the previous example. However, it turns out that there is no available analytical solution for this problem, being required therefore to resort to some numerical approximations. This was done specifically for this problem using the Arnoldi method, which has been implemented in Matlab and executed through the command *eigs*. In general, especially for large matrices, it is known that this could be a formidable numerical task in itself. In fact, here it was only possible to compute such eigenvalues for the smallest problem, which corresponds to the size domain denoted as $scale = 4$ in Table 4. The real part of the eigenvalues, since the matrix is unsymmetric, are given by $\mathrm{Re}(\lambda_{max}) = 2.08 \times 10^3$, $\mathrm{Re}(\lambda_{min}) = 3.47$.

The computational time spent for computing the solution at a single point and time inside the domain for different number cores is shown in Table 3. Note that both the MC method and the MLMC method scale well with the number of cores, while the computational time spent with Matlab rapidly saturates when increasing the number of cores, due to the heavier intercommunication overhead of the Krylov-based algorithm.

In Table 4 the computational time spent when computing the solution for different size domains is shown, being now the number of cores kept fixed to the maximum number of cores available.

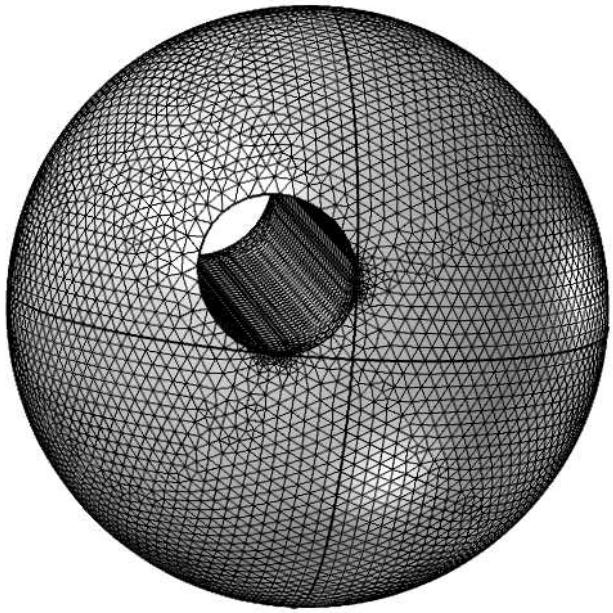

Fig. 6. Computational mesh describing the domain used for solving the 3D convection-diffusion equation.

Table 3
Elapsed time spent for computing the solution of the 3D convection-diffusion equation at a single point, and for time $t = 1$ as a function of the number of cores. The spatial point where the solution is computed, has been chosen to be the nodal point of the computational mesh closer to the physical point $(0, 0, 0)$. The accuracy was kept fixed to 5×10^{-4}. The radius of the sphere was $r = 4 \times scale$, with $scale = 12$, being the total number of nodes of the computational mesh $n = 2,375,211$.

Cores	**Time MC (s)**	**Time MLMC (s)**	**Time Matlab (s)**
1	1,519(33)	423(15)	152
4	378(8)	106(3)	107
8	215(4)	77(10)	91
12	167(5)	64(9)	89

It is remarkable that the computational cost of the MC and MLMC method appears to be almost independent of the size of the domain, while it increases almost linearly for the Krylov-based method. As it was already explained in [7] for the specific case of complex networks, this is mainly due to the similar matrix structure observed for any value of the matrix size. Because of this,

the error becomes mostly independent of the size, and consequently it is not required to modify further the value of the sample size M, or the time step Δt for increasingly larger matrix sizes (assuming a given prescribed accuracy for the solution), making therefore the computational cost of the algorithm almost independent of the size of the domain. This does not happen with the Krylov-based method, allowing specifically for the MLMC method to achieve a computational performance higher than the Matlab solution for large scale problems.

Table 4
Elapsed time spent for computing the solution of the 3D convection-diffusion equation at a single spatial point, and time $t = 1$ for different sizes of the domain. This has been done rescaling both, the sphere and cylinder, choosing different values of the scale parameter. The number of cores was kept fixed to 12 cores. The spatial point where the solution was computed consisted in the nodal points closer to the physical point $(0, 0, 0)$. The accuracy was kept fixed to 5×10^{-4}.

Scale	**n**	**Time MC (s)**	**Time MLMC (s)**	**Time Matlab (s)**
4	$83,813$	125(3)	59(11)	2
8	$694,751$	146(5)	66(10)	24
12	$2,375,211$	167(5)	64(9)	90

Even though, the MC and MLMC methods were proposed initially to compute the solution at single temporal points, it turns out that they can be used as well to obtain the solution at intermediate instants of time. As a remarkable feature this can be done without any additional computational cost, as it was already explained in Sec. 5. For the Krylov-based methods, it is worth pointing out that there were also some recent attempts [32] to improve the performance of the method for computing the solution in a finite time interval, being however the performance of the resulting algorithm slightly worse than the performance of the algorithm for computing the solution at a single time. To test the accuracy of the obtained solution for intermediate times, in Fig. 7 the solution computed using the Monte Carlo method is compared with the solution obtained using the Krylov-based method. Note the excellent agreement between both solutions for any value of time.

Example B: Complex networks

Small-world networks. In Table 5 the computational time required to compute the total communicability of a small-world network of size $n = 10^8$ is shown as a function of the number of cores for the Monte Carlo, MLMC method, and Matlab.

In Table 6 the results corresponding to a sort of weak scalability analysis of the MLMC algorithm are shown. For this purpose the algorithm was run for an increasing number of cores, searching for the value of the accuracy ε that equals

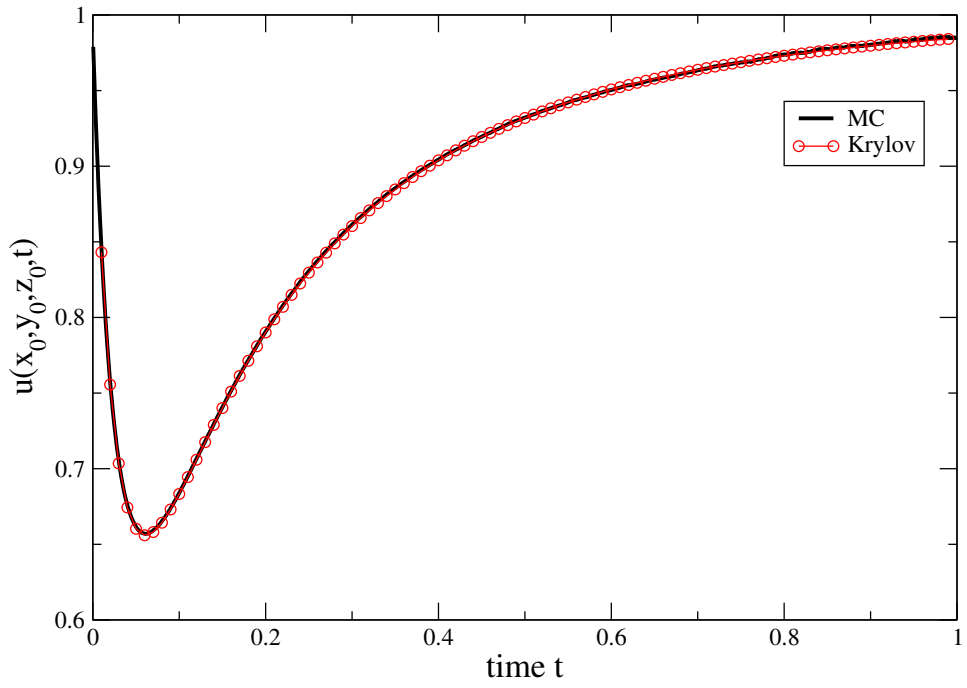

Fig. 7. Solution of the 3D convection-diffusion equation evaluated at the nodal points closer to the physical point $(0, 0, 0)$, and for different values of time. The initial value function was $f(\mathbf{x}) = e^{-(x^2+y^2+z^2)}$, and the velocity field β $(-1, -1, -1)$. The solid line denotes the solution obtained with the MLMC method, and the dotted line corresponds to the Krylov-based solution.

Table 5
Elapsed time spent for computing the total communicability of a small-world network as a function of the number of cores. The size of the matrix is $n = 10^8$, and the accuracy ε was kept fixed to 6.25×10^{-4}. The simulations were run on the commodity server.

Cores	**Time MC (s)**	**Time MLMC (s)**	**Time Matlab (s)**
1	825(2)	508(4)	348
4	225(2)	137(7)	258
8	114(2)	71(8)	257
12	76(2)	47(3)	282

the simulation time. Note that when the number of used cores increases, the accuracy ε should be reduced accordingly. Moreover, since the computational cost of the MLMC algorithm is of order $\mathcal{O}(\varepsilon^{-2})$, the workload of the algorithm increases when reducing the value of ε, being therefore required to increase

conveniently the number of used cores to keep approximately constant the overall execution time.

From the results in Table 6 it can be seen, for instance, that whenever the number of used cores increases 24 times (passing from 1 to 24 cores), the value of ε should be reduced by a factor of approximately 0.22 for the same execution time. It is worth observing that for such a reduction of ε, the workload of the algorithm increases by a factor of 21, which can be fully mitigated by increasing the number of used cores up to 24. This is due to the remarkable scalability of the parallel algorithm. Similar conclusions can be drawn from the results obtained when using 48 cores.

Table 6
Weak scalability analysis of the MLMC algorithm for computing the total communicability of a small-world network. The elapsed execution time was kept fixed around 1100 seconds, and the size of the matrix was $n = 10^8$. The simulations were run on the MareNostrum supercomputer.

Cores	ε	**Time MLMC (s)**
1	4.05×10^{-4}	1107
24	8.8×10^{-5}	1095
48	6.25×10^{-5}	1115

Table 7
Elapsed time spent for computing the total communicability of a small-world network as a function of the network size. The accuracy ε was kept fixed to 6.25×10^{-4}, and the number of cores on the commodity server to 12.

Size n	**Time MC (s)**	**Time MLMC (s)**	**Time Matlab (s)**
10^5	36(1)	26(2)	0.2
10^6	38(1)	30(1)	1.9
10^7	75(1)	45(2)	21
10^8	76(1)	47(3)	282

Table 7 shows the results corresponding to the computational time when computing the total communicability for different network sizes. Here the number of cores was fixed to the maximum number of cores available. It is remarkable to note that the computational cost of the MLMC method appears to be almost independent of the size of the network, while it increases almost linearly for the Krylov-based method.

Concerning the largest and smallest eigenvalues of the matrix, since again no analytical solution is available, we resort here to numerical approximations for the smallest network size, $n = 10^5$, and the results are $\lambda_{max} = 3.14$, and $\lambda_{min} = 2.6959 \times 10^{-5}$.

Scale-free networks. In Table 8 the results corresponding to a scale-free network for an arbitrarily large size are shown for different number of cores. Similar to the results obtained for the small-world network, the MLMC method outperforms the Krylov-based method for large size networks and cores. Since the network size used in this example has a matrix extremely large, no numerical approximations for the largest and smallest eigenvalues could be computed.

Table 8
Elapsed time spent for computing the total communicability of a scale-free network as a function of the number of cores. The size of the matrix was $n = 10^8$, and the accuracy ε was kept fixed to 2.5×10^{-8}. The simulations were run on the commodity server.

Cores	**Time MC (s)**	**Time MLMC (s)**	**Time Matlab (s)**
1	136(1)	87(10)	98
4	35(1)	23(10)	88
8	18(1)	11(3)	89
12	12(1)	8(3)	95

5.2 Distributed memory architecture

The simulations for a distributed memory architecture were carried out on the MareNostrum Supercomputer of the Barcelona Supercomputing Center (BSC) and on the Marconi Supercomputer at CINECA. In both cases, two processes were launched on each node (one per processor), with as many threads as physical cores available (24 threads on MareNostrum and 18 on Marconi). Up to 200 nodes (a total of 9600 cores) were used on MareNostrum and up to 160 nodes (5760 cores) on Marconi, which are respectively the maximum we had access to.

To the best of our knowledge, no parallel code suitable for distributed memory architecture capable of computing the action of a matrix exponential over a vector is currently available. Therefore, in the following, only results corresponding to the proposed multilevel method implemented in MPI are given.

Example A: Partial Differential equations.

The computational time spent by the multilevel method for computing the solution of the boundary value problem for the 3D heat equations at a single point is shown in Table 9 for different number of cores. These results were all obtained in the Marconi system. The speedup column indicates how much faster the execution is relative to half the number of cores (previous row in the table).

Table 9
Elapsed time spent for computing the solution of the 3D heat equation at the single point $(0, 0, 0)$, and for time $t = 1$ as a function of the number of cores. The accuracy ε was kept fixed at 10^{-5}. The length of the domain was $\delta = 4$, and three different numbers of discretization points, n_x, were used. Note that the matrix size for the system is given by $n_x^3 \times n_x^3$.

n_x	**Cores**	**Time MLMC (s)**	**Speedup**
	720	153	
128	1440	82	1.9
	2880	41	2.0
	5760	21	2.0
	720	774	
256	1440	395	2.0
	2880	196	2.0
	5760	107	1.8
	720	3577	
512	1440	1773	2.0
	2880	906	2.0
	5760	467	1.9

In all cases, the speedup is very close to the ideal, even for such a large number of cores. This is because most of the calculations are totally independent, corresponding to the Monte Carlo simulations performed at the each level of the method. For the defined level of accuracy ε, a very large number of samples is required, exceeding the number of 10^9 for the coarsest level. Communication is required between levels, but the overhead is negligible.

Example B: Complex networks

Small-world networks. In Table 10 the computational time required to compute the total communicability of a small-world network of size $n = 10^8$ is shown as a function of different number of cores for the multilevel method.

As in the case of the partial differential equation, the scalability of the method is almost perfect.

Table 10
Elapsed time spent for computing the total communicability of a small-world network as a function of the number cores. The size of the matrix was $n = 10^8$, and the accuracy ε was kept fixed at 10^{-7}.

	Cores	**Time MLMC (s)**	**Speedup**
MareNostrum	1200	315	
	2400	175	1.8
	4800	87	2.0
	9600	50	1.7
Marconi	720	320	
	1440	166	1.9
	2880	86	1.9
	5760	44	2.0

6 Conclusion

The multilevel Monte Carlo method was conveniently recast to be able to compute the action of a matrix exponential over a vector. As the main ingredient of the method, the leading probabilistic method 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.

This new method extends the previous work in three respects. First, the probabilistic method proposed in [7] has been generalized allowing now to be applied to any class of matrices (not only adjacency matrices). Second, it allows now to compute much more efficiently a highly accurate solution. In fact the computational complexity has been proved in this paper to be significantly better than that of the classical Monte Carlo method. Third, the underlying algorithm after parallelization has been shown to be highly scalable, which in practice enables simulation of large-scale problems for extremely large number of cores. We analyzed the performance of the algorithm running several benchmarks of interest in science and engineering. These consist in computing the total communicability of the network for a variety of complex networks (real and synthetic), and in solving at single points inside the domain a boundary-value problem for parabolic partial differential equations. Finally, whenever available, simulations based on a standard Krylov-based method have been conducted, and the performance compared with the multilevel MC method. In particular, the multilevel MC 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.

To conclude, an interesting question deserving further investigation is whether the proposed method can be extended to deal with other matrix functions such as trigonometric functions arising in oscillatory problems, and even hyperbolic functions appearing in coupled hyperbolic systems of partial differential equations.

Acknowledgments

The work has been performed under the Project HPC-EUROPA3 (INFRAIA-2016-1-730897), with the support of the EC Research Innovation Action under the H2020 Programme; in particular, the authors gratefully acknowledge the support of the Computer Architecture Department at Universitat Politècnica de Catalunya (UPC) and the computer resources and technical support provided by Barcelona Supercomputing Center (BSC). We acknowledge PRACE for awarding us access to Marconi at CINECA, through grant 2010PA4246. This work was also supported by Fundação para a Ciência e a Tecnologia under Grant No. UIDB/50021/2020, by the Spanish Ministry of Science and Technology through TIN2015-65316-P project and by the Generalitat de Catalunya (contract 2017-SGR-1414).

References

[1] T. Jahnke, and C. Lubich, Error bounds for exponential operator splittings, *BIT*,40 (2000) 735-744.

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

[3] J.A. Acebrón, M.P. Busico, P. Lanucara, and R. Spigler,Domain decomposition solution of elliptic boundary-value problems *SIAM J. Sci. Comput*, 27 (2005) 440-457.

[4] J.A. Acebrón, and A. Rodríguez-Rozas, A new parallel solver suited for arbitrary semilinear parabolic partial differential equations based on generalized random trees, *Journal of Computational Physics*, 230 (2011) 7891-7909.

[5] J.A. Acebrón, and A. Rodríguez-Rozas, Highly efficient numerical algorithm based on random trees for accelerating parallel Vlasov-Poisson simulations, *Journal of Computational Physics*, 250 (2013) 224-245.

[6] S. Mancini, F. Bernal, and J.A. Acebrón, An Efficient Algorithm for Accelerating Monte Carlo Approximations of the Solution to Boundary Value Problems, *J. Sci. Comput.*, 66 (2016) 577-597.

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

[8] 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) 146-179.

[9] R.N. Bhattacharya, and E.C. Waymire, Stochastic Processes with Applications, SIAM, 2009.

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

[11] M. Benzi, and C. Klymko, Total communicability as a centrality measure, *J. Complex Networks*, 1 (2013) 124-149.

[12] 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).

[13] M.A. Botchev, V. Grimm, and M. Hochbruck, Residual, restarting, and Richardson iteration for the matrix exponential, *SIAM J. Sci. Comput*, 35 (2013) A1376-A1397

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

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

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

[17] 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) 15-35.

[18] 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.

[19] 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.

[20] M. Evans, and T. Swartz, Approximating Integrals Via Monte Carlo and Deterministic Methods, Oxford University Press, 2000.

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

[22] G. Forsythe, and R. Leibler, *Matrix inversion by a Monte Carlo method*, Math. Tables Other Aids Comput., 4 (1950) pp. 127-129.

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

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

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

[26] N.J. Higham, and A. H. Al-Mohy, Computing matrix functions, *Acta Numerica*, 19 (2010) 159-208.

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

[28] A. Martinez, L. Bergamaschi, M. Caliari, and M. Vianello, A massively parallel exponential integrator for advection-diffusion models, J. Comput. Appl. Math. 231 (2009) 8291.

[29] 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) 2107-2122.

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

[31] S. Mazunder, Numerical Methods for Partial Differential Equations, Academic Press, 2015.

[32] A. H. Al-Mohy, and N.J. Higham, Computing the action of the matrix exponential, with an application to exponential integrators, SIAM J. Sci. Comput. 33 (2011) 488-511.

[33] G. Ökten, *Solving linear equations by Monte Carlo simulation*, SIAM J. Sci. Comput. 27 (2005) 511-531.

[34] M. Pusa, and J. Leppänen, Computing the Matrix Exponential in Burnup Calculations, *Nuclear Sci. and Eng*,164 (2010) 140-150.

[35] M.N.O. Sadiku, Monte Carlo methods for electromagnetics, *CRC press*, 2009.

[36] R.B. Sidjea, and W.J. Stewart, A numerical study of large sparse matrix exponentials arising in Markov chains, *Comput. Stat. Data Anal.*, 29 (1999) 345-368.

[37] `http://www.sprng.org/`

[38] G. Strang, Computational Science and Engineering, Wellesley-Cambridge Press, 2007.

[39] `https://sparse.tamu.edu/`

[40] `https://www.top500.org/`

[41] S.H. Weng, Q. Chen, and C.K. Cheng, Circuit Simulation by Matrix Exponential Method, *IEEE ASIC Conference*, (2011) 369-372.

[42] H. Zhuang, S.H. Weng, and C.K Cheng, Power Grid Simulation using Matrix Exponential Method with Rational Krylov Subspaces, *IEEE ASIC Conference*, (2013).

[43] O.C Zienkiewicz, R.L. Taylor, and J.Z Zhu, The Finite Element Method: Its basis and fundamentals, Elsevier , 2005.