Numerical solution of the incompressible Navier-Stokes equation by a deep branching algorithm

Jiang Yu Nguwi* Guillaume Penent† Nicolas Privault‡

Division of Mathematical Sciences

School of Physical and Mathematical Sciences

Nanyang Technological University

21 Nanyang Link, Singapore 637371

April 29, 2023

Abstract

We present an algorithm for the numerical solution of systems of fully nonlinear PDEs using stochastic coded branching trees. This approach covers functional nonlinearities involving gradient terms of arbitrary orders, and it requires only a boundary condition over space at a given terminal time T instead of Dirichlet or Neumann boundary conditions at all times as in standard solvers. Its implementation relies on Monte Carlo estimation, and uses neural networks that perform a meshfree functional estimation on a space-time domain. The algorithm is applied to the numerical solution of the Navier-Stokes equation and is benchmarked to other implementations in the cases of the Taylor-Green vortex and Arnold-Beltrami-Childress flow.

Keywords: Fully nonlinear PDEs, systems of PDEs, Navier-Stokes equations, Monte Carlo method, deep neural network, branching process, random tree.
Mathematics Subject Classification (2020): 35G20, 76M35, 76D05, 60H30, 60J85, 65C05.

1 Introduction

This paper is concerned with the numerical solution of systems of $d+1$ fully nonlinear coupled parabolic partial differential equations (PDEs) and Poisson equations on $[0,T]\times\mathbb{R}^d$,

*nguw0003@e.ntu.edu.sg
†pene0001@e.ntu.edu.sg
‡nprivault@ntu.edu.sg

of the form

$$\begin{cases} \partial_t u_i(t,x) + \nu \Delta u_i(t,x) + f_i\big(\partial_{\bar{\alpha}^1} u_0(t,x), \dots, \partial_{\bar{\alpha}^q} u_0(t,x), \partial_{\bar{\alpha}^{q+1}} u_{\beta_{q+1}}(t,x), \dots, \partial_{\bar{\alpha}^n} u_{\beta_n}(t,x)\big) = 0, \\ \Delta u_0(t,x) = f_0\big(\partial_{\bar{\alpha}^{q+1}} u_{\beta_{q+1}}(t,x), \dots, \partial_{\bar{\alpha}^n} u_{\beta_n}(t,x)\big), \\ u_i(T,x) = \phi_i(x), \quad (t,x) = (t,x_1,\dots,x_d) \in [0,T] \times \mathbb{R}^d, \quad i = 1,\dots,d, \end{cases} \tag{1.1}$$

where $q \in \{0,1,\dots,n\}$, $\partial_t u(t,x) = \partial u(t,x)/\partial t$, $\nu > 0$, $\Delta = \sum_{i=1}^{d} \partial^2/\partial x_i^2$ is the standard d-dimensional Laplacian, $1 \leq \beta_j \leq d$ for $q < j \leq n$, $\bar{\alpha}^i = (\alpha_1^i, \dots, \alpha_d^i) \in \mathbb{N}^d$, $i = 0,1,\dots,n$, $f_i(x_1,\dots,x_n)$ and $f_0(x_{q+1},\dots,x_n)$ are smooth functions of the derivatives

$$\partial_{\bar{\alpha}^i} u(t,x) = \frac{\partial^{\alpha_1^i}}{\partial x_1^{\alpha_1^i}} \cdots \frac{\partial^{\alpha_d^i}}{\partial x_d^{\alpha_d^i}} u(t,x_1,\dots,x_d), \qquad (x_1,\dots,x_d) \in \mathbb{R}^d,$$

and $(\phi_i)_{i=1,\dots,d}$ is a smooth terminal condition. We note that the problem (1.1) is posed using the terminal time boundary condition $u_i(T,x) = \phi_i(x)$ in $(x_1,\dots,x_d) \in \mathbb{R}^d$, instead of assuming Dirichlet and Neumann boundary conditions at all times as is usually the case in the finite difference and mesh-based literature.

As is well known, standard numerical schemes for solving (1.1) by e.g. finite differences or finite elements suffer from a high computational cost which typically grows exponentially with the dimension d. This motivates the study of probabilistic representations of (1.1), which, combined with meshfree Monte Carlo approximation, can overcome the curse of dimensionality. In addition, it is not clear how the standard numerical schemes can be applied when boundary conditions are not available.

Probabilistic representations for the solutions of first and second order nonlinear PDEs can be obtained by writing $u(t,x) \in \mathbb{R}$ as $u(t,x) = Y_t^{t,x}$, where $(Y_s^{t,x})_{t \leq s \leq T}$ is the solution of first or second order backward stochastic differential equation (BSDE), see [PP92], [CSTV07], [STZ12], and [HJE18] for a deep learning implementation. See also [LS97] for the use of stochastic branching processes for the probabilistic representation of solutions of the Navier-Stokes equation.

Stochastic branching diffusion mechanisms ([Sko64], [INW69], [McK75]) have also been applied to the probabilistic representation of the solutions of nonlinear PDEs, see e.g. [HL12], [HLOT+19] for the case of polynomial first order gradient nonlinearities, and [FTW11], [Tan13], [GZZ15], [HLZ20] for finite difference schemes combined with Monte Carlo estimation for fully nonlinear PDEs with gradients of order up to two. However, extending the

above approaches to nonlinearities in higher order derivatives involves technical difficulties linked to the integrability of the Malliavin-type weights used in repeated integration by parts argument, see page 199 of [HLOT+19].

In [NPP23], a stochastic branching method that carries information on (possibly functional) nonlinearities along a random tree has been introduced, with the aim of providing Monte Carlo schemes for the numerical solution of fully nonlinear PDEs with gradients of arbitrary orders on the real line. This method has been implemented on $\mathbb{R}^d$ in [NPP22] using a neural network approach to efficiently approximate the PDE solution $u(t,x) \in \mathbb{R}$ over a bounded domain in $[0,T] \times \mathbb{R}^d$.

In this paper, we extend the approaches in [NPP22] and [NPP23] to treat the case of systems of fully nonlinear PDEs of the form (1.1), and we apply our algorithm to the incompressible Navier-Stokes equation

$$\begin{cases} \partial_t u(t,x) + \nu \Delta u(t,x) = \nabla p(t,x) + (u \cdot \nabla) u, \\ \Delta p(t,x) = -\operatorname{div} \operatorname{div} (u \otimes u), \\ u(T,x) = \phi(x), \quad (t,x) = (t, x_1, \ldots, x_d) \in [0,T] \times \mathbb{R}^d, \end{cases}$$

with pressure term $p(t,x) = u_0(t,x)$. This equation is a special case of (1.1) obtained by taking $n = d(d+2)$ and $q = d$, see Section 4, and can be rewritten as the divergence-free problem

$$\begin{cases} \partial_t u(t,x) + \nu \Delta u(t,x) = \nabla p(t,x) + (u \cdot \nabla) u, \\ \operatorname{div} u = 0, \\ u(T,x) = \phi(x), \quad (t,x) = (t, x_1, \ldots, x_d) \in [0,T] \times \mathbb{R}^d. \end{cases}$$

Probabilistic representations for the solution of the Navier-Stokes equation using BSDEs have been considered in e.g. [AB10], [CC07], [CS09], and Monte Carlo numerical algorithms based on BSDEs have been designed and implemented in [DQT15] and [LG20]. The BSDE approach is however restricted to first order nonlinear PDE systems for which we have $\max_j \sum_{i=1}^d \alpha_i^j \leq 1$ in (1.1), and its numerical implementation involves errors from both Monte Carlo estimation and time discretization, thus reducing its effectiveness. The Navier-Stokes equation can also be solved in the framework of physics-informed neural networks (PINN) using the Galerkin method [RPK19] and high quality solution data usually obtained from

an existing solver over a given training domain. On the other hand, our branching algorithm belongs to the family of solvers that do not use existing training data.

In Section 4 we compare our numerical results to those obtained in [LG20] using BSDEs and the Monte Carlo method, and in [APFC17] using finite-difference and finite-element methods. We note in particular that our method is more stable and much faster than the BSDE approach of [LG20] which has been implemented on a computer cluster with a few tens of cores.

We also compare our results to those of [APFC17] in which the 2D Taylor-Green example has been treated by finite-difference and finite-element methods with viscosities $\nu = 10^{-1}$ and $\nu = 10^{-2}$, see Section 5 therein. Although we cannot fully match the speed and precision of state of the art finite element methods, we would like to stress the following points.

- Our neural network approach yields a full functional estimation over a whole time-space domain instead of pointwise estimates on a grid as in mesh-based methods.

- Monte Carlo estimation provides an intuitive interpretation of the solution of partial differential equations via the diffusion of heat mechanism, as such it makes sense to test their applicability, which also opens the door to future applications to the solution of higher dimensional systems.

  In particular, our branching algorithm overcomes the curse of dimensionality because the number of tree branches is not sensitive to dimension, see the comments at the end of Section 2. For example, in [NPP22], [NPP23] this branching method has been applied to PDE examples in dimension 100, which may not be treated using mesh-based methods.

- Our results compare favorably to other Monte Carlo algorithms such as [LG20], in which computations for a single time step can require several hours.

In [Mat21], the Deep Galerkin Method (DGM) has been applied to the numerical solution of compressible Navier-Stokes equations with Reynolds numbers around 1,000, and in [LYZD22], the DGM method has been applied to time-independent Stokes equations. However, we have not encounter applications of the DGM method to the incompressible Navier-Stokes equation in the literature, including for the Taylor-Green vortex and the

Arnold-Beltrami-Childress flow. In Section 4.3 we compare the output of our method to that of the deep Galerkin method [SS18], see Figures 7-10. We note that the DGM method performs correctly if one reduces the domain of study from $[0, 2\pi]^2$ to $[0, 1]^2$ as done in e.g. [LYZD22] for Stokes equations, and uses space-time boundary conditions on $[0, 1]^2 \times [0, T]$. On the other hand, we observe that the DGM algorithm loses its accuracy when only a condition at terminal time T is used as in our method, or when the domain is extended from $[0, 1]^2$ to $[0, 2\pi]^2$, see Figures 8-10.

Although our method does not use domain boundary conditions, such conditions can be taken into account by replacing the standard Gaussian kernel by specialized heat kernels, see e.g. § III-4 of [Bor17] for explicit heat kernel expressions with rectangle boundary conditions.

In addition to dealing with the Navier-Stokes equation, the framework of Equation (1.1) is general enough to potentially cover equations of non-Newtonian fluid mechanics in which viscosity may depend on the gradient of the solution, as, for example, in the non-Newtonian Navier-Stokes equation

$$\partial_t u(t,x) + \xi\nu|\partial_x u(t,x)|^{\xi-1}\Delta u(t,x) = \nabla p(t,x) + (u\cdot\nabla)u,$$

for a power-law non-Newtonian flow, here in dimension $d = 1$, $\xi > 0$.

We proceed as follows. In Section 2, we present the construction of the probability representation (2.10) for the PDE system (1.1) with the corresponding algorithm. Then, in Section 3 we outline the deep branching method for the estimation of (2.10). Then in Section 4 we apply the deep branching method to the examples of Taylor-Green vortex and Arnold-Beltrami-Childress flow, and we present further examples using rotating flows.

The Python codes used for our numerical illustrations are available at `https://github.com/nguwijy/deep_navier_stokes`.

Notation

We denote by $\mathbb{N} = \{0, 1, 2, \dots\}$ the set of natural numbers. We let $\mathcal{C}^{0,\infty}([0, T] \times \mathbb{R}^d; \mathbb{R}^k)$ be the set of functions $u : [0, T] \times \mathbb{R}^d \to \mathbb{R}^k$ such that $u(t, x)$ is continuous in t for all $x \in \mathbb{R}^d$, and infinitely x-differentiable for all $t \in [0, T]$. For a vector $x = (x_1, \dots, x_d)^\top \in \mathbb{R}^d$, we let $|x| = \sum_{i=1}^{d} |x_i|$ and $\mathbf{1}_p$ be the indicator vector made of 1 at position $p \in \{1, \dots, d\}$, and

0 elsewhere. As in [CS96], for use in the multivariate Faà di Bruno formula we also define a linear order $\prec$ on $\mathbb{R}^d$ such that $(k_1,\dots,k_d) = k \prec l = (l_1,\dots,l_d)$ if one of the following holds:

i) $|k| < |l|$;

ii) $|k| = |l|$ and $k_1 < l_1$;

iii) $|k| = |l|$, $k_1 = l_1, \dots k_i = l_i$, and $k_{i+1} < l_{i+1}$ for some $1 \le i < d$.

Given $\mu \in \mathbb{N}^d$, $f \in \mathcal{C}^\infty(\mathbb{R}^n)$ and $v \in \mathcal{C}^{0,\infty}([0,T]\times\mathbb{R}^d;\mathbb{R}^n)$, we will use the multivariate Faà di Bruno formula

$$\partial_\mu(f(v(t,x))) = \left(\prod_{i=1}^d \mu_i!\right) \sum_{\substack{1\le\lambda_1+\cdots+\lambda_n\le|\mu|\\ 1\le s\le|\mu|}} \partial_\lambda f(t,x) \sum_{\substack{1\le|k_1|,\dots,|k_s|,\ 0\prec l^1\prec\cdots\prec l^s\\ k_1^i+\cdots+k_s^i=\lambda_i,\ i=1,\dots,n\\ |k_1|l_j^1+\cdots+|k_s|l_j^s=\mu_j,\ j=1,\dots,d}} \prod_{\substack{1\le i\le n\\ 1\le r\le s}} \frac{(\partial_{l^r} v_i(t,x))^{k_r^i}}{k_r^i!\,(l_1^r!\cdots l_d^r!)^{k_r^i}}, \tag{1.2}$$

$x = (x_1,\dots,x_d) \in \mathbb{R}^d$, see Theorem 2.1 in [CS96].

2 Fully nonlinear Feynman-Kac formula

In this section we extend the construction of [NPP23], [NPP22] to the case of systems of fully nonlinear coupled parabolic and Poisson equations PDEs of the form (1.1). For this, we rewrite (1.1) in integral form for $i = 1,\dots,d$ as

$$\begin{cases} u_0(t,x) = \dfrac{\Gamma(d/2)}{2\pi^{d/2}} \displaystyle\int_{\mathbb{R}^d} \frac{N(y)}{|y|^d} f_0\big(\partial_{\bar\alpha^{q+1}} u_{\beta_{q+1}}(t,x+y),\dots,\partial_{\bar\alpha^n} u_{\beta_n}(t,x+y)\big)dy, & (2.1)\\ u_i(t,x) = \displaystyle\int_{\mathbb{R}^d} \varphi_{2\nu}(T-t,y-x)\phi_i(y)dy & (2.2)\\ + \displaystyle\int_t^T \int_{\mathbb{R}^d} \varphi_{2\nu}(s-t,y-x) f_i\big(\partial_{\bar\alpha^1} u_0(s,y),\dots,\partial_{\bar\alpha^q} u_0(s,y),\partial_{\bar\alpha^{q+1}} u_{\beta_{q+1}}(s,y),\dots,\partial_{\bar\alpha^n} u_{\beta_n}(s,y)\big)dyds,\\ u_i(T,x) = \phi_i(x), \quad (t,x) = (t,x_1,\dots,x_d) \in [0,T]\times\mathbb{R}^d, \quad i = 0,1,\dots,d,\ (t,x)\in[0,T]\times\mathbb{R}^d, \end{cases}$$

under appropriate integrability condition as in e.g. Lemma 1.6 in [MB02], where $\varphi_{\sigma^2}(t,x) = \mathrm{e}^{-x^2/(2\sigma^2 t)}/\sqrt{2\pi\sigma^2 t}$,

$$\phi_0(x) := u_0(T,x) = \frac{\Gamma(d/2)}{2\pi^{d/2}} \int_{\mathbb{R}^d} \frac{N(y)}{|y|^d} f_0\big(\partial_{\bar\alpha^{q+1}}\phi_{\beta_{q+1}}(x+y),\dots,\partial_{\bar\alpha^n}\phi_{\beta_n}(x+y)\big)dy,$$

$\Gamma(y) := \int_0^\infty x^{z-1}\mathrm{e}^{-x}dx$ is the Gamma function, and $N(y)$ is the Poisson kernel

$$N(y) = \begin{cases} |y|^2 \log|y|, & d = 2, \\ \dfrac{|y|^2}{2-d}, & d \geq 3, \quad y \in \mathbb{R}^d. \end{cases}$$

Our fully nonlinear Feynman-Kac formula relies on the construction of a branching coding tree, based on the definition of a set $\mathcal{C}$ of codes and its associated branching mechanism $\mathcal{M}$. In what follows, for any function $g : \mathbb{R}^n \to \mathbb{R}$, we let g^* be the operator mapping $\mathcal{C}^{0,\infty}([0,T] \times \mathbb{R}^d)$ to $\mathcal{C}^{0,\infty}([0,T] \times \mathbb{R}^d)$ and defined by

$$\begin{aligned} g^*(u)(t,x) &:= g\big(\partial_{\bar{\alpha}^1} u_0(t,x), \ldots, \partial_{\bar{\alpha}^q} u_0(t,x), \partial_{\bar{\alpha}^{q+1}} u_{\beta_{q+1}}(t,x), \ldots, \partial_{\bar{\alpha}^n} u_{\beta_n}(t,x)\big) \\ &= g\big(\partial_{\bar{\alpha}^1} u_{\beta_1}(t,x), \ldots, \partial_{\bar{\alpha}^n} u_{\beta_n}(t,x)\big), \end{aligned}$$

with $\beta_1 = \cdots = \beta_q = 0$, from which (1.1) can be rewritten as

$$\partial_t u_i(t,x) + \nu \Delta u_i(t,x) + f_i^*(u)(t,x) = 0, \qquad i = 1, \ldots, d,$$

$(t,x) = (t, x_1, \ldots, x_d) \in [0,T] \times \mathbb{R}^d$. In the sequel, for $\lambda = (\lambda_1, \ldots, \lambda_n) \in \mathbb{N}^n$ we let

$$\partial_\lambda f_i(x_1, \ldots, x_n) = \frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}} \cdots \frac{\partial^{\lambda_n}}{\partial x_n^{\lambda_n}} f_i(x_1, \ldots, x_n), \qquad (x_1, \ldots, x_n) \in \mathbb{R}^n,$$

$$\partial_\lambda f_0(x_{q+1}, \ldots, x_n) = \frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}} \cdots \frac{\partial^{\lambda_n}}{\partial x_n^{\lambda_n}} f_0(x_{q+1}, \ldots, x_n), \qquad (x_1, \ldots, x_n) \in \mathbb{R}^n.$$

Definition 2.1 *We let $\mathcal{C}$ denote the set of operators from $\mathcal{C}^{0,\infty}([0,T] \times \mathbb{R}^d; \mathbb{R}^{d+1})$ to $\mathcal{C}^{0,\infty}([0,T] \times \mathbb{R}^d; \mathbb{R})$ called codes, and defined as*

$$\mathcal{C} := \big\{ \mathrm{Id}_i,\ (a\partial_\lambda f)^*,\ (a\partial_\mu, i),\ (\partial_\mu, -1)\ :\ \lambda \in \mathbb{N}^n,\ \mu \in \mathbb{N}^d,\ a \in \mathbb{R},\ i = 0, \ldots, d \big\}. \tag{2.4}$$

The codes c in $\mathcal{C}$ are operators acting on $(u_0, u_1, \ldots, u_d) = u \in \mathcal{C}^{0,\infty}\left([0,T] \times \mathbb{R}^d; \mathbb{R}^{d+1}\right)$ as

$$c(u)(t,x) = \begin{cases} u_i(t,x), & \text{if } c = \mathrm{Id}_i, \\ a\partial_\lambda f\left(\partial_{\bar{\alpha}^1} u_{\beta_1}(t,x), \ldots, \partial_{\bar{\alpha}^n} u_{\beta_n}(t,x)\right), & \text{if } c = (a\partial_\lambda f)^*, \\ a\partial_\mu u_i(t,x), & \text{if } c = (a\partial_\mu, i), \\ \partial_\mu(\partial_t + \nu\Delta) u_0(t,x), & \text{if } c = (\partial_\mu, -1). \end{cases}$$

In the branching algorithm implementation, the quantities $\partial_\lambda f\left(\partial_{\bar{\alpha}^1} u_{\beta_1}(t,x), \ldots, \partial_{\bar{\alpha}^n} u_{\beta_n}(t,x)\right)$ and $\partial_\mu u_i(t,x)$ will be computed by estimating $u_i(t,x)$ recursively using the integral form of

(1.1) for $i \in \{1, 2, \ldots, d\}$. No such recursion is needed for $\partial_\mu u_0(t,x)$ and $\partial_\mu(\partial_t + \nu\Delta)u_0(t,x)$ which will be computed by solving the corresponding Poisson equation using integral expressions, see (A.4) and (A.7) in appendix.

This recursion will be implemented using the *branching mechanism* $\mathcal{M}$ defined below, which is based on the multivariate Faà di Bruno formula (1.2). For the description and implementation of the algorithm we will enumerate the terms appearing in (1.2) applied to the index set $\mu \in \mathbb{N}^n$ and function f on $\mathbb{R}^n$ using the set $\mathrm{fdb}(\mu, f, (c_1, \ldots, c_m))$ of code sequences defined as

$$\begin{aligned}
&\mathrm{fdb}(\mu, f, (c_1, \ldots, c_m)) \\
&:= \bigcup_{\substack{1\le s\le|\mu|,\ 1\le\lambda_1+\cdots+\lambda_n\le|\mu| \\ 1\le|k_1|,\ldots,|k_s|,\ 0\prec l^1\prec\cdots\prec l^s \\ k_1^i+\cdots+k_s^i=\lambda_i,\ i=1,\ldots,n \\ |k_1|l_j^1+\cdots+|k_s|l_j^s=\mu_j,\ j=1,\ldots,d}} \left\{ (c_1, \ldots, c_m) \bigcup \left(\frac{\prod_{j=1}^d \mu_j!}{\prod_{\substack{1\le i\le n \\ 1\le r\le s}} k_r^i!\,(l_1^r!\cdots l_d^r!)^{k_r^i}} (\partial_\lambda f)^* \right) \right. \\
&\qquad\qquad \left. \bigcup \bigcup_{\substack{1\le i\le n \\ 1\le r\le s}} \big(\underbrace{(\partial_{l^r+\bar\alpha^i}, \beta_i), \ldots, (\partial_{l^r+\bar\alpha^i}, \beta_i)}_{k_r^i \text{ times}} \big) \right\},
\end{aligned}$$

where $(c_1, \ldots, c_m)$ is any sequence of codes in $\mathcal{C}$ and we use the notation

$$(a_1, \ldots, a_n) \cup (b_1, \ldots, b_m) := (a_1, \ldots, a_n, b_1, \ldots, b_m)$$

and $(a_1, \ldots, a_n) \cup \emptyset = (a_1, \ldots, a_n)$ for any sequences $(a_1, \ldots, a_n)$, $(b_1, \ldots, b_m)$. The next definition provides a way to enumerate the terms appearing in (1.2) and in (A.4)-(A.7) below.

Definition 2.2 *We define a* mechanism $\mathcal{M}$ *that maps any code* c *in* $\mathcal{C}$ *to a set* $\mathcal{M}(c)$ *of code sequences, by letting*

$$\begin{aligned}
&\mathcal{M}(\mathrm{Id}_i) := \{f_i^*\}, \quad i = 0, 1, \ldots, d, \\
&\mathcal{M}\left((\partial_\mu, i)\right) := \mathrm{fdb}\left(\mu, f_i, \emptyset\right), \qquad \mu \in \mathbb{N}^n, \quad i = 0, 1, \ldots, d, \\
&\mathcal{M}(g^*) := \bigcup_{q<r\le n} \mathrm{fdb}\left(\bar\alpha^r, f_{\beta_r}, ((\partial_{\mathbf{1}_r} g)^*)\right) \bigcup \bigcup_{\substack{i,j=1,\ldots,n \\ k=1,\ldots,d}} \left\{ \left(-\nu(\partial_{\mathbf{1}_i+\mathbf{1}_j} g)^*, (\partial_{\bar\alpha^i+\mathbf{1}_k}, \beta_i), (\partial_{\bar\alpha^j+\mathbf{1}_k}, \beta_j)\right) \right\} \\
&\qquad\qquad \bigcup \bigcup_{1\le r\le q} \left\{ \left(-(\partial_{\mathbf{1}_r} g)^*, (\partial_{\bar\alpha^r}, -1)\right) \right\}, \qquad g^* \in \mathcal{C},
\end{aligned}$$

and

$$
\begin{aligned}
&\mathcal{M}((\partial_\mu,-1)) \\
&:= \bigcup_{\substack{q<i,j\le n\\ k=1,\dots,d\\ 0\le \ell_p\le\gamma_p\le\mu_p\\ p=1,\dots,d}} \mathrm{fdb}\left(\gamma,\partial_{\mathbf{1}_i+\mathbf{1}_j}f_0,\left(\left(\left(\nu\prod_{r=1}^d\binom{\mu_r}{\ell_r}\binom{\ell_r}{\gamma_r}\right)\partial_{\mu-\ell+\bar\alpha^i+\mathbf{1}_k},\beta_i\right),\left(\partial_{\ell-\gamma+\bar\alpha^j+\mathbf{1}_k},\beta_j\right)\right)\right) \\
&\bigcup \bigcup_{\substack{q<i\le n\\ 0\le\ell_p\le\mu_p\\ p=1,\dots,d}} \bigcup_{\substack{1\le s\le|\ell|,\ 1\le\lambda_1+\cdots+\lambda_n\le|\ell|\\ 1\le|k_1|,\dots,|k_s|,\ 0\prec l^1\prec\cdots\prec l^s\\ k_1^i+\cdots+k_s^i=\lambda_i,\ i=1,\dots,n\\ |k_1|l_j^1+\cdots+|k_s|l_j^s=\ell_j,\ j=1,\dots,d}} \\
&\mathrm{fdb}\left(\mu-\ell+\bar\alpha^i,f_{\beta_i},\left(-\frac{\prod_{j=1}^d\frac{\mu_j!}{(\mu_j-\ell_j)!}}{\prod_{\substack{1\le i\le n\\ 1\le r\le s}}k_r^i!\,(l_1^r!\cdots l_d^r!)^{k_r^i}}(\partial_{\lambda+\mathbf{1}_i}f_0)^*\right)\bigcup\bigcup_{\substack{1\le i\le n\\ 1\le r\le s}}\Big(\underbrace{(\partial_{l^r+\bar\alpha^i},\beta_i),\dots,(\partial_{l^r+\bar\alpha^i},\beta_i)}_{k_r^i\ \text{times}}\Big)\right).
\end{aligned}
$$

In order to motivate the construction of the mechanism $\mathcal{M}$, we note that

- $\mathcal{M}\left((\partial_\mu,i)\right)$, $i=1,\dots,d$, is used to model the Faà di Bruno formula (1.2) via $\mathrm{fdb}\left(\mu,f_i,\emptyset\right)$,
- $\mathcal{M}(\mathrm{Id}_0)$ is used to model the Poisson integral equation (2.1),
- $\mathcal{M}(\mathrm{Id}_i)$, $i=1,\dots,d$, is used to model the integral equation (2.2),
- $\mathcal{M}(g^*)$ is used to model the integral equation (A.2),
- $\mathcal{M}((\partial_\mu,0))$ is used to model the Poisson integral equation (A.4),
- $\mathcal{M}((\partial_\mu,-1))$ is used to model the integral equation (A.6).

Example - semilinear PDEs

To illustrate our method, consider the simpler case of a semilinear PDE of the form

$$
\begin{cases}
\partial_t u(t,x)+\dfrac{1}{2}\partial_x^2 u(t,x)+f(u(t,x))=0\\
u(T,x)=\phi(x), \qquad (t,x)\in[0,T]\times\mathbb{R},
\end{cases}
\tag{2.5}
$$

in dimension $d=1$, with the integral formulation

$$
u(t,x)=\int_{-\infty}^{\infty}\varphi(T-t,y-x)\phi(y)dy+\int_t^T\int_{-\infty}^{\infty}\varphi(s-t,y-x)f(u(s,y))dyds. \tag{2.6}
$$

In order to iterate (2.6) into a tree-based recursion, we will find a PDE satisfied by $f(u(s,y))$ by differentiating

$$\begin{aligned}\partial_s f(u(s,y)) + \frac{1}{2}\partial_y^2 f(u(s,y)) &= \left(\partial_s u(s,y) + \frac{1}{2}\partial_y^2 u(s,y)\right) f'(u(s,y)) + \frac{1}{2}(\partial_y u(s,y))^2 f''(u(s,y)) \\ &= -f(u(s,y))f'(u(s,y)) + \frac{1}{2}(\partial_y u(s,y))^2 f''(u(s,y)),\end{aligned}$$

showing that $f(u(s,y))$ satisfies the integral equation

$$\begin{aligned}f(u(s,y)) &= \int_{-\infty}^{\infty} \varphi(T-s, z-x) f(\phi(z)) dz \\ &+ \int_s^T \int_{-\infty}^{\infty} \varphi(w-s, z-x) \left(f(u(w,z))f'(u(w,z)) - \frac{1}{2}(\partial_z u(w,z))^2 f''(u(w,z)) \right) dz dw,\end{aligned} \tag{2.7}$$

More generally, we use (2.6) and (2.7) to expand $u(t,x)$ and $af^{(k)}(u(t,x))$ into a consistent set of equations which are suitable for a recursive estimation of $u(t,x)$, as

$$\begin{cases} u(t,x) = \displaystyle\int_{-\infty}^{\infty} \varphi(T-t, y-x)\phi(y)dy + \int_t^T \int_{-\infty}^{\infty} \varphi(s-t, y-x) f(u(s,y)) dy ds \\ af^{(k)}(u(t,x)) = \displaystyle\int_{-\infty}^{\infty} \varphi(T-t, y-x) af^{(k)}(\phi(y)) dy \\ \quad + \displaystyle\int_t^T \int_{-\infty}^{\infty} \varphi(s-t, y-x) \left(af(u(s,y)) f^{(k+1)}(u(s,y)) - \frac{a}{2}(\partial_y u(s,y))^2 f^{(k+2)}(u(s,y)) \right) dy ds \end{cases}$$

$a \in \mathbb{R} \setminus \{0\}$, $k \geq 0$, and we expand $\partial_x u(t,x)$ as

$$\partial_x u(t,x) = \int_{-\infty}^{\infty} \varphi(T-t, y-x)\partial_x \phi(y) dy + \int_t^T \int_{-\infty}^{\infty} \varphi(s-t, y-x) f'(u(s,y)) \partial_y u(s,y) dy ds.$$

In this case, the set of $\mathfrak{C}$ codes in (2.4) reads

$$\mathfrak{C} := \left\{ \mathrm{Id},\ \partial_x,\ af^{(k)},\ a \in \mathbb{R} \setminus \{0\},\ k \in \mathbb{N} \right\},$$

and the branching mechanism $\mathcal{M}$ in Definition 2.2 is given by

$$\mathcal{M}(\mathrm{Id}) := \{f^*\},\ \ \mathcal{M}(g^*) := \left\{ (f^*, (\partial_{z_0} g)^*); \left(\partial_x, \partial_x, -\frac{1}{2}(\partial_{z_0}^2 g)^* \right) \right\},\ \ \mathcal{M}(\partial_x) := \left\{ \left((\partial_{z_0} f)^*, \partial_x \right) \right\}, \tag{2.8}$$

for $g \in \mathcal{C}^\infty(\mathbb{R}^{n+1})$ of the form $g = a\partial_{z_0}^k f$, $a \in \mathbb{R} \setminus \{0\}$, $k \geq 0$. Figure 1 presents a sample of the random coded tree $\mathcal{T}_{t,x,\mathrm{Id}}$ started from $c = \mathrm{Id}$ for a semilinear PDE of the form (2.5).

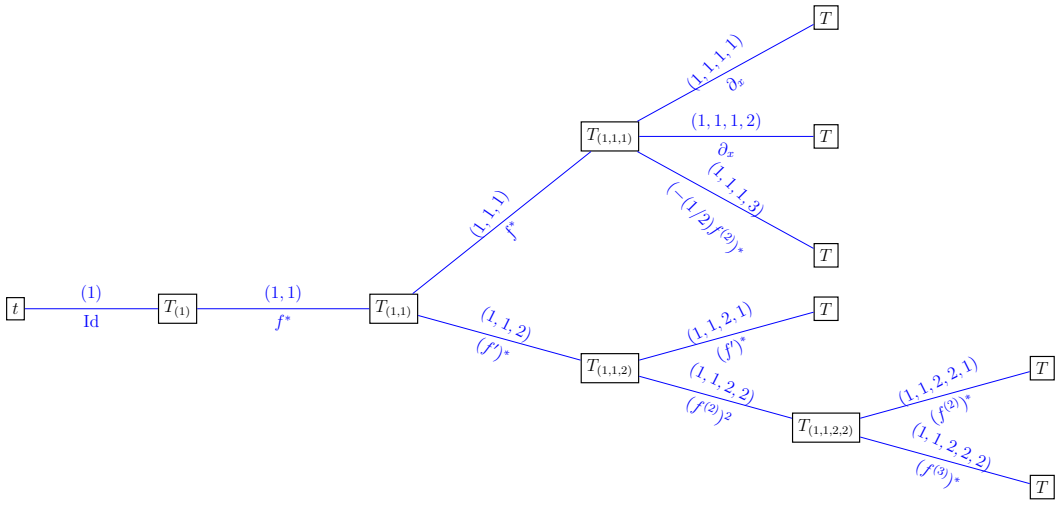


Figure 1: Sample coding tree.

Implementation

The probabilistic representation of PDE solutions will be implemented using the functional $\mathcal{H}(t, x, c)$ constructed in Algorithm 1 below along a random coding tree started at $(t, x, c) \in [0, T] \times \mathbb{R}^d \times \mathcal{C}$. We consider two probability density functions (PDF) $\rho, \widetilde{\rho} : \mathbb{R}_+ \to (0, \infty)$ on $\mathbb{R}_+$, and denote by $\overline{F}$ the tail distribution function of ρ, and let $\mathcal{N}(0, \sigma^2 \mathrm{I}_d)$ denote the d-dimensional centered normal distribution with variance σ^2 and independent components.

Algorithm 1 Coding tree algorithm TREE(t, x, c).

Input: $t \in [0, T]$, $x \in \mathbb{R}^d$, $c \in \mathcal{C}$
Output: $\mathcal{H}(t, x, c) \in \mathbb{R}$
  $\mathcal{H}(t, x, c) \leftarrow 1$
  $\tau \leftarrow$ a random variable drawn from the distribution of ρ
  $\widetilde{\tau} \leftarrow$ a random variable drawn from the distribution of $\widetilde{\rho}$
  if $t + \tau > T$ **then**
    $W_{2\nu(T-t)} \leftarrow$ a random vector drawn from $\mathcal{N}(0, 2\nu(T-t))$
    $\mathcal{H}(t, x, c) \leftarrow \mathcal{H}(t, x, c) \times c(u)(T, x + W_{2\nu(T-t)})/\overline{F}(T-t)$
  else if $c \in \left\{(a\partial_\mu, i) : a \in \mathbb{R}, \mu \in \mathbb{N}^d, i = 0, -1\right\}$ **then**
    $W_{\widetilde{\tau}} \leftarrow$ a random vector drawn from $\mathcal{N}(0, \widetilde{\tau})$
    $r_c \leftarrow$ the size of the mechanism set $\mathcal{M}(c)$
    $I_c \leftarrow$ a random element drawn uniformly from $\mathcal{M}(c)$
    $\mathcal{H}(t, x, c) \leftarrow \mathcal{H}(t, x, c) \times N(W_{\widetilde{\tau}}) \times r_c \times (2\widetilde{\tau}\,\widetilde{\rho}(\widetilde{\tau}))^{-1}$
    for all $cc \in I_c$ **do**
      $\mathcal{H}(t, x, c) \leftarrow \mathcal{H}(t, x, c) \times \text{TREE}(t, x + W_{\widetilde{\tau}}, cc)$
    end for
  else
    $W_{2\nu\tau} \leftarrow$ a random vector drawn from $\mathcal{N}(0, 2\nu\tau)$
    $r_c \leftarrow$ the size of the mechanism set $\mathcal{M}(c)$
    $I_c \leftarrow$ a random element drawn uniformly from $\mathcal{M}(c)$
    $\mathcal{H}(t, x, c) \leftarrow \mathcal{H}(t, x, c) \times r_c \times \rho^{-1}(\tau)$
    for all $cc \in I_c$ **do**
      $\mathcal{H}(t, x, c) \leftarrow \mathcal{H}(t, x, c) \times \text{TREE}(t + \tau, x + W_{2\nu\tau}, cc)$
    end for
  end if

As in Theorem 3.2 in [NPP23], the following Feynman-Kac type identity holds for the solution of (1.1) holds under suitable integrability conditions on $\mathcal{H}(t, x, \mathrm{Id}_i)$ and smoothness assumptions on the coefficients of (1.1).

Proposition 2.3 *Let $T > 0$ such that $\mathbb{E}[|\mathcal{H}(t, x, c)|] < \infty$, $c \in \mathcal{C}$, $(t, x) \in [0, T] \times \mathbb{R}^d$, and consider the system of equations*

$$\begin{cases} c(u)(t,x) = \displaystyle\sum_{Z \in \mathcal{M}(c)} \int_0^\infty \int_{\mathbb{R}^d} \varphi_1(s,y) \frac{N(y)}{2s} \prod_{z \in Z} z(u)(t, x+y) dy ds, \quad c = (\partial_\mu, 0),\ c = (\partial_\mu, -1), \\ (t,x) \in [0,T] \times \mathbb{R},\ \textit{and} \\ c(u)(t,x) = \displaystyle\int_{\mathbb{R}^d} \varphi_{2\nu}(T-t, y-x) c(u)(T,y) dy + \sum_{Z \in \mathcal{M}(c)} \int_t^T \int_{\mathbb{R}^d} \varphi_{2\nu}(s-t, y-x) \prod_{z \in Z} z(u)(s,y) dy ds, \\ \textit{for all remaining codes } c \in \mathcal{C},\ (t,x) \in [0,T] \times \mathbb{R}. \end{cases} \tag{2.9}$$

If the solution of the above system is unique, then the solution of (1.1) *admits the probabilistic representation*

$$u_i(t,x) = \mathbb{E}\left[\mathcal{H}\left(t,x,\mathrm{Id}_i\right)\right], \qquad (t,x) \in [0,T] \times \mathbb{R}^d, \tag{2.10}$$

$i = 0, 1, \ldots, d$.

The proof of Proposition 2.3 is given in appendix. It proceeds as in the proof of Theorem 3.2 of [NPP23], by showing that

$$c(u)(t,x) = \mathbb{E}[\mathcal{H}(t,x,c)], \qquad (t,x) \in [0,T] \times \mathbb{R},$$

for all codes $c \in \mathcal{C}$, which implies (2.10) by taking $c = \mathrm{Id}_i$, $i = 1, \ldots, d$.

In numerical applications, the expected value $\mathbb{E}[\mathcal{H}(t,x,c)]$ in Proposition 2.3 is estimated as the average

$$\frac{1}{N}\sum_{k=1}^{N} \mathcal{H}(t,x,c)^{(k)}$$

where $\mathcal{H}(t,x,c)^{(1)}, \ldots, \mathcal{H}(t,x,c)^{(N)}$ are independent samples of $\mathcal{H}(t,x,c)$. In this case, the error on the estimate of $\mathbb{E}[\mathcal{H}(t,x,c)]$ from the Monte Carlo method can be estimated as the standard deviation

$$\left(\mathbb{E}\left[\left(\mathbb{E}[\mathcal{H}(t,x,c)] - \frac{1}{N}\sum_{k=1}^{N} \mathcal{H}(t,x,c)^{(k)}\right)^2\right]\right)^{1/2} = \frac{1}{\sqrt{N}}\sqrt{\mathrm{Var}[\mathcal{H}(t,x,c)]}.$$

The main tunable parameter in the stochastic branching algorithm is the distribution ρ of the random branching time τ. Higher mean branching times result into shorter trees on average, therefore requiring a higher number of Monte Carlo samples in order to achieve the same precision level. For example, in the case of an exponentially distributed branching time with parameter λ, the average depth of binary branching trees until time $t > 0$ is of order $e^{\lambda t}$, see e.g. § 4 of [PP22].

Overall, the impact of dimension d is on the number of sequences in the mechanism $\mathcal{M}(c)$, i.e. on the number of possible ways of branching. On the other hand, the complexity of the algorithm is determined by the number of branches at each branching time, i.e. on the lengths of coding sequences, which do not depend on the dimension d. As a result, the complexity of our method has polynomial growth as a (small) power of the dimension d, mostly due to the use of $d+1$ coding trees in the algorithm.

3 Deep branching solver

Instead of evaluating (2.10) at a given point $(t,x) \in [0,T] \times \mathbb{R}^d$, we use the L^2-minimality property of expectation to obtain a functional estimation of $u = (u_1, \dots, u_d)$ as $u(\cdot,\cdot) = v^*(\cdot,\cdot)$ on the support of a random vector (ζ, X) on $[0,T] \times \mathbb{R}^d$ such that $\mathcal{H}(\zeta, X, \mathrm{Id}_i) \in L^2$, where

$$v^* = \underset{\left\{v:\mathbb{R}^{d+1}\to\mathbb{R}^d \,:\, v(\zeta,X)\in L^2\right\}}{\arg\min} \sum_{i=1}^{d} \mathbb{E}\left[\left(\mathcal{H}\left(\zeta, X, \mathrm{Id}_i\right) - v_i(\zeta, X)\right)^2\right]. \tag{3.1}$$

To evaluate (2.10) on $[0,T] \times \Omega$, where Ω is a bounded domain of $\mathbb{R}^d$, we can choose (ζ, X) to be a uniform random vector on $[0,T] \times \Omega$.

In order to implement the deep learning approximation, we parametrize $v(\cdot,\cdot)$ using a functional space described below. Given $\sigma : \mathbb{R} \to \mathbb{R}$ an activation function such as $\sigma_{\mathrm{ReLU}}(x) := \max(0,x)$, $\sigma_{\tanh}(x) := \tanh(x)$, $\sigma_{\mathrm{Id}}(x) := x$, we define the set of layer functions $\mathbb{L}^{\sigma}_{d_1,d_2}$ by

$$\mathbb{L}^{\sigma}_{d_1,d_2} := \left\{L : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2} \;:\; L(x) = \sigma(Wx+b),\ x \in \mathbb{R}^{d_1},\ W \in \mathbb{R}^{d_2\times d_1},\ b \in \mathbb{R}^{d_2}\right\}, \tag{3.2}$$

where $d_1 \geq 1$ is the input dimension, $d_2 \geq 1$ is the output dimension, and the activation function σ is applied component-wise to $Wx+b$. Similarly, when the input dimension and the output dimension are the same, we define the set of residual layer functions $\mathbb{L}^{\rho,\mathrm{res}}_{d}$ by

$$\mathbb{L}^{\sigma,\mathrm{res}}_{d} := \left\{L : \mathbb{R}^{d} \to \mathbb{R}^{d} \;:\; L(x) = x + \sigma(Wx+b),\ x \in \mathbb{R}^{d},\ W \in \mathbb{R}^{d\times d},\ b \in \mathbb{R}^{d}\right\}, \tag{3.3}$$

see [HZRS16]. Then, we denote by

$$\mathbb{NN}^{\sigma,l,m}_{d_1,d_2} := \left\{L_l \circ \cdots \circ L_0 : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2} \;:\; L_0 \in \mathbb{L}^{\sigma}_{d_1,m}, L_l \in \mathbb{L}^{\sigma_{\mathrm{Id}}}_{m,d_2}, L_i \in \mathbb{L}^{\sigma,\mathrm{res}}_{m}, 1 \leq i < l\right\}$$

the set of feed-forward neural networks with one output layer, $l \geq 1$ hidden residual layers each containing $m \geq 1$ neurons, and the activation functions of the output layer and the hidden layers being respectively the identity function σ_{Id} and σ. Any $v(\cdot;\theta) \in \mathbb{NN}^{\sigma,l,m}_{d_1,d_2}$ is fully determined by the sequence

$$\theta := \left(W_0, b_0, W_1, b_1, \dots, W_{l-1}, b_{l-1}, W_l, b_l\right)$$

of $((d_1+1)m + (l-1)(m+1)m + (m+1)d_2)$ parameters.

Since by the universal approximation theorem, see e.g. Theorem 1 of [Hor91], $\bigcup_{m=1}^{\infty} \mathbb{NN}_{d_1,d_2}^{\sigma,l,m}$ is dense in L^2 functional space, the optimization problem (3.1) can be approximated by

$$v^* \approx \operatorname*{arg\,min}_{v \in \mathbb{NN}_{d+1,d}^{\sigma,l,m}} \sum_{i=1}^{d} \mathbb{E}\left[\left(\mathcal{H}\left(\zeta, X, \mathrm{Id}_i\right) - v_i(\zeta, X)\right)^2\right]. \tag{3.4}$$

By the law of large numbers, (3.4) can be further approximated by

$$v^* \approx \operatorname*{arg\,min}_{v \in \mathbb{NN}_{d+1,d}^{\sigma,l,m}} \sum_{i=1}^{d} N^{-1} \sum_{j=1}^{N} \left(\mathcal{H}_{i,j} - v_i(\zeta_j, X_j)\right)^2, \tag{3.5}$$

where for all $j = 1, \ldots, N$, (ζ_j, X_j) is drawn independently from the distribution of (ζ, X) and $\mathcal{H}_{i,j}$ is drawn from $\mathcal{H}_{\zeta_j, X_j, \mathrm{Id}_i}$ using Algorithm 1. However, the approximation (3.5) may perform poorly when the variance of $\mathcal{H}_{i,j}$ is too high. To address this issue, we perform

$$v^* \approx \operatorname*{arg\,min}_{v \in \mathbb{NN}_{d+1,d}^{\sigma,l,m}} \sum_{i=1}^{d} \frac{1}{N} \sum_{j=1}^{N} \left(\frac{1}{M} \sum_{k=1}^{M} \mathcal{H}_{i,j,k} - v_i(\zeta_j, X_j)\right)^2, \tag{3.6}$$

where for all $k = 1, \ldots, M$, $\mathcal{H}_{i,j,k}$ is drawn independently from $\mathcal{H}_{\zeta_j, X_j, \mathrm{Id}_i}$ using Algorithm 1.

Finally, the deep branching algorithm using the gradient descent method to solve the optimization in (3.6) is summarized in Algorithm 2.

Algorithm 2 Deep branching algorithm.

Input: The learning rate η and the number of epochs P
Output: $v(\cdot, \cdot; \theta) \in \mathbb{NN}_{d+1,d}^{\sigma,l,m}$

$(\zeta_j, X_j)_{1 \le j \le N} \leftarrow$ random vectors drawn from the distribution of (ζ, X)
$(\mathcal{H}_{i,j,k})_{\substack{1 \le i \le d \\ 1 \le j \le N \\ 1 \le k \le M}} \leftarrow$ random variables generated by $\mathrm{TREE}(\zeta_j, X_j, \mathrm{Id}_i)$ in Algorithm 1
Initialize θ
for $i \leftarrow 1, \ldots, P$ **do**
  $L \leftarrow \sum_{i=1}^{d} N^{-1} \sum_{j=1}^{N} \left(M^{-1} \sum_{k=1}^{M} \mathcal{H}_{i,j,k} - v_i(\zeta_j, X_j; \theta)\right)^2$
  $\theta \leftarrow \theta - \eta \nabla_\theta L$
end for

Since no closed form expression may be available for the function

$$\phi_0(x) = \frac{\Gamma(d/2)}{2\pi^{d/2}} \int_{\mathbb{R}^d} \frac{N(y)}{|y|^d} f_0\big(\partial_{\bar{\alpha}^{q+1}} \phi_{\beta_{q+1}}(x+y), \ldots, \partial_{\bar{\alpha}^n} \phi_{\beta_n}(x+y)\big) dy,$$

we approximate it using the neural network function and Monte Carlo method for the numerical integration. More precisely, we approximate

$$\phi_0(x) = \int_0^\infty \int_{\mathbb{R}^d} \varphi_1(s,y) \frac{N(y)}{2s} f_0\big(\partial_{\bar{\alpha}^{q+1}} \phi_{\beta_{q+1}}(x+y), \dots, \partial_{\bar{\alpha}^n} \phi_{\beta_n}(x+y)\big) dyds \tag{3.7}$$

using

$$\phi_0 \approx \underset{v \in \mathbb{NN}_{d,1}^{\sigma,l,m}}{\arg\min} N^{-1} \sum_{i=1}^{N} \Bigg(M^{-1} \sum_{j=1}^{M} N(Y_{i,j}) \times (2\widetilde{\tau}_{i,j}\, \widetilde{\rho}(\widetilde{\tau_{i,j}}))^{-1} \times$$
$$f_0\big(\partial_{\bar{\alpha}^{q+1}} \phi_{\beta_{q+1}}(X_i + Y_{i,j}), \dots, \partial_{\bar{\alpha}^n} \phi_{\beta_n}(X_i + Y_{i,j})\big) - v_i(\zeta_i, X_i) \Bigg)^2,$$

where X_i is the uniform vector on $[x_{\min} - (x_{\max} - x_{\min})/2, x_{\max} + (x_{\max} - x_{\min})/2]^d$, $\widetilde{\tau}_{i,j}$ is the random variable drawn independently from the distribution of $\widetilde{\rho}$, and $Y_{i,j}$ is the random vector drawn independently from $\mathcal{N}(0, \widetilde{\tau}_{i,j})$, see (A.4) for the derivation of (3.7).

Algorithm 2 is implemented with the following parameters:

a) ρ is chosen to be the PDF of exponential distribution with rate $-(\log 0.95)/T$,

b) $\widetilde{\rho}$ is chosen to be the PDF of uniform distribution $\widetilde{\rho}(x) = (6 - 10^{-5})^{-1} \mathbf{1}_{[10^{-5},6]}(x)$,

c) given $x_{\min} < x_{\max}$, we let (ζ, X) be a uniformly distributed random vector on $[0,T] \times \Omega$, where $\Omega := [x_{\min}, x_{\max}]^d$,

d) the activation function $\sigma_{\tanh}(x) := \tanh(x)$ is used instead of ReLu because the target PDE solution (1.1) is smooth,

e) the optimal learning rate η for gradient update is obtained by trial and error, given that that a lower η means slow convergence to a possibly local suboptimum, while a higher η can lead to instability,

f) standard parameters without tuning were used for Adam optimization and batch normalization,

and we perform the following additional steps:

g) $\eta \leftarrow \eta/10$ at epoch $1,000$ and $2,000$.

h) Instead of using η to update θ directly, the Adam algorithm is used to update θ, see [KB14].

i) A batch normalization layer is added before the every layer of (3.2)-(3.3), see [IS15].

4 Application to the Navier-Stokes equation

The incompressible Navier-Stokes equation

$$\begin{cases} \partial_t u_i(t,x) + \nu \Delta u_i(t,x) = \partial_{\mathbf{1}_i} p(t,x) + \sum_{j=1}^d u_j(t,x) \partial_{\mathbf{1}_j} u_i(t,x), \\ \Delta p(t,x) = -\sum_{i,j=1}^d \partial_{\mathbf{1}_j} u_i(t,x) \partial_{\mathbf{1}_i} u_j(t,x), \\ u_i(T,x) = \phi_i(x), \quad (t,x) = (t,x_1,\dots,x_d) \in [0,T] \times \mathbb{R}^d, \quad i = 1,\dots,d, \end{cases}$$

with pressure term $p(t,x) = u_0(t,x)$ can be obtained as a particular case of the system (1.1). For this, we take $n = d(d+2)$, $q = d$, and let

$$f_0\big(y_1,\dots,y_d,z_1^{(1)},\dots,z_d^{(1)},\dots,z_1^{(d)},\dots,z_d^{(d)}\big) = -\sum_{i,j=1}^d z_i^{(j)} z_j^{(i)}$$

and

$$f_i\big(x_1,\dots,x_d,y_1,\dots,y_d,z_1^{(1)},\dots,z_d^{(1)},\dots,z_1^{(d)},\dots,z_d^{(d)}\big) = -x_i - \sum_{j=1}^d y_j z_i^{(j)},$$

$i = 1,\dots,d$, with $\bar\alpha^i = \mathbf{1}_i$, $i = 1,\dots,d$, $\bar\alpha^{d+1} = \cdots = \bar\alpha^{2d} = 0$, $\beta_{d+i} = i$, $i = 1,\dots,d$, $\bar\alpha^{i+(j+1)d} = \mathbf{1}_j$, $\beta_{i+(j+1)d} = i$, $i,j = 1,\dots,d$, so that

$$\begin{aligned} f_0\big(\partial_{\bar\alpha^{q+1}} u_{\beta_{q+1}}(t,x),\dots,\partial_{\bar\alpha^n} u_{\beta_n}(t,x)\big) &= f_0\big(u_1(t,x),\dots,u_d(t,x),(\partial_{\mathbf{1}_j} u_k(t,x))_{1\le j,k\le d}\big) \\ &= -\sum_{i,j=1}^d \partial_{\mathbf{1}_j} u_i(t,x) \partial_{\mathbf{1}_i} u_j(t,x), \end{aligned}$$

and

$$\begin{aligned} &f_i\big(\partial_{\bar\alpha^1} u_0(t,x),\dots,\partial_{\bar\alpha^q} u_0(t,x),\partial_{\bar\alpha^{q+1}} u_{\beta_{q+1}}(t,x),\dots,\partial_{\bar\alpha^n} u_{\beta_n}(t,x)\big) \\ &\quad = f_i\big(\partial_{\mathbf{1}_1} u_0(t,x),\dots,\partial_{\mathbf{1}_d} u_0(t,x),u_1(t,x),\dots,u_d(t,x),(\partial_{\mathbf{1}_j} u_k(t,x))_{1\le j,k\le d}\big) \\ &\quad = -\partial_{\mathbf{1}_i} u_0(t,x) - \sum_{j=1}^d u_j(t,x) \partial_{\mathbf{1}_j} u_i(t,x). \end{aligned}$$

In order to apply Proposition 2.3, we note that Theorem 3.1 of [DQT15] ensures the existence of a unique strong solution to the Navier-Stokes equation on a sufficiently small time interval $[t_0,T]$, provided that the terminal condition ϕ belongs to the Sobolev space of order $\lceil d/2 \rceil$

on $\mathbb{R}^d$. See also Theorem 4.1 of [LS97] for the existence and uniqueness of a global solution under sufficient conditions on the terminal data ϕ.

The following numerical examples in Sections 4.1-4.2 are implemented in PYTHON using PYTORCH on a computer with a 3.60 GHz AMD Ryzen 5 3500 processor, a 16 GB at 3200 MHz DDR4-SDRAM, and a GeForce RTX 3080 Ti graphics card with 12 GB memory. The default PYTORCH initialization scheme for θ is used, together with the default values $N = 100,000$, $M = 1,000$, $P = 10,000$, $\eta = 0.01$, $l = 3$, $m = 100$, $x_{\min} = 0$, $x_{\max} = 2\pi$. For any $\delta > 0$, we let $C_\delta := \delta\mathbb{Z}^d$ and perform the analysis of error on the grid of $\Omega \cap C_\delta$ at time $t_k = kT/10$ for $k = 0, 1, \ldots, 9$. Our benchmarking to [APFC17] and [LG20] uses the following errors:

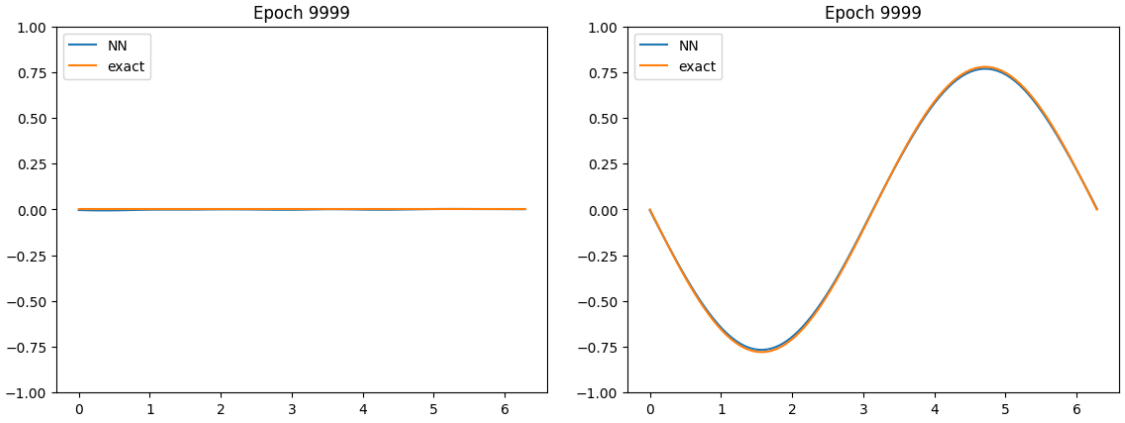
$$
\begin{aligned}
e_i(t_k) &= \sup_{x \in \Omega \cap C_\delta} |u_i(t_k, x) - v_i(t_k, x; \theta)|^2, \\
e(t_k) &= \sup_{x \in \Omega \cap C_\delta} \sum_{i=1}^{d} |u_i(t_k, x) - v_i(t_k, x; \theta)|^2, \\
\mathrm{erru}(t_k) &= \left(\frac{\sum_{i=1}^{d} \sum_{x \in \Omega \cap C_\delta} |u_i(t_k, x) - v_i(t_k, x; \theta)|^2}{\sum_{i=1}^{d} \sum_{x \in \Omega \cap C_\delta} |u_i(t_k, x)|^2} \right)^{1/2}, \\
\mathrm{errgu}(t_k) &= \left(\frac{\sum_{i,j=1}^{d} \sum_{x \in \Omega \cap C_\delta} |\partial_{\mathbf{1}_j} u_i(t_k, x) - \partial_{\mathbf{1}_j} v_i(t_k, x; \theta)|^2}{\sum_{i,j=1}^{d} \sum_{x \in \Omega \cap C_\delta} |\partial_{\mathbf{1}_j} u_i(t_k, x)|^2} \right)^{1/2}, \\
\mathrm{errdivu}(t_k) &= \left((x_{\max} - x_{\min})^d |\Omega|^{-1} \sum_{i=1}^{d} \sum_{x \in \Omega \cap C_\delta} |\partial_{\mathbf{1}_i} u_i(t_k, x)|^2 \right)^{1/2}, \\
\mathrm{errp}(T) &= \left(\frac{\sum_{x \in \Omega \cap C_\delta} |p(T, x) - v_0(T, x; \theta) + |\Omega|^{-1} \sum_{x \in \Omega \cap C_\delta} v_0(T, x; \theta)|^2}{\sum_{x \in \Omega \cap C_\delta} |p(T, x)|^2} \right)^{1/2}.
\end{aligned}
$$

4.1 Taylor-Green vortex

In this section we consider the 2-dimensional Taylor-Green [TG37] vortex

$$\begin{cases} u_1(t,x) = -\cos(x_1)\sin(x_2)\mathrm{e}^{-2\nu(T-t)}, \\ u_2(t,x) = \sin(x_1)\cos(x_2)\mathrm{e}^{-2\nu(T-t)}, \\ u_0(t,x) = -\dfrac{1}{4}\left(\cos(2x_1)+\cos(2x_2)\right)\mathrm{e}^{-4\nu(T-t)} + c \end{cases} \tag{4.1}$$

$x = (x_1, x_2) \in [0, 2\pi]^2$, with Reynolds numbers in the range $[1, 100]$. We first let $\nu = 1$, $\delta = \pi/126$, $T = 1/4$, and present the results in Figure 2 and Table 1. In this example and the next one, our method provides a solution on $[0, T] \times \mathbb{R}^d$ by only imposing a terminal condition at terminal time $t = T$. As those examples are periodic we only provide solution values on a given interval of periodicity as in [LG20], however our solver can be used to yield estimates on larger intervals as well.



(a) Comparison for $u_1(x_1, \pi)$. (b) Comparison for $u_2(x_1, \pi)$.

Figure 2: Comparison with the exact solution (4.1) by taking $t = T/2$ and $x_2 = \pi$.

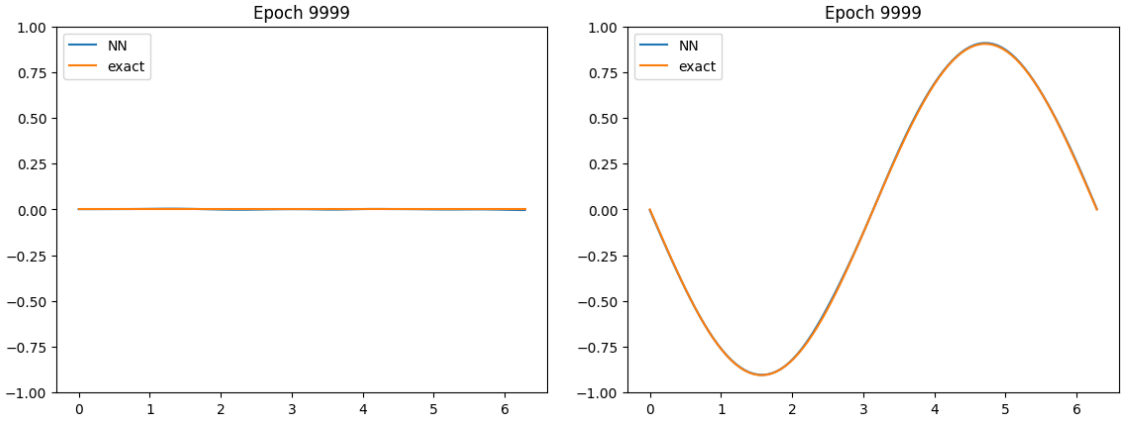
k	0	1	2	3	4	5	6	7	8	9	10
$e_0(t_k)$	1.90E-04	1.94E-04	1.91E-04	2.03E-04	2.51E-04	2.96E-04	3.28E-04	3.53E-04	3.79E-04	4.64E-04	—
$e_1(t_k)$	1.99E-04	1.95E-04	2.24E-04	2.34E-04	2.44E-04	2.56E-04	2.63E-04	2.44E-04	2.40E-04	3.63E-04	—
$e(t_k)$	2.32E-04	1.98E-04	2.27E-04	2.40E-04	2.58E-04	3.06E-04	3.39E-04	3.64E-04	3.86E-04	4.65E-04	—
$\mathrm{erru}(t_k)$	1.57E-02	1.43E-02	1.51E-02	1.64E-02	1.73E-02	1.76E-02	1.72E-02	1.64E-02	1.56E-02	1.56E-02	—
$\mathrm{errgu}(t_k)$	3.24E-02	2.75E-02	2.50E-02	2.40E-02	2.37E-02	2.34E-02	2.28E-02	2.20E-02	2.14E-02	2.20E-02	—
$\mathrm{errdivu}(t_k)$	2.03E-02	1.38E-02	9.92E-03	9.13E-03	1.02E-02	1.12E-02	1.14E-02	1.06E-02	9.81E-03	1.33E-02	—
$\mathrm{errp}(t_k)$	—	—	—	—	—	—	—	—	—	—	1.75E-02

Table 1: Error comparison.

Our simulation runtime on the full grid $[0, T] \times \Omega$ is approximately 22 minutes for the Taylor-Green vortex, after 20 minutes of pre-computation for the training of the terminal condition

$p(T, x)$. Table 1 above can be compared[1] to Table 1 in [LG20] where computing a single time step by BSDEs and Monte Carlo on a computer cluster with a few tens of cores took approximately 2 hours, whereas our neural network approach yields a functional estimate on $[0, 2\pi]^2 \times [0, T]$.

Next, we let $\nu = 0.1$, $T = 1$, and present the results in Figure 3 and Table 2.



(a) Comparison for $u_1(x_1, \pi)$. (b) Comparison for $u_2(x_1, \pi)$.

Figure 3: Comparison with the exact solution (4.1) by taking $t = T/2$ and $x_2 = \pi$.

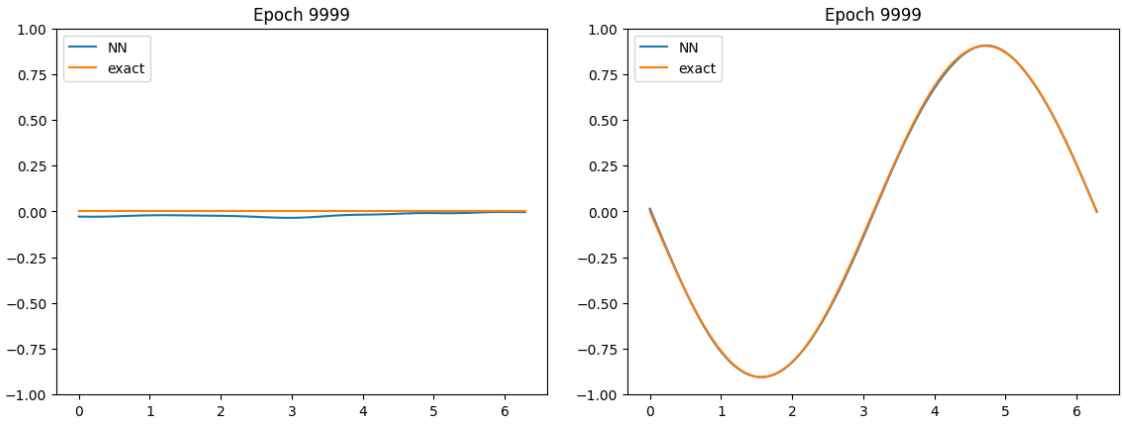
k	0	1	2	3	4	5	6	7	8	9	10
$e_0(t_k)$	3.27E-04	2.48E-04	1.73E-04	1.41E-04	1.53E-04	1.62E-04	1.67E-04	1.73E-04	1.89E-04	2.07E-04	—
$e_1(t_k)$	3.27E-04	2.04E-04	1.34E-04	1.28E-04	1.17E-04	1.12E-04	1.26E-04	1.54E-04	1.93E-04	2.54E-04	—
$e(t_k)$	3.84E-04	2.72E-04	1.79E-04	1.67E-04	1.75E-04	1.78E-04	1.75E-04	1.77E-04	2.14E-04	2.70E-04	—
erru(t_k)	1.29E-02	1.04E-02	8.90E-03	8.34E-03	8.27E-03	8.28E-03	8.18E-03	8.03E-03	8.17E-03	9.18E-03	—
errgu(t_k)	3.03E-02	2.56E-02	2.17E-02	1.88E-02	1.70E-02	1.61E-02	1.62E-02	1.73E-02	1.94E-02	2.25E-02	—
errdivu(t_k)	2.52E-02	1.76E-02	1.32E-02	1.21E-02	1.29E-02	1.35E-02	1.34E-02	1.29E-02	1.36E-02	1.78E-02	—
errp(t_k)	—	—	—	—	—	—	—	—	—	—	1.75E-02

Table 2: Error comparison.

Table 2 above can be compared to Tables 17 and 19 in Section 5 of [APFC17], which use mesh-based methods running a 20 core CPU under Ubuntu 16.04 with 32 Go RAM. Our results are comparable in terms of errgu(t_k) to the rectangular meshes 1 to 4 in Table 19 therein, which require up to 5 seconds. Those results are also comparable in terms of erru(t_k) to the triangular meshes 1 to 3 in Table 17 therein, which require up to 44 seconds.

We now let $\nu = 0.01$, $T = 10$, and present the results in Figure 4 and Table 3.

[1]The numbers in Table 1 above should be multiplied by 10^3 for comparison with Table 1 in [LG20].



(a) Comparison for $u_1(x_1, \pi)$. (b) Comparison for $u_2(x_1, \pi)$.

Figure 4: Comparison with the exact solution (4.1) by taking $t = T/2$ and $x_2 = \pi$.

k	0	1	2	3	4	5	6	7	8	9	10
$e_0(t_k)$	4.99E-03	4.51E-03	4.01E-03	3.20E-03	3.16E-03	2.28E-03	1.74E-03	1.33E-03	1.56E-03	1.69E-03	—
$e_1(t_k)$	2.54E-03	2.14E-03	1.73E-03	1.27E-03	8.45E-04	5.77E-04	4.96E-04	5.29E-04	8.22E-04	1.63E-03	—
$e(t_k)$	5.55E-03	4.94E-03	4.29E-03	3.39E-03	3.39E-03	2.42E-03	1.84E-03	1.42E-03	1.56E-03	1.88E-03	—
$\mathrm{erru}(t_k)$	6.22E-02	5.55E-02	4.94E-02	4.36E-02	3.83E-02	3.36E-02	2.97E-02	2.72E-02	2.65E-02	2.81E-02	—
$\mathrm{errgu}(t_k)$	4.39E-02	3.86E-02	3.43E-02	3.09E-02	2.85E-02	2.72E-02	2.72E-02	2.87E-02	3.16E-02	3.59E-02	—
$\mathrm{errdivu}(t_k)$	6.28E-02	5.11E-02	4.20E-02	3.48E-02	2.90E-02	2.48E-02	2.33E-02	2.59E-02	3.23E-02	4.16E-02	—
$\mathrm{errp}(t_k)$	—	—	—	—	—	—	—	—	—	—	1.75E-02

Table 3: Error comparison.

Table 3 above is comparable in terms of $\mathrm{errgu}(t_k)$ to the rectangular meshes 1 to 4 in Table 20 in Section 5 of [APFC17], which require up to 10 seconds. Those results are also comparable in terms of $\mathrm{erru}(t_k)$ to the triangular meshes 1 to 3 in Table 18 therein, which require up to one minute.

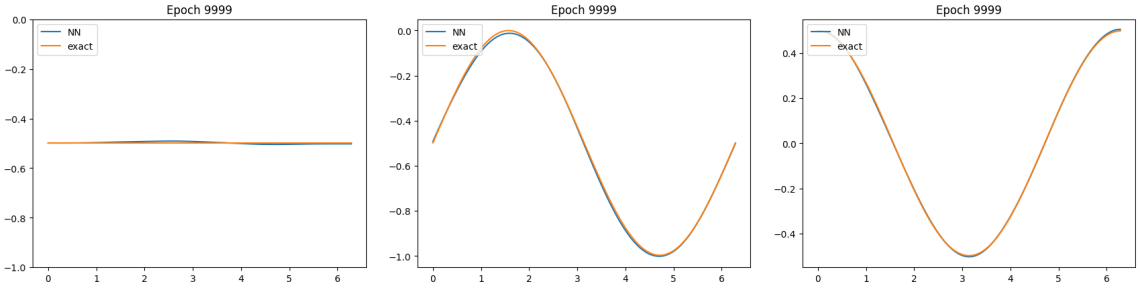
4.2 Arnold-Beltrami-Childress flow

Here, we consider the following 3-dimensional Arnold-Beltrami-Childress [Arn65], [Chi70] flow

$$\begin{cases} u_1(t,x) = (A\sin(x_3) + C\cos(x_2))\,\mathrm{e}^{-\nu(T-t)}, \\ u_2(t,x) = (B\sin(x_1) + A\cos(x_3))\,\mathrm{e}^{-\nu(T-t)}, \\ u_3(t,x) = (C\sin(x_2) + B\cos(x_1))\,\mathrm{e}^{-\nu(T-t)}, \\ u_0(t,x) = -\left(AC\sin(x_3)\cos(x_2) + BA\sin(x_1)\cos(x_3) + CB\sin(x_2)\cos(x_1)\right)\mathrm{e}^{-2\nu(T-t)} + c. \end{cases} \tag{4.2}$$

$x = (x_1, x_2, x_3) \in [0, 2\pi]^3$. We first let $\nu = 0.01$, $A = B = C = 0.5$, $T = 0.7$, $\delta = \pi/45$, which corresponds to a Reynolds numbers in the range $[1, 100]$, and we present the results

in Figure 5 and Table 4.



(a) Comp. for $u_1(x_1, \pi, \pi)$. (b) Comp. for $u_2(x_1, \pi, \pi)$. (c) Comp. for $u_3(x_1, \pi, \pi)$.

Figure 5: Comparison with the exact solution (4.2) by taking $t = T/2$ and $x_2 = x_3 = \pi$.

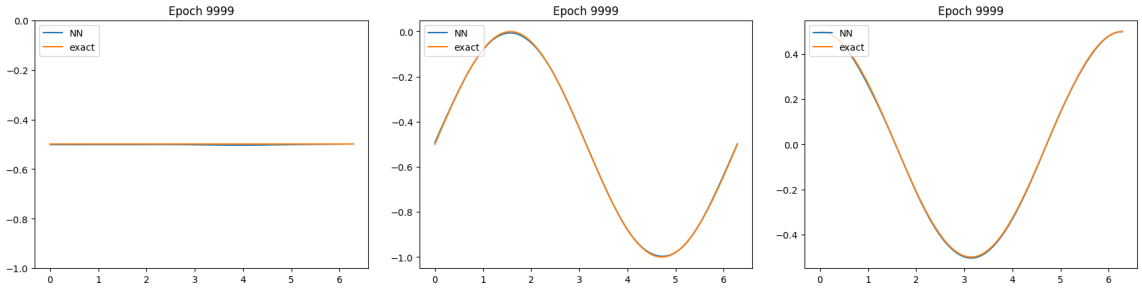
k	0	1	2	3	4	5	6	7	8	9	10
$e_0(t_k)$	9.68E-04	4.27E-04	5.92E-04	5.29E-04	6.02E-04	5.42E-04	5.61E-04	4.11E-04	6.28E-04	5.96E-04	—
$e_1(t_k)$	2.20E-03	1.01E-03	8.25E-04	9.41E-04	7.28E-04	7.18E-04	7.90E-04	8.11E-04	6.64E-04	7.05E-04	—
$e_2(t_k)$	1.09E-03	5.06E-04	4.61E-04	6.83E-04	9.28E-04	4.70E-04	6.65E-04	8.29E-04	4.90E-04	5.68E-04	—
$e(t_k)$	2.97E-03	1.34E-03	1.12E-03	1.50E-03	1.20E-03	1.05E-03	1.04E-03	1.10E-03	6.99E-04	9.08E-04	—
$\text{erru}(t_k)$	1.64E-02	1.14E-02	1.18E-02	1.22E-02	1.32E-02	1.28E-02	1.24E-02	1.14E-02	1.11E-02	1.09E-02	—
$\text{errgu}(t_k)$	3.72E-02	3.16E-02	3.19E-02	3.34E-02	3.59E-02	3.52E-02	3.41E-02	3.15E-02	3.01E-02	3.02E-02	—
$\text{errdivu}(t_k)$	8.74E-02	5.42E-02	5.30E-02	5.84E-02	6.91E-02	7.12E-02	6.02E-02	5.45E-02	5.15E-02	5.05E-02	—
$\text{errp}(t_k)$	—	—	—	—	—	—	—	—	—	—	1.93E-02

Table 4: Error comparison.

Our simulation runtime on the full grid $[0, T] \times \Omega$ is approximately 60 minutes for the Arnold-Beltrami-Childress flow after 30 minutes of pre-computation for the training of the terminal condition $p(T, x)$. Table 4 can be compared[2] to Table 5 in [LG20] where a single time step by BSDEs and Monte Carlo took approximately 20 hours. Our results have a significantly lower runtime, and are at least one order of magnitude more accurate than [LG20]. In addition, the neural network approach yields a functional estimate on $[0, 2\pi]^3 \times [0, T]$ instead of estimating the solution at discrete time instants.

Finally, we let $\nu = 10^{-4}$, which corresponds to a Reynolds number of order 10,000, and we present the results in Figure 6 and Table 5.

[2]The numbers in Table 4 above should be multiplied by 10^2 for comparison with Table 5 in [LG20].



(a) Comp.for $u_1(x_1, \pi, \pi)$. (b) Comp. for $u_2(x_1, \pi, \pi)$. (c) Comp. for $u_3(x_1, \pi, \pi)$.

Figure 6: Comparison with the exact solution (4.2) by taking $t = T/2$ and $x_2 = x_3 = \pi$.

k	0	1	2	3	4	5	6	7	8	9	10
$e_0(t_k)$	2.86E-04	1.46E-04	1.29E-04	1.15E-04	1.75E-04	1.47E-04	1.73E-04	9.17E-05	1.31E-04	1.15E-04	—
$e_1(t_k)$	3.22E-04	1.58E-04	1.56E-04	1.67E-04	1.27E-04	1.37E-04	1.14E-04	1.11E-04	1.55E-04	2.12E-04	—
$e_2(t_k)$	2.68E-04	1.72E-04	2.48E-04	2.09E-04	1.72E-04	1.73E-04	1.71E-04	1.42E-04	1.15E-04	1.52E-04	—
$e(t_k)$	4.05E-04	2.46E-04	2.76E-04	2.20E-04	2.64E-04	2.55E-04	2.51E-04	1.58E-04	1.93E-04	2.95E-04	—
$\text{erru}(t_k)$	8.08E-03	6.03E-03	5.90E-03	6.15E-03	6.41E-03	6.50E-03	5.88E-03	5.64E-03	6.48E-03	6.53E-03	—
$\text{errgu}(t_k)$	2.41E-02	2.18E-02	2.13E-02	2.13E-02	2.14E-02	2.20E-02	2.07E-02	2.01E-02	1.98E-02	2.00E-02	—
$\text{errdivu}(t_k)$	4.11E-02	2.49E-02	2.24E-02	2.38E-02	2.88E-02	3.09E-02	2.48E-02	2.30E-02	2.19E-02	2.15E-02	—
$\text{errp}(t_k)$	—	—	—	—	—	—	—	—	—	—	1.93E-02

Table 5: Error comparison.

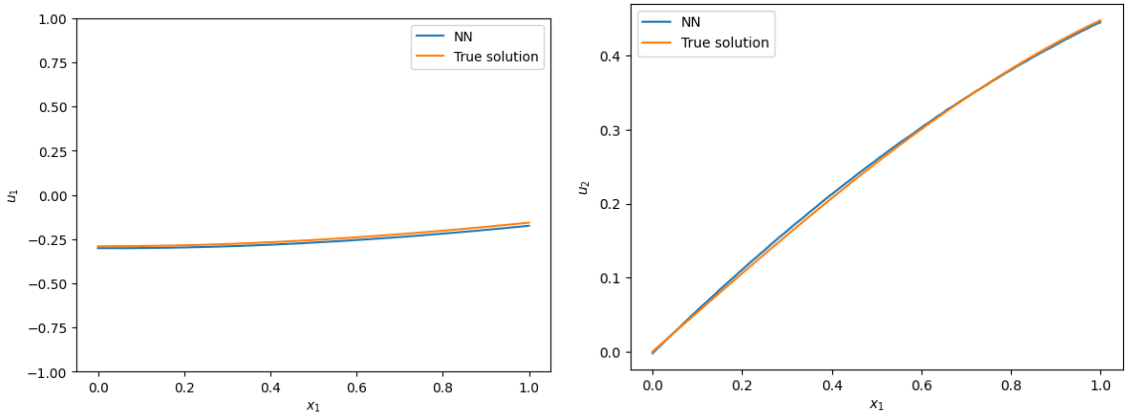
4.3 Comparison with the deep Galerkin method (DGM)

In this section, we compare the output of our method applied to the Taylor-Green vortex to that of the deep Galerkin method which has been developed in [SS18] using neural networks. In the following simulations we take $t = 0$, $T = 1/4$, and use the same number of neural network epochs as our deep branching (DB) algorithm, i.e. 20,000 epochs, and the computation times are comparable, as seen in Table 6. Note that the pre-computation of $p(T, x)$ is part of the terminal boundary condition, and can be re-used for a different equation.

	Deep Branching (Taylor-Green)	DGM (Taylor-Green)
$p(T, x)$	1200s	2400s
$u(t, x)$	1300s	

Table 6: Comparison of computation times in seconds.

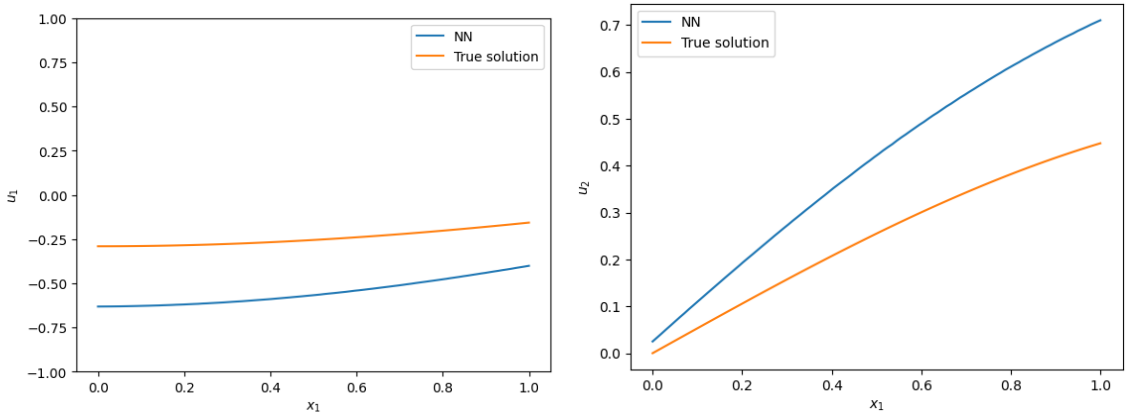
In Figure 7, we start with boundary conditions given by (4.1) on the space-time domain $[0, 1]^2 \times [0, T]$ used in [LYZD22].



(a) Comparison for $u_1(x_1, 1)$. (b) Comparison for $u_2(x_1, 1)$.

Figure 7: Comparison of DGM and (4.1) with space-time boundary condition and $x_2 = 1$.

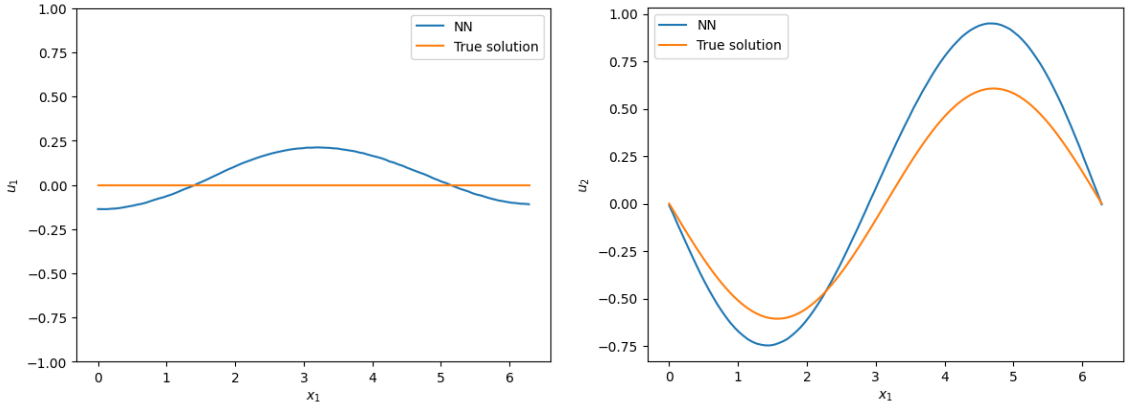
Next, in Figure 8 we only use a spatial boundary condition on $[0, 1]^2$ at the terminal time T, and we observe that accuracy of the output is lost.



(a) Comparison for $u_1(x_1, 1)$. (b) Comparison for $u_2(x_1, 1)$.

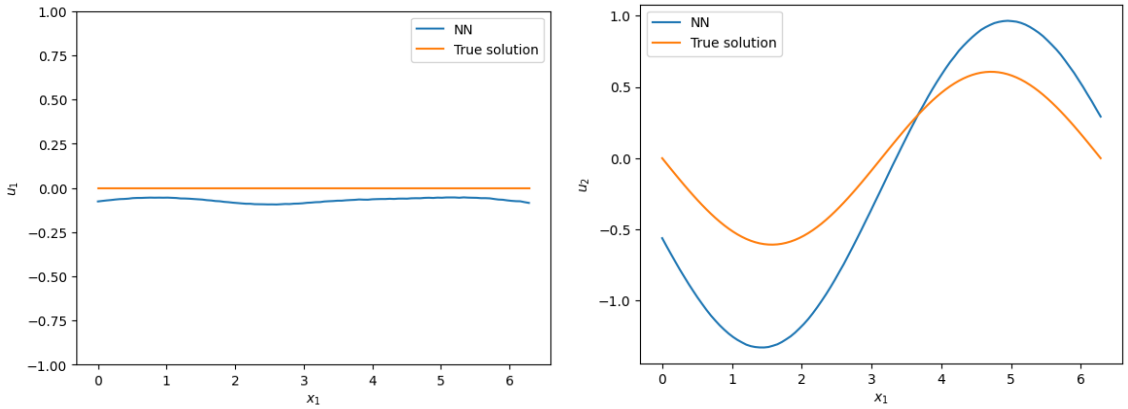
Figure 8: Comparison of DGM and (4.1) with terminal boundary condition and $x_2 = 1$.

To conclude our assessment of the DGM method to the Taylor-Green vortex, in Figures 9 and 10 we extend the domain $[0, 1]^2$ used in [LYZD22] to $[0, 2\pi]^2$ as in Section 4.1, and we observe that accuracy is lost in this case, for both the space-time boundary condition on $[0, 2\pi]^2 \times [0, T]$ and the terminal boundary condition on $[0, 2\pi]^2$ at time T.



(a) Comparison for $u_1(x_1, 2\pi)$. (b) Comparison for $u_2(x_1, 2\pi)$.

Figure 9: Comparison of DGM and (4.1) with space-time boundary condition and $x_2 = 2\pi$.



(a) Comparison for $u_1(x_1, 2\pi)$. (b) Comparison for $u_2(x_1, 2\pi)$.

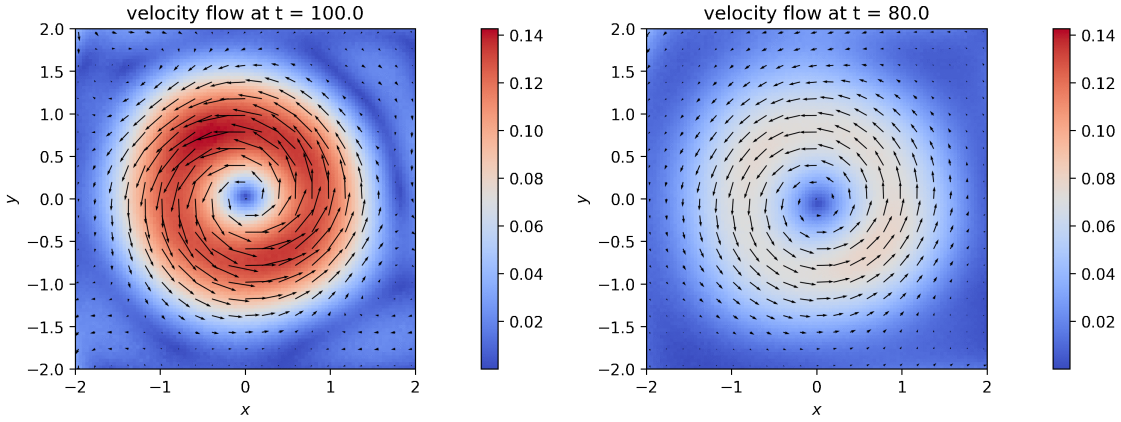
Figure 10: Comparison of DGM and (4.1) with terminal boundary condition and $x_2 = 2\pi$.

4.4 Rotating flows

In this section we propose two other examples of dimensional flows on $\mathbb{R}^2$ that can be solved with vanishing boundary conditions at infinity, by taking a terminal condition ϕ of the form

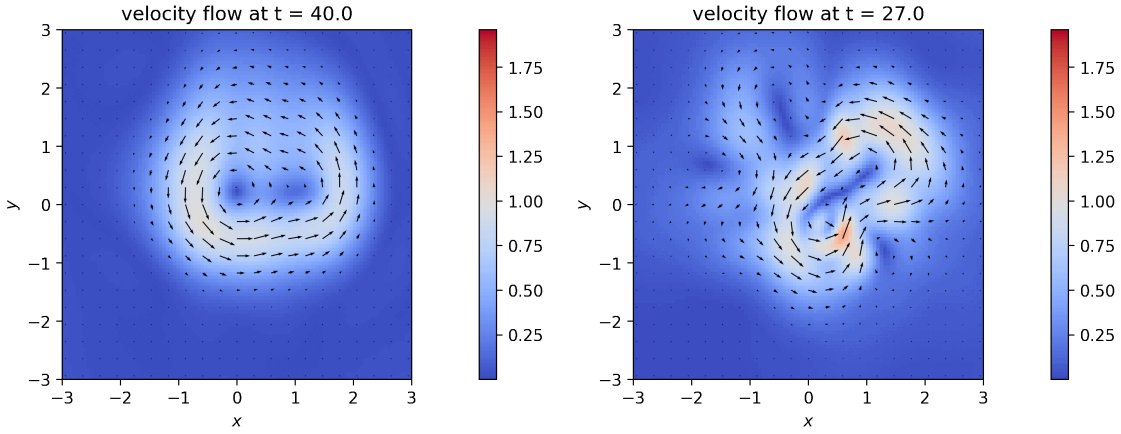
$$\begin{cases} \phi_1(x_1, x_2) = \dfrac{f'(x_2)}{f(x_2)} \exp\left(-\dfrac{g(x_1)}{f(x_2)}\right) \\ \phi_2(x_1, x_2) = \dfrac{g'(x_2)}{g(x_2)} \exp\left(-\dfrac{g(x_1)}{f(x_2)}\right), \end{cases}$$

which satisfies the divergence-free condition $\operatorname{div} \phi(x_1, x_2) = 0$, $(x_1, x_2) \in \mathbb{R}^2$, with a vanishing spatial condition at infinity. This yields two-dimensional quiver velocity plots at different times with $T = 100$ in Figures 11 and 12 below.



(a) $t = 100$. (b) $t = 80$.

Figure 11: Case $f(x_2) = 1 + x_2^2$, $g(x_1) = 1/(1 + x_1^2)$.



(a) $t = 40$. (b) $t = 27$.

Figure 12: Case $f(x_2) = (2 + \sin(x_2))/(1 + x_2^2)$, $g(x_1) = \mathrm{e}^{x_1^2}/(2 + x_1^3 + x_1^4)$.

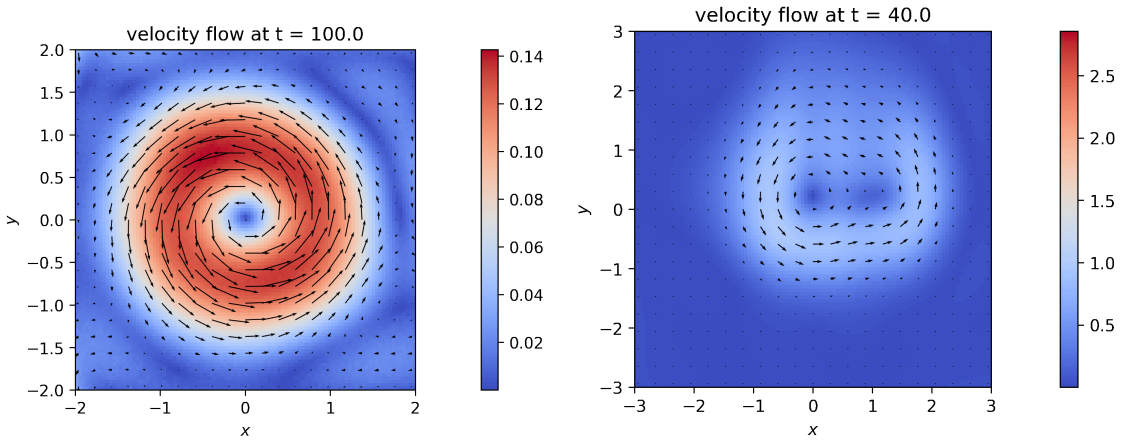


Figure 13: Rotating flows.

A Branching solution of PDE systems

In this section we present the extension of the arguments of [NPP23], [NPP22] to systems of partial differential equations, which leads to the probabilistic representation (2.10). The following proof uses the notation of Algorithm 1.

Proof of Proposition 2.3. (i) Consider $c \in \mathcal{C}$ a code of the form $c = (\partial_\lambda f)^*$. From the Faà di Bruno formula (1.2) applied to the function f_{β_r}, for $g \in \mathcal{C}^\infty(\mathbb{R}^d)$ we have

$$
\begin{aligned}
&\partial_t g^*(u) + \nu \Delta g^*(u) \\
&= \sum_{w=1}^{n} \partial_{\bar\alpha^w} \left(\partial_t u_{\beta_w} + \nu \Delta u_{\beta_w}\right) (\partial_{\mathbf{1}_w} g)^* + \nu \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{d} \left(\partial_{\bar\alpha^i + \mathbf{1}_k} u_{\beta_i}\right) \left(\partial_{\bar\alpha^j + \mathbf{1}_k} u_{\beta_j}\right) (\partial_{\mathbf{1}_i + \mathbf{1}_j} g)^* \\
&= \sum_{w=1}^{q} (\partial_{\mathbf{1}_w} g)^* \partial_{\bar\alpha^w} \left(\partial_t + \nu\Delta\right) u_0 - \sum_{w=q+1}^{n} (\partial_{\mathbf{1}_w} g)^* \partial_{\bar\alpha^w} (f^*_{\beta_w}(u)) \\
&\qquad\qquad + \nu \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{d} \left(\partial_{\bar\alpha^i + \mathbf{1}_k} u_{\beta_i}\right) \left(\partial_{\bar\alpha^j + \mathbf{1}_k} u_{\beta_j}\right) (\partial_{\mathbf{1}_i + \mathbf{1}_j} g)^* \qquad \text{(A.1)} \\
&= - \sum_{w=q+1}^{n} (\partial_{\mathbf{1}_w} g)^* \left(\prod_{i=1}^{d} \alpha_i^w !\right) \sum_{\substack{1 \le \lambda_1 + \cdots + \lambda_n \le |\bar\alpha^w| \\ 1 \le s \le |\bar\alpha^w|}} (\partial_\lambda f_{\beta_w})^* \sum_{\substack{1 \le |k_1|, \ldots, |k_s|, \ 0 \prec l^1 \prec \cdots \prec l^s \\ k_1^i + \cdots + k_s^i = \lambda_i, \ i=1,\ldots,n \\ |k_1| l_j^1 + \cdots + |k_s| l_j^s = \alpha_j^w, \ j=1,\ldots,d}} \prod_{\substack{1 \le i \le n \\ 1 \le r \le s}} \frac{(\partial_{l^r + \bar\alpha^i} u_{\beta_i})^{k_r^i}}{k_r^i! \, (l_1^r! \cdots l_d^r!)^{k_r^i}} \\
&\qquad\qquad + \sum_{r=1}^{q} (\partial_{\mathbf{1}_r} g)^* \partial_{\bar\alpha^r} \left(\partial_t + \nu\Delta\right) u_0 + \nu \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{d} \left(\partial_{\bar\alpha^i + \mathbf{1}_k} u_{\beta_i}\right) \left(\partial_{\bar\alpha^j + \mathbf{1}_k} u_{\beta_j}\right) (\partial_{\mathbf{1}_i + \mathbf{1}_j} g)^*.
\end{aligned}
$$

Rewriting the above equation in integral form yields

$$
\begin{aligned}
g^*(u)(t,x) &= \int_{\mathbb{R}^d} \varphi_{2\nu}(T-t, y-x) g^*(\phi)(y) dy \qquad \text{(A.2)} \\
&+ \int_t^T \int_{\mathbb{R}^d} \varphi_{2\nu}(s-t, y-x) \Bigg(-\nu \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{d} \left(\partial_{\bar\alpha^i + \mathbf{1}_k} u_{\beta_i}(s,y)\right) \left(\partial_{\bar\alpha^j + \mathbf{1}_k} u_{\beta_j}(s,y)\right) (\partial_{\mathbf{1}_i + \mathbf{1}_j} g)^* \\
&- \sum_{r=1}^{q} (\partial_{\mathbf{1}_r} g)^* \partial_{\alpha^r} \left(\partial_t + \nu\Delta\right) u_0(s,y) \\
&+ \sum_{w=q+1}^{n} (\partial_{\mathbf{1}_w} g)^* \left(\prod_{i=1}^{d} \alpha_i^w !\right) \sum_{\substack{1 \le \lambda_1 + \cdots + \lambda_n \le |\bar\alpha^w| \\ 1 \le s \le |\bar\alpha^w|}} (\partial_\lambda f_{\beta_w})^* \sum_{\substack{1 \le |k_1|, \ldots, |k_s|, \ 0 \prec l^1 \prec \cdots \prec l^s \\ k_1^i + \cdots + k_s^i = \lambda_i, \ i=1,\ldots,n \\ |k_1| l_j^1 + \cdots + |k_s| l_j^s = \bar\alpha_j^w, \ j=1,\ldots,d}} \prod_{\substack{1 \le i \le n \\ 1 \le r \le s}} \frac{\left(\partial_{l^r + \bar\alpha^i} u_{\beta_i}(s,y)\right)^{k_r^i}}{k_r^i! \, (l_1^r! \cdots l_d^r!)^{k_r^i}} \Bigg) dy ds,
\end{aligned}
$$

which shows the equation

$$c(u)(t,x)=\int_{\mathbb{R}^d}\varphi_{2\nu}(T-t,y-x)c(u)(T,y)dy+\sum_{Z\in\mathcal{M}(c)}\int_t^T\int_{\mathbb{R}^d}\varphi_{2\nu}(s-t,y-x)\prod_{z\in Z}z(u)(s,y)dyds, \tag{A.3}$$

$(t,x)\in[0,T]\times\mathbb{R}$, for any code $c\in\mathcal{C}$ of the form $c=(\partial_\lambda f)^*$. Also, (A.3) holds directly from (2.2) for the code $c=\mathrm{Id}_i$, $i=1,\dots,d$.

(ii) For $c=(\partial_\mu,0)$, by (2.2) we have

$$\begin{aligned}\partial_\mu u_0(t,x)&=\frac{\Gamma(d/2)}{2\pi^{d/2}}\int_{\mathbb{R}^d}\frac{N(y)}{|y|^d}\partial_\mu f_0\big(\partial_{\bar\alpha^{q+1}}u_{\beta_{q+1}}(t,x+y),\dots,\partial_{\bar\alpha^n}u_{\beta_n}(t,x+y)\big)dy\\&=\int_{\mathbb{R}^d}N(y)\int_0^\infty\frac{(2\pi s)^{-d/2}}{2s}\mathrm{e}^{-|y|^2/(2s)}ds\partial_\mu f_0\big(\partial_{\bar\alpha^{q+1}}u_{\beta_{q+1}}(t,x+y),\dots,\partial_{\bar\alpha^n}u_{\beta_n}(t,x+y)\big)dy\\&=\int_0^\infty\int_{\mathbb{R}^d}\varphi_1(s,y)\frac{N(y)}{2s}\partial_\mu f_0\big(\partial_{\bar\alpha^{q+1}}u_{\beta_{q+1}}(t,x+y),\dots,\partial_{\bar\alpha^n}u_{\beta_n}(t,x+y)\big)dyds\\&=\sum_{Z\in\mathcal{M}((\partial_\mu,0))}\int_0^\infty\int_{\mathbb{R}^d}\varphi_1(s,y)\frac{N(y)}{2s}\prod_{z\in Z}z(u)(t,x+y)dyds,\end{aligned} \tag{A.4}$$

which shows that

$$c(u)(t,x)=\sum_{Z\in\mathcal{M}(c)}\int_0^\infty\int_{\mathbb{R}^d}\varphi_1(s,y)\frac{N(y)}{2s}\prod_{z\in Z}z(u)(t,x+y)dyds, \tag{A.5}$$

for the code $c=(\partial_\mu,0)$. Also, (A.5) holds directly for the code $c=\mathrm{Id}_0$ from (2.2).

(iii) Next, for the code $c=(\partial_\mu,-1)$, from (A.1) applied to $g=f_0$, we have

$$\begin{aligned}&\partial_\mu(\partial_t+\nu\Delta)u_0(t,x)\\&=\int_0^\infty\int_{\mathbb{R}^d}\varphi_1(s,y)\frac{N(y)}{2s}\partial_\mu(\partial_t+\nu\Delta)f_0\big(\partial_{\bar\alpha^{q+1}}u_{\beta_{q+1}}(t,x+y),\dots,\partial_{\bar\alpha^n}u_{\beta_n}(t,x+y)\big)dyds\\&=\int_0^\infty\int_{\mathbb{R}^d}\varphi_1(s,y)\frac{N(y)}{2s}\\&\qquad\partial_\mu\left(\nu\sum_{i=q+1}^n\sum_{j=q+1}^n\sum_{k=1}^d\left(\partial_{\bar\alpha^i+\mathbf{1}_k}u_{\beta_i}\right)\left(\partial_{\bar\alpha^j+\mathbf{1}_k}u_{\beta_j}\right)\left(\partial_{\mathbf{1}_i+\mathbf{1}_j}f_0\right)^*-\sum_{i=q+1}^n\left(\partial_{\mathbf{1}_i}f_0\right)^*\partial_{\bar\alpha^i}f^*_{\beta_i}(u)\right)dyds\\&=\int_0^\infty\int_{\mathbb{R}^d}\varphi_1(s,y)\frac{N(y)}{2s}\\&\qquad\left(\sum_{\substack{0\le\gamma_i\le\ell_i\le\mu_i\\1\le i\le d}}\left(\nu\prod_{r=1}^d\binom{\mu_r}{\ell_r}\binom{\ell_r}{\gamma_r}\right)\sum_{i=q+1}^n\sum_{j=q+1}^n\sum_{k=1}^d\left(\partial_{\mu-\ell+\bar\alpha^i+\mathbf{1}_k}u_{\beta_i}\right)\left(\partial_{\ell-\gamma+\bar\alpha^j+\mathbf{1}_k}u_{\beta_j}\right)\partial_\gamma\left(\partial_{\mathbf{1}_i+\mathbf{1}_j}f_0\right)^*\right.\end{aligned}$$

$$
\left. - \sum_{\substack{0\le \ell_i \le \mu_i \\ 1\le i\le d}} \left(\prod_{r=1}^{d} \binom{\mu_r}{\ell_r} \right) \sum_{i=q+1}^{n} \partial_\ell (\partial_{\mathbf{1}_i} f_0)^* \partial_{\mu-\ell+\bar{\alpha}^i} f^*_{\beta_i}(u) \right) dyds
$$

$$
= \int_0^\infty \int_{\mathbb{R}^d} \varphi_1(s,y) \frac{N(y)}{2s}
$$

$$
\left(\sum_{\substack{0\le \gamma_i \le \ell_i \le \mu_i \\ 1\le i\le d}} \left(\nu \prod_{r=1}^{d} \binom{\mu_r}{\ell_r} \binom{\ell_r}{\gamma_r} \right) \sum_{i=q+1}^{n} \sum_{j=q+1}^{n} \sum_{k=1}^{d} \left(\partial_{\mu-\ell+\bar{\alpha}^i+\mathbf{1}_k} u_{\beta_i} \right) \left(\partial_{\ell-\gamma+\bar{\alpha}^j+\mathbf{1}_k} u_{\beta_j} \right) \partial_\gamma (\partial_{\mathbf{1}_i+\mathbf{1}_j} f_0)^* \right.
$$

$$
\left. - \sum_{\substack{0\le \ell_i \le \mu_i \\ 1\le i\le d}} \left(\prod_{w=1}^{d} \binom{\mu_w}{\ell_w} \ell_w! \right) \sum_{\substack{q<i\le n \\ 1\le \lambda_1+\cdots+\lambda_n \le |\ell| \\ 1\le s\le |\ell|}} \partial_{\mu-\ell+\bar{\alpha}^i} f^*_{\beta_i}(u) (\partial_{\lambda+\mathbf{1}_i} f_0)^*(u) \sum_{\substack{1\le |k_1|,\ldots,|k_s|,\ 0\prec l^1 \prec \cdots \prec l^s \\ k_1^i+\cdots+k_s^i=\lambda_i,\ i=1,\ldots,n \\ |k_1| l_j^1 + \cdots + |k_s| l_j^s = \ell_j,\ j=1,\ldots,d}} \prod_{\substack{1\le i\le n \\ 1\le r\le s}} \frac{(\partial_{l^r+\bar{\alpha}^i} u_{\beta_i})^{k_r^i}}{k_r^i! \, (l_1^r! \cdots l_d^r!)^{k_r^i}} \right) dyds \tag{A.6}
$$

$$
= \sum_{Z\in\mathcal{M}((\partial_\mu,-1))} \int_0^\infty \int_{\mathbb{R}^d} \varphi_1(s,y) \frac{N(y)}{2s} \prod_{z\in Z} z(u)(t,x+y) dyds, \tag{A.7}
$$

according to the definition of $\mathcal{M}((\partial_\mu,-1))$. Hence, we have shown that

$$
c(u)(t,x) = \sum_{Z\in\mathcal{M}(c)} \int_0^\infty \int_{\mathbb{R}^d} \varphi_1(s,y) \frac{N(y)}{2s} \prod_{z\in Z} z(u)(t,x+y) dyds \tag{A.8}
$$

for the code $c = (\partial_\mu, -1)$.

(iv) By the Faà di Bruno formula (1.2), Equation (A.3) is also satisfied by $c = (\partial_\mu, i)$ for $i = 1, \ldots, d$, since $\mathcal{M}\left((\partial_\mu, i)\right) = \mathrm{fdb}(\mu, f_i, \emptyset)$.

For any $c \in \mathcal{C}$, we now let

$$
u_c(t,x) := \mathbb{E}[\mathcal{H}(t,x,c)], \qquad (t,x) \in [0,T] \times \mathbb{R}^d.
$$

(v) Starting from a code of the form $c = (\partial_\lambda f)^*$ or $c = (\partial_\mu, i)$ for $i = 1, \ldots, d$, we draw a sample of I_c uniformly in $\mathcal{M}(c)$. As each code in the tuple I_c yields a new branch at time τ, we have

$$
u_c(t,x) = \mathbb{E}\big[\mathcal{H}(t,x,c)\mathbb{1}_{\{t+\tau>T\}} + \mathcal{H}(t,x,c)\mathbb{1}_{\{t+\tau\le T\}}\big]
$$

$$
= \mathbb{E}\left[\frac{c(u)(T, x+W_{2\nu(T-t)})}{\overline{F}(T-t)} \mathbb{1}_{\{t+\tau>T\}} + \mathbb{1}_{\{t+\tau\le T\}} \sum_{Z\in\mathcal{M}(c)} \mathbf{1}_{\{I_c=Z\}} r_c \frac{\prod_{z\in Z} u_z(t+\tau, x+W_{2\nu\tau})}{\rho(\tau)} \right]
$$

$$= \int_{-\infty}^{\infty} \varphi_{2\nu}(T-t, y-x) c(u)(T,y) dy + \sum_{Z \in \mathcal{M}(c)} \int_t^T \int_{-\infty}^{\infty} \varphi_{2\nu}(s-t, y-x) \prod_{z \in Z} u_z(s,y) dy ds, \tag{A.9}$$

which yields the same system of equations as (A.3).

(*vi*) Similarly, starting a code of the form $c = (\partial_\mu, 0)$ or $c = (\partial_\mu, -1)$ we draw a sample of I_c uniformly in $\mathcal{M}(c)$ with probability $1/r_c$, where r_c is the size of $\mathcal{M}(c)$. As each code in the tuple I_c yields a new branch at time $\widetilde{\tau}$, we obtain

$$\begin{aligned} u_c(t,x) &= \mathbb{E}[\mathcal{H}(t,x,c)] \\ &= \mathbb{E}\left[\frac{r_c N(W_{\widetilde{\tau}})}{2\widetilde{\tau}\widetilde{\rho}(\widetilde{\tau})} \sum_{Z \in \mathcal{M}(c)} \mathbf{1}_{\{I_c = Z\}} \prod_{z \in Z} u_z(t, x + W_{\widetilde{\tau}}) \right] \\ &= \sum_{Z \in \mathcal{M}(c)} \int_0^\infty \int_{\mathbb{R}^d} \varphi_1(s,y) \frac{N(y)}{2s} \prod_{z \in Z} z(u)(t, x+y) dy ds, \end{aligned} \tag{A.10}$$

which coincides with (A.5) or (A.8), respectively for $c = (\partial_\mu, 0)$ and $c = (\partial_\mu, -1)$.

(*vii*) From (A.9)-(A.10) and (A.3)-(A.5)-(A.8) we conclude that for any code $c \in \mathcal{C}$, $u_c(t,x)$ and $c(u)(t,x)$ satisfy the same system of equations (2.9). As by assumption the system (2.9) has a unique solution we conclude that $(c(u))_{c \in \mathcal{C}} = (u_c)_{c \in \mathcal{C}}$, and therefore

$$u_c(t,x) = \mathbb{E}[\mathcal{H}(t,x,c)] = c(u)(t,x), \qquad (t,x) \in [0,T] \times \mathbb{R}, \quad c \in \mathcal{C}.$$

In particular, for $c = \mathrm{Id}_i$ this yields

$$u_i(t,x) = \mathrm{Id}_i(u)(t,x) = \mathbb{E}[\mathcal{H}(t,x,\mathrm{Id}_i)], \qquad (t,x) \in [0,T] \times \mathbb{R}, \quad i = 0, 1, \ldots, d,$$

which is (2.10). □

References

[AB10] S. Albeverio and Ya. Belopolskaya. Generalized solutions of the Cauchy problem for the Navier-Stokes system and diffusion processes. *Cubo*, 12(2):77–96, 2010.

[APFC17] P.-E. Angeli, M.-A. Puscas, G. Fauchet, and A. Cartalade. FVCA8 Benchmark for the Stokes and Navier–Stokes equations with the TrioCFD code-benchmark session. In *FVCA 2017: Finite Volumes for Complex Applications VIII - Methods and Theoretical Aspects*, volume 199 of *Springer Proceedings in Mathematics & Statistics*, pages 181–202. Springer Verlag, 2017.

[Arn65] V. Arnol′d. Sur la topologie des écoulements stationnaires des fluides parfaits. *C. R. Acad. Sci. Paris*, 261:17–20, 1965.

[Bor17] A.N. Borodin. *Stochastic processes*. Probability and its Applications. Birkhäuser/Springer, Cham, 2017. Original Russian edition published by LAN Publishing, St. Petersburg, 2013.

[CC07] F. Cipriano and A.B. Cruzeiro. Navier-Stokes equation and diffusions on the group of homeomorphisms of the torus. *Comm. Math. Phys.*, 275:255–269, 2007.

[Chi70] S. Childress. New solutions of the kinematic dynamo problem. *J. Math. Phys.*, 11(10):3063–3076, 1970.

[CS96] G.M. Constantine and T.H. Savits. A multivariate Faa di Bruno formula with applications. *Trans. Amer. Math. Soc.*, 348(2):503–520, 1996.

[CS09] A.B. Cruzeiro and E. Shamarova. Navier-Stokes equations and forward-backward SDEs on the group of diffeomorphisms of a torus. *Stochastic Process. Appl.*, 119(12):4034–4060, 2009.

[CSTV07] P. Cheridito, H.M. Soner, N. Touzi, and N. Victoir. Second-order backward stochastic differential equations and fully nonlinear parabolic PDEs. *Comm. Pure Appl. Math.*, 60(7):1081–1110, 2007.

[DQT15] F. Delbaen, J. Qiu, and S. Tang. Forward-backward stochastic differential systems associated to Navier-Stokes equations in the whole space. *Stochastic Process. Appl.*, 125(7):2516–2561, 2015.

[FTW11] A. Fahim, N. Touzi, and X. Warin. A probabilistic numerical method for fully nonlinear parabolic PDEs. *Ann. Appl. Probab.*, 21(4):1322–1364, 2011.

[GZZ15] W. Guo, J. Zhang, and J. Zhuo. A monotone scheme for high-dimensional fully nonlinear PDEs. *Ann. Appl. Probab.*, 25(3):1540–1580, 2015.

[HJE18] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[HL12] P. Henry-Labordère. Counterparty risk valuation: a marked branching diffusion approach. Preprint arXiv:1203.2369, 2012.

[HLOT+19] P. Henry-Labordère, N. Oudjane, X. Tan, N. Touzi, and X. Warin. Branching diffusion representation of semilinear PDEs and Monte Carlo approximation. *Ann. Inst. H. Poincaré Probab. Statist.*, 55(1):184–210, 2019.

[HLZ20] S. Huang, G. Liang, and T. Zariphopoulou. An approximation scheme for semilinear parabolic PDEs with convex and coercive Hamiltonians. *SIAM J. Control Optim.*, 58(1):165–191, 2020.

[Hor91] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[HZRS16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[INW69] N. Ikeda, M. Nagasawa, and S. Watanabe. Branching Markov processes I, II, III. *J. Math. Kyoto Univ.*, 8-9:233–278, 365–410, 95–160, 1968-1969.

[IS15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[KB14] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Preprint arXiv:1412.6980*, 2014.

[LG20] A. Lejay and H.M. González. A forward-backward probabilistic algorithm for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 420:109689, 19, 2020.

[LS97] Y. Le Jan and A. S. Sznitman. Stochastic cascades and 3-dimensional Navier-Stokes equations. *Probab. Theory Related Fields*, 109(3):343–366, 1997.

[LYZD22] J. Li, J. Yue, W. Zhang, and W. Duan. The deep learning Galerkin method for the general Stokes equations. *J. Sci. Comput.*, 93(1):Paper No. 5, 20, 2022.

[Mat21] M. Matsumoto. Application of Deep Galerkin Method to solve compressible Navier-Stokes equations. *Trans. Japan Soc. Aero. Space Sci.*, 64(6):348–357, 2021.

[MB02] A.J. Majda and A.L. Bertozzi. *Vorticity and incompressible flow*, volume 27 of *Cambridge Texts in Applied Mathematics*. Cambridge University Press, Cambridge, 2002.

[McK75] H.P. McKean. Application of Brownian motion to the equation of Kolmogorov-Petrovskii-Piskunov. *Comm. Pure Appl. Math.*, 28(3):323–331, 1975.

[NPP22] J.Y. Nguwi, G. Penent, and N. Privault. A deep branching solver for fully nonlinear partial differential equations. Preprint arXiv:2203.03234, 17 pages, 2022.

[NPP23] J.Y. Nguwi, G. Penent, and N. Privault. A fully nonlinear Feynman-Kac formula with derivatives of arbitrary orders. *Journal of Evolution Equations*, 23(22), 2023. https://doi.org/10.1007/s00028-023-00873-3.

[PP92] É. Pardoux and S. Peng. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In *Stochastic partial differential equations and their applications (Charlotte, NC, 1991)*, volume 176 of *Lecture Notes in Control and Inform. Sci.*, pages 200–217. Springer, Berlin, 1992.

[PP22] G. Penent and N. Privault. Numerical evaluation of ODE solutions by Monte Carlo enumeration of Butcher series. *BIT Numerical Mathematics*, 62:1921–1944, 2022.

[RPK19] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[Sko64] A.V. Skorokhod. Branching diffusion processes. *Teor. Verojatnost. i. Primenen.*, 9:492–497, 1964.

[SS18] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[STZ12] H.M. Soner, N. Touzi, and J. Zhang. Wellposedness of second order backward SDEs. *Probab. Theory Related Fields*, 153(1-2):149–190, 2012.

[Tan13] X. Tan. A splitting method for fully nonlinear degenerate parabolic PDEs. *Electron. J. Probab.*, 18:no. 15, 24, 2013.

[TG37] G.I. Taylor and A.E. Green. Mechanism of the production of small eddies from large ones. *Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci.*, 158(895):499–521, 1937.