

A new parallel solver suited for arbitrary semilinear parabolic partial differential equations based on generalized random trees

Juan A. Acebrón^a and Ángel Rodríguez-Rozas^a

^a*Center for Mathematics and its Applications, Department of Mathematics, Instituto Superior Técnico Av. Rovisco Pais 1049-001 Lisboa, Portugal*

Abstract

A probabilistic representation for initial value semilinear parabolic problems based on generalized random trees has been derived. Two different strategies have been proposed, both requiring generating suitable random trees combined with a Pade approximant for approximating accurately a given divergent series. Such series are obtained by summing the partial contribution to the solution coming from trees with arbitrary number of branches. The new representation greatly expands the class of problems amenable to be solved probabilistically, and was used successfully to develop a generalized probabilistic domain decomposition method. Such a method has been shown to be suited for massively parallel computers, enjoying full scalability and fault tolerance. Finally, a few numerical examples are given to illustrate the remarkable performance of the algorithm, comparing the results with those obtained with a classical method.

Key words: Monte Carlo methods, domain decomposition, semilinear parabolic problems, parallel computing, fault-tolerant algorithms, random trees

PACS: 65C05, 65C30, 65M55, 65N55

1 Introduction

Nowadays, most of the more popular numerical methods developed for solving partial differential equations (PDE) are based as a rule in generating a computational mesh, discretizing the given problem using the nodes of a computational mesh as discretization nodes, and solving the ensuing linear algebra

Email addresses: juan.acebron@ist.utl.pt (Juan A. Acebrón),
angel.rodriguez@ist.utl.pt (Ángel Rodríguez-Rozas).

problem for such nodes. As an alternative, the so-called meshless methods have started to be exploited more recently to avoid the need of building a computational mesh, but unfortunately they cannot exclude solving a corresponding linear algebra problem. Moreover, both numerical methods share also an important disadvantage, consisting namely in the impossibility of computing the solution of the problem at a single point inside the domain. Clearly this is due to the globally coupled nature behind these numerical methods. In the past such a feature did not constitute any serious limitation of the methods, being implemented frequently in sequential computers, but currently with the advent of parallel computers, this can be seen as a strong limiting factor to the overall efficiency of the corresponding numerical algorithms.

In fact, there are three sources of problem that have been observed. First, the tightly coupled nature of the algorithms induces a strong communication among the large number of processors currently present in the more advanced high performance supercomputing, and thus reducing the effective performance. Second, the chance to get a failure in one or several processors during the computation time increases with the number of processors involved. There is indeed a non negligible probability that a small percentage of processors or the network connecting them experience a failure. Most existing algorithms simply stop and are aborted as a consequence of such failures, and the proposed remedies usually require some kind of storing and restarting procedures, which degrades seriously the overall performance. Finally, in case of the grid computing and heterogeneous distributed computing, things can be even worse due to the high degree of systems heterogeneity and high network latency. Three major and recently published studies about the current source of problems in scientific computing can be found in [17], [24], and [25].

A successful alternative to such traditional methods consists of developing parallel numerical algorithms based on probabilistic numerical methods [1–4,12,20,22]. Such methods are inherently parallel, and naturally fault-tolerant, hence overcome all the obstacles mentioned above. Moreover, they allow to compute the solution at a single point inside the domain without the need of a computational mesh for solving the entire problem. However, they are not recommended to be applied in every point of the mesh due to the slow convergence rate, but rather to be combined with a classical Domain Decomposition [9,21] to compute merely the solution in a few points along some interfaces inside the domain. This is essentially the method called Probabilistic Domain Decomposition (PDD) proposed for the first time in [1,2] for solving linear elliptic problems, and generalized further to deal with some particular semi-linear parabolic problems in [3,4]. In short, the idea consists of generating only few interfacial values using probabilistic methods along a given possibly artificial interfaces inside the domain, obtaining approximate values interpolating on such interfaces, and then use such values as boundary data in order to split the original problem into fully decoupled sub-problems.

The aforementioned problems, successfully resolved by the PDD method, are very common and generally recognized by the global high performance computing community. At the present time, the situation becomes even more dramatic because of the number of scientific and engineering challenging problems requiring levels of petaflops, even exaflops, thus computing performance tends to increase very fast. A compelling and comprehensive reference about this fact can be found in the recently published roadmap of the *International Exascale Software Project* (IESP) [10]. In this project a worldwide joint effort is being conducted trying to overcome the current problems looking for a high quality computational environment for petascale/exascale systems. Incidentally in [10], it is claimed the need of rethinking completely new algorithms, and in particular it was suggested to reconsider the use of Monte Carlo based approaches, which is actually the case of the PDD method.

A key ingredient for implementing any PDD method requires having a probabilistic representation of the solution, since it will allow to compute the solution at the interfacial values. Probabilistic representations do exist for some elementary *semilinear* parabolic equations. Indeed, in [18] H.P. McKean derived the representation formula

$$u(x, t) = E\left[\prod_{i=1}^{k_t(\omega)} f(x_i(\omega, t))\right] \quad (1)$$

for the KPP equation

$$u_t = u_{xx} + u(u - 1), \quad x \in \mathbf{R}, \quad t > 0, \quad (2)$$

subject to the initial value $u(x, 0) = f(x)$; see also [13,19,23]. It is understood that in (1) the point $x_i(\omega, t)$ is the position of the i th branch of a branching stochastic process surviving at time t , ω denoting the chance variable. The quantity $k_t(\omega)$ is the random number of descendants at time t . A similar representation has been recently found in [3,4] for the solution of a more general semilinear parabolic problem, given by

$$\frac{\partial u}{\partial t} = Lu - cu + \sum_{j=2}^m \alpha_j u^j, \quad (3)$$

where L is a general linear elliptic operator, say $L := a_{ij}(\mathbf{x}, t)\partial_i\partial_j + b_i(\mathbf{x}, t)\partial_i$ (using the summation convention), with continuous bounded coefficients, $m \geq 2$ is an integer, $\alpha_j \geq 0$, $\sum_{j=2}^m \alpha_j = 1$, and c is a positive constant. Such a representation is based on generating *branching* diffusion processes, associated with the elliptic operator in Eq. (3), and governed by an exponential random time, S , with probability density $p(S) = c \exp(-cS)$.

In this paper this method is extended to deal with a wider class of semilinear parabolic problems, whose general form now is given by

$$\begin{aligned} \frac{\partial u}{\partial t} &= Lu + f(u, x, t), \quad x \in \mathbf{R}^n, \quad t > 0 \\ u(x, 0) &= g(x), \end{aligned} \tag{4}$$

where

$$f(u, x, t) = \sum_{j=2}^m c_j(x, t)u^j. \tag{5}$$

It is worth to observe that this generalizes further the previous representation obtained in [3,4], since it accounts for the following aspects: A constant potential term such as $-cu$ is not required anymore; the coefficients multiplying the nonlinear terms, $c_j(x, t)$, can be now chosen arbitrarily, hence overcoming the constraint imposed in the previous representation consisting in $\sum_{j=2}^m c_j(x, t) = 1$, and finally the initial data $g(x)$ may now be chosen negative, or greater than 1.

Moreover, using such a generalized probabilistic representation, the PDD method will be generalized further increasing notably the type of semilinear parabolic problems capable to be numerically solved in a highly efficient way. Finally, in order to assess the computational feasibility of the algorithm, we have compared our results with those obtained using competitive (freely available) parallel numerical codes, which are widely used by the high-performance scientific computing community.

Here it is the outline of the paper. In Sec. 2 a generalized probabilistic representation is presented, discussing two different possible strategies based on suitable random trees. Moreover, a qualitative study of the numerical errors is accomplished analyzing a few relevant test examples. Sec. 3 is devoted to numerical examples, where the high efficiency of the PDD method comparing with classical methods is illustrated. Finally, we summarize the more relevant findings to close the paper.

2 A generalized probabilistic representation

In order to generalize the class of parabolic problems amenable to a probabilistic representation in terms of branching diffusion processes, it becomes more convenient to rewrite Eq.(4) in an integral form. This can be done readily resorting to the Duhamel principle [11] for inhomogeneous initial-value parabolic

problems, and yields

$$u(x, t) = \int_{\mathbf{R}^n} dy g(y) p(x, t, y, 0) + \int_0^t \int_{\mathbf{R}^n} ds dy f(u(y, s), y, s) p(x, t, y, s), \quad (6)$$

where $p(x, t, y, \tau)$ is the associated Green's function, satisfying the equation

$$\begin{aligned} \frac{\partial p}{\partial t} &= Lp, \quad x \in \mathbf{R}^n, \quad t > \tau \\ p(x, \tau, y, \tau) &= \delta(x - y). \end{aligned} \quad (7)$$

The main difference with the previous representation obtained in (3) rests on the absence of the constant potential term $-cu(x, t)$. Such a term was crucial, since it allowed to obtain a probabilistic representation based on generating *branching* diffusion processes governed by an exponential random time, S , with density probability $p(S) = c \exp(-cS)$ in [3,4]. In the following we propose two different strategies capable to overcome such a constraint generalizing further the aforementioned representation.

2.1 Strategy A

This first strategy consists in inserting artificially a constant potential term into the PDE, by simply changing variables as follows

$$u(x, t) = v(x, t)e^t. \quad (8)$$

Then, $v(x, t)$ should satisfy the following equation:

$$\begin{aligned} v(x, t) &= e^{-t} \int_{\mathbf{R}^n} dy g(y) p(x, t, y, 0) \\ &+ \sum_{j=2}^m \int_0^t \int_{\mathbf{R}^n} ds dy e^{-s} c_j(y, t-s) e^{(j-1)(t-s)} v^j(y, t-s) p(x, s, y, 0). \end{aligned} \quad (9)$$

The Green function can be obtained probabilistically by means of the celebrated Feynman-Kac formula [15] as follows

$$p(x, t, y, \tau) = E[\delta(\beta(t) - y)], \quad (10)$$

where $\beta(t)$ is the solution of an initial-value problem for the stochastic differential equation (SDE) of the Ito type, related to the elliptic operator in (4), i.e.,

$$d\beta = b(\beta, t) dt + \sigma(\beta, t) dW(t), \quad \beta(\tau) = x. \quad (11)$$

Here $W(t)$ represents the N -dimensional standard Brownian motion (also called Wiener process); see [15], e.g., for generalities, and [16,19] for related numerical treatments. The drift, b , and the diffusion, σ , in (11), are related to the coefficients of the elliptic operator in (4) by $\sigma^2 = a$, with $a > 0$. Substituting (10) into Eq.(9), and introducing $\mathbf{1}_{[S>t]}$ as the indicator (or characteristic) function, being 1 or 0 depending whether S is or is not greater than t , Eq.(9) can be rewritten as follows

$$v(x, t) = E \left[g(\beta(t)) \mathbf{1}_{[S>t]} \right] \quad (12) \\ + E \left[\sum_{j=2}^m c_j(\beta(S), t - S) e^{(t-S)(j-1)} v^j(\beta(S), t - S) \mathbf{1}_{[S \leq t]} \right].$$

Here the time S is a random time, drawn from the exponential density distribution $p(S) = \exp(-S)$.

The equation above can be recursively solved, replacing the last term on the right-hand side with the solution $v(x, t)$, obtaining in such way an expansion in terms of multiple exponential random times, S_i , similarly as it was done in [3,4]. However, rather here the procedure is much more involved since now the integral equation contains both, variable coefficient terms, $c_j(\beta(S), t - S) e^{(t-S)(j-1)}$, and various nonlinear terms labeled by j . The latter can be reformulated probabilistically introducing a new discrete random variable α taken values between 2 and m , and governed by a uniform probability distribution with probability $p = 1/(m - 1)$, as follows

$$v(x, t) = E \left[g(\beta(t)) \mathbf{1}_{[S>t]} \right] \quad (13) \\ + E \left[c'_\alpha(\beta(S), t - S) e^{(t-S)(\alpha-1)} v^\alpha(\beta(S), t - S) \mathbf{1}_{[S \leq t]} \right],$$

where $c'_i = (m - 1)c_i$. Note that the probability distribution for the random variable α can be chosen in principle arbitrarily, however in order to minimize the statistical error it turns out convenient to assume α uniformly distributed since in such way all the nonlinear terms are equally sampled. Expanding recursively the equation above, we obtain:

$$v(x, t) = E \left[g(\beta(t)) \mathbf{1}_{[S_0>t]} \right]$$

$$\begin{aligned}
& + E \left[h^{(\alpha_1)}(y_1(S_0), S_0) \prod_{i=1}^{\alpha_1} g(x_i(t - S_0)) \mathbf{1}_{[S_i > t - S_0]} \mathbf{1}_{[S_0 < t]} \right] \\
& + E \left[h^{(\alpha_1)}(y_1(S_0), S_0) h^{(\alpha_2)}(y_2(S_0 + S_1), S_0 + S_1) \mathbf{1}_{[S_i > t - S_1]} \right. \\
& \times \left. \prod_{i=2}^{\alpha_1} g(x_i(t - S_0)) \prod_{i=\alpha_1+1}^{\alpha_1+\alpha_2} g(x_i(t - S_0 - S_1)) \mathbf{1}_{[S_i > t - S_0 - S_1]} \mathbf{1}_{[S_1 < t - S_0]} \mathbf{1}_{[S_0 < t]} \right] \\
& + \dots, \tag{14}
\end{aligned}$$

where $h^{(\alpha)}(y, s) = c'_\alpha(y, t-s) e^{(t-s)(\alpha-1)}$. Here $x_i(t)$, $y_i(t)$ denote the position of the i th path of the stochastic process $\beta(t)$ surviving or expiring, respectively, at time t .

Note that in Eq. (14) the solution can be evaluated by simply summing each partial contribution, and similarly to the class of equations studied in [3,4], the computational tool based on generating random trees turns out to be very useful since it allows to rapidly obtain analytically or compute numerically such contributions.

In the following, we assume the reader is familiar with the terminology used here, like the concepts of *branch*, *random tree*, etc. However, it is useful at this point to recall some terminology pertaining to trees, usually directed trees. A tree is a connected graph, i.e., a set of nodes linked by *edges*, with only one starting node, called *root*, a number of final nodes, called *leaves*, while the other (internal) nodes are called *vertices*, and we call here *branch* every set of edges joining vertices to leaves. Therefore, in this sense the number of leaves in a tree is equivalent to the number of branches. Finally, the number of nodes linked to a given node is called the number of *children*.

The algorithm works as follows: We first generate a random exponentially distributed time, S_0 , and a random path belonging to the stochastic process β . If S_0 is less than the final time t , then we split the given path into as many branches as those corresponding to the randomly chosen value α , the degree of nonlinearity. They depart from the position where the previous path was at time S_0 , and continue along independent trajectories until the next splitting event takes place. Whenever one of the possible branches reaches the final time, t , the initial value, g , is evaluated at the position where the path was located. Finally, the partial contribution to the solution is reconstructed multiplying all contributions coming from each branch and the coefficients $h^{(\alpha)}(y, s)$ conveniently evaluated at specific points. For the purpose of illustration, we sketch a typical configuration in the second picture of Fig. 1, corresponding to two different splitting events. Note that such a configuration represents graphically what appears in the last term of Eq. (14). Moreover, it can be seen clearly the structure of the random tree, where the dots denotes the splitting events, being marked in black or white depending on whether the splitting time obtained is less or not than the final time, T . Therefore, the

white dots correspond to the leaves of the tree, while the black dots are the corresponding vertices. Note that the number of children associated to a given node depends on the value of α , which has been randomly chosen.

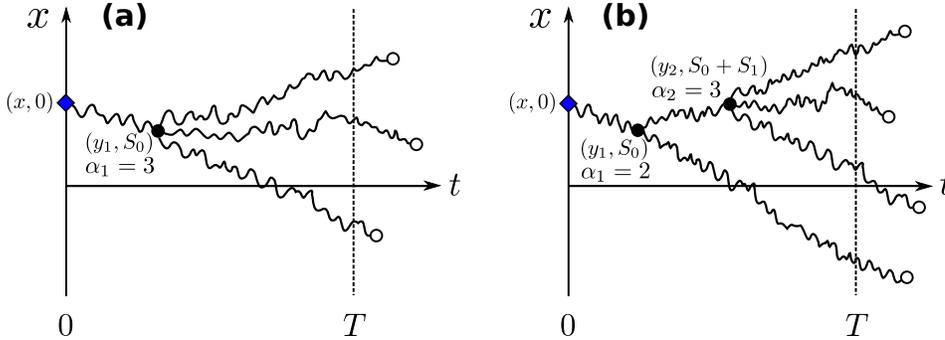


Fig. 1. Two possible configuration diagrams. In (a) only one splitting event occurs, with $\alpha_1 = 3$. Rather in (b) two of such events occur, the first one with $\alpha_1 = 2$, and the second one with $\alpha_2 = 3$.

Hence the solution $v(x, t)$ in Eq. (13) can be rewritten as a expectation value over random trees of a suitable multiplicative functional of the initial data $g(x)$, the random times S_i , and the variable coefficients c'_j , as follows,

$$v(x, t) = E \left[\prod_{i=1}^{Ne(\omega)} h^{(\alpha_i(\omega))}(y_i(\omega), \bar{S}_i(\omega)) \prod_{l=1}^{k(\omega)} g(x_l) \right]. \quad (15)$$

Here $k(\omega)$, and $Ne(\omega)$ are the random number of branches at final time t , and the number of splitting events obtained when generating the random tree, respectively. By \bar{S} we denote the corresponding global random time obtained by summing conveniently the random times S_i according to the specific structure of the generated random tree. It is worth to observe that such trees are used as a tool to construct the structure representing a given partial contribution to the solution, allowing afterward to follow easily how the arguments of the functions $h^{(\alpha)}(y, s)$ are exchanged when solving recursively Eq.(13).

Even though such a new representation allows to expand further the class of equations suited to be computed probabilistically, however rather than in the representation obtained in [3,4], a major drawback now should be faced. This is because the coefficients multiplying the nonlinear terms v^j might be greater than 1, therefore being the convergence of the numerical procedure not guaranteed. In fact, the series obtained by expanding Eq. (13) could be divergent, and in general cannot be summed simply by a sequence of partial sums. Additionally, the *pruning* techniques presented in [3,4] cannot be applied for the same reason. Nevertheless, numerical experiments show that in many cases the asymptotic series can be summed up resorting to summation methods such as the Euler's formula, or approximation techniques based on

the Pade approximant. In this paper, we consider merely the latter one, because from numerical experiments seems to be more robust in dealing with the unavoidable numerical errors affecting the coefficients of the series computed numerically. In Sec. 2.4 a few test problems have been investigated to illustrate the robustness and convergence of the Pade approximant.

2.2 Strategy B

A different strategy for obtaining a probabilistic representation for the problem in Eq. (4) consists in sampling both terms of the integral equation (6), by introducing a two-point discrete random variable ξ taking the values 0, and 1, with probability $P(0) = q$, $P(1) = 1 - q$. Therefore, the integral equation (6) can be rewritten as follows,

$$u(x, t) = q \int_{\Omega} dy \tilde{g}(y) p(x, t, y, 0) + (1 - q) \int_0^t \int_{\Omega} ds dy \sum_{j=2}^m \tilde{c}_j(y, t - s) u^j(y, t - s) p(x, s, y, 0), \quad (16)$$

where $\tilde{g}(x) = g(x)/q$, and $\tilde{c}_j(x, t) = c_j(x, t)/(1 - q)$. The probabilistic representation can be readily found and has the form

$$u(x, t) = E [\tilde{g}(\beta(t))\delta(\xi)] + E [\eta(t)\tilde{c}'_{\alpha}(\beta(tS), t(1 - S)) u^{\alpha}(\beta(tS), t(1 - S))\delta(\xi - 1)], \quad (17)$$

where the time S is a random time between 0 and 1 uniformly distributed, α a discrete random variable taking values between 2 and m with equal probability $p = 1/(m - 1)$, $\tilde{c}'_{\alpha} = (m - 1)\tilde{c}_{\alpha}$, and $\eta(t) = t$. Similarly to the previous strategy, the equation above can be recursively expanded, and yields

$$\begin{aligned} u(x, t) &= E [\tilde{g}(\beta(t)) \delta(\xi_0)] \\ &+ E \left[\eta(t)\tilde{c}'_{\alpha_1}(y_1(tS_0), t(1 - S_0)) \prod_{i=1}^{\alpha_1} \tilde{g}(x_i(t(1 - S_0))) \delta(\xi_i)\delta(\xi_0 - 1) \right] \\ &+ E \left[\eta(t)\eta(t(1 - S_0))\tilde{c}'_{\alpha_1}(y_1(tS_0), t(1 - S_0)) \right. \\ &\times \tilde{c}'_{\alpha_2}(y_2(t(1 - S_0)S_1), t(1 - S_0)(1 - S_1)) \prod_{i=2}^{\alpha_1} g(x_i(t(1 - S_0))) \delta(\xi_i) \\ &\left. \times \prod_{j=\alpha_1+1}^{\alpha_1+\alpha_2+1} g(x_j(t(1 - S_0)(1 - S_1)))\delta(\xi_j)\delta(\xi_1 - 1)\delta(\xi_0 - 1) \right] + \dots \quad (18) \end{aligned}$$

Therefore, as in the strategy A, the solution can be obtained as the expectation value over suitable random trees of a given multiplicative functional of the initial condition, being given now as follows,

$$u(x, t) = E \left[\prod_{i=1}^{Ne(\omega)} \eta(t\bar{S}_i(\omega)) c_{\alpha_i(\omega)}(y_i(\omega), t\bar{S}_i(\omega)) \prod_{l=1}^{k(\omega)} g(x_l) \right]. \quad (19)$$

Here \bar{S} is the global time random variable associated to the generated random tree. It is obtained multiplying the different random times S_i according to the specific structure of the tree.

In view of the probabilistic representation obtained for both strategies, it is worth to point out that both strategies require generating random trees to evaluate numerically the partial contribution to the solution, however in practice the computational procedure needed is quite different. In fact, while the random trees in the strategy A are constructed by generating a unique random number, the random time S , which governs how the trees branch off in time, rather those in the strategy B require two independent random numbers for the same purpose. This is because in the strategy A the change of variable introduces a time dependent exponential coefficient, which can be used to construct a probabilistic representation for Eq. (12) based on an exponential random time, and therefore the random trees can be fully characterized by such random time. On the contrary, the probabilistic representation in Eq. (17) requires both, a random number ξ which governs the branching process, and a random time S uniformly distributed, for evaluating numerically the partial contribution to the solution corresponding to a given random tree.

Similarly to the strategy A, the series obtained using the strategy B in Eq. (18) may be divergent, being therefore necessary to resort to approximation techniques, such as the Pade approximant, to approximate conveniently the sum of the series.

To illustrate how both strategies can be implemented in practice for solving an initial value semilinear parabolic problem, let consider the following equation,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + u^2, \quad x \in \mathbf{R}, t > 0 \quad (20)$$

$$u(x, 0) = f(x). \quad (21)$$

Since the procedure underlying the strategy A is formally similar to that followed when a constant potential term is present, we refer to the reader to [3,4] for a practical implementation for such a case, and here we focus merely on the strategy B. From Eq. (17), the probabilistic representation is given by

$$u(x, t) = E [\tilde{g}(\beta(t))\delta(\xi)] + E [\eta(t) u^2(\beta(tS), t(1 - S))\delta(\xi - 1)], \quad (22)$$

or in a more compact format, using Eq.(19) for the expectation value over random trees of a given multiplicative functional in Eq.(22), by

$$u(x, t) = E \left[\prod_{i=1}^{Ne(\omega)} \eta(t\bar{S}_i(\omega)) \prod_{l=1}^{k(\omega)} g(x_l) \right]. \quad (23)$$

Every random tree is built generating a sequence of interconnected binary random variables, ξ_i , branching off from the previous one as follows: Let ξ_1 the random variable associated to the root of the tree. Only when ξ_1 takes value 1 with probability $P(1) = 1 - q$, two new random variables denoted by $\xi_{2,3}$ (child nodes of the root), are created. These new variables proceed further creating other nodes governed by the same rule, until no random number ξ_i takes anymore the value 1. At this point the procedure is concluded, giving rise to a random tree characterized by k branches or leaves, and Ne splitting events.

The nodes of the tree are labeled in binary format according to their ancestors as follows: A given node with label $[a_0a_1a_2\dots a_N]$, where $a_i = 0, 1$, is connected to the set of nodes $\{[a_0], [a_0a_1], [a_0a_1a_2], \dots, [a_0a_1a_2 \dots a_{N-1}]\}$. The global time random variable \bar{S} associated to a given tree with k branches is given by

$$\bar{S} = \prod_{i=1}^{2^{k-1}-1} S_j^{\gamma_j}, \quad \gamma_l = \sum_{j=l+1}^{2^{k-1}-1} \nu_j \langle j|l \rangle, \quad l = 1, \dots, 2^{k-1} - 1 \quad (24)$$

where ν_l is 0, or 1 depending on whether the tree contains or not the node l . The function $\langle \cdot | \cdot \rangle$ is defined as follows,

$$\langle j|l \rangle := \begin{cases} 1 & \text{if } T_j^{[l]} = l \\ 0 & \text{otherwise.} \end{cases}, \quad (25)$$

where both, j and l are numbers written in binary format, and $T_j^{[l]}$ is an operator that truncates the number j to their most significant $[l]$ digits, where $[l]$ is the number of digits of l . By example, let $j = [a_0a_1a_2\dots a_N]$, then $T_j^{[l]} = [a_0a_1\dots a_{[l]-1}]$.

Figure 2 shows the different random trees obtained with $k = 4$, and $Ne = 3$, and their corresponding labels according to the rule defined above.

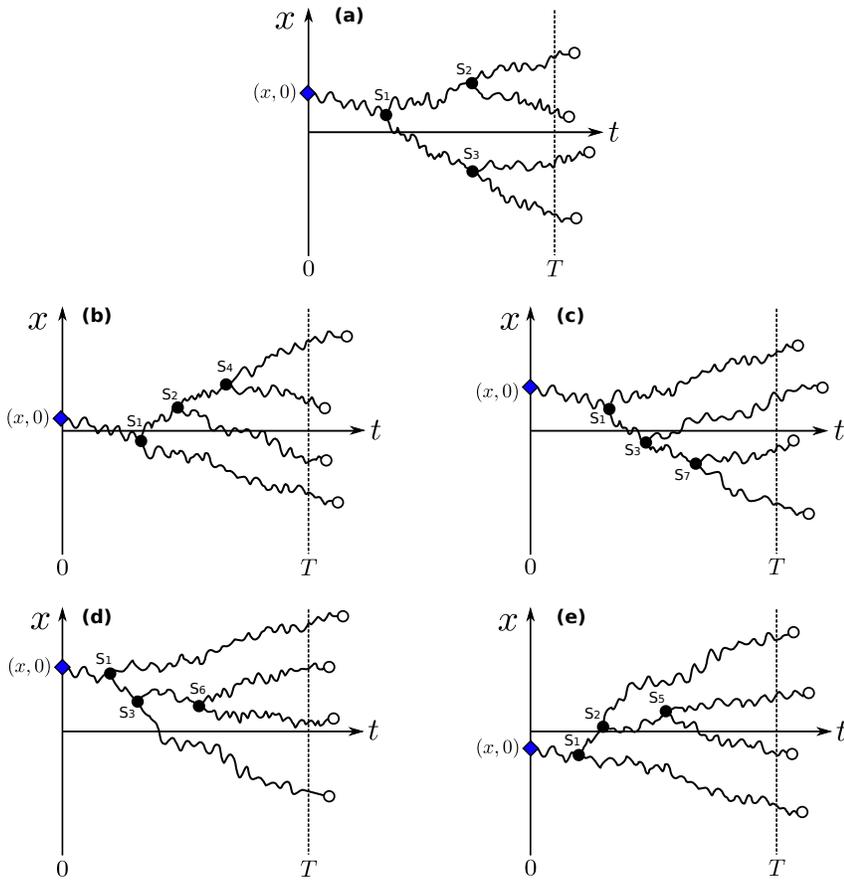


Fig. 2. Configuration diagram for the case of 4 branches and 3 splitting events. Here S_i is a random time uniformly distributed between the previous generated time, and the final time, T . The corresponding labels i of the random time S_i are defined according to the rule explained in the text.

2.3 Computational complexity of the strategy B

In this subsection we estimate the computational complexity in terms of the computational time required to compute probabilistically the solution at a single point (x, t) based solely on the strategy B, since the computational complexity of the strategy A coincides with that analyzed already in [3,4].

The branching stochastic process associated to the nonlinear term u^j , requires creating j branches every time a splitting event occurs, being therefore more costly whenever the power of the nonlinearity is higher. Then, it is worth to observe that for the general nonlinear function in Eq. (4), the overall computational time is governed by the computational time spent by the nonlinear coefficient with the highest power, say u^m .

The computational time spent to generate any given branch, is a function of the final time, t , as well as of the time step, Δt , chosen to solve numerically the

associated stochastic differential equation in (11). In addition, we should take into account the random times Δt_s responsible for branching. From Eq. (17), it holds that $\Delta t_s = tS$, where S is a random number picked up from the uniform distribution $U(0, 1)$. It is thus necessary that the time-step discretization, used to solve (11), also captures the instants when the random exponential times occur. In practice, the actual time step is chosen according to the minimum value between Δt and Δt_s . Since Δt_s is chosen randomly, the probability of being less than Δt can be easily estimated, and turns out to be $(\Delta t)/t$. When this occurs, the actual time step used for the numerical solution of (11) should be chosen to be Δt_s . Averaging over all random trees, we obtain the most probable time step to be used, which is given by

$$\overline{\Delta t_s} = \int_0^{\Delta t} ds s \frac{1}{t} + \Delta t \int_{\Delta t}^t ds \frac{1}{t} = \Delta t - \frac{1}{2} \frac{(\Delta t)^2}{t}. \quad (26)$$

The computational time can be measured, typically, in terms of the number of iterations in time, required to fully generate a random tree with k branches up to the final time, t . Defining t_c as the time spent per iteration, such computational time can be estimated as $kt_c t / \overline{\Delta t_s}$. In case of N random trees, the average computational time, t_b (b standing for ‘‘branching’’), turns out to be

$$t_b = N \sum_{k=1}^{\infty} kt_c \frac{t}{\overline{\Delta t_s}} P(k, m), \quad (27)$$

where $P(k, m)$ is the probability of finding a random tree with k branches, being m the number of children.

Such a probability can be evaluated by first enumerating and then summing up the various probabilities, $p_i(k)$, of having k branches as a final configuration, that is

$$P(k, m) = \sum_{i=1}^{N_D(k, m)} p_i(k, m). \quad (28)$$

Here $N_D(k, m)$ denotes the total number of possible diagrams characterized by k branches and m children, and $p_i(k, m)$ the probability of obtaining each of them. In Fig. 3 we show, for the purpose of illustration, some diagrams for $k = 2, 3$. In particular, for $k = 4$ the total number of possibilities of obtaining 4 branches has been shown in Fig. 2. It is reasonable to assume that each diagram contributes equally to the probability function (28). Therefore, such a probability can be obtained by simply counting the number of possible diagrams with k branches, $N_D(k, m)$, and then multiplying by the probability

of having one of them, that is $P(k, m) = N_D(k, m)p_1(k, m)$. For convenience, we consider the special diagram shown in Fig. 4. The probability of obtaining such a diagram as a final configuration is given by

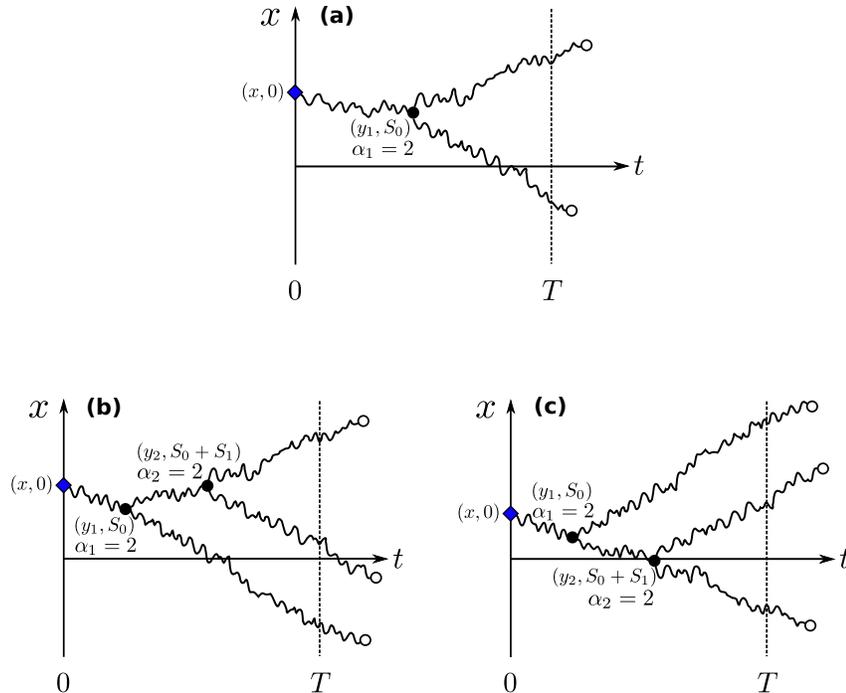


Fig. 3. Configuration diagrams for the case of 2, and 3 branches, illustrating the notation used in the probabilistic representation obtained for Strategy A in Eq. (15). Here α_i denotes the power of the nonlinearity, chosen randomly between 2 and m , and governing the number of branches created every time a splitting event i takes place; y_i denotes the position of the i th path of the stochastic process expiring at a given time \bar{S} . By \bar{S} we denote the corresponding global time obtained by summing conveniently the random times S_i according to the rules described in the text.

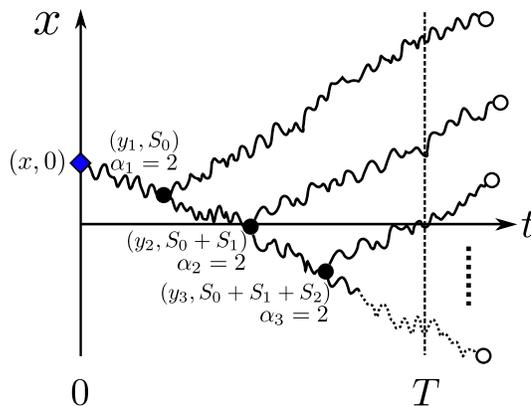


Fig. 4. Configuration diagram with k branches. See Fig. 3 for a detailed explanation of the corresponding labels.

$$p_1(k) = q^k(1 - q)^{Ne}. \quad (29)$$

Note that the number of branches, k , is related to the number of splitting events, N_e , by $k = (m - 1)N_e + 1$.

Concerning the number of possible diagrams, N_D , this should be a function of the number of branches, k . A simple strategy to compute the all possible configurations requires analyzing the distribution of leaves at any depth level of the random tree.

For simplicity, in the following let consider first the case of $m = 2$, which corresponds to binary trees, and a tree composed of k branches. When the tree is generated, the first splitting event gives rise to two new sub-trees, each one having at least one leaf, and at most $k - 1$ leaves, since the total number of leaves should be k . The different possible ways of distributing the k leaves among the two sub-trees determines the different possible configurations at this level, namely let assign 1 leaf to the first sub-tree, and $k - 1$ to the second one; or 2 to the first one, and $k - 2$ to the second one, and so on. Clearly, this procedure should be recursively applied to every sub-tree, since any of them exhibits different type of configurations. As a result the following nonlinear, full memory, convoluted recurrence is obtained

$$N_D(k, 2) = \sum_{j=1}^{k-1} N_D(k - j, 2) N_D(j, 2),$$

$$N_D(0, 2) = 0, \quad N_D(1, 2) = 1,$$

see [14] e.g. This recurrence can be solved by means of generating functions, yielding

$$N_D(k, 2) = \frac{1}{k} \binom{2k - 2}{k - 1}.$$

Moreover, this can be generalized further to any number of children by considering the Fuss-Catalan numbers, see [6]. Recall that the number of children is given by the power of the nonlinearity m in Eq.(4), and therefore the corresponding tree should be in general an m -ary tree. For this case it becomes more convenient to describe the number of configurations N_D in terms of the number of splitting events N_e instead of using the number of branches k , since when m is different from 2, k is not a consecutive integer number. The solution is given by

$$N_D(N_e, m) = \frac{1}{N_e m + 1} \binom{N_e m + 1}{N_e}.$$

Summarizing, the probability of obtaining a random tree with k branches is given by

$$P(k, m) = q^k (1 - q)^{Ne} \frac{1}{m Ne + 1} \binom{m Ne + 1}{Ne}. \quad (30)$$

However, it turns out that the function $P(k, m)$ in Eq.(30) can be considered a probability function for a particular range of values of q , since only for such values $\sum_{Ne=0}^{\infty} P(k, m) = 1$ is satisfied, being different from one or even unbounded otherwise. In fact, let $u = \sum_{Ne=0}^{\infty} P(k, m)$, $v = u/q$, and $x = q^{m-1}(1 - q)$, then

$$v = \sum_{Ne=0}^{\infty} x^{Ne} \frac{1}{m Ne + 1} \binom{m Ne + 1}{Ne}. \quad (31)$$

Note that the power series above coincides with the generalized binomial series [14], thus $v = B_m(x)$. From the properties of the generalized binomial series, it holds that

$$v = 1 + x v^m. \quad (32)$$

Among the m possible solutions of Eq.(32), only that one satisfying $\lim_{x \rightarrow 0} v = 1$ is meaningful, which corresponds to the trivial probability function $P(k) = \delta_{k0}$ being $q = 1$. The solution v can be constructed iteratively applying a Picard iteration as follows,

$$v_n = 1 + x v_{n-1}^m, \quad (33)$$

with v_0 arbitrarily chosen. Note that this can be seen as a dynamical map $v_n = f(v_{n-1})$, with $f(v) = 1 + x v^m$. Such a map has as a fixed point $v^* = 1/q$, being stable whenever $f'(v^*) \leq 1$. This corresponds to values of q satisfying

$$q \geq (m - 1)/m. \quad (34)$$

This can be generalized further for a nonlinear function as in Eq. (4), obtaining the range of allowed values of q for which a probability function $P(k, m)$ can be found. Let define now the function $u = \sum_{k=1}^{\infty} \sum_{l=\lceil (k-1)/(m-1) \rceil}^{k-1} P(k, l)/(m - 1)$. Similarly to the previous case, it can be readily proved that u satisfies the following equation,

$$u = q + \frac{(1-q)}{m-1} \sum_{i=2}^m u^i. \quad (35)$$

The solution $u = 1$ can be iteratively constructed by applying a Picard iteration, thus obtaining the following nonlinear map,

$$u_n = f(u_{n-1}), \quad (36)$$

where $f(u) = q + \frac{(1-q)}{m-1} \sum_{i=2}^m u^i$. Since $f'(1) = (1-q)[1 + (m+1)/2]$, the fixed point $u = 1$ turns out to be stable when

$$q \geq m/(m+2). \quad (37)$$

The validity of Eq. (30) can be confirmed by some numerical simulations, consisting in generating N random trees with a given q satisfying the specific constraint described above, and counting the number of branches obtained. A comparison between the probability $P(k, m)$ as function of k , obtained both numerically and theoretically, is plotted in Fig. 5. This has been done for the case $m = 2$ in Fig. 5(a), and $m = 3$ in Fig. 5(b). The perfect agreement validates the formula (30).

Once the probability function, $P(k, m)$ is known, the computational time t_b spent by the probabilistic part of the algorithm, can be evaluated. From (27), we have

$$t_b \leq N t_c \frac{t}{\Delta t_s} \langle k \rangle_{P(k, m)}, \quad (38)$$

where $\langle k \rangle_{P(k, m)}$ denotes the mean number of leaves, that is $\sum_{k=1}^{\infty} k P(k, m)$. Such a number can be easily computed exploiting the following relation, obtained from Eq.(31) simply deriving with respect to x and then multiplying by x ,

$$\sum_{N_e=0}^{\infty} N_e x^{N_e} \frac{1}{m N_e + 1} \binom{m N_e + 1}{N_e} = x \frac{dv}{dx}. \quad (39)$$

Since for the allowed values of q , v is given by $1/q$, then from Eq. (32) $\langle k \rangle_{P(k, m)}$ can be readily obtained and is given by

$$\langle k \rangle_{P(k, m)} = \frac{q}{1 - m(1 - q)}. \quad (40)$$

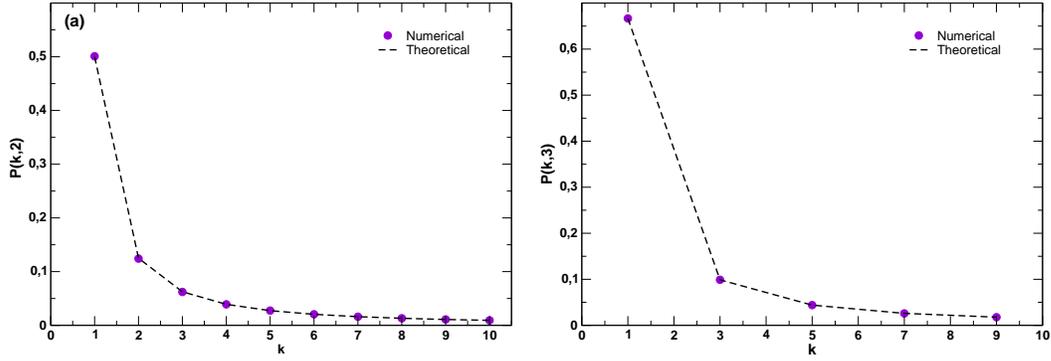


Fig. 5. Comparison between the probability function $P(k, m)$ obtained analytically and numerically simulating 10^6 random trees for $m = 2$ in (a), and $m = 3$ in (b).

Hence an estimate of the computational time t_b satisfies the following bound,

$$t_b \leq N t_c \frac{t}{\Delta t_s} \frac{q}{1 - m(1 - q)}. \quad (41)$$

Note that the estimate obtained above exhibits a linear growth on t . This contrasts with the theoretical estimates of the computational time obtained for the strategy A [3,4], which grows instead unboundedly in time. Such a remarkable feature of this strategy in comparison with the strategy A allows to speed up notably the simulations. Moreover, the bound obtained for the computational time suggests that decreasing q may decrease further such a time, becoming singular however when $q = (m - 1)/m$, and therefore useless when such an occurrence happens.

In Fig. 6, a comparison between the theoretical estimates obtained for $m = 2$, and two different values of q , and the measured computational times are shown as function of the final time. Note the good agreement with the theoretical results.

2.4 Qualitative study of the numerical errors

The numerical errors appearing when solving parabolic problems by some of the probabilistic strategies described above, are essentially the same as those analyzed in [3,4]. In fact, the most important source of numerical errors arises from replacing the expected value in Eq.(19) by a finite sum, and when the stochastic paths are actually simulated resorting to suitable numerical schemes. Thus, approximately,

$$u(\mathbf{x}, t) \approx \frac{1}{N} \sum_{j=1}^N f(\beta_j^*(t)), \quad (42)$$

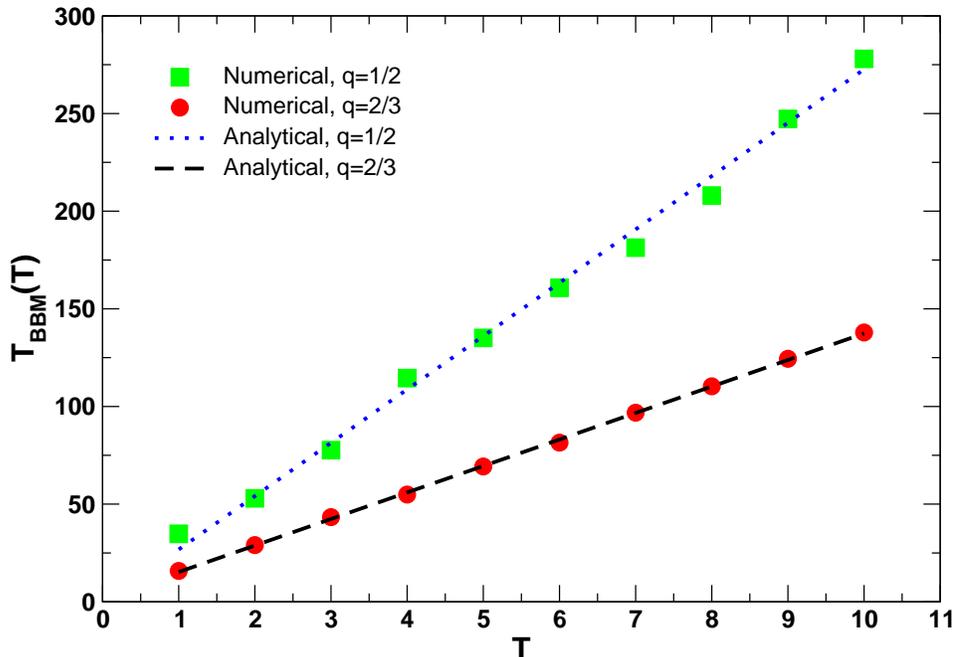


Fig. 6. Comparison between the computational time spent in solving Eq. (21) at a single point $x = 0$, and the estimated computational time obtained theoretically in (41) for $m = 2$. This has been done for $q = 1/2$, and $q = 2/3$. The numerical results have been fitted to a line with correlation coefficients $r = 0.9973364$, and $r = 0.9998173$, respectively. Parameters are $N = 10^6$.

where f denotes the multiplicative functional in Eq.(19), N is the sample size, and β^* is the stochastic path with discretized time. Clearly, such a discretization procedure unavoidably introduces two sources of numerical error. The first one is the pure Monte Carlo statistical error, which it is known to be of order $O(1/\sqrt{N})$ when N goes to infinity. The second error is due to the fact that the ideal stochastic path, $\beta_j(\cdot)$, has to be approximated, discretizing time, by some numerical scheme yielding the paths $\beta_j^*(\cdot)$. The truncation error made when solving numerically the stochastic differential equation (11), obviously depends on the specific scheme chosen, see [16], e.g. Among these are the Euler scheme, which was used here to simulate numerically Eq. (11). Such scheme is well known to have a truncation error of order $O(\Delta t^\alpha)$, where $\alpha = 1/2$ or $\alpha = 1$ depending on whether the scheme being of the “strong” or “weak” type, respectively [16].

Concerning the first error for the strategy B, the freedom to choose the value of q from a set of allowed values can be exploited in order to minimize such an error. In fact, rather than the strategy A, the strategy B requires choos-

ing specifically a given value of q satisfying the constraints mentioned in the previous section. Among the set of allowed values, in the following we show that there exists an optimal one such that the statistical error made in computing probabilistically the coefficients of the Pade approximant turns out to be minimum.

Clearly, the statistical error becomes larger when computing the contributions coming from trees with arbitrarily large number of branches, since for such a case the number of generated trees is expected to be smaller, and consequently the associated multidimensional integral in (16) to be computed may be affected by a large statistical error. The number of generated trees N_k with k branches can be readily obtained, known the probability distribution $P(k, m)$, and is given by $N_k = NP(k, m)$, where N is the sample size. Therefore the statistical error can be estimated, and it turns out to be of order of $O(N_k^{-1/2})$.

For simplicity let consider first the case of a nonlinear function in (4) with a single nonlinear term, u^j . The value of q minimizing the statistical error is attained when the number of random trees N_k is maximum. Such a value can be obtained simply evaluating dN_k/dq , using the probability function in Eq. (30), and looking for the global maximum q^* . The result is given by

$$q^* = \frac{k(j-1)}{kj-1}. \quad (43)$$

Since the number of generated random trees N_k is minimum, and consequently the statistical error maximum, for trees with large number of branches, the optimal value of q can be obtained considering in particular the limiting case $\lim_{k \rightarrow \infty} q^*$, yielding $q_{opt} = (j-1)/j$. Note that this coincides precisely with the minimum value from the range of allowed values obtained in Eq. (34).

This result can be generalized further for an arbitrary nonlinear function $f(u) = q + \frac{(1-q)}{m-1} \sum_{i=2}^m u^i$. Recall that for such a function the random trees obtained may be composed in general of $m-1$ different type of vertices, each one possessing from 2 to m children. A crucial hint is to realize that the most probable configuration of any generated random tree with arbitrary number of branches k occurs when the number of different type of vertices are identical, or in other words the number of children in the random tree are uniformly distributed. To illustrate through numerical simulations the observation above, N random trees are generated for a nonlinear function with $m=3$, giving rise therefore to trees composed of vertices with two and three children. For each tree the number of vertices with two children n_2 , and with three children n_3 , were recorded, and computed the difference between them. In Fig. 7 a histogram showing the number of configurations obtained as a function of $n_2 - n_3$ is shown, and this has been done for two different values of the number of branches k . Note, as expected, that the maximum number of configurations

corresponds to generated trees composed of the same number of vertices with two, and three children, that is, when $n_2 - n_3 = 0$.

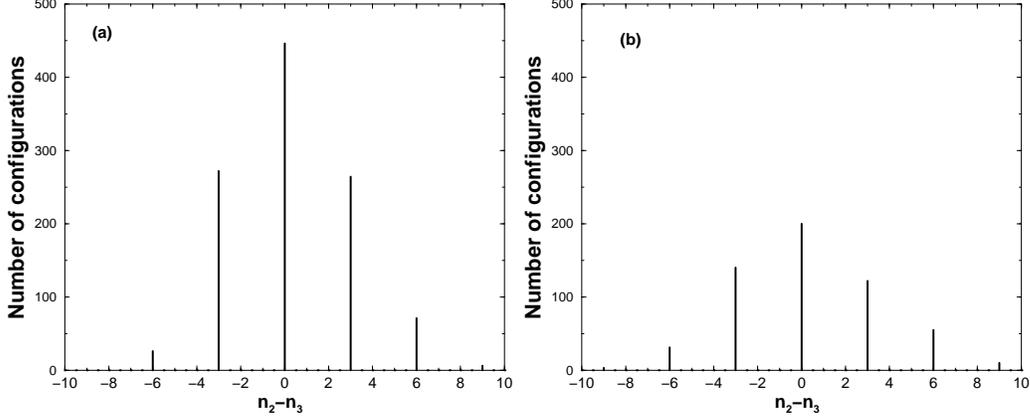


Fig. 7. Number of configurations of generated random trees as function of $n_2 - n_3$, being n_2, n_3 the number of vertices with two and three children, respectively. This has been done for trees with (a) $k = 13$, and (b) $k = 19$. Parameters are $N = 10^5$, and $m = 3$

For the general function $f(u)$ with $m - 1$ nonlinear terms the most probable configuration of a random tree with k branches satisfies $n_2 = n_3 = \dots n_m = n$, being n_j the number of vertices with j children. Then, the global number of splitting events Ne for such a configuration is given by $Ne = (m - 1)n$. Clearly the number of branches k of such a tree is related with the number of splitting events Ne , and this relation can be obtained readily, yielding for k

$$k = 1 + \sum_{i=1}^{m-1} (m - i)n = n \frac{m - 1}{2} m + 1. \quad (44)$$

Given $Ne = (m - 1)n$, it holds that $k = \frac{Ne}{2} m + 1$. For a given finite sample size N , the number of random trees N_k with k branches should be maximal, and consequently the statistical error minimal, provided that the value of q is chosen such as $(dN_k/dq)|_{q^*} = 0$. This yields,

$$q^* = \frac{k m}{k(m + 2) - 2}. \quad (45)$$

Similarly to the simple case analyzed above, the optimal value of q can be obtained considering in particular the limiting case $\lim_{k \rightarrow \infty} q^*$, yielding now $q_{opt} = m/(m + 2)$. Note that such a value coincides again with the smallest value from the range of allowed values in Eq. (37).

Apart from the errors discussed above, a new source of error now should be taken into account. This consists of the numerical error made in approximati-

ing divergent series by a Pade approximant, since for the class of problems considered in this paper, both strategies proposed require dealing with series that in general may be divergent. Since finding theoretical estimates of such an error for any given problem may be a formidable task, our goal in this section consists merely to gain some insight of such an error, illustrating how well Pade approximation actually works by analyzing a few relevant test problems. More specifically, our aim is twofold.

On one hand we show that the statistical error made in evaluating the partial contributions to the solution in Eq. (18) by Monte Carlo propagates to the coefficients of the Pade approximant. However, when the statistical error is sufficiently small, the coefficients of the Pade approximant can be obtained within a reasonable accuracy, being therefore the Pade approximant rather robust at least for the examples here considered.

On the other hand, the convergence of the Pade approximant to the solution is analyzed for such examples. Since for computational purpose the expansion generated with both strategies must be truncated, it becomes essential to determine whether the Pade approximant converges rapidly to the solution for the finite number of terms involved, or rather it is required to increase further the order of the approximation by considering more terms in the expansion. In [4], a pruning technique of the full random tree was proposed, which in practice amounts to keep only few trees possessing a certain number of branches. This is because it was observed that truncating the expansion in (3) up to only a certain number of branches, might not affect appreciably the result, since the partial contributions to the global solution decay very rapidly as the number of branches increases. However, for the class of problems discussed here, the expansion in Eq.(18) may give rise to a divergent series with coefficients, associated to the partial contributions to the solution, growing as the number of branches increases. Truncating the expansion, or equivalently pruning the trees, might be applied, but because it turns out to be uncontrollable, a special care should be taken. In particular the effect of including more coefficients in the expansion, increasing further the order of the Pade approximation, will be analyzed for the test examples considered below. Recall that in general the convergence of the Pade approximant can be affected by artificial poles present in the denominator of the approximant, but not being own by the function to be approximated, see e.g. [8]. Therefore, to assess properly the validity of our findings, it becomes essential to compute the Pade approximant for different number of coefficients.

Concerning the apparent robustness of the Pade approximant against the statistical error affecting the coefficients of the power series in (18), a main reason could be that the solution of the test examples seems to be apparently locally Lipschitz. Thus, the error made in computing the coefficients of the Pade approximant should be bounded. In fact, in [26] it has been proved the following

related theorem

$$\| P_f - P_{f'} \| \leq K \| c - c' \|, \quad (46)$$

provided that $\| c - c' \| \leq d$. Here P_f , and $P_{f'}$ are the Pade approximants of order (m, n) in $[a, b]$ of a given power series f and f' with coefficients c_j , and c'_j respectively, being $\| c \| = \max_{i \leq i \leq n+m} |c_i|$, f locally Lipschitz, and K and d constants depending only on c_i and $[a, b]$.

Moreover, it is worth to observe that both errors described above may be alleviated in any case by increasing conveniently the sample size N , and considering more coefficients in the expansion in order to compute the Pade approximant.

In the following, we present several test examples concerning one-dimensional initial value parabolic problems to illustrate what it was described before. The solution was computed probabilistically at the points (x, t) , where $x = 0$, and t several values chosen to be distributed between 0 and T . In absence of an analytical solution the results were compared with the solution obtained upon applying an implicit finite difference scheme with a very fine mesh, and solving the ensuing algebra linear problem, characterized by a banded matrix, with LAPACK.

Example 1. An IV parabolic problem with a purely quadratic negative non-linear term. Consider the problem

$$\begin{aligned} u_t &= u_{xx} - u^2, \quad x \in \mathbf{R}, \\ u(x, 0) &= \frac{e^{-\frac{x^2}{4(t+1)}}}{\sqrt{4\pi(t+1)}} \end{aligned} \quad (47)$$

The numerical error made when solving probabilistically Example 1 at a few points with $x = 0$, and $t \in [0, 1]$, using both strategies, A and B, are depicted in Fig. 8 and 9, respectively. Note that for both strategies, truncating the expansion to only four coefficients, that is pruning the trees to $kmax = 4$ branches, is already close to convergence for any purpose. Although for this example the number of coefficients to be included in the expansion could be any number above $kmax = 4$, it becomes clear that choosing a larger number rather than improving accuracy, it acts reversely degrading them. In fact larger number of coefficients corresponds to contributions to the solution coming from random trees with large number of branches, and as it was explained above such contributions are affected by larger statistical error. Moreover, it turns out to be disadvantageous as well under a computational point of view, since generating trees with large number of branches have been proved to be rather inefficient.

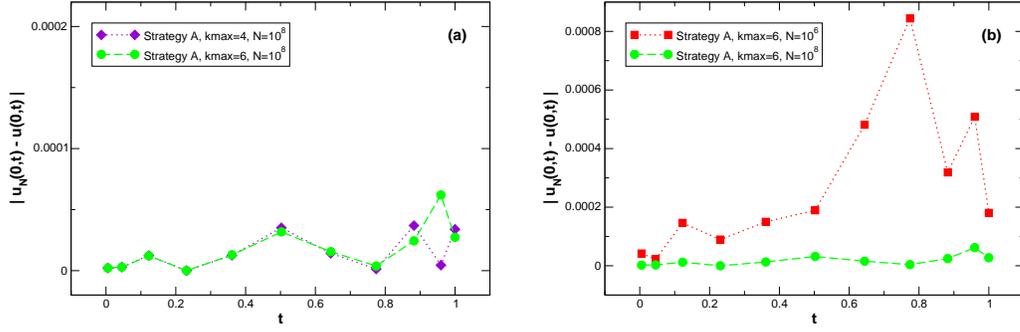


Fig. 8. Numerical error made when solving Example 1 with the strategy A. This has been done for two different values of the number of coefficients in the corresponding expansion, denoted by k_{max} . The number of realizations N was kept fixed in (a), while in (b) k_{max} was kept fixed.

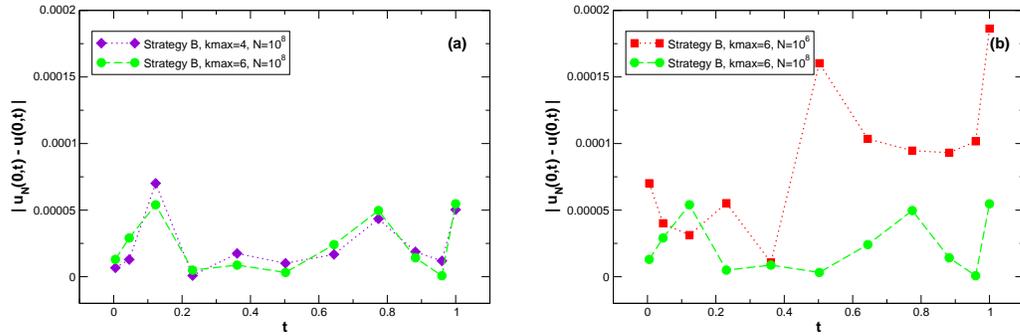


Fig. 9. Numerical error made when solving Example 1 with the strategy B. Identical values as in Fig. 8 have been used to obtain the results plotted in (a) and (b).

Finally, note that keeping fixed the number of coefficients, and increasing the sample size, N , reduces accordingly the statistical error as expected.

Similar results are shown for the strategy B in Fig. 9, and therefore identical conclusions hold for this case.

Example 2. An IV problem with a negative initial condition, $u(x, 0) < 0$. Consider the problem

$$\begin{aligned}
 u_t &= u_{xx} + u^2, \quad x \in \mathbf{R}, \\
 u(x, 0) &= -6 \frac{e^{-\frac{x^2}{4(t+1)}}}{\sqrt{4\pi(t+1)}}
 \end{aligned} \tag{48}$$

Note that the initial condition is now defined negative, and greater than 1 in absolute value. Results are depicted in Fig. 10 and 11, corresponding to strategies A and B, respectively. As in Example 1, similar conclusions can be reached.

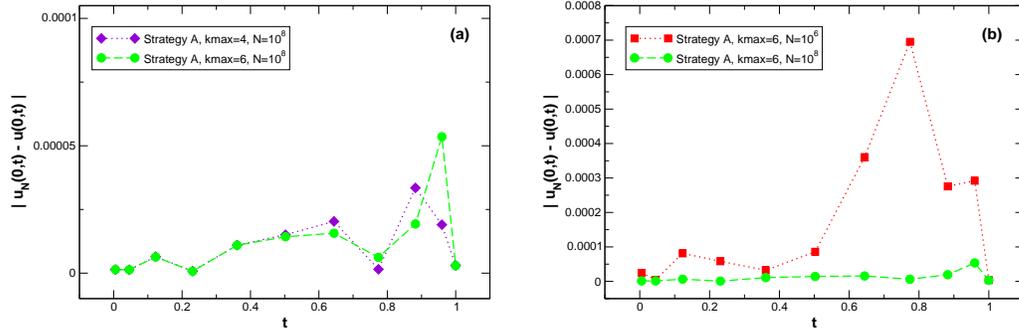


Fig. 10. Numerical error made when solving Example 2 with the strategy A. This has been done for two different number of coefficients in the corresponding expansion, denoted by k_{max} . The number of realizations N was kept fixed in (a), while in (b) it was kept fixed k_{max} .

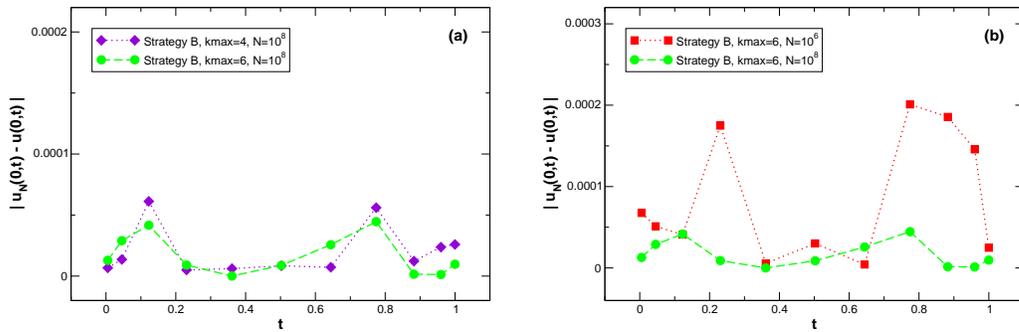


Fig. 11. Numerical error made when solving Example 2 with the strategy B. Identical values as in Fig. 10 have been used to obtain the results plotted in (a) and (b).

Example 3. An IV problem with two nonlinear terms. Consider the more general problem

$$\begin{aligned}
 u_t &= u_{xx} - (1+a)u^2 - u^3, \quad x \in \mathbf{R}, \\
 u(x, 0) &= \frac{1}{1 + e^{-\frac{x + \sqrt{2}(0.5-a)t}{\sqrt{2}}}}, \quad (49)
 \end{aligned}$$

where the parameter a has been chosen arbitrarily to be 0.25.

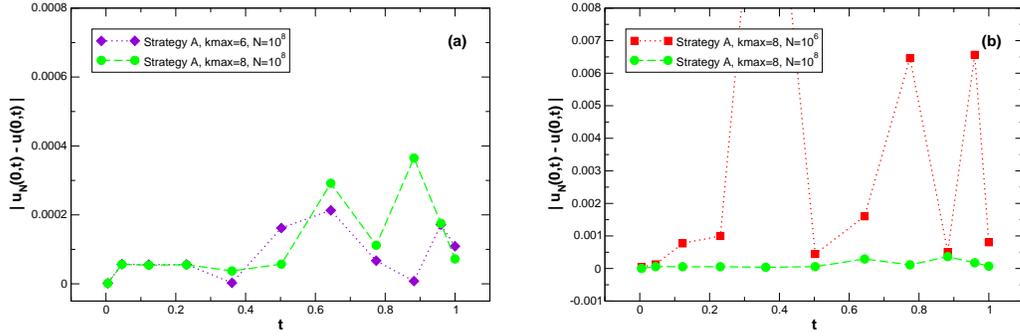


Fig. 12. Numerical error made when solving Example 3 with the strategy A. This has been done for two different number of coefficients in the corresponding expansion, denoted by k_{max} . The number of realizations N was kept fixed in (a), while in (b) it was kept fixed k_{max} .

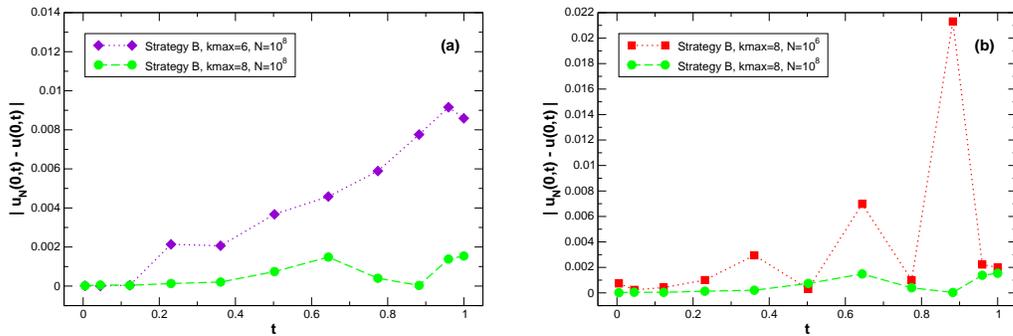


Fig. 13. Numerical error made when solving Example 3 with the strategy B. Identical values as in Fig. 12 have been used to obtain the results plotted in (a) and (b).

Clearly this consists of a more involved example compared with the previous cases, since now the nonlinear function is composed of two different terms. Moreover, the coefficients multiplying both terms appear to be negative, and one of them even greater than 1. Obviously, the joint effect of both terms gives rise to a more complex solution, suggesting the need of considering a larger number of coefficients in the expansion in order to reach convergence for the Pade approximant. This is indeed what it is observed in Fig. 12(a) and 13(a). In particular for this example, it can be seen that the strategy B seems to require more coefficients than the strategy A.

Again as in the examples above, in Figs. 12(b) and 13(b) it can be observed that increasing the sample size, N , for both strategies reduces largely the statistical error, and in turn improves the convergence of the Pade approximant to the solution.

To conclude, both strategies showed similar performance in all test examples analyzed, however the strategy B turns out to be advantageous in any case, because when implemented in practice, the computational time increases linearly with the final time, growing unboundedly rather for the strategy A.

3 Numerical examples

The probabilistic representation described in Sec. 2 can be hardly used for solving efficiently semilinear parabolic problems in a whole domain, due to the high computational cost of evaluating the solution at single points. However, such a representation can be combined successfully with a classical domain decomposition method, as it was proposed in [1,2]. The method was called probabilistic domain decomposition (PDD for short), and consists of a hybrid algorithm which requires generating only few interfacial values along given, possibly artificial interfaces inside the domain, then obtaining approximate values upon interpolation on such interfaces. Such values are used as boundary data to split the original problem into a number of fully decoupled sub-problems. The main advantage of this method is that the corresponding codes are especially suited for *massively parallel computing* [20]. In fact, being the solution obtained probabilistically through an expected value over a given finite sample whose elements are independent from each other, and then after the domain decomposition the corresponding sub-problems fully decoupled, the implemented parallel codes are characterized by an extremely low communication overhead among the various processors, affecting positively crucial properties such as *scalability* and *fault tolerance*. In the following, we describe briefly the main parts of the PDD algorithm, and for more details we refer the reader to [3], e.g.

Probabilistic part. This is the first step to be carried out, and consists of computing the solution of the PDE at a few suitable points by some of the probabilistic strategies described in Sec. 2.

Interpolation. Once the solution has been computed at few points on each interface, a second step consists of interpolating on such points, being used as nodal points, thus obtaining continuous approximations of interfacial values of the solution. For this purpose, since the examples analyzed below corresponds to two-dimensional problems a tensor product interpolation based on cubic spline [5] was used. The computational cost of this part turns out to be negligible compared with the time spent in the other parts of the algorithm. The nodal points are uniformly distributed on each plane, and a not-a-knot condition is imposed.

Local solver. The third and final step consists of computing the solution in-

side each subdomain, this task being assigned to different processors. This can be accomplished resorting to local solvers, which may use classical numerical schemes, such as implicit finite differences for simple geometries or finite elements methods for more complex configurations. For the former case, sub-routines based on LAPACK for solving the ensuing linear algebra problems has been chosen, since the corresponding matrices are banded. Therefore, each processor can be devoted only to the solution of its local linear system, whose banded associated matrix is smaller. Concerning the memory consumption per processor, including an extra fill-in space, the total amount is considerably reduced [4].

In Fig. 14 we sketch a diagram, illustrating how the algorithm works in practice for a two-dimensional case. Here the solution is obtained probabilistically at a few points pertaining to some “interfaces” conveniently chosen inside the space-time domain $D := \Omega \times [0, T]$, with $\Omega \subset \mathbb{R}^2$. Such interfaces divide the domain into p subdomains, Ω_i , from $i = 1, \dots, p$, being assigned to different processors, $p_i, i = 1, \dots, p$. The more convenient way to parallelize this part is splitting in independent sets of points. Since the number of points where the solution is computed is larger than the number of processors p , computing such a solution can be assigned as a task to different processors. This can be seen as a coarse-grain parallelization, and even though other finest strategies can be adopted, this one turns out to be the more convenient for the examples analyzed in this section.

Here we present some numerical examples for 2D initial value problems to illustrate the PDD algorithm, being the probabilistic part built up with the two strategies A and B discussed in the previous section. All simulations were carried out on the Matrix supercomputer, belonging to the Inter-University Consortium for the Application of Super-Computing for Universities and Research (CASPUR) located in Rome (Italy), using up to 512 processors. This supercomputer consists of a Linux cluster based on multi-core Opteron processor nodes with Infiniband interconnection, and it was ranked in the Top500 list with a peak performance of 22 TFlops.

As in [3,4], a comparison was made solving the same problems by some other classical numerical methods in order to asses the performance of both methods. For the space-time domain as well as for the subdomains in our decomposition, we used the Crank-Nicolson (implicit) finite difference method. On the various decoupled subdomains obtained by the PDD algorithm we used LAPACK for solving the ensuing linear algebra problem, while the full domain solution was computed by ScaLAPACK. This widely used and freely available numerical package has been considered extremely efficient for the parallel solution of banded linear systems. For more details concerning the computational cost of both methods, LAPACK and ScaLAPACK, see [4], e.g.

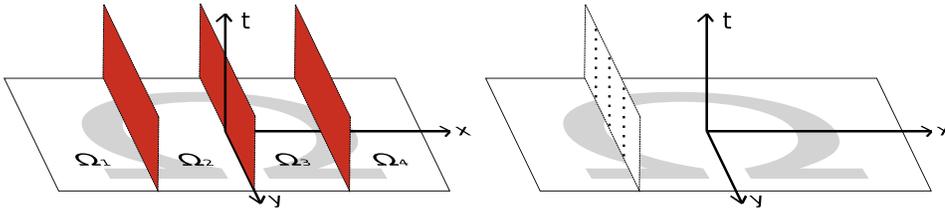


Fig. 14. A sketchy diagram illustrating the main steps of the algorithm in 2D: The figure on the left shows how the domain decomposition is done in practice. The figure on the right shows the points where the solution is computed probabilistically; these are used afterward as nodal points for interpolation.

Example 4. An IV problem with two nonlinear terms. Consider the problem

$$\begin{aligned}
 u_t &= u_{xx} + u_{yy} - (1 + a)u^2 - u^3, \quad (x, y) \in \mathbf{R}^2, \\
 u(x, y, 0) &= -2 \cos^2\left(\frac{\pi x}{2A_x}\right) \cos^2\left(\frac{\pi y}{2A_y}\right).
 \end{aligned} \tag{50}$$

where $a = 0.25$, $A_x = 10$, $A_y = 40$. The space and time discretization step has been chosen to be $\Delta x = \Delta y = 0.25$, $\Delta t = 10^{-3}$, and the solution was computed for a final time $T = 0.5$.

Note that the unbounded domain should be truncated conveniently to a bounded domain in order to be able to solve numerically the problem using a finite difference scheme. This requires introducing some artificial boundary conditions to confine the computational domain. Since the problem is formulated as a pure initial value problem, the artificial boundary conditions should be prescribed in such a way no additional data are imposed on such boundaries. In practice, this can be done readily imposing Dirichlet boundary conditions on the artificial boundaries, such that the boundary conditions are automatically satisfied by the solution of the problem. However, being the solution of the problem unknown, one should resort to several type of approximations of the solution to be used as boundary conditions. For the problem above, the solution of the problem is assumed to decay sufficiently fast to infinity, and being the computational domain chosen to be large enough $\Omega \in [-L_x, L_x] \times [-L_y, L_y]$, with $L_x = 40$ and $L_y = 160$, a zero Dirichlet boundary condition can be properly imposed at $x = \pm L_x, y = \pm L_y$.

When a probabilistic representation is available, such a representation were used as well to obtain much more accurate approximations for the artificial boundary conditions, since it allows to obtain the solution at any single point arbitrarily chosen. This is remarkable feature of the probabilistic representation, not owned by any other numerical method.

In Fig. 15 and Fig. 16 the pointwise numerical error around the origin made with the strategy A and strategy B, respectively, is shown. Here the value of

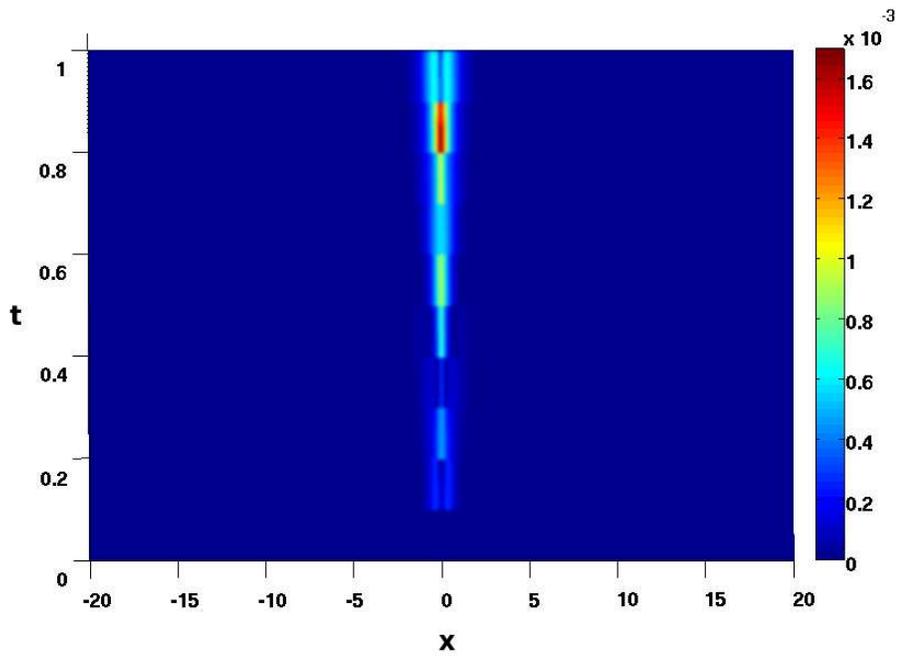


Fig. 15. Pointwise numerical error made when solving Example 4 with the strategy A.

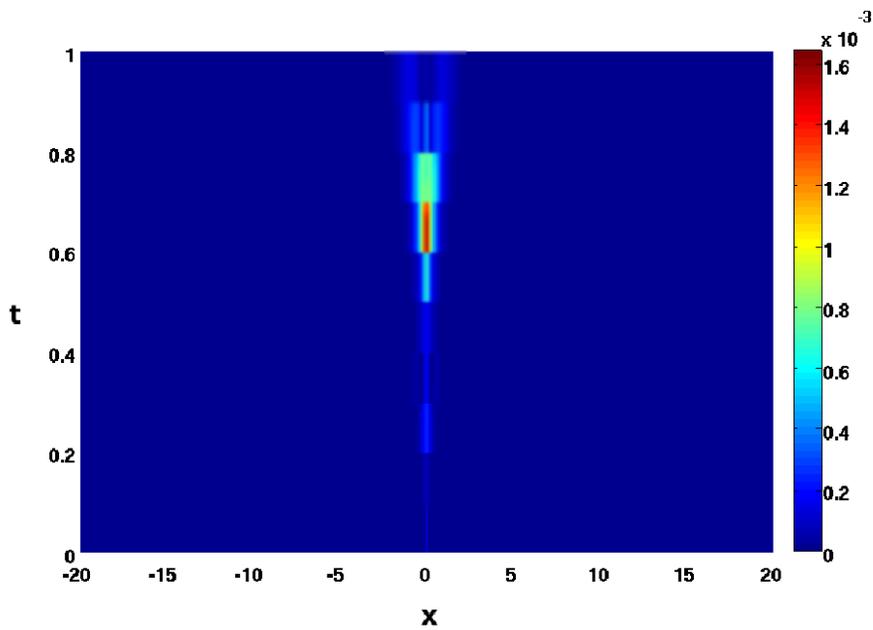


Fig. 16. Pointwise numerical error made when solving Example 4 with the strategy B.

the parameters were kept fixed to $\Delta x = \Delta y = 10^{-2}$, $\Delta t = 10^{-3}$, $T = 1$. For clarity, only the maximum absolute value of the error obtained in the y -axis is plotted.

Table 1

Example 4: T_{PDD} , $T_{ScaLAPACK}$ denotes the computational time spent in seconds by PDD with the strategy A, and ScaLAPACK, respectively. T_{MC} , and T_{INT} correspond to the time spent by the probabilistic and the interpolation part, respectively; *Memory* denotes the total memory consumption.

<i>Procs.</i>	T_{MC}	T_{INT}	<i>Memory</i>	T_{PDD}	$T_{ScaLAPACK}$
128	902"	<1"	0.86 GBs	5572"	29881"
256	998"	<1"	0.19 GBs	2086"	23953"
512	1018"	<1"	0.06 GBs	1327"	23334"

Table 2

Example 4: T_{PDD} , $T_{ScaLAPACK}$ denotes the computational time spent in seconds by PDD with the strategy B, and ScaLAPACK, respectively. T_{MC} , and T_{INT} correspond to the time spent by the probabilistic and the interpolation part, respectively; *Memory* denotes the total memory consumption.

<i>Procs.</i>	T_{MC}	T_{INT}	<i>Memory</i>	T_{PDD}	$T_{ScaLAPACK}$
128	736"	<1"	0.86 GBs	5413"	29881"
256	824"	<1"	0.19 GBs	1917"	23953"
512	837"	<1"	0.06 GBs	1152"	23334"

In Table 1 and Table 2, the computational times obtained when solving the example 4 using the PDD algorithm with the strategy A and the strategy B, respectively, are shown. The partial computational times spent by the probabilistic part and the interpolation part of the algorithm, as well as the computational time spent by ScaLAPACK, have also been displayed. The two methods were compared correspondingly to the same maximum error, 10^{-3} .

It is worth to observe that the computational times obtained with the strategy B are significantly smaller than those obtained with the strategy A. This can be explained in view of the less computational cost of the probabilistic representation based on the strategy B, as it was already theoretically shown in the previous section.

Example 5. An IV problem with a single nonlinear term and variable coefficients. Consider the problem

$$\begin{aligned}
 u_t &= u_{xx} + u_{yy} - e^{-x^2} \frac{1}{t+0.1} u^2, \quad (x, y) \in \mathbf{R}^2, \\
 u(x, y, 0) &= \frac{e^{-\frac{x^2}{4}}}{\sqrt{4\pi}}.
 \end{aligned} \tag{51}$$

The space and time steps are $\Delta x = \Delta y = 0.25$, $\Delta t = 10^{-3}$, and the solution

was computed at the final time $T = 0.5$.

Note that in this example a variable coefficient depending on time and space, multiplying the nonlinear term u^2 has been considered, and it may be in general taken values larger than one.

The computational times are shown in Table 3 and Table 4, comparing the strategy A and B, respectively, with ScaLAPACK. Note that the computational times turns out to be slightly smaller than those obtained in the previous example, and this is because the generated random trees now are purely binary. The strategy B wins again over the strategy A, and the same reason of the previous example holds also for the present case.

Table 3

Example 5: T_{PDD} , $T_{SCALAPACK}$ denotes the computational time spent in seconds by PDD with the strategy A, and ScaLAPACK, respectively. T_{MC} , and T_{INT} correspond to the time spent by the Monte Carlo and the interpolation part, respectively; *Memory* denotes the total memory consumption.

<i>Procs.</i>	T_{MC}	T_{INT}	<i>Memory</i>	T_{PDD}	$T_{ScaLAPACK}$
128	807"	<1"	0.86 GBs	5461"	27466"
256	889"	<1"	0.19 GBs	1954"	22070"
512	906"	<1"	0.06 GBs	1102"	21327"

Table 4

Example 5: T_{PDD} , $T_{ScaLAPACK}$ denotes the computational time spent in seconds by PDD with the strategy B, and ScaLAPACK, respectively. T_{MC} , and T_{INT} correspond to the time spent by the probabilistic and the interpolation part, respectively; *Memory* denotes the total memory consumption.

<i>Procs.</i>	T_{MC}	T_{INT}	<i>Memory</i>	T_{PDD}	$T_{ScaLAPACK}$
128	511"	<1"	0.86 GBs	5148"	27466"
256	562"	<1"	0.19 GBs	1623"	22070"
512	569"	<1"	0.06 GBs	882"	21327"

4 Summary

The class of semilinear parabolic problems amenable to a probabilistic solution has been expanded by introducing suitable generalized random trees. The probabilistic computation consists of evaluating averages on the generated random tree, which plays a role similar to that of a random path in linear problems. The new representation allows treatment of semilinear problems without a potential term, with arbitrary coefficients multiplying the nonlinear term, and arbitrary initial data, including negative definite and greater than one.

The implementation uses two different strategies, which require computing the solution through a series where the coefficients represent the partial contribution to the solution coming from generated random trees with any number of branches. Since such a series might be divergent, in general it cannot be summed simply by a sequence of partial sums. Nevertheless, numerical experiments show that, in many cases, the asymptotic series can be approximated quite accurately by techniques based on the Pade approximant. A qualitative analysis of the error done has been carried out, showing that for the test problems analyzed so far, considering a few coefficients of the series suffices to obtain a reasonable accuracy.

Moreover, it has been shown that the strategy termed B greatly reduces the computation time compared with strategy A, which is based rather on generating random trees governed by an exponential random time. The new probabilistic representation has been used successfully as a crucial element for implementing a suitable probabilistic domain decomposition method. In contrast to the classical deterministic method for solving partial differential equations, the probabilistic approach computes the solution at single points internal to the domain, without first generating a computational mesh and solving the full problem. The generalized PDD method has been shown to be suited for massively parallel computers. In fact, some numerical examples have been run that show excellent scalability properties of the PDD algorithm in large-scale simulations, using up to 512 processors on a high performance supercomputer. Finally, the performance of the algorithm has been compared with other efficient, freely available parallel algorithms, showing a striking difference.

Acknowledgments

This work was supported by the Portuguese FCT under grant PTDC/EIA-CCO/098910/2008. The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Rome Supercomputing center (CASPUR).

References

- [1] Acebrón, J.A., Busico, M.P., Lanucara, P., and Spigler, R., *Domain decomposition solution of elliptic boundary-value problems*, SIAM J. Sci. Comput. **27** (2005), 440-457.
- [2] Acebrón, J.A., Busico, M.P., Lanucara, P., and Spigler, R., *Probabilistically induced domain decomposition methods for elliptic boundary-value problems*, J. Comput. Phys. **210** (2005), 421-438.

- [3] Acebrón, J.A., R., Rodríguez-Rozas, A., and Spigler, R., *Domain decomposition solution of nonlinear two-dimensional parabolic problems by random trees*, J. Comput. Phys. **15** (2009), 5574-5591.
- [4] Acebrón, J.A., R., Rodríguez-Rozas, A., and Spigler, R., *Efficient Parallel Solution of Nonlinear Parabolic Partial Differential Equations by a Probabilistic Domain Decomposition*, J. Sci. Comput. **43** (2010), 135-157.
- [5] Antia, H.M., *Numerical methods for scientists and engineers*, Tata McGraw-Hill, New Delhi, 1995.
- [6] Aval, J.C., *Multivariate Fuss-Catalan numbers*, Discrete Math. **308** (2008), 46604669.
- [7] Baker, G.A., and Gammel, J.L., *The Padé approximant*, Journal of Mathematical Analysis and Applications, **2** (1961) 21-30.
- [8] Bender, C., and Orszag, S.A., *Advanced Mathematical Methods for Scientists and Engineers*, McGraw Hill, New York (1978)
- [9] Chan, Tony F., and Mathew, Tarek P., *Domain decomposition algorithms*. Acta Numerica (1994), 61-143 [Cambridge University Press, Cambridge, 1994].
- [10] Dongarra, J., Beckman, P., et al, *International Exascale Software Project Roadmap*, UT-CS-10-652 (2010)
- [11] DuChateau, P. and Zachmann, D., *Applied Partial Differential Equations*. Dover Publications (2002).
- [12] Floriani, E., Lima, R., and Vilela Mendes, R., *Poisson-Vlasov: Stochastic representation and numerical codes*, Eur. Phys. Journal. D **46** (2008) 295-302.
- [13] Freidlin, M., *Functional Integration and Partial Differential Equations*. Annals of Mathematics Studies no. 109, Princeton Univ. Press, Princeton (1985)
- [14] Graham, R.L., Knuth, D.E., and Patashnik, O., *Concrete Mathematics: a Foundation for Computer Science*. 2nd ed., Addison-Wesley Professional, 1994.
- [15] Karatzas, I., and Shreve, S.E., *Brownian Motion and Stochastic Calculus*. 2nd ed., Springer, Berlin (1991)
- [16] Kloeden, P.E., and Platen, E., *Numerical Solution of Stochastic Differential Equations*. Springer, Berlin (1992)
- [17] Kogge, P.M., and et al, *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, DARPA Information Processing Techniques Office, Washington, DC, pp.278, September 28, 2008.
- [18] McKean, H.P., *Application of brownian motion to the equation of Kolmogorov-Petrovskii-Piskunov*. Comm. on Pure and Appl. Math., **28** (1975), 323-331.
- [19] Milstein, G.N., and Tretyakov, M.V., *Stochastic Numerics for Mathematical Physics*. Springer (2004)

- [20] Petersen, W., and Arbenz, P., *Introduction to parallel computing. A practical guide with examples in C*. Oxford Univ. Press, (2004).
- [21] Quarteroni, A., and Valli, A., *Domain Decomposition Methods for Partial Differential Equations*. Oxford Science Publications, Clarendon Press, Oxford (1999)
- [22] Ramirez, J.M., *Multiplicative cascades applied to PDEs (two numerical examples)*, J. Comput. Phys., **214** (2006), 122-136.
- [23] Regnier, H., and Talay, D., *Convergence rate of the Sherman and Peskin branching stochastic particle method*. Proc. Royal Soc. Lond. Ser. A Math. Phys. Eng. Sci. **460** 199-220 (2004).
- [24] Sarkar, V., Harrod, W., and Snavely, A.E., *Software challenges in extreme scale systems*. Journal of Physics: Conference Series pp. 012045 (2009)
- [25] Stevens, R., Zacharia, T., and Simon, H., *Modeling and Simulation at the Exascale for Energy and the Environment Town Hall Meetings Report*. Department of Energy Office of Advance Scientific Computing Reserach, Washington, DC, pp. 174 (2008)
- [26] Wuytack, L., *On the conditioning of the Pade approximant problem*, Lect. Notes Math.**888** (1981), 78-89.