Hybrid PDE solver for data-driven problems and modern branching†

F R A N C I S C O   B E R N A L[1],   G O N Ç A L O   D O S   R E I S[2,3]   and   G R E I G   S M I T H[2,4]

[1]*CMAP - Centre de Mathématiques Appliquées, Ecole Polytechnique, Route de Saclay,*
91128 Palaiseau Cedex, France
email: Francisco.Bernal@polytechnique.edu
[2]*School of Mathematics, University of Edinburgh, Edinburgh, EH9 3FD, UK*
email: G.dosReis@ed.ac.uk
[3]*Centro de Matemática e Aplicações (CMA), FCT, UNL, 2829-516 Caparica, Portugal*
[4]*Maxwell Institute Graduate School in Analysis and its Applications, University of Edinburgh,*
Edinburgh, EH9 3FD, UK
email: G.Smith-13@sms.ed.ac.uk

The numerical solution of large-scale PDEs, such as those occurring in data-driven applications, unavoidably require powerful parallel computers and tailored parallel algorithms to make the best possible use of them. In fact, considerations about the parallelization and scalability of realistic problems are often critical enough to warrant acknowledgement in the modelling phase. The purpose of this paper is to spread awareness of the Probabilistic Domain Decomposition (PDD) method, a fresh approach to the parallelization of PDEs with excellent scalability properties. The idea exploits the stochastic representation of the PDE and its approximation via Monte Carlo in combination with deterministic high-performance PDE solvers. We describe the ingredients of PDD and its applicability in the scope of data science. In particular, we highlight recent advances in stochastic representations for non-linear PDEs using branching diffusions, which have significantly broadened the scope of PDD. We envision this work as a dictionary giving large-scale PDE practitioners references on the very latest algorithms and techniques of a non-standard, yet highly parallelizable, methodology at the interface of deterministic and probabilistic numerical methods. We close this work with an invitation to the fully non-linear case and open research questions.

Key words: Probabilistic Domain Decomposition, high-performance parallel computing, Marked branching diffusions, hybrid non-linear PDE solvers, Monte Carlo methods.

2010 AMS Subject Classification: Primary: 65C05, 65C30, Secondary: 65N55, 60H35, 91-XX, 35CXX

1 Introduction

Partial Differential Equations (PDEs) are ubiquitous in modelling, appearing in image analysis and processing, inverse problems, shape analysis and optimization, filtering, data assimilation and optimal control. They are used in Math–Biology to model population dynamics with competition or growth of tumours; or to model complex dynamics of movement of persons in crowds or to model (ir)rational decisions of players in games and they feature in many complex problems in Mathematical Finance. Underpinning all these applications is the necessity of solving numerically such equations either in bounded or unbounded domains.

Deterministic Domain Decomposition. The standard example of a Boundary Value Problem (BVP) is Laplace's equation with Dirichlet Boundary Conditions (BCs):

$$\Delta u(\mathbf{x}) = 0 \text{ if } \mathbf{x} \in \Omega \subset \mathbb{R}^d, \qquad u(\mathbf{x}) = g(\mathbf{x}) \text{ if } \mathbf{x} \in \partial\Omega. \qquad (1.1)$$

The large data sets involved in realistic applications nearly always imply that the discretization of a BVP such as (1.1) leads to algebraic systems of equations that can only be solved on a parallel computer with a large number (say $p \gg 1$) of processors. Not only does parallelization require multiple processors but also parallel algorithms. The classical Schwarz's alternating method was the first and remains the paradigm of such algorithms which we refer to as "Deterministic Domain Decomposition" (DDD) [58]. While state-of-the-art DDD algorithms outperform Schwarz's alternating method in every respect, the latter nonetheless serves to illustrate the crucial difficulty they all face. The idea of Schwarz's algorithm is to divide Ω into a set of p overlapping subdomains, and have processor $j = 1, \ldots, p$ solve the restriction of the PDE to the subdomain, Ω_j, see Figure 1.

Since the solution is not known in the first place, the BCs on the fictitious interfaces of Ω_j are also unknown, therefore, an initial guess has to be made in order to give processor j a well-posed (yet incorrect) problem. The BCs along the fictitious interfaces of Ω_j are then updated from the solution of the surrounding subdomains in an iterative way until convergence.

Inter-processor communication and the scalability limit of DDD. Since the inter-processor communication involved in DDD's updating procedure is intrinsically sequential, it sets a limit to the scalability of the algorithm by virtue of the well-known Amdahl's law.
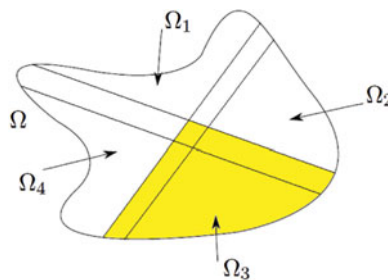

FIGURE 1. Domain decomposition on an arbitrary domain Ω, split into four overlapping subdomains Ω_i as required for Schwarz's alternating method. The subdomain Ω_3 is highlighted.

A simple illustrative example is as follows. A fully scalable algorithm would take half the time (say $T/2$) to run if the number of processors was doubled. If there is a fraction $v < 1$ of the algorithm which is sequential, then the completion time will not drop below Tv, regardless of how many processors are added. For instance, in Schwarz's method, if 5% (i.e. $v = 0.05$) of the execution time of one given processor is lost by waiting for the artificial BCs to be ready, then the execution time could be shortened by at most a factor of 20 (with infinitely many processors; a factor of about 19 would already take over 1,000 processors). In other words, Schwarz's alternating algorithm, or *any* DDD algorithm for that matter, cannot exploit the full capabilities of a parallel computer due to the idle time wasted in (essential) communication.

We emphasize this point further by borrowing an example from David Keyes[1]. The Gordon Bell prizes are annually awarded to numerical schemes which achieve a breakthrough in performance when solving a realistic problem. In 1999, one such problem was the simulation of the compressible Navier–Stokes equations around the wing of an airplane. With 128 processors, the winning code took 43 minutes to solve the task. On the other hand, with 3,072 processors, it took it 2.5 minutes instead of 1.79 as would have been the case with a fully parallelizable algorithm. The remaining 28% of computer time were lost to interprocessor communication. At this point, adding more processors would have led to a faster loss of scalability.

The Probabilistic Domain Decomposition (PDD) method. A conceptual breakthrough was achieved by Acebrón *et al.* with the PDD method (or rather, the PDD framework) [1], based on a previous, unpublished idea by Renato Spigler. PDD is the only domain decomposition method potentially free of communication, and thus potentially fully scalable. It does so by splitting the simulation in two separate stages, the first stage recasts the BVP into a stochastic formulation (via the so-called Feynman–Kac formula) which allows to compute the solution of the PDE at certain specific points in time/space. Thus, we can compute the "true" solution values of the DDD's fictitious interfaces for the Ω_j's. Therefore, the fictitious boundaries that were previously unknown are now known!

Consequently, the subdomains are now completely independent of each other, the second stage then involves solving for the solution over the subdomains in a full parallel way. PDD calculations will be affected by two independent sources of numerical error: the subdomain solver and the statistical error of Monte Carlo simulations.

PDD is currently well understood for the linear case, although recent advances in stochastic representations have opened the door for using PDD with non-linear PDEs. A class of non-linear PDEs strongly amenable to PDD are those whose solution can be represented by the so-called branching processes, as introduced by [51, 57, 61] and recently extended by [43] and [42]. This methodology avoids backward regressions, the so-called "Monte Carlo of Monte Carlo simulation" problem, see Section 4 below or [41, Section 3.1]. To illustrate the potential of branching in PDD, numerical examples are worked out in Section 3. For general mildly non-linear PDEs, the straightforward (and often efficient) approach of linearization and solution of each of the linear iterates, would be easily implementable with PDD, without resorting to non-linear representations.

The general case of systems of non-linear second-order parabolic/elliptic PDEs or fully non-linear PDEs are, to the best of our knowledge, yet been addressed in the framework

---

[1] See http://www.mcs.anl.gov/research/projects/petsc-fun3d/Talks/bellTalk.ppt

of PDD. These types of PDEs admit a probabilistic representation in terms of Forward Backward SDEs (FBSDEs [28]) and numerical methods for FBSDEs is currently the subject of extensive research.

The remainder of the paper is organized as follows:

- Section 2 is an overview of PDD. We illustrate the connection between stochastic and BVPs in the linear case, and provide pointers for the numerical methods required. We stress the notion of balancing the various aspects of PDD in order to speed up the overall algorithm. The section closes with a survey of reported results on PDD.
- In Section 3, non-linear equations that can be represented probabilistically in terms of branching processes are discussed, and an account of recent developments is given. We give examples with convincing numerical tests using branching.
- Section 4 is an invitation to new, more challenging problems than those tackled so far. In particular, the stochastic representation of systems of non-linear parabolic equations with FBSDEs is succinctly discussed. We highlight open research problems.

2 An overview of probabilistic domain decomposition

An alternative to deterministic methods which is specifically designed to circumvent the scalability issue of DDD is the PDD method [1,2]. First, the domain Ω under consideration is divided into non-overlapping subdomains. (Rather than overlapping ones, such as with Schwarz's alternating algorithm.) PDD then consists of two stages, first, the solution is calculated only on a set of interfacing nodes along the artificial interfaces, by solving the stochastic representation of the BVP with the Monte Carlo method, using the so-called Feynman–Kac representations. The stochastic representation is the crux of PDD and can be highly non-trivial. Nonetheless, it can also be extremely simple, for instance, in (1.1), it is well known (see [45]) that the solution u can be represented as

$$u(\mathbf{x}) = \mathbb{E}_{\mathbf{x},0}\left[g(\mathbf{X}_\tau) \right] \tag{2.1}$$

where $\mathbf{X}$ is the solution to the simplest Stochastic Differential Equation (SDE)

$$\text{for } t \geqslant 0 \quad \mathrm{d}\mathbf{X}_t = \mathrm{d}\mathbf{W}_t, \quad \mathbf{X}_0 = \mathbf{x} \in \mathbb{R}^d \quad \Leftrightarrow \quad \mathbf{X}_t = \mathbf{x} + \mathbf{W}_t, \tag{2.2}$$

where $(\mathbf{W}_t)_{t \geqslant 0}$ is a d-dimensional Brownian motion; $\mathbf{X}_\tau$ is the point on $\partial\Omega$ where the trajectory of $\mathbf{X}_t$ *first* hits the boundary and $\mathbf{X}$ starts at $t = 0$ from $\mathbf{x}$; and last, $\mathbb{E}[\cdot]$ is the usual expectation operator (in (2.1), $\mathbb{E}_{\mathbf{x},0}[\cdot]$ emphasizes that the diffusion $(\mathbf{X}_t)_{t \geqslant 0}$ starts at time $t = 0$ from position $\mathbf{x}$). See Figure 2 for an illustration.

The expected value above is approximated via a Monte Carlo method (the mean over many independent realizations of the SDE (2.2), which can be carried out in a fully parallelizable way), introducing some statistical error. A numerical scheme is required to solve the SDE, we refer to it as "stochastic numerics" [46].

By using the stochastic representation (2.1), one can compute the equation's solution at each point at the artificial boundaries $\Omega_i \cap \Omega_j$. It is then possible to reconstruct (approximately) the solution on the interfaces (interpolating the values at interfacing nodes
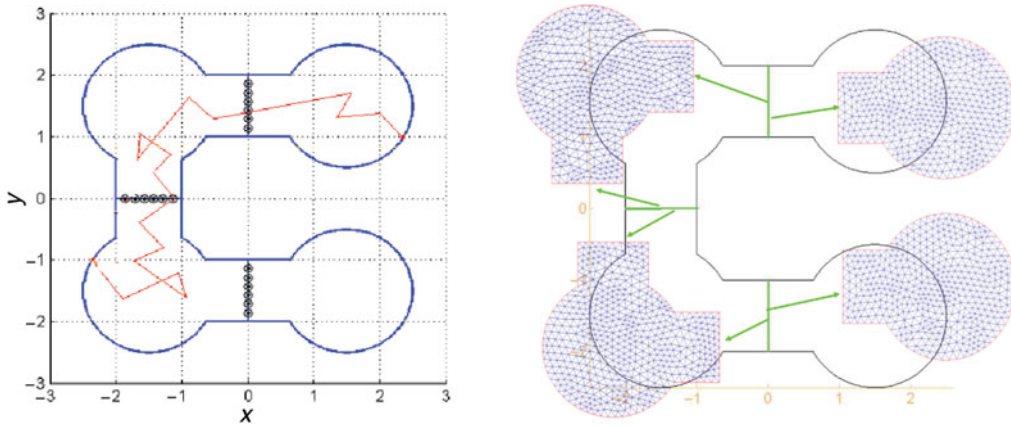
FIGURE 2. An illustration of the two stages of PDD. (Left) First, the solution is computed on each of the interface nodes (black circles) between subdomains by numerically solving an appropriate SDE, which involves many independent realizations from the same interfacing node. Two such realizations are the trajectories shown in red. (Right) After the nodal solutions are known, the artificial interfaces (green segments) are provided with a Dirichlet BC by interpolation, and the BVPs on the various subdomains are decoupled. They can then be solved independently from one another with a deterministic method, such as FEM (meshes depicted).

with Chebyshev polynomials, say), so that the PDE restricted to each of the subdomains is now well posed. Now, the p Laplace's equations on each of the p subdomains are separate problems, and can be independently solved "deterministically", the second stage of PDD. Note that both stages in the PDD are *embarrassingly parallel* by construction. We stress that this has been made possible only by the unique capability of stochastic representations of PDEs for solving a BVP at an isolated point (time-space) of the domain. On the other hand, DDD methods are matrix based and deliver the solution everywhere; but by doing this are bound to move around information.

Remark 2.1 (Further features: Fault tolerance and boundary design) *Another feature of PDD is that it is, by construction,* fault tolerant: *the outage of an individual processor does not lead to substantial — let alone fatal — damage to the overall simulation. Given that modern parallel computers boast millions of processors, this is a definite plus (which stands in stark contrast to DDD algorithms).*

Last, it is up to the user to choose the artificial boundaries. These can be chosen in such a way as to lessen effects of kinks, sharp corners in the numerics. For instance, by better interplaying with the solutions being provided by the stochastic representations and Monte Carlo simulation.

2.1 The ingredients of PDD

PDD requires four ingredients: an interpolator, a subdomain solver, a stochastic representation of the PDE at hand, and an efficient numerical method for exploiting the latter with the Monte Carlo method. Let us briefly comment on them.

▷ The *interpolator* scheme joins the (approximate) solutions at the interface nodes (computed with Monte Carlo) in order to yield an (approximate) Dirichlet BC on the whole interface. (We remark that, in time-dependent problems, the interface runs along the time component, as well as the spatial ones.) A flexible and accurate interpolation scheme well suited to PDD is the RBF interpolation — see [12] and references therein. We note that less stringent approximation schemes than interpolation, such as "denoising" splines or least squares, are also possible.

▷ The *subdomain solver* is any state-of-the-art numerical scheme to solve a PDE (or a BVP) on a finite domain, without parallelization. It takes the approximation to the solution on the boundary delivered by the interpolator as Dirichlet BCs and produces a solution everywhere within the subdomain. All the subdomain solutions are later "glued together" to yield a complete solution of the large-scale original problem. Common examples of solvers are finite differences, finite elements, and meshless methods.

▷ The *stochastic (or probabilistic) representation of the PDE*, like that in (2.1), is the pointwise characterization of the solution of a PDE (stationary or time dependent) as the expected value of the functional of a system of SDEs. The paradigm of stochastic representations is the well-known Feynman–Kac formula. More details will follow in Section 2.2.

▷ *Efficient stochastic numerical methods* to exploit the stochastic representation. With the Monte Carlo approach, the expected value which yields the solution at an interfacing node is approximated as the average over many independent samples and this has to be repeated for every interfacing node. Contrary to deterministic methods, in the stochastic setting the errors are two-fold, brought about by both discretization of time and by the substitution of the expected value by a sample mean. (One can say that the estimator of the expected value is biased due to the discretization of the time variable.) In order to attain a target accuracy a, both sources of errors must be balanced and this leads to lengthy simulations: the computational effort is $\mathcal{O}(a^{-4})$ if naive numerics are used (such as the Euler–Maruyama scheme plus boundary test for linear BVPs, see [44]). This cost can be substantially reduced (to around $\mathcal{O}(a^{-2})$ in some cases) with sophisticated, novel schemes. The new integrators for bounded stopped diffusions put forward by Gobet [35] and Gobet and Menozzi [39] have doubled the convergence rate of the weak error associated to the interaction with the boundary, see also [11]. For reflected diffusions, Lèpingle's method [35] or Milstein's bounded methods [53] are recommended. Two further promising directions are the incorporation of Giles' multilevel method [34, 44] and the use of pathwise control variates for variance reduction [12]. In the later sections, pointers will be given to the numerics of non-linear probabilistic representations. In any case, the numerics of SDEs (and generalizations thereof) are underdeveloped compared to those of deterministic equations. A positive aspect is that many new ideas being currently developed for SDEs in financial mathematics can be transplanted to PDD.

Ideally, a numerical method tailored to the probabilistic representation of the given PDE exists, resulting in huge gains in efficiency. An example is the Walk on Spheres algorithm for the stopped Brownian motion, which is the probabilistic representation of Laplace's equation with Dirichlet BCs. This and similar, specific numerical schemes should be used whenever possible (see [13] for references).

2.2 Probabilistic representation of linear BVPs

We proceed now to give the probabilistic representation of linear BVPs (a generalization of the well-known Feynman–Kac formula). Let us start by considering the general linear parabolic BVP of second order with mixed BCs:

$$
\begin{cases}
\frac{\partial u}{\partial t} = \mathcal{L}u + c(\mathbf{x},t)u + f(\mathbf{x},t) & \text{if } t > 0, \mathbf{x} \in \Omega, \\
u = p(\mathbf{x}) & \text{if } t = 0, \mathbf{x} \in \Omega, \\
u = g(\mathbf{x},t) & \text{if } t > 0, \mathbf{x} \in \partial\Omega_A, \\
\frac{\partial u}{\partial N} = \varphi(\mathbf{x},t)u + \psi(\mathbf{x},t) & \text{if } t > 0, \mathbf{x} \in \partial\Omega_R.
\end{cases}
\tag{2.3}
$$

The notation $\partial\Omega_A$ stands for a portion of the boundary where Dirichlet BCs are imposed, while on $\partial\Omega_R = \partial\Omega \backslash \partial\Omega_A$, BCs involving the normal derivative (Neumann or Robin) hold. In SDE literature, $\partial\Omega_A$ and $\partial\Omega_R$ are known as absorbing and reflecting boundaries, respectively. In (2.3), $\mathcal{L}$ is the second-order differential operator defined by

$$
\mathcal{L} := \frac{1}{2} \sum_{i,j=1}^{d} A_{ij}(\mathbf{x},t) \frac{\partial^2}{\partial x_i \partial x_j} + \sum_{i=1}^{d} b_i(\mathbf{x},t) \frac{\partial}{\partial x_i},
\tag{2.4}
$$

where the matrix A is positive definite and hence can be decomposed as $A := [A_{ij}] = \sigma\sigma^T$ (for instance, by Cholesky factorization). For future reference, we point out the importance of this operator $\mathcal{L}$ in obtaining the (main) driving SDE, one can compare the coefficients of $\mathcal{L}$ with $\mathbf{X}$ in (2.8) below. It is assumed that $\varphi \leqslant 0$ and that the remaining coefficients and boundary in (2.3) are smooth enough (in particular, the outward normal vector $\mathbf{N}$ to $\partial\Omega_R$ is well defined) so that a unique solution exists [21, 33].

In order to introduce the stochastic representation of (2.3), let $\mathbf{X}_t, 0 \leqslant t \leqslant T$ be a diffusion (see (2.8) below) starting at $\mathbf{X}_0 = \mathbf{x}_0 \in \Omega \subset \mathbb{R}^d, (d \geqslant 1)$, and consider the linear functional, in fact, a random variable,

$$
\phi = \mathbf{1}_{\{\tau \geqslant T\}} p(\mathbf{X}_T) \Phi(T) \mathbf{1}_{\{\tau < T\}} g(\mathbf{X}_\tau, T - \tau) \Phi(\tau)
$$
$$
+ \int_0^{\min(T,\tau)} f(\mathbf{X}_t, T - t) \Phi(t) dt + \int_0^{\min(T,\tau)} \psi(\mathbf{X}_t, T - t) \Phi(t) d\xi_t,
\tag{2.5}
$$

where

$$
\Phi(t) = \exp\left(\int_0^t c(\mathbf{X}_s, s) ds + \int_0^t \varphi(\mathbf{X}_s, s) d\xi_s \right).
\tag{2.6}
$$

In (2.5) and (2.6), $\mathbf{1}_{\{H\}}$ is the indicator function (1 if H is true and 0 otherwise); $\tau = \inf_t \{\mathbf{X}_t \in \partial\Omega_A\}$ is the "first exit (or first passage) time" from Ω; which occurs at the "first exit point" $\mathbf{X}_\tau \in \partial\Omega_A$; ξ_t is the "local time" (the amount of time the trajectory spends infinitesimally close to $\partial\Omega_R$); and all functions are assumed continuous and bounded. We want to calculate the expectation of ϕ in (2.5), which can also be expressed as

$$
\mathbb{E}[\phi|\mathbf{X}_0 = \mathbf{x}_0] = \mathbb{E}\left[q(\mathbf{X}_\tau) Y_\tau + Z_\tau \right], \text{ s. th. } q(\mathbf{X}_\tau) = \begin{cases} g(\mathbf{X}_\tau, T - \tau), & \text{if } \tau < T, \\ p(\mathbf{X}_T), & \text{if } \tau \geqslant T, \end{cases}
\tag{2.7}
$$

where the processes $(\mathbf{X}_t, Y_t, Z_t, \xi_t)$ are governed by a set of SDEs driven by a d-dimensional

Wiener process $\mathbf{W}_t$:

$$\begin{cases} d\mathbf{X}_t = \mathbf{b}(\mathbf{X}_t, T - t)dt + \sigma(\mathbf{X}_t, T - t)d\mathbf{W}_t - \mathbf{N}(\mathbf{X}_t)d\xi_t & \mathbf{X}_0 = \mathbf{x}_0, \\ dY_t = c(\mathbf{X}_t, T - t)Y_t dt + \varphi(\mathbf{X}_t, T - t)Y_t d\xi_t & Y_0 = 1, \\ dZ_t = f(\mathbf{X}_t, T - t)Y_t dt + \psi(\mathbf{X}_t, T - t)Y_t d\xi_t & Z_0 = 0, \\ d\xi_t = \mathbf{1}_{\{\mathbf{X}_t \in \partial\Omega_R\}} dt & \xi_0 = 0. \end{cases} \tag{2.8}$$

Then, formulas (2.5) and (2.7) are the solution of the parabolic linear BVP, i.e.

$$u(t, \mathbf{x}_0) = \mathbb{E}\big[\phi|\mathbf{X}_0 = \mathbf{x}_0\big] = \mathbb{E}\Big[q(\mathbf{X}_\tau)Y_\tau + Z_\tau\Big]. \tag{2.9}$$

The representation in terms of the SDE system (2.8), favoured by Milstein, is not the most usual one, but it can be programmed in a straightforward manner. If $c(\mathbf{x}) \leqslant 0$, elliptic BVPs can be formally derived from (2.3), the Feynman–Kac formulas are then known as Dynkin's formulas. By taking $T \to \infty$, removing p, and dropping the time dependence from the surviving coefficients, the SDEs in (2.8) are now autonomous as long as τ is finite. (This fails to happen, for instance, with a purely reflecting boundary ($\partial\Omega = \partial\Omega_R$), when the solution $u(\mathbf{x})$ is defined up to an arbitrary constant and requires one more compatibility condition [33].) In the case of purely Dirichlet BCs, $\xi_.$ and the last equation in (2.8) drop out. The representation of linear BVPs can be simplified in many cases, such as (2.1) and (2.2) for (1.1).

2.3 An illustration of the effect of improved numerics

Remarkable research into the numerical methods to solve representations of linear BVPs given above during the last 15 years has meant that they can be solved today at a fraction of the cost required when PDD was introduced. To highlight the effect of improved stochastic numerics on the PDD methodology, let us take an example from [12]. There the authors study the BVP

$$\nabla^2 u + \frac{\cos(x + y)}{1.1 + \sin(x + y)}\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) - \frac{x^2 + y^2}{1.1 + \sin(x + y)}u + f(x, y) = 0, \tag{2.10}$$

with $f(x, y)$ such that $u(x, y) = 2\cos\big(2(y - 2)x\big) + \sin\big(3(x - 2)y\big) + 3.1$ is the exact solution, as well as the Dirichlet BC. The BVP domain can be seen in Figure 2.

This problem was solved with the most current version of PDD (called IterPDD in [12]). IterPDD is a numerical suite which includes variance reduction techniques with iterative control variates based on nested, increasingly accurate global solutions.

The speedup of a method A over a method B to solve a BVP is defined as

$$S(A, B) = \frac{\text{Time taken by method B}}{\text{Time taken by method A}}. \tag{2.11}$$

Table 1 shows the speedup of IterPDD over the previous versions of PDD. Note that IterPDD retains all the advantages of any previous version of the PDD algorithm and hence it is not necessary to compare it against DDD solvers, since $S(IterPDD, DDD) = S(IterPDD, PDD) \times S(PDD, DDD)$. The theoretical speedup in Table 1 is the optimal

Table 1. *Acceleration of the most current version of PDD (IterPDD), with improved stochastic numerics, over the previous PDD algorithm. a_0 is the error tolerance on interfacing nodes. The BVP being solved is (2.10)*

a_0	Theoretical speedup	Observed speedup (approx.)
0.04	13.93	15
0.02	28.34	29
0.01	57.42	60
0.005	116.00	125
0.0025	233.84	250

speedup predicted by a sensitivity algorithm presented in [12]. Importantly, the latter is designed in such a way that it relies on fast warm up Monte Carlo sampling, and runs in parallel so as not to defeat the purpose of PDD. The agreement with the experimentally observed speedup is consistently conservative.

The acceleration (speedup) grows larger as the target nodal accuracy a_0 (the largest admissible error of the numerical solution on the interface nodes, obtained via the probabilistic representation) decreases, approximately in an inversely proportional way. In summary, nearly every previously reported result using PDD, which were already faster than deterministic solvers, could be additionally accelerated by one to two orders of magnitude thanks to improved numerics. Further acceleration improvements are expected to follow in combination with multilevel formulations [34].

2.3.1 *Some applications of PDD*

The initial undertaking of the PDD programme was due to J. A. Acebrón and his research group [1–5]. Some realistic large-scale simulations carried out in supercomputers at the Barcelona Supercomputer Center and Rome's CASPUR proved the superiority of PDD over ScaLAPACK in terms of both total time and observed scalability when solving a non-linear equations including Kolmogorov–Petrovsky–Piskunov (KPP) [4,7,8]. Independently, Gobet and Mairé proposed a PDD-like domain decomposition scheme for the Poisson equation in [38]. Bihlo and Haynes have applied PDD to the generation of meshes for finite elements, a PDE-based task very well suited to stochastic representations [14–16]. Moreover, in [5], the Vlasov–Poisson equations were tackled, which are of unquestionable interest in plasma physics. The PDD treatment of non-linear PDEs like this one is further discussed in Section 3.

3 Branching diffusions and the KPP equation

The PDEs covered in Section 2, are all linear, unfortunately though PDEs arising in applications are typically not linear and standard SDE arguments are insufficient to address the general settings. That being said, one can derive stochastic representations for more general PDEs using so-called FBSDEs, but FBSDEs are computationally expensive and difficult to handle (see Section 4). However, [41,43,56] and [42] have further developed

the "branching diffusions" methodology as an efficient method to solve wider classes of nonlinear PDEs than the originally proposed by McKean.

Branching diffusions allow one to tackle PDEs which are not linear without the FBSDE machinery. Although the classical results were somewhat restrictive on the form of the PDE (see (3.1) below), the recent developments allow one to address in an efficient way more general classes of PDEs. For example, when non-linearities appear in the solution, or even gradients of the solution (see (3.3) and Section 3.3 below). Throughout this section, we supplement the theory with examples of PDEs that are amenable to branching and whose representations open the door for PDD as a viable algorithm for a far larger class of problems than previously available.

In Section 2, we discussed stochastic representations for non-Dirichlet BCs, unfortunately such BCs have not been considered in this more general setting, but are future work for the authors. For the interested reader, we mention recent analysis of the Monte Carlo Branching methodology to elliptic PDEs [9].

3.1 Reaction–diffusion: KPP equation

The branching diffusion idea is based on the works [51,57,61], to solve the KPP equation (in one spatial dimension with Dirichlet BC),

$$\begin{cases} \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} - u(u-1) = 0, & x \in (x_1, x_2), \ x_1, x_2 \in \mathbb{R}, \ t > 0, \\ u(x,0) = \psi(x), & u(x_1,t) = g(x_1,t), \quad u(x_2,t) = g(x_2,t). \end{cases}$$

This was later generalized to allow for non-linear PDEs such as

$$\frac{\partial u}{\partial t} - \mathcal{L}u - c \left(\sum_{i=0}^{\infty} \alpha_i u^i - u \right) = 0, \tag{3.1}$$

where c is a positive constant. Solutions to such non-linear PDEs can be represented through so-called *branching diffusion processes*. There has also been work for non-integer powers of u, typically u^α for $\alpha \in [0,2]$, such process are referred to as *super diffusions*, although we will not discuss these here to focus on more recent work, see [26], [43], and references therein for further details. Again the SDE process is governed by $\mathcal{L}$, see (2.4).

Assumption 3.1 (Classical branching diffusions [51]) *In the classical setting, the stochastic representation of the above PDE relies on the following two conditions for the coefficients* α: $\alpha_i \geqslant 0$ *and* $\sum_{i=0}^{\infty} \alpha_i = 1$.

To explain the stochastic representation to (3.1), for point $(\mathbf{x},t)$, we need to consider a set of particles. We start a particle at $\mathbf{x}$ at time 0, this particle has a life time τ which is exponentially distributed with intensity c (sometimes known as the "branching rate"), we then simulate this particle until $\min(t, \tau, \tau_{\partial\Omega})$, where $\tau_{\partial\Omega}$ is the particle hitting the spatial boundary, if τ is this smallest time, then the particle "branches" into i particles with probability α_i. These particles all start at the same point in space, however, they are all equipped with the own independent exponential random variable (life time) and driving
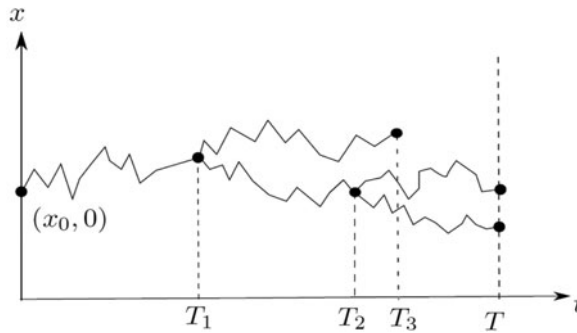
FIGURE 3. Representation of a branching diffusion, the first particle starts at point x_0 at time $t = 0$, then after time T_1 branches into two particles. The first of these later dies at T_3 with no descendants, the other particle branches again in two particles at time T_2. Both of these (third generation) particles hits the terminal time, T boundary.

Brownian motion. We then continue to simulate every particle until it has hit the space boundary or is alive at time t. We provide a detailed algorithm for branching diffusions (without space boundaries) in Section 3.1.1.

Consider the same initial and boundary value as above. The solution u can then be written as the expected value of the product of the surviving particles or particles that hit the space boundary (see [7]),

$$u(\mathbf{x}, t) = \mathbb{E}_{\mathbf{x},0} \left[\prod_{i=1}^{N_t} \left(\psi\left(X_t^{(i)}\right) \mathbf{1}_{\{t < \tau_{\partial\Omega}\}} + g\left(X_{\tau_{\partial\Omega}}^{(i)}, t - \tau_{\partial\Omega}\right) \mathbf{1}_{\{t \geqslant \tau_{\partial\Omega}\}} \right) \right], \tag{3.2}$$

where $X_s^{(i)}$ is the location of the ith particle surviving at time s and N_t is the number of surviving particles at time t (a particle that hits the boundary is still "alive" at time t).

Although this set up allows for more general PDEs, it is not difficult to see that the computational complexity is greater for these types of problems (though still smaller than that for FBSDEs). Moreover, because of the nature of the branching, we are mainly confined to cases which are not "overly" non-linear with time horizons that are not too long. Otherwise, we may end up with "explosions" whereby we are required to keep track of an extremely large number of particles, which will also tend to increase the number of branchings. We postpone discussion of this to Section 3.2.

To help give an idea of branching diffusions, we illustrate a sample path for a given PDE. Let us consider the PDE (one spatial dimension),

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} - \frac{1}{2}\left(1 + u^2\right) + u = 0, \qquad u(x, 0) = \psi(x), \quad x \in \mathbb{R}, \ t > 0.$$

From the work above, there are two possible outcomes at a branching time (compare with (3.1)), either the particle dies, or splits into two descendants, the probability of each of these events is $\frac{1}{2}$. A possible trajectory (to compute $u(x_0, T)$) is shown in Figure 3.

3.1.1 A PDD numerical example with the KPP equation

Since we already have enough machinery to consider some interesting PDEs, let us show an example which comes from the area of reaction diffusion equations, this class of equations are used in many areas such as physics and biology [31, 32, 48]. For illustration purposes, we consider the most well known of these, the KPP equation [47], linked to applications in plasma physics and ecology. The general form of the KPP is the following non-linear parabolic PDE:

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} + ru(1-u) = 0, \qquad (x,t) \in \mathbb{R} \times [0,\infty)$$

where D and r are constants. Following [7], taking $r = D = 1$ and the initial value as

$$u(x,0) = \psi(x) = 1 - \left(1 + \exp\left(x/\sqrt{6}\right)\right)^{-2},$$

we can write the true solution of this problem as

$$u(x,t) = 1 - \left(1 + \exp\left(\frac{x}{\sqrt{6}} - \frac{5t}{6}\right)\right)^{-2}.$$

The advantage of such an example is that we can also compare the error of each method. Following (3.2), the stochastic representation of the solution at (x,t) is simply,

$$u(x,t) = \mathbb{E}_{\mathbf{x},0}\left[\prod_{i=1}^{N_t} u\left(W_t^{(i)} + x, 0\right)\right] = \mathbb{E}_{\mathbf{x},0}\left[\prod_{i=1}^{N_t}\left(1 - \left(1 + \exp\left(\left(W_t^{(i)} + x\right)/\sqrt{6}\right)\right)^{-2}\right)\right],$$

where N_t is the number of particles at time t, and $W_t^{(i)}$ is the Brownian motion for particle i at time t. Since the operator $\mathcal{L}$ is $\partial^2/\partial x^2$, the process X is Brownian motion started at point x (compare with (2.3) and (2.8)).

3.1.2 Algorithm for the problem

We solve this problem[2] over the domain $x \in [-2000, 2000]$ for $t \in [0, 1]$. For the PDD algorithm, this corresponds to choosing points in space and solving them over a sequence of times to construct the artificial space time boundary, we create $p + 1$ boundaries to obtain p subdomains. Hence, with five processors we split the domain in four subdomains $[-2000, -1000]$, $[-1000, 0]$, $[0, 1000]$, and $[1000, 2000]$. This may not be the most optimal approach, but our goal is to show how even a simple PDD approach can dramatically improve the computational time required. We further calculate the solution at 11 equally spaced time points (including $t = 0$), which we denote by the set of T_i, satisfying $0 = T_0 < T_1 < \cdots < T = 1$. Denote by Γ the set of nodes at which we approximate

---

[2] MATLAB was used for the implementation. The simulations ran on a Dell PowerEdge R430 with four intel xeon E5-2680 processors. All polynomial interpolation were carried out using MATLAB's "polyfit" and the PDE is solved using MATLAB's "pdepe" function. To keep this consistent with the error we expect from Monte Carlo we have set the relative and absolute error in the solver as 10^{-3}.

the boundaries (true and artificial), hence for D domain points, with Θ time points, Γ is an D-by-Θ matrix. We denote by Γ_k for $k \in \{1, \ldots, D\}$ the column vectors of Γ, i.e. the approximation of the boundary at the kth domain point through time.

Remark 3.2 (Pruning — controlling the growth of the branching) *A useful technique to control the branching is the "pruning", which is a method whereby we truncate the number of branches and comes from the observation that the pruned branches contribute very little to the expectation and are extremely computationally expensive. We do not discuss this further here, but direct the interested reader to [6].*

It is clear from the PDE that all branching types will be two (the non-linear part is a squared — compare with (3.1)). Moreover, the driving process for the branching diffusions is a Brownian motion, this allows for us to take large step sizes since the simulation of Brownian motion is unbiased. Algorithm 3.1.2 provides a summary of the method.

Algorithm 3.1 Outline to solve the KPP equation using PDD over p processors

Split Ω into $(p-1)$ subdomains Ω_i and select set of nodes Γ on the artificial boundary.
Take a series points $T_0 < T_1 < \cdots < T$ and set the "pruning" level P_r.
for Each $\Gamma_k \in \Gamma$ **do**
    for N Monte Carlo simulations **do**
        while $t < T$ **do**
            Set N_p as the number of particles and t as the current time.
            Find T_i, such that $T_{i-1} \leqslant t < T_i$ and simulate $t_c \sim \text{Exp}(c \times N_p)$.
            Simulate each particle over $\min(t + t_c, T_i)$
            if we reach time $t + t_c$ **then**
                Pick a particle (with equal probability) to branch into two
            else
                Evaluate $\psi^{(s)}(T_i) = \prod_{j=1}^{N_p} \psi(X_{T_i}^{(j)})$.
            end if
            if $N_p > P_r$ **then**
                Restart this iteration of the for loop
            end if
         end while
        end for
    end for
for Each T_i **do**
    Calculate $\Gamma_k(T_i) = \frac{1}{N} \sum_{s=1}^{N} \psi^{(s)}(T_i)$
end for
Interpolate over each Γ_k to create artificial boundaries and solve the PDE on each Ω_i.

3.1.3 *Error analysis*

Remark 3.3 (Straightforward implementation for Monte Carlo) *For this example, we used only a standard Monte Carlo implementation. Therefore, the results presented should be seen*
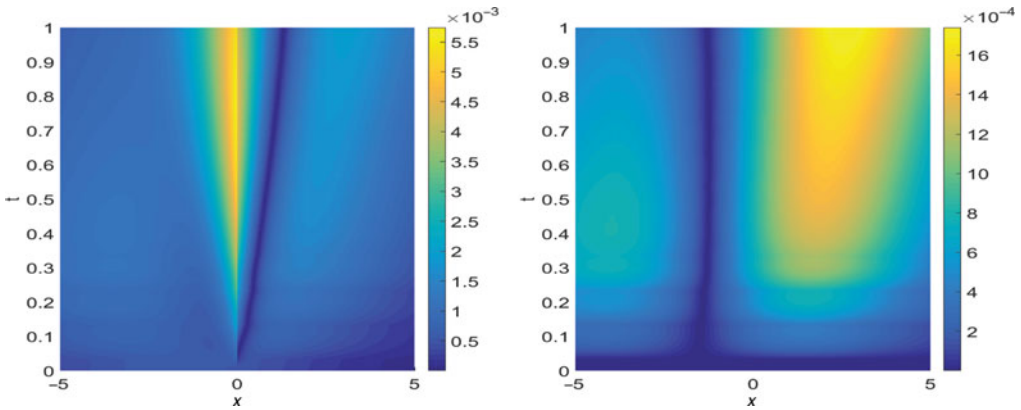
as the lower bound for what is achievable. It possible to improve the speed and accuracy of the algorithm by using methods touched on in Section 2.

Figure 4 compares the absolute error of Monte Carlo versus using only the PDE solver over the domain $[-5, 5]$. We observe the maximum error of the pure PDE solver is approximately $1/3$ that of PDD. However, the largest error for PDD is concentrated in one region to the left-hand side of the boundary and outside this small region the two errors are comparable.

3.1.4 *Scalability and running times of the algorithm*

We conclude by considering how the time taken to solve the problem changes as more cores are used. We assume that we have access to the $p + 1$ processors for p subdomains. Since PDD consists of two main steps (Monte Carlo then PDE solver), we want to consider how these change as we increase the number cores. These results are presented in Figure 5. Note, to check the scaling, we use the number of domains rather than the number of processors and PDD is not used when the number of domains is one.

Figure 5, clearly shows the scaling capabilities of PDD. As one increases the number of processors the Monte Carlo time stays close to constant (as expected) and the PDE solver time drops dramatically as the number of processors increases. In fact, as is presented on the right graph, this decrease is constant with the number of subdomains with no apparent sign of levelling off; this follows from the no interprocessor communication.

3.2 Marked branching diffusions

The idea of *marked branching diffusions* was developed by [41] as a solution to pricing credit valuation adjustments (see Section 3.2.2), but since then has been generalized by [43] and [42]; similar ideas appeared in [56]. Following [43], we consider a terminal value
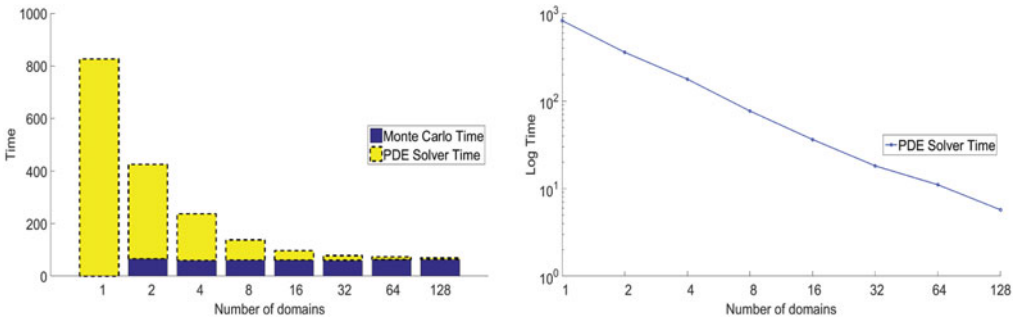
FIGURE 5. On the left, wall-clock running times (in seconds) for the PDE solver and PDD solver (Monte Carlo simulation and the PDE solver) as the number of subdomains (processors) increases. See Remark 3.3. On the right, the running times in log-scale showing perfect scalability.

PDE of the form (recall the time-inversion argument in Section 2.2):

$$\frac{\partial u}{\partial t} + \mathcal{L}u + c\left(\sum_{i=0}^{L} \alpha_i(\mathbf{x}, t)u^i - u\right) = 0, \qquad u(\mathbf{x}, T) = \psi(\mathbf{x}). \tag{3.3}$$

The notation follows that in Section 3 with L a positive integer.

We now discuss sufficient conditions for our stochastic representation to be the unique viscosity solution of this PDE. The technique in [43] is to use the fact that a large class of PDEs can be represented by FBSDEs, see [54]. The idea is to derive sufficient conditions for the FBSDE to be the unique bounded viscosity solution to a PDE of the form (3.3) and use this to obtain the results on branching diffusions; here, FBSDEs are used only as a theoretical method to enable the branching argument. The key assumption is the following [43, Assumption 2.2] denote the two functions for $s \in [0, \infty)$:

$$l_0(s) := \sum_{k \geqslant 0} \|\alpha_k\|_\infty s^k \quad \text{and} \quad l(s) := c\left(\frac{l_0\big(s\|\psi\|_\infty\big)}{\|\psi\|_\infty} - s\right),$$

where $\|\cdot\|_\infty$ is the usual sup-norm and the constants/functions follow from (3.3), then,

Assumption 3.4 (Assumptions for Marked branching)
  (1) *The power series l_0 has a radius of convergence $0 < R \leqslant \infty$. Moreover, the function l satisfies one of the following conditions:*
    (a) *$l(1) \leqslant 0$,*
    (b) *$l(1) > 0$ and for some $\hat{s} > 1$, $l(s) > 0$, $\forall s \in [1, \hat{s})$ and $l(\hat{s}) = 0$,*
    (c) *$l(s) > 0 \; \forall s \in [1, \infty)$ and $\int_1^{\bar{s}} \big(l(s)\big)^{-1} ds = T$, for some constant $\bar{s} \in (1, R/\|\psi\|_\infty)$.*
  (2) *The terminal function satisfies $\|\psi\|_\infty < R$.*

Under the above assumption, the stochastic equations associated to (3.3) have a unique bounded solution and yield the unique viscosity solution for PDE (3.3) (see Proposition 2.3 and Theorem 2.13 of [43]). Thus, the above assumption provides sufficient conditions for the branching diffusions to be used.

We are now most of the way to stating the marked branching diffusion representation. Of course, α needs not be a probability distribution anymore, so let us introduce a probability distribution q, where q is chosen so $q_i > 0$ if $\alpha_i \neq 0$. It will beneficial for us to introduce notation to keep track of the particles. Let T_n the nth branching time of the system where at time T_n one of the particles branches into k particles, with probability q_k. We denote by I_n the number of descendants created at time T_n, hence $I_n \in \{0, \ldots, L\}$. Denote by $M_{T-t} := \sup\{n : t + T_n \leqslant T\}$ the number of branches that occurred between time t and T. Further denote by $\mathcal{K}_t$ the index of all particles alive at time t, therefore $\mathcal{K}_{T-t}$ corresponds to the index of all particles alive at time $T - t$. Finally, denote by K_n the index of the particle that has branched at time T_n (hence, $K_n \in \mathcal{K}_{T_n}$). The representation for the solution of the PDE (3.3) at $(\mathbf{x}, t)$ is given as

$$u(\mathbf{x}, t) = \mathbb{E}_{\mathbf{x}, t} \left[\prod_{k \in \mathcal{K}_{T-t}} \psi(\mathbf{X}_T^{(i)}) \prod_{n=1}^{M_{T-t}} \left(\frac{\alpha_{I_n}(\mathbf{X}_{t+T_n}^{(K_n)}, t + T_n)}{q_{I_n}} \right) \right].$$

Remark 3.5 (Variance of the estimator) *It is also of note that the specific choice of probability distribution q (subject to the conditions above) does not change the representation; however, from the view point of variance reduction an optimal choice exists, see [41].*

Although the set up here has been carried out using exponential random variables as the lifetime of particles, more recent work suggest that other distributions yield smaller variances, see [25, 42, 60] and [19].

3.2.1 *More general PDEs through approximation*

The stochastic representations shown in this section can be used to deal with many different types of PDEs. Namely, *these representations allow us to consider PDEs whose source term f can be well approximated by polynomials.* An example of this is considered in Section 3.2.2.

Although we mention credit valuation adjustments, many other financial contracts rely on maximums between two entries (so called obstacle PDEs), these appear in American options, for example. Computing the solution to certain obstacle PDEs in stochastics is deeply related to optimal stopping problems. We write such PDEs in their variational formulation: for a exercise payoff φ, the price function u solves

$$\max \left(\frac{\partial u}{\partial t} + \mathcal{L}u, \varphi(\mathbf{x}) - u \right) = 0, \qquad u(\mathbf{x}, T) = \varphi(\mathbf{x}), \qquad (\mathbf{x}, t) \in \mathbb{R}^d \times [0, T].$$

In [10], the authors showed how this PDE can be converted into a semi-linear PDE

$$\frac{\partial u}{\partial t} + \mathcal{L}u = \mathbf{1}_{\{\varphi(\mathbf{x}) \geqslant u\}} \mathcal{L}\varphi, \qquad u(\mathbf{x}, T) = \varphi(\mathbf{x}), \qquad (\mathbf{x}, t) \in \mathbb{R}^d \times [0, T].$$

A stochastic representation for these equations is well-understood and the methodology we have presented so far can be applied to them, see [41, Section 2.3]. Moreover, [17] use the polynomial representation of trigonometric functions to calculate the non-linear Poisson–Boltzmann equation.

3.2.2 *Numerical example with option pricing: Credit valuation adjustment (CVA)*

The next example we consider is a different problem than the KPP above, but highlights the marked branching diffusion and its application scope very well. This example concerns option (derivative) pricing with credit valuation adjustments. Since the financial crisis in 2008, risk, especially default risk (the risk that a counter party fails to meet future obligations) has been at the forefront of many policy decisions and regulation changes for financial firms. Credit valuation adjustments play a crucial role here by adjusting the price of a derivative (a financial contract) with the knowledge that the counter party may default. [40] give a derivation of the PDEs arising in this problem. The specific form of the PDE depends on the way in which one chooses to model the mark-to-market value (the fair value) of the derivative at the time of default, but we obtain PDEs of the form

$$\frac{\partial u}{\partial t} + \mathcal{L}u + r_0 u + r_1 u^+ = 0, \qquad u(x, T) = \psi(x), \qquad x \in \mathbb{R},$$

where the r_i are functions of x and t and we denote by $y^+ := \max(0, y)$.

Although at first glance such a PDE seems beyond the scope of the stochastic representations in Section 3.2, [41] provided a simple yet powerful trick to deal with such PDEs (see also [56]). For simplicity, we will consider only the one-dimensional case. The first trick, is to rescale the problem, by assuming the payoff is bounded. Then we may consider the following function $v := u/\|\psi\|_\infty$, such a function satisfies the PDE

$$\frac{\partial v}{\partial t} + \mathcal{L}v + r_0 v + r_1 v^+ = 0 \tag{3.4}$$

with terminal value bounded above by 1. Let us consider instead a PDE of the form

$$\frac{\partial v}{\partial t} + \mathcal{L}v + c\left(F(v) - v\right) = 0, \qquad v(x, T) = \psi/\|\psi\|_\infty$$

where F is a polynomial, hence this equation is of the type considered previously. The great insight in [41] was to use a polynomial to approximate the semi-linear component v^+. Therefore, we have transformed the PDE from one outside the capabilities of marked branching diffusions, into one which can be. [41] uses a polynomial of order 4 to do this, which provides a good approximation as can be seen in Figure 6.

Thus, the PDE we want to calculate is

$$\frac{\partial v}{\partial t} + \mathcal{L}v + c\left(0.0586 + 0.5v + 0.8199v^2 - 0.4095v^4 - v\right) = 0, \qquad v(x, T) = \psi/\|\psi\|_\infty.$$

Since we have constant coefficients and a bounded terminal condition, this PDE easily satisfies Assumption 3.4, hence we can use the stochastic representation. Remark the negative coefficient associated to the fourth-power term; the classical Assumption 3.1 does not hold here and hence the motivation behind marked branching diffusions.

Of course, there does not seem much scope for PDD here since the domain is rather small. However, in higher dimensional problems (basket options), the PDD method can very easily provide the computational advantages highlighted in earlier sections.
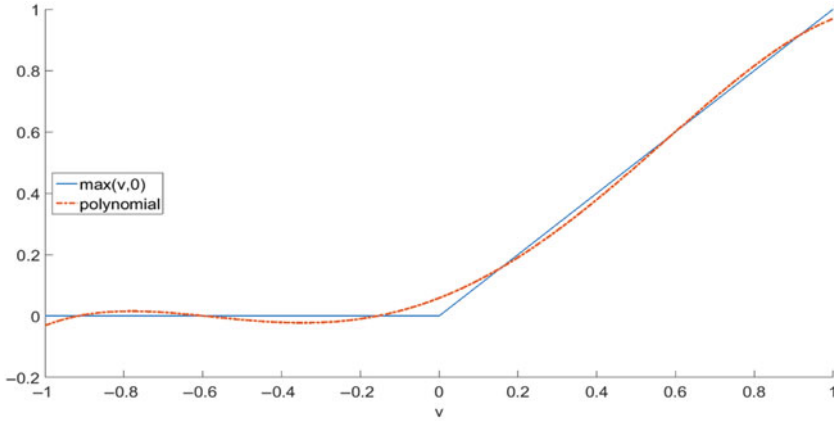
FIGURE 6. Approximation of $\max(v, 0)$ over $v \in [-1, 1]$ by a fourth-order polynomial.

3.3 Further generalizations: Aged marked branching diffusions

In [42], the authors generalize the representation to a wider class of semi-linear PDEs through so called *age-marked branching diffusions*, still building on the paradigm of semi-linear PDEs with polynomial source terms f. Consider for some integer $m \geqslant 0$ a set $L \subset \mathbb{N}^{m+1}$ and a sequence of functions $(c_l)_{l \in L}$ and $(h_i)_{i=1,\dots,m}$, where $c_l : [0, T] \times \mathbb{R}^d \to \mathbb{R}$ and $h_i : [0, T] \times \mathbb{R}^d \to \mathbb{R}$. Let the source term function $f : [0, T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ be of the form

$$f(t, \mathbf{x}, y, z) = \sum_{l=(l_0,\dots,l_m) \in L} c_l(t, \mathbf{x}) y^{l_0} \prod_{i=1}^{m} (h_i(t, \mathbf{x}) \cdot z)^{l_i}.$$

With such a function in mind (under some regularity assumptions), [42] provides a stochastic representation for PDEs of the form

$$\frac{\partial u}{\partial t} + \mathcal{L}u + f(\cdot, u, \nabla u) = 0, \quad \text{with} \quad u(\cdot, T) = \psi(\cdot), \quad t \in [0, T], \ \mathbf{x} \in \mathbb{R}^d,$$

where $\psi : \mathbb{R}^d \to \mathbb{R}$ is a bounded Lipschitz function. The assumptions required for a stochastic representation are more technical than the previous case considered. We thus omit them here and point the reader to [42] for the details. Under this set up, we can handle systems of PDE and also popular PDEs such as Burger's equation. In fact, using the polynomial approximation trick discussed in Section 3.2.1, we can handle PDEs containing $f(\cdot, u, \nabla u) = u|\nabla u|^2$ terms, which appear in many applications, for instance, to construct harmonic homotopic maps [59]; in the theory of ferromagnetic materials through the Landau–Lifschitz–Gilbert equation or models of magnetostriction.

Remark 3.6 (Fully non-linear case) *Recently, [60] constructed an algorithm from techniques developed in [42] to approximate a fully non-linear PDEs. This work is still experimental (at the present time) and thus we only reference it here for interested readers.*

3.3.1 *Some other PDE examples in data science solvable by branching diffusions*

We focus on branching diffusions, since they are amenable to PDD. In mathematical biology, one often wants to consider the affect on a population of a species given the presence of another species and the interaction between them. Such models have been considered in works such as [24], see [37] as well. In the paper by Escher and Matioc [29], they present a model to describe the growth of tumours. The model is a non-linear two-dimensional system with two decoupled Dirichlet problems.

As highlighted in Section 3.2.1, the methodology described in this work can also be used to deal with PDE obstacle problems. Apart from the well-known problem of pricing American type options, we point out to the so-called *Travel agency problem* [37, Section 5.2] where a travel agency needs to decide when to offer travels depending on currency and weather forecast.

If one returns to finance, the approximation method highlighted in Section 3.2.1 combined with the age-marked branching diffusions can be used to tackle the problems in actuarial science, namely the "Reinsurance and Variable Annuity" problem (see [40]) or the pricing of contracts with different borrowing/lending rates [22, Section 3.3].

4 An invitation to more challenging problems

In this Section, we shortly review probabilistic representations of non-linear problems based on FBSDEs for which branching arises as a particular case. FBSDEs provide stochastic representations for a wide class of PDEs, wider than branching techniques allow, including systems of (fully non-linear, parabolic, or elliptic) PDEs (or BVPs) combining any type of mixed BC.

Consider the system of $K \geqslant 1$ quasi-linear coupled PDEs in $\mathbb{R}^d \times [0, T]$, $d \geqslant 1$:

$$
\begin{cases}
\frac{\partial u^{(1)}}{\partial t}(\mathbf{x}, t) + \mathcal{L}u^{(1)}(\mathbf{x}, t) + f^{(1)}(t, \mathbf{x}, u^{(1)}, \ldots, u^{(K)}, \nabla u^{(1)}, \ldots, \nabla u^{(K)}) = 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad u^{(1)}(\mathbf{x}, T) = \psi^{(1)}(\mathbf{x}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
\frac{\partial u^{(K)}}{\partial t}(\mathbf{x}, t) + \mathcal{L}u^{(K)}(\mathbf{x}, t) + f^{(K)}(t, \mathbf{x}, u^{(1)}, \ldots, u^{(K)}, \nabla u^{(1)}, \ldots, \nabla u^{(K)}) = 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad u^{(K)}(\mathbf{x}, T) = \psi^{(K)}(\mathbf{x})
\end{cases}
\tag{4.1}
$$

The associated SDE to $\mathcal{L}$, $\mathbf{X}_t$, is as in equation (2.8) (with the same requirements on the coefficients as there). Under the proper conditions on the coefficients in (4.1) and adequate smoothness of the $u^{(1)}(\mathbf{x}, t), \ldots, u^{(K)}(\mathbf{x}, t)$, the pointwise solution of the system at $(\mathbf{x}_0, 0)$ is given by $(1 \leqslant k \leqslant K)$

$$
\begin{aligned}
u^{(k)}(\mathbf{x}_0, 0) = \mathbb{E}\Big[&\psi^{(k)}(\mathbf{X}_T) \\
&+ \int_0^T f^{(k)}\big(s, \mathbf{X}_s, Y_s^{(1)}, \ldots, Y_s^{(K)}, \mathbf{Z}_s^{(1)}\sigma^{-1}(\mathbf{X}_s, s), \ldots, \mathbf{Z}_s^{(K)}\sigma^{-1}(\mathbf{X}_s, s)\big)\, ds \Big],
\end{aligned}
\tag{4.2}
$$

where the processes $Y_t^{(1)}, \ldots, Y_t^{(K)}, \mathbf{Z}_t^{(1)}, \ldots, \mathbf{Z}_t^{(K)}$ are the solution to the system of FBSDEs

(see [36, chapter VII])

$$\begin{cases} Y_t^{(1)} = \psi^{(1)}(\mathbf{X}_T) - \int_t^T \mathbf{Z}_s^{(1)} \cdot d\mathbf{W}_s \\ \qquad + \int_t^T f^{(1)}\big(s, \mathbf{X}_s, Y_s^{(1)}, \dots, Y_s^{(K)}, \mathbf{Z}_s^{(1)}\sigma^{-1}(\mathbf{X}_s, s), \dots, \mathbf{Z}_s^{(K)}\sigma^{-1}(\mathbf{X}_s, s)\big)\,ds, \\ \quad\vdots \\ Y_t^{(K)} = \psi^{(K)}(\mathbf{X}_T) - \int_t^T \mathbf{Z}_s^{(K)} \cdot d\mathbf{W}_s \\ \qquad + \int_t^T f^{(K)}\big(s, \mathbf{X}_s, Y_s^{(1)}, \dots, Y_s^{(K)}, \mathbf{Z}_s^{(1)}\sigma^{-1}(\mathbf{X}_s, s), \dots, \mathbf{Z}_s^{(K)}\sigma^{-1}(\mathbf{X}_s, s)\big)\,ds. \end{cases} \tag{4.3}$$

One can show via Itô's formula that $Y_t^{(k)} = u^{(k)}(\mathbf{X}_t, t)$ and $\mathbf{Z}_t^{(k)} = \nabla u^{(k)}(\mathbf{X}_t, t)\sigma(\mathbf{X}_t, t)$, $(1 \leqslant k \leqslant K)$ — this lifts the apparent inconsistency of having more processes than equations in (4.3). As with linear parabolic equations, the terminal condition can be transformed into an initial one by reversing time, i.e. by letting $\hat{t} \to T - t$ and $\partial/\partial\hat{t} \to -\partial/\partial t$. System (4.1) accommodates many important applications in biochemistry, stochastic control, finance, and physics [36, chapter VII]. Multiple extensions of the above system are possible, including different generators for each of the equations in the system (i.e. different $\mathcal{L}^{(1)}, \dots, \mathcal{L}^{(K)}$), as well as BCs of various types. The complexity of the probabilistic representation grows accordingly.

In order to simplify the discussion and focus on difficulties, let $K = 1$ in the remainder of this section (thus $Y_t = Y_t^{(1)}$, $\psi = \psi^{(1)}$, $f = f^{(1)}$), and $f = f(t, \mathbf{x}, u)$. It can be shown

$$Y_t = u(\mathbf{X}_t, t) = \mathbb{E}\left[\psi(\mathbf{X}_T) + \int_t^T f(s, \mathbf{X}_s, Y_s)\,ds \,\big|\, \mathbf{X}_t\right]. \tag{4.4}$$

The manipulation of the conditional expectation (4.4) requires the introduction of filtered probability spaces, which lies beyond the scope of this paper. The numerical treatment is equally delicate. After a canonical time discretization over a uniform time partition with step $h = T/N$, we have

$$Y_{t_N} = \psi(\mathbf{X}_T), \qquad Y_{t_i} = \mathbb{E}\left[Y_{t_{i+1}} + hf(t_i, \mathbf{X}_{t_i}, Y_{t_{i+1}}) \,\big|\, \mathbf{X}_{t_i}\right] \tag{4.5}$$
$$\Leftrightarrow u(\mathbf{X}_{t_i}, t_i) = \mathbb{E}\left[Y_{t_{i+1}} + hf\big(t_i, \mathbf{X}_{t_i}, u(\mathbf{X}_{t_{i+1}}, t_{i+1})\big) \,\big|\, \mathbf{X}_{t_i}\right].$$

Here, we emphasize that to compute $u(\mathbf{x}, 0) = Y_0$ one needs, at each time step t_i, to compute an approximation of $u(\mathbf{x}, t_i)$ for x over the domain of the PDE. Comparing with the result when branching is possible, see equation (3.2), the gain of branching over the general FBSDE machinery is clear. Nonetheless, branching techniques have their limits and the general case must, in principle, be tackled through FBSDEs and its variants. How to carry out a PDD methodology within the framework of FBSDEs that still retains PDD's characteristic scalability is an open question.

In general, the timestep-per-timestep backward iteration and the computation and/or approximation of the conditional expectation in (4.5) can be done in many ways, for instance, by resorting to Bellman's dynamic programming principle, whereby the solution is projected on a functional basis by regression (see [36, chapter VIII]). Overviews on numerics of FBSDE can be found in [18, 22] and PDE inspired problems dealt by FBSDEs can be found in [32, 48, 49].

A brief remark on FBSDEs and stochastic representations

We close this section with comments on various probabilistic representations not mentioned elsewhere in this paper.

There is theoretical work relating FBSDEs to the Navier–Stokes equations [23]. While, in general, transport equations do not have a stochastic representation, there are some important exceptions, of which we cite two: the one-dimensional telegraph equation [3]; and the Vlasov–Poisson system of equations. For the latter, it was proposed in [52] a probabilistic representation in the Fourier space.

In [17], the authors deal with representations (and Monte Carlo simulation) for nonlinear divergence-form elliptic Poisson–Boltzmann PDE over the whole $\mathbb{R}^3$. Quasi-linear parabolic PDEs (multidimensional and systems) admit a stochastic representation in terms of FBSDEs [54, 55] and many extensions exist ranging from representations for obstacle PDE problems [27] to representations of fully non-linear elliptic and parabolic PDEs [20]. See as well [32, 48] for stochastic representations for reaction–diffusion PDEs, PDEs with quadratic gradients terms and of Burger's type nonlinearities. Complex-valued PDEs and the FBSDE connection are handled in [62].

Systems of fully non-linear systems of second-order PDEs (i.e. including possible nonlinearities on the highest derivatives i.e. f can depend on "Δu" terms) have recently found a representation in terms so-called 2BSDEs [20]; numerical schemes have already been proposed [30]. Last, much of the theory (on stochastic representations) can be extended to frameworks where the coefficients b and σ depend on u and ∇u (see [50]).

References

[1] Acebrón, J. A., Busico, M. P., Lanucara, P. & Spigler, R. (2005a) Domain decomposition solution of elliptic boundary-value problems via Monte Carlo & quasi-Monte Carlo methods. *SIAM J. Sci. Comput.* **27**(2), 440–457.

[2] Acebrón, J. A., Busico, M. P., Lanucara, P. & Spigler, R. (2005b) Probabilistically induced domain decomposition methods for elliptic boundary-value problems. *J. Comput. Phys.* **210**(2), 421–438.

[3] Acebrón, J. A. & Ribeiro, M. A. (2016) A Monte Carlo method for solving the one-dimensional Telegraph equations with boundary conditions. *J. Comput. Phys.* **305**, 29–43.

[4] Acebrón, J. A. & Rodríguez-Rozas, Á. (2011) A new parallel solver suited for arbitrary semilinear parabolic partial differential equations based on generalized random trees. *J. Comput. Phys.* **230**(21), 7891–7909.

[5] Acebrón, J. A. & Rodríguez-Rozas, Á. (2013) Highly efficient numerical algorithm based on random trees for accelerating parallel Vlasov–Poisson simulations. *J. Comput. Phys.* **250**, 224–245.

[6] Acebrón, J. A., Rodríguez-Rozas, Á. & Spigler, R. (2009) Domain decomposition solution of nonlinear two-dimensional parabolic problems by random trees. *J. Comput. Phys.* **228**(15), 5574–5591.

[7] Acebrón, J. A., Rodríguez-Rozas, Á. & Spigler, R. (2010a) Efficient parallel solution of nonlinear parabolic partial differential equations by a probabilistic domain decomposition. *J. Sci. Comput.* **43**(2), 135–157.

[8] Acebrón, J. A., Rodríguez-Rozas, Á. & Spigler, R. (2010b) A fully scalable algorithm suited for petascale computing and beyond. *Comput. Sci. Res. Dev.* **25**(1–2), 115–121.

[9] AGARWAL, A. & CLAISSE, J. (2017) Branching diffusion representation of quasi-linear elliptic pdes and estimation using monte carlo method. *preprint, arXiv:1704.00328.*

[10] BENTH, F. E., KARLSEN, K. H. & REIKVAM, K. (2003) A semilinear Black and Scholes partial differential equation for valuing American options. *Finance Stoch.* **7**(3), 277–298.

[11] BERNAL, F. & ACEBRÓN, J. A. (2016a) A comparison of higher-order weak numerical schemes for stopped stochastic differential equations. *Comm. Comput. Phys.* **20**(3), 703–732.

[12] BERNAL, F. & ACEBRÓN, J. A. (2016b) A multigrid-like algorithm for probabilistic domain decomposition. *Comput. Math.Appl.* **72**(7), 1790–1810.

[13] BERNAL, F., ACEBRÓN, J. A. & ANJAM, I. (2014) A stochastic algorithm based on fast marching for automatic capacitance extraction in non-Manhattan geometries. *SIAM J. Imag. Sci.* **7**(4), 2657–2674.

[14] BIHLO, A. & HAYNES, R. D. (2014) Parallel stochastic methods for PDE based grid generation. *Comput. Math. Appl.* **68**(8), 804–820.

[15] BIHLO, A. & HAYNES, R. D. (2016) A stochastic domain decomposition method for time dependent mesh generation. In: *Domain Decomposition Methods in Science and Engineering XXII.* T. Dickopf, M. J. Gander, L. Halpern, R. Krause, & L. F. Pavarino (editors), Springer, Vol. 104, pp. 107–115.

[16] BIHLO, A., HAYNES, R. D. & WALSH, E. (2015) Stochastic domain decomposition for time dependent adaptive mesh generation. *J. Math. Stud.* **48**(2), 106–124.

[17] BOSSY, M., CHAMPAGNAT, N., LEMAN, H., MAIRE, S., VIOLEAU, L. & YVINEC, M. (2015) Monte Carlo methods for linear and non-linear Poisson-Boltzmann equation. *ESAIM: Proc. Surv.* **48**, 420–446.

[18] BOUCHARD, B., ELIE, R. & TOUZI, N. (2009) Discrete-time approximation of BSDEs and probabilistic schemes for fully nonlinear PDEs. In: *Advanced Financial Modelling,* H. Albrecher, W. J. Runggaldier & W. Schachermayer (editors), Radon Ser. Comput. Appl. Math., Vol. 8, Walter de Gruyter, Berlin, pp. 91–124.

[19] BOUCHARD, B., TAN, X. & ZOU, Y. (2016) Numerical approximation of BSDEs using local polynomial drivers and branching processes. *arXiv:1612.06790.*

[20] CHERIDITO, P., SONER, H. M., TOUZI, N. & VICTOIR, N. (2007) Second-order backward stochastic differential equations and fully nonlinear parabolic PDEs. *Comm. Pure Appl. Math.* **60**(7), 1081–1110.

[21] COSTANTINI, C., PACCHIAROTTI, B. & SARTORETTO, F. (1998) Numerical approximation for functionals of reflecting diffusion processes. *SIAM J. Appl. Math.* **58**(1), 73–102.

[22] CRISAN, D. & MANOLARAKIS, K. (2010) Probabilistic methods for semilinear partial differential equations. Applications to finance. *Math. Modelling Numer. Anal.* **44**(5), 1107.

[23] CRUZEIRO, A. B. & SHAMAROVA, E. (2009) Navier–Stokes equations and forward–backward SDEs on the group of diffeomorphisms of a torus. *Stoch. Process. Appl.* **119**(12), 4034–4060.

[24] DANCER, E. N. & DU, Y. H. (1994) Competing species equations with diffusion, large interactions, and jumping nonlinearities. *J. Differ. Equ.* **114**(2), 434–475.

[25] DOUMBIA, M., OUDJANE, N., & WARIN, X. (2017) Unbiased Monte Carlo estimate of stochastic differential equations expectations. *ESAIM: Probability and Statistics* **21**, 56–87.

[26] DYNKIN, E. B. (2004) *Superdiffusions and Positive Solutions of Nonlinear Partial Differential Equations,* University Lecture Series, Vol. 34, American Mathematical Society, Providence, RI. Appendix A by J.-F. Le Gall and Appendix B by I. E. Verbitsky.

[27] EL KAROUI, N., KAPOUDJIAN, C., PARDOUX, E., PENG, S. & QUENEZ, M. C. (1997) Reflected solutions of backward SDEs, and related obstacle problems for PDEs. *Ann. Probab.* **25**(2), 702–737.

[28] EL KAROUI, N., PENG, S. & QUENEZ, M. C. (1997) Backward stochastic differential equations in finance. *Math. Finance* **7**(1), 1–71.

[29] ESCHER, J. & MATIOC, A.-V. (2010) Radially symmetric growth of nonnecrotic tumors. *Nonlinear Differ. Equ. Appl. NoDEA* **17**(1), 1–20.

[30] FAHIM, A., TOUZI, N. & WARIN, X. (2011) A probabilistic numerical method for fully nonlinear parabolic PDEs. *Ann. Appl. Probab.* **21**(4), 1322–1364.

[31] FISHER, R. A. (1937) The wave of advance of advantageous genes. *Ann. eugenics* **7**(4), 355–369.

[32] FREI, C. & DOS REIS, G. (2013) Quadratic FBSDE with generalized Burgers' type nonlinearities, perturbations and large deviations. *Stoch. Dynam.* **13**(2), 1250015.

[33] FREIDLIN, M. (1985) *Functional integration and Partial Differential Equations*, Annals of Mathematics Studies, Vol. 109, Princeton University Press, Princeton, NJ.

[34] GILES, M. B. & BERNAL, F. (2017) Multilevel estimation of expected exit times and other functionals of stopped diffusions. Submitted.

[35] GOBET, E. (2001) Euler schemes and half-space approximation for the simulation of diffusion in a domain. *ESAIM Probab. Statist.* **5**, 261–297.

[36] GOBET, E. (2016) *Monte–Carlo Methods and Stochastic Processes: From Linear to Non-Linear*, CRC Press.

[37] GOBET, E., LIU, G. & ZUBELLI, J. (2016) A non-intrusive stratified resampler for regression Monte Carlo: Application to solving non-linear equations. hal-01291056.

[38] GOBET, E. & MAIRE, S. (2005) Sequential Monte Carlo domain decomposition for the Poisson equation. In: *Proceedings of the 17th IMACS World Congress, Scientific Computation, Applied Mathematics and Simulation* (11–15 July 2005, Paris).

[39] GOBET, E. & MENOZZI, S. (2010) Stopped diffusion processes: Boundary corrections and overshoot. *Stoch. Process. Appl.* **120**(2), 130–162.

[40] GUYON, J. & HENRY-LABORDÈRE, P. (2013) *Nonlinear Option Pricing*, CRC Press.

[41] HENRY-LABORDERE, P. (2012) Counterparty risk valuation: A marked branching diffusion approach. Available at SSRN: https://ssrn.com/abstract=1995503.

[42] HENRY-LABORDERE, P., OUDJANE, N., TAN, X., TOUZI, N. & WARIN, X. (2016) Branching diffusion representation of semilinear PDEs and Monte Carlo approximation. *arXiv:1603.01727*.

[43] HENRY-LABORDERE, P., TAN, X. & TOUZI, N. (2014) A numerical algorithm for a class of BSDEs via the branching process. *Stoch. Process. Appl.* **124**(2), 1112–1140.

[44] HIGHAM, D. J., MAO, X., ROJ, M., SONG, Q. & YIN, G. (2013) Mean exit times and the multilevel Monte Carlo method. *SIAM/ASA J. Uncertain. Quantification* **1**(1), 2–18.

[45] KARATZAS, I. & SHREVE, S. (1991) *Brownian Motion and Stochastic Calculus*, Vol. 113, Springer-Verlag, New York.

[46] KLOEDEN, P. E. & PLATEN, E. (1992) *Numerical Solution of Stochastic Differential Equations*, Applications of Mathematics (New York), Vol. 23, Springer-Verlag, Berlin.

[47] KOLMOGOROV, A. N., PETROVSKY, I. G. & PISKUNOV, N. S. (1937) Étude de l'équation de la diffusion avec croissance de la quantité de matiere et son application a un probleme biologique. *Moscow Univ. Math. Bull* **1**(1–25), 129.

[48] LIONNET, A., DOS REIS, G. & SZPRUCH, L. (2015) Time discretization of FBSDE with polynomial growth drivers and reaction–diffusion PDEs. *Ann. Appl. Probab.* **25**(5), 2563–2625.

[49] LIONNET, A., DOS REIS, G. & SZPRUCH, L. (2016) Convergence and properties of modified explicit schemes for BSDEs with polynomial growth. arXiv:1607.06733.

[50] MA, J. & YONG, J. (1999) *Forward-Backward Stochastic Differential Equations and Their Applications*, Lecture Notes in Mathematics, Vol. 1702, Springer-Verlag, Berlin.

[51] MCKEAN, H. P. (1975) Application of Brownian motion to the equation of Kolmogorov-Petrovskii-Piskunov. *Commun. Pure Appl. Math.* **28**(3), 323–331.

[52] MENDES, R. V. (2010) Poisson–Vlasov in a strong magnetic field: A stochastic solution approach. *J. Math. Phys.* **51**(4), 043101.

[53] MILSTEIN, G. N. & TRETYAKOV, M. V. (2004) *Stochastic Numerics for Mathematical Physics*, Springer-Verlag, Berlin.

[54] PARDOUX, É. & PENG, S. (1992) Backward stochastic differential equations and quasilinear parabolic partial differential equations. In: *Stochastic partial differential equations and*

their Applications (Charlotte, NC, 1991), Lec. Notes in Control and Inform. Sci., Vol. 176, Springer, Berlin, pp. 200–217.

[55] PENG, S. (1991) Probabilistic interpretation for systems of quasilinear parabolic partial differential equations. *Stoch. Stoch. Rep.* **37**(1–2), 61–74.

[56] RASULOV, A., RAIMOVA, G. & MASCAGNI, M. (2010) Monte Carlo solution of Cauchy problem for a nonlinear parabolic equation. *Math. Comput. Simul.* **80**(6), 1118–1123.

[57] SKOROKHOD, A. V. (1964) Branching diffusion processes. *Teor. Verojatnost. i Primenen.* **9**(3), 492–497.

[58] SMITH, B., BJORSTAD, P. & GROPP, W. (2004) *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge, Cambridge University Press.

[59] STRUWE, M. (1996) Geometric evolution problems. In: *Nonlinear Partial Differential Equations in Differential Geometry (Park City, UT, 1992)*. IAS/Park City Math. Ser., Vol. 2, Amer. Math. Soc., Providence, RI, pp. 257–339.

[60] WARIN, X. (2017) Variations on branching methods for nonlinear PDEs. *arXiv:1701.07660*.

[61] WATANABE, S. (1965) On the branching process for Brownian particles with an absorbing boundary. *J. Math. Kyoto Univ.* **4**(2), 385–398.

[62] XU, Y. (2015) A complex Feynman-Kac formula via linear backward stochastic differential equations. *arXiv:1505.03590*.