# Numerical solution of the modified and non-Newtonian Burgers equations by stochastic coded trees

Jiang Yu Nguwi* Nicolas Privault†

Division of Mathematical Sciences

School of Physical and Mathematical Sciences

Nanyang Technological University

21 Nanyang Link, Singapore 637371

December 13, 2023

**Abstract**

We present the numerical application of a meshfree algorithm for the solution of fully nonlinear PDEs by Monte Carlo simulation using branching diffusion trees coded by the nonlinearities appearing in the equation. This algorithm is applied to the numerical solution of modified and non-Newtonian Burgers equations, and to a problem with boundary conditions in fluid dynamics, by the computation of a Poiseuille flow. Our implementation uses neural networks that yield a functional space-time domain estimation, and includes numerical comparisons with the deep Galerkin (DGM) and deep backward stochastic differential equation (BSDE) methods.

*Keywords*: Fully nonlinear PDEs, Monte Carlo method, branching process, random tree, Burgers equation, modified Burgers equation, non-Newtonian Burgers equation, Poiseuille flow, power law, deep neural network.

*Mathematics Subject Classification (2020):* 35G20, 60H30, 60J85, 76M35, 65C05.

## 1 Introduction

This paper presents an application of the stochastic branching method of [NPP23b] to the numerical solution of nonlinear partial differential equations (PDEs) such as the modified

*nguw0003@e.ntu.edu.sg

†nprivault@ntu.edu.sg

and non-Newtonian Burgers equation, and to a fluid dynamics boundary problem using a Poiseuille flow.

Standard numerical schemes for the solution of partial differential equations (PDEs) by e.g. finite differences or finite elements, which generally suffer from the curse of dimensionality. Deep learning methods for the solution of high dimensional PDEs have been introduced in [SS18] using the deep Galerkin method (DGM), see [LZCC22] for recent improvements. Probabilistic schemes, also provide a promising direction to overcome the curse of dimensionality. Probabilistic representations for the solutions of nonlinear PDEs with first and second order gradient nonlinearities can also be obtained using backward stochastic differential equation (BSDE), see [Pen91], [PP92], [CSTV07], and [HJE17, HJE18, BEJ19] for deep learning implementations, see [CKSY23] for a recent survey.

Stochastic branching diffusion mechanisms [Sko64], [INW69], represent an alternative to the DGM and BSDE methods, see [HL12, HLTT14, HLOT$^+$19]. In [NPP23b], a stochastic branching method that carries information on (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. In this paper, we present a deep learning implementation of the method of [NPP23b] which relies on Monte Carlo sampling, the law of large number and the universal approximation theorem. In this framework,

i) the PDE solution is given by the conditional expectation of a functional of a random coding tree via the fully nonlinear Feynman-Kac formula Theorem 3.2 in [NPP23b].

ii) The conditional expectation can be approximated by a neural network function through the $L^2$-minimality property and the universal approximation theorem.

This algorithm carries information on (possibly functional) nonlinearities along a random tree for the numerical solution of fully nonlinear PDEs with gradients of arbitrary orders by the Monte Carlo method. It has been implemented on $\mathbb{R}^d$ in [NPP23a] using a neural network approach to efficiently approximate the PDE solution $u(t,x) \in \mathbb{R}$ over a bounded domain in $[0,T] \times \mathbb{R}^d$, see also [NPP23c] for an application to the numerical solution of the incompressible Navier-Stokes equation. This approach extends the stochastic branching mechanisms [HLOT$^+$19] and references therein, which applies only to first order gradient

nonlinearities. We note however that branching diffusion methods suffer in general from a finite explosion time depending on the smoothness of the nonlinearities involved in the equation, which is linked to the non-integrability of stochastic weight defined by a branching process.

Other approaches to the solution of differential equations by carrying information on nonlinearities along trees include the use of B-series for ordinary differential equations (ODEs), see [But10], Chapters 4-6 of [DB02] and references therein, with applications ranging to geometric numerical integration and stochastic differential equations [HLW06], see also [Gub10], and [BHZ19] for decorated trees and stochastic partial differential equations. On the other hand, the stochastic branching method does not use series truncations, and it can be used to estimate an infinite series, see [PP22] for an application to ODEs.

Numerical solutions of semilinear PDEs have also been obtained by the multilevel Picard method [EHJK19, EHJK21, HJK22], and [BBH+20] for numerical experiments. We also refer to [BBC+21, HPW20, PWG21] and [LLP23], for related deep learning-based numerical algorithms.

Our method is applied to the Burgers and modified Burgers equations, as well as to the generalized power-law Burger's equation for non-Newtonian fluids, which has applications to the study of aneurysms [BLS18]. We also provide comparisons of our results to the outputs of the deep Galerkin algorithm of [SS18], and to the deep BSDE method of [HJE18, BEJ19], and consider an application to a fluid dynamics boundary problem by the computation of a Poiseuille flow. We note that all three methods considered in this paper are providing a functional estimation of PDE solutions using neural networks.

After a summary of our stochastic coded tree approach in Section 2, we consider in Sections 3-4 the Burgers and modified Burgers equations

$$\partial_t u(t,x) + \frac{\nu}{2}\partial_{xx} u(t,x) - u^p(t,x)\partial_x u(t,x) = 0,$$

for $p = 1, 2, 3$, which incorporate nonlinearity and diffusive dissipation, where $\nu > 0$ is the viscosity parameter.

The numerical solution of the (modified) Burgers equation has been the object of several studies using time and space splitting methods, see e.g. [RED05] and [SD08] for the use of quintic splines, and [Bra10] for a fourth-order finite difference method.

We note in Figures 1 and 3-4 that the Galerkin, BSDE and branching methods perform similarly up to time $T = 1$ in dimension $d = 1$, and up to time $T = 0.2$ in dimension $d = 3$. On the other hand, for the version (3.3) of the multidimensional Burgers equation, the deep branching method is more stable than the deep Galerkin and deep BSDE methods in dimension $d = 20$, see Figure 2-$b$).

Next, in Section 5 we consider the one-dimensional non-Newtonian Burgers equation

$$\partial_t u(t,x) + n\frac{\nu}{2}|\partial_x u(t,x)|^{n-1}\Delta u(t,x) - u(t,x)\partial_x u(t,x) = 0,$$

$(t,x) \in [0,T] \times \mathbb{R}$, where $n \in (0,\infty)$. We note that while all three methods perform comparably for $n = 2, 3$, see Figure 5, it turns out that the deep branching method is more stable than the deep Galerkin and deep BSDE methods for the fractional orders $n = 1/2$ and $n = 1/3$, see Figure 6. Indeed, dealing with functional nonlinearities appears to be challenging for the BSDE and Galerkin algorithms, which perform better for polynomial nonlinearities in Sections 3 and 4. On the other hand, the deep branching method is natively designed to handle a class of general nonlinearities, by propagating them along a random tree.

In Section 6 we consider the case of boundary conditions with application to a fluid dynamics problem using the equation of a Poiseuille flow.

The codes and executions of our numerical experiments are available at `https://github.com/nguwijy/deep_branching_jjiam`. In order to ensure a fair performance comparison, our numerical experiments are tuned to have comparable runtimes, while ensuring that the deep branching method takes the shortest time, see Table 1 which lists the respective computation times in the figures of Sections 3 to 5. The numerical outputs of the deep BSDE and deep Galerkin may be improved by further increasing their runtimes.

| Method \ Figure | 1a | 1b | 2a | 2b | 3a | 3b | 4a | 4b | 5a | 5b | 6a | 6b |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Deep branching | 34 | 92 | 126 | 241 | 38 | 37 | 38 | 38 | 36 | 55 | 54 | 55 |
| Deep Galerkin [SS18] | 87 | 140 | 155 | 598 | 83 | 84 | 85 | 87 | 83 | 91 | 87 | 87 |
| Deep BSDE [BEJ19] | 142 | 145 | 137 | 145 | 141 | 142 | 143 | 141 | 150 | 157 | 148 | 157 |

Table 1: Comparison of runtimes in minutes for Figures 1a to 6b.

We proceed as follows. In Section 2 we recall the framework of [NPP23a] for the probabilistic representation of fully nonlinear PDE solutions using branching trees coded by nonlinearities. This is followed by numerical experiments for the modified and non-Newtonian

Burgers equations in Sections 3-5, and by an application to PDE problems with boundary domain conditions in Section 6. The appendix contains technical derivations that are used in Section 2.

## 2 Stochastic coded trees

In this section, we summarize the construction of the stochastic branching solver of [NPP23b], and we extend it from one real variable to $d$-dimensional variables, $d \geq 1$, for nonlinear PDEs of the form

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x) + \dfrac{\nu}{2}\Delta u(t,x) + f\left(u(t,x), \dfrac{\partial u}{\partial x_1}(t,x), \ldots, \dfrac{\partial u}{\partial x_d}(t,x), \dfrac{\partial^2 u}{\partial x_1^2}(t,x), \ldots, \dfrac{\partial^2 u}{\partial x_d^2}(t,x)\right) = 0, \\ u(T,x) = \phi(x), \qquad (t,x) \in [0,T] \times \mathbb{R}^d, \end{cases} \tag{2.1}$$

where $\nu > 0$, $\Delta = \sum_{k=1}^d \partial_{x_k}^2$ is the standard Laplacian on $\mathbb{R}^d$, and $f(z_0, z_1, \ldots, z_{2d})$ is a smooth functional nonlinearity involving the partial derivatives $\partial u/\partial x_k$, $\partial^2 u/\partial {x_k}^2$, $k = 1, \ldots, d$.

For this, we start by rewriting (2.1) in Duhamel integral representation, as

$$\begin{aligned} u(t,x) &= \int_{\mathbb{R}^d} \varphi(T-t, y-x)\phi(y)dy \\ &+ \int_t^T \int_{\mathbb{R}^d} \varphi(s-t, y-x) f\left(u(t,y), \frac{\partial u}{\partial y_1}(t,y), \ldots, \frac{\partial u}{\partial y_d}(t,y), \frac{\partial^2 u}{\partial y_1^2}(t,y), \ldots, \frac{\partial^2 u}{\partial y_d^2}(t,y)\right) dyds, \end{aligned} \tag{2.2}$$

where $\varphi(t,x) := \mathrm{e}^{-x^2/(2\nu^2 t)}/\sqrt{2\pi\nu^2 t}$ is the Gaussian kernel with variance $\nu^2 > 0$ and $(t,x) \in [0,T] \times \mathbb{R}^d$. Next, using the notation

$$g^*(u)(t,y) := g\left(u(t,y), \frac{\partial u}{\partial y_1}(t,y), \ldots, \frac{\partial u}{\partial y_d}(t,y), \frac{\partial^2 u}{\partial y_1^2}(t,y), \ldots, \frac{\partial u^2}{\partial y_d^2}(t,y)\right), \tag{2.3}$$

$y = (y_1, \ldots, y_d) \in \mathbb{R}^d$, for any function $g : \mathbb{R}^{2d+1} \to \mathbb{R}$, we consider the set $\mathfrak{C}$ of operators from $\mathcal{C}^{0,\infty}([0,T] \times \mathbb{R}^d)$ to $\mathcal{C}^{0,\infty}([0,T] \times \mathbb{R}^d)$, defined as

$$\mathfrak{C} := \left\{ \mathrm{Id}, \quad \left(a\partial_\lambda f\right)^*, \ \partial_\mu \ : \ \lambda \in \mathbb{N}^{2d+1}, \ \mu \in \mathbb{N}^d, \ a \in \mathbb{R} \setminus \{0\} \right\},$$

where Id denotes the identity on $\mathcal{C}^{0,\infty}([0,T] \times \mathbb{R}^d)$, and

$$\partial_\lambda := \partial_{z_0}^{\lambda_0} \cdots \partial_{z_{2d}}^{\lambda_{2d}}, \quad \partial_\mu := \partial_{x_1}^{\mu_1} \cdots \partial_{x_d}^{\mu_d}, \quad \lambda = (\lambda_0, \ldots, \lambda_{2d}) \in \mathbb{N}^{2d+1}, \ \mu = (\mu_1, \ldots, \mu_d) \in \mathbb{N}^d.$$

We also consider the mechanism $\mathcal{M}$, which is a mapping defined on $\mathfrak{C}$ by $\mathcal{M}(\mathrm{Id}) := \{f^*\}$ and

$$\mathcal{M}(g^*) := \left\{\left(f^*, \left(\frac{\partial g}{\partial z_0}\right)^*\right)\right\} \tag{2.4}$$
$$\bigcup_{\substack{1\le p\le 2d,\ 1\le s\le|\lambda^p|\\ 1\le\nu_0+\cdots+\nu_{2d}\le|\lambda^p|\\ 1\le|k_1|,\ldots,|k_s|,\\ 0\prec l^1\prec\cdots\prec l^s\\ k_1^i+\cdots+k_s^i=\nu_i,\ i=0,\ldots,2d\\ |k_1|l_j^1+\cdots+|k_s|l_j^s=\lambda_j^p,\ j=0,\ldots,2d}} \left\{\left((\partial_{\mathbf{1}_p}g)^*, \frac{\prod_{i=1}^d \lambda_i^p!(\partial_\nu f)^*}{\prod_{\substack{1\le r\le s\\ 1\le q\le 2d}} k_r^q!\,(l_1^r!\cdots l_d^r!)^{k_r^q}}\right) \bigcup_{\substack{1\le r\le s\\ 1\le q\le 2d}} \Big(\underbrace{\partial_{l^r+\lambda^q},\ldots,\partial_{l^r+\lambda^q}}_{k_r^q\ \text{times}}\Big)\right\}$$
$$\bigcup_{\substack{i,j=1,\ldots,2d\\ k=1,\ldots,d}} \left\{\left(-\frac{\nu}{2}(\partial_{\mathbf{1}_i+\mathbf{1}_j}g)^*, \partial_{\lambda^i+\mathbf{1}_k}, \partial_{\lambda^j+\mathbf{1}_k}\right)\right\}, \qquad g^*\in\mathcal{C},$$

$g^*\in\mathfrak{C}$, where we let $\lambda^0=(0,\ldots,0)\in\mathbb{N}^{2d+1}$, $\lambda^i=\mathbf{1}_i\in\mathbb{N}^{2d+1}$ for $i=1,\ldots,d$, and $\lambda^i=2\mathbf{1}_i\in\mathbb{N}^{2d+1}$ for $i=d+1,\ldots,2d$, where $\mathbf{1}_i$ denotes the vector made of 1 at the position $q\in\{1,\ldots,2d\}$ and 0 elsewhere, and by

$$\mathcal{M}(\partial_\mu) \tag{2.5}$$
$$:= \bigcup_{\substack{1\le s\le|\mu|,\ 1\le\eta_0+\cdots+\eta_d\le|\mu|\\ 1\le|k_1|,\ldots,|k_s|,\ 0\prec l^1\prec\cdots\prec l^s\\ k_1^i+\cdots+k_s^i=\eta_i,\ i=0,\ldots,d\\ |k_1|l_j^1+\cdots+|k_s|l_j^s=\mu_j,\ j=1,\ldots,d}} \left\{\left(\frac{\prod_{p=1}^d \mu_p!}{\prod_{\substack{1\le r\le s\\ 1\le q\le d}} k_r^q!\,(l_1^r!\cdots l_d^r!)^{k_r^q}}(\partial_\eta f)^*\right) \bigcup_{\substack{1\le r\le s\\ 1\le q\le d}} \Big(\underbrace{\partial_{l^r+\mathbf{1}_q}u,\ldots,\partial_{l^r+\mathbf{1}_q}u}_{k_r^q\ \text{times}}\Big)\right\},$$

$\mu\in\mathbb{N}^d$. The elements of $\mathfrak{C}$ are called *codes*, and from the calculations presented in appendix we note that the application $c(u)$ of $c\in\mathfrak{C}$ to $u(t,x)$ satisfies the system of equations

$$c(u)(t,x) = \int_{\mathbb{R}^d}\varphi(T-t,y-x)c(u)(T,y)dy + \sum_{Z\in\mathcal{M}(c)}\int_t^T\int_{\mathbb{R}^d}\varphi(s-t,y-x)\prod_{z\in Z}z(u)(s,y)dyds, \tag{2.6}$$

$(t,x)\in[0,T]\times\mathbb{R}^d$, see also Lemma 2.3 of [NPP23b]. Next, we rewrite (2.6) as the recursive relation

$$c(u)(t,x) = \mathbb{E}\Bigg[\mathbf{1}_{\{\tau_1>T-t\}}\frac{c(u)(T,x+W_{\nu(T-t)})}{\overline{F}(T-t)} \tag{2.7}$$
$$+\,\mathbf{1}_{\{\tau_1\le T-t\}}\sum_{Z\in\mathcal{M}(c)}\int_t^T\mathbf{1}_{\{I_c=Z\}}\frac{\prod_{z\in Z}z(u)(s,x+W_{\nu(s-t)})}{q(I_c)\rho(\tau_1)}ds\Bigg],$$

where $(\tau_1, W_{\sigma^2}, I_c)$ is a triple of independent random variables, such that

i) $\tau_1$ is a random time with probability density function $\rho : \mathbb{R}_+ \to (0, \infty)$ and tail distribution function $\overline{F}$,

ii) $W_{\sigma^2}$ is a $d$-dimensional $\mathcal{N}(0, \sigma^2)$ independent centered normal random sample with variance $\sigma^2$, and

iii) $I_c$ is a random variable with uniform distribution $q_c(b) := \mathbb{P}(I_c = b)$ on $\mathcal{M}(c)$, $b \in \mathcal{M}(c)$.

Given an initial code $c$ and initial condition $(t, x) \in [0, T] \times \mathbb{R}^d$, we build a random tree $\mathcal{T}_{t,x,c}$ which branches at random time intervals whose lengths are independent copies of $\tau_1$. In addition, a code in $\mathfrak{C}$ and a Brownian path are assigned to every branch of $\mathcal{T}_{t,x,c}$. Namely, after branching at the node $\overline{k} \in \mathcal{T}_{t,x,c}$, a branch with code $c_{\overline{k}} \in \mathfrak{C}$ generates a sequence of offsprings with corresponding code sequence $I_{c_{\overline{k}}}$ chosen in $\mathcal{M}(c_{\overline{k}})$ with the probability distribution $(q_{c_{\overline{k}}}(b))_{b \in \mathcal{M}(c_{\overline{k}})}$, and every new branch develops an independent Brownian random evolution over the random branch time $\tau_{\overline{k}}$. The creation of new branches from a node $\overline{k} \in \mathcal{T}_{t,x,c}$ stops when the new branching time exceeds $T$. In this case, the last branching time is denoted by $T_{\overline{k}-}$ and the corresponding terminal Brownian value is denoted by $X^{\bar{k}}_{T,x}$. Finally, the set of leaves of the random tree $\mathcal{T}_{t,x,c}$ is denoted by $\mathcal{T}^{\partial}_{t,x,c}$, and we let $\mathcal{T}^{\circ}_{t,x,c} = \mathcal{T}_{t,x,c} \setminus \mathcal{T}^{\partial}_{t,x,c}$.

As in Definition 4.1 of [NPP23b], we consider the random functional

$$\mathcal{H}(t, x, \mathrm{Id}) := \prod_{\overline{k} \in \mathcal{K}^{\circ}} \frac{1}{q_{c_{\overline{k}}}(I_{c_{\overline{k}}}) \rho(\tau_{\overline{k}})} \prod_{\overline{k} \in \mathcal{K}^{\partial}} \frac{c_{\overline{k}}(u)\big(T, X^{\bar{k}}_{T,x}\big)}{\overline{F}(T - T_{\overline{k}-})}$$

of the random coding tree $\mathcal{T}_{t,x,c}$ started at time $t \in [0, T]$, location $x \in \mathbb{R}$ and code $c = \mathrm{Id}$. The functional $\mathcal{H}(t, x, c)$ is constructed by recursion in Algorithm 1 along the random tree $\mathcal{T}_{t,x,c}$ started at $(t, x, c) \in [0, T] \times \mathbb{R}^d \times \mathcal{C}$.

**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 $\rho$
  **if** $t + \tau > T$ **then**
    $W \leftarrow$ a random vector drawn from $\mathcal{N}(0, \nu(T - t))$
    $\mathcal{H}(t, x, c) \leftarrow \mathcal{H}(t, x, c) \times c(u)(T, x + W)/\overline{F}(T - t)$
  **else**
    $q \leftarrow$ the size of the mechanism set $\mathcal{M}(c)$
    $I \leftarrow$ a random element drawn uniformly from $\mathcal{M}(c)$
    $\mathcal{H}(t, x, c) \leftarrow \mathcal{H}(t, x, c) \times q/\rho(\tau)$
    **for all** $cc \in I$ **do**
      $W \leftarrow$ a random vector drawn from $\mathcal{N}(0, \nu\tau)$
      $\mathcal{H}(t, x, c) \leftarrow \mathcal{H}(t, x, c) \times \text{TREE}(t + \tau, x + W, cc)$
    **end for**
  **end if**

Under suitable regularity conditions on $f$ and $\phi$, see Assumption (A) in [NPP23b], taking $c = \text{Id}$ we obtain the probabilistic representation

$$u(t, x) := \mathbb{E}\big[\mathcal{H}(t, x, \text{Id})\big], \quad (t, x) \in [0, T] \times \mathbb{R}^d, \tag{2.8}$$

for the solution $u(t, x)$ of (2.1) as a consequence of (2.7), see Theorem 1 in [NPP23b].

Instead of performing a Monte Carlo evaluation (2.8) at given points $(t, x) \in [0, T] \times \mathbb{R}^d$, we use the $L^2$-minimality property of expectation to obtain a functional estimation of $u = (u_1, \ldots, u_d)$ as $u(\cdot, \cdot) = v^*(\cdot, \cdot)$ on the support of a random vector $(\tau, X)$ on $[0, T] \times \mathbb{R}^d$ such that $\mathcal{H}(\tau, X, \text{Id}_i) \in L^2$, where

$$v^* = \underset{\{v : \mathbb{R}^{d+1} \to \mathbb{R}^d \,:\, v(\tau, X) \in L^2\}}{\arg\min} \sum_{i=1}^{d} \mathbb{E}\left[\left(\mathcal{H}(\tau, X, \text{Id}_i) - v_i(\tau, X)\right)^2\right]. \tag{2.9}$$

To evaluate (2.9) on $[0, T] \times \Omega$, where $\Omega$ is a bounded domain of $\mathbb{R}^d$, we can choose $(\tau, X)$ to be a uniform random vector on $[0, T] \times \Omega$.

We note that the complexity of the algorithm increases linearly with the dimension $d$, as $d$ trees are used to generate a $d$-dimensional solution, whereas the complexity of finite difference methods is generally polynomial in the dimension $d$. Time complexity can be estimated from the mean number of leaves of a branching tree with exponential interbranching times, which

grows exponentially as a function of the terminal time $t > 0$ when the number of branches is fixed, see [PP22].

# 3 Multidimensional Burgers equation

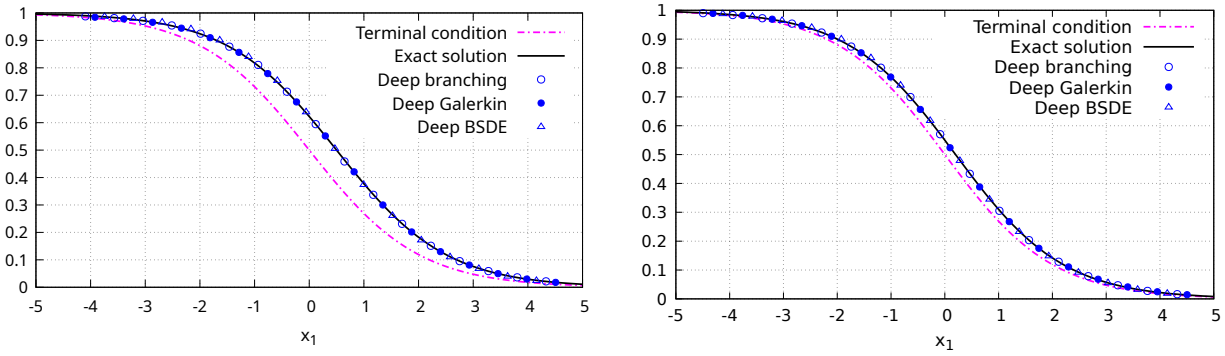
In this section, we consider the standard multidimensional Burgers equation

$$\partial_t u(t,x) + \frac{\nu}{2}\Delta u(t,x) - u(t,x)\sum_{k=1}^{d}\partial_{x_k} u(t,x) = 0, \quad x = (x_1,\dots,x_d) \in \mathbb{R}^d,\ t \in [0,T], \tag{3.1}$$

where $\Delta = \sum_{k=1}^{d}\partial_{x_k}^2$ denotes the standard Laplacian on $\mathbb{R}^d$, $d \geq 1$. Equation (3.1) is known to admit the traveling wave solution

$$u(t,x) = u_2 + \frac{u_1 - u_2}{1 + \exp\big((u_1-u_2)\big(\sum_{i=1}^{d} x_i - (u_1+u_2)(T-t)d/2\big)/\nu\big)}, \tag{3.2}$$

$(t,x) \in [0,T]\times\mathbb{R}^d$, where $u_1 < u_2$, see [WB11]. Figure 1 presents estimates of the solution of the Burgers equation (3.1) with solution (3.2) in dimensions $d = 1$ and $d = 3$, with comparisons with the outputs of the deep Galerkin method [SS18] and of the deep BSDE method [HJE18].




(a) Dimension $d = 1$ with $T = 1$.

(b) Dimension $d = 3$ with $T = 0.2$.

Figure 1: Numerical solution of (3.1) and comparison to (3.2) with $\nu = 1$.

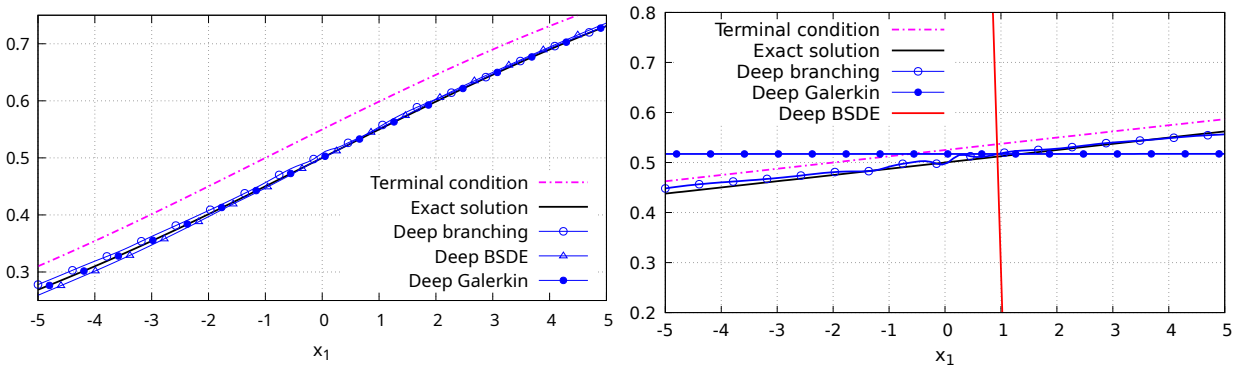
In Figure 1-*b*) and in Figure 2 below we set $(x_2,\dots,x_d) = (0,\dots,0)$ and plot $u(t,x)$ as a function of $x_1$, with $x = (x_1,0,\dots,0)$. Next, we consider the multidimensional Burgers equation

$$\partial_t u(t,x) + \frac{d^2}{2}\Delta u(t,x) + \left(u(t,x) - \frac{2+d}{2d}\right)\left(d\sum_{k=1}^{d}\partial_{x_k} u(t,x)\right) = 0, \tag{3.3}$$

with traveling wave solution

$$u(t,x) = \frac{\exp\left(t + d^{-1}\sum_{i=1}^{d} x_i\right)}{1 + \exp\left(t + d^{-1}\sum_{i=1}^{d} x_i\right)}, \qquad x = (x_1, \ldots, x_d) \in \mathbb{R}^d,\ t \in [0,T], \tag{3.4}$$

see § 4.5 of [HJE17], and § 4.2 of [Cha13]. Figure 2 presents estimates of the solution of the Burgers equation (3.3) with solution (3.4) in dimensions $d = 5$ and $d = 20$, with comparisons with the outputs of the deep Galerkin method [SS18] and of the deep BSDE method [HJE18].




(a) Dimension $d = 5$ with $T = 0.2$.

(b) Dimension $d = 20$ with $T = 0.1$.

Figure 2: Numerical solution of (3.3) and comparison to (3.4) with $\nu = d^2$.

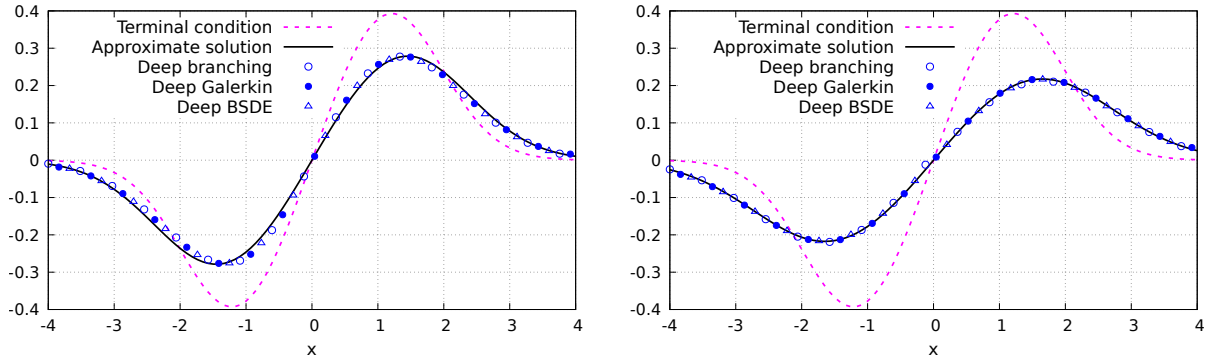
We note in Figure 2−*b*) that the deep branching method is more stable than the deep Galerkin and deep BSDE methods in dimension $d = 20$.

# 4 Modified Burgers equation

We note that in dimension $d = 1$ the standard Burgers equation (3.1) also admits the traveling wave solution

$$u(t,x) = \frac{x}{1+T-t}\left(1 + \alpha\sqrt{1+T-t}\exp\left(\frac{x^2}{2\nu(1+T-t)}\right)\right)^{-1}, \tag{4.1}$$

$(t,x) \in [0,T] \times \mathbb{R}$, for any $\alpha > 0$, see Section 8.3 of [Deb12]. Figure 3 presents estimates of the solution of the Burgers equation (3.1) with solution (4.1).



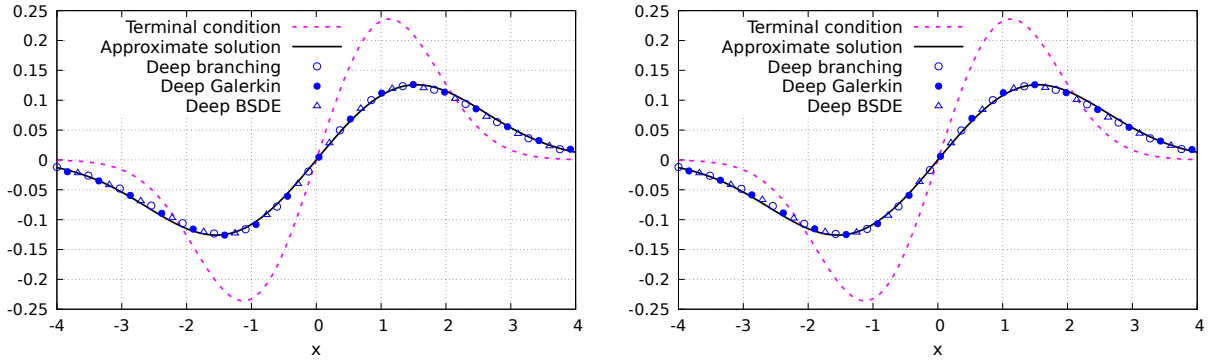

(a) Time $T = 0.5$. (b) Time $T = 1$.

Figure 3: Numerical solution of (3.1) and comparison to (4.1) with $\alpha = 1$, and $\nu = 1$.

We consider the modified Burgers equations

$$\partial_t u(t,x) + \frac{\nu}{2}\partial_{xx} u(t,x) - u^p(t,x)\partial_x u(t,x) = 0, \quad x \in \mathbb{R}, \qquad p = 2, 3, \tag{4.2}$$

which incorporate nonlinearity and diffusive dissipation, and are known to admit (4.1) as approximate solution when $\alpha = 2$, see [Har96], [RED05], [SD08], [Bra10]. Figure 4 presents estimates of the solution of the modified Burgers equation (4.2).




(a) MBE2 at time $T = 1$, $p = 2$. (b) MBE3 at time $T = 1$, $p = 3$.

Figure 4: Numerical solution of the Burgers equation (4.2) with $\alpha = 2$ and $\nu = 1$.

# 5 Non-Newtonian Burgers equation

In this section, we consider the non-Newtonian Burgers equation in dimension $d = 1$

$$\partial_t u(t,x) + \frac{\nu}{2}\partial_x \left( |\partial_x u(t,x)|^{n-1} \partial_x u(t,x) \right) - u(t,x)\partial_x u(t,x) = 0, \tag{5.1}$$

with $n > 0$, i.e. when $\partial_x u(t,x) > 0$ we have

$$\partial_t u(t,x) + n\frac{\nu}{2}(\partial_x u(t,x))^{n-1}\Delta u(t,x) - u(t,x)\partial_x u(t,x) = 0,$$

$(t,x) \in [0,T]\times\mathbb{R}$. In order to apply the algorithm described in Section 2 and the deep BSDE method to (5.1), we further rewrite the equation as

$$\partial_t u(t,x) + \frac{\nu}{2}\Delta u(t,x) + n\frac{\nu}{2}(\partial_x u(t,x))^{n-1}\Delta u(t,x) - \frac{\nu}{2}\partial_x^2 u(t,x) - u(t,x)\partial_x u(t,x) = 0, \quad (5.2)$$

i.e. the nonlinearity $f$ in (2.1) is given by

$$f(u,v,w) = n\frac{\nu}{2}wv^{n-1} - \frac{\nu}{2}w - uv.$$

For any $u_1, u_2 \in \mathbb{R}$ with $u_1 < u_2$, the non-Newtonian Burgers equation (5.1) is known to admit a solution of the form

$$u(t,x) = h^{-1}\left(x - (u_1+u_2)\frac{T-t}{2}\right), \qquad (t,x) \in [0,T]\times\mathbb{R}, \quad (5.3)$$

with

$$\lim_{x\to-\infty} u(t,x) = u_2, \quad \lim_{x\to\infty} u(t,x) = u_1, \quad \text{and} \quad \lim_{|x|\to\infty} \partial_x u(t,x) = 0,$$

where

$$\begin{aligned} h(x) &:= \int_x^{(u_1+u_2)/2} \left(\frac{\nu}{(v-u_1)(u_2-v)}\right)^{1/n} dv \\ &= -\nu^{1/n}\frac{n}{n-1}\frac{(x-u_1)}{(u_2-u_1)^{1/n}(x-u_1)^{1/n}}\,{}_2F_1\left(\frac{1}{n}, 1-\frac{1}{n}, 2-\frac{1}{n}; \frac{u_1-x}{u_1-u_2}\right), \end{aligned}$$

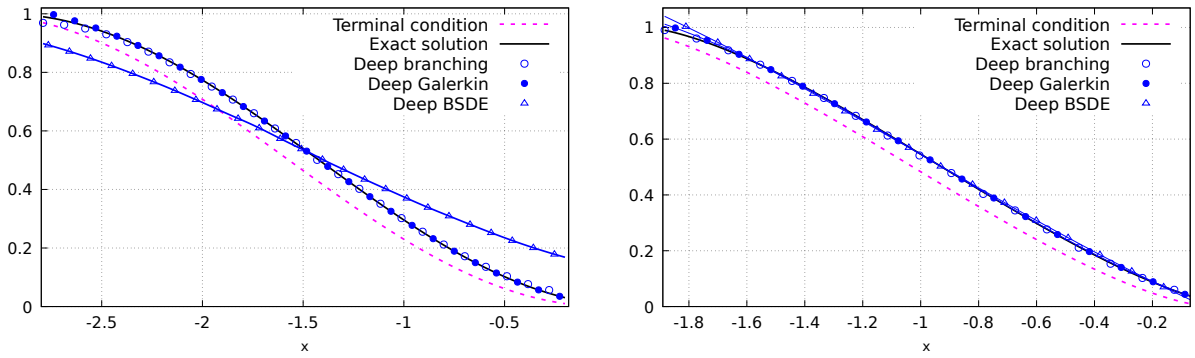
$u_1 < x < u_2$, and ${}_2F_1$ is the Gaussian hypergeometric function, see [WB11], [WH12]. When $n = 1$, (5.1) coincides with the standard Burgers equation (3.1) with

$$h(u) = \frac{\nu}{u_2-u_1}\log\left(\frac{u_2-u}{u-u_1}\right),$$

which recovers the traveling wave solution (3.4) in dimension $d = 1$.

When the inverse function of $h$ is not available in closed form, we use the library 'pynverse' to obtain pointwise samples of $h^{-1}$ in (5.3). Then, we use a neural network coded in PyTorch to learn from the pointwise samples of $h^{-1}$ in order to gain full knowledge of the derivatives of the terminal condition $\phi(x) = u(T,x) = h^{-1}(x)$, for use by the deep branching solver.

Figure 5 presents estimates of the solution of the non-Newtonian Burgers equation (5.1) with explicit solution given by (5.3) with $n = 2$ and $n = 3$.




(a) Equation (5.1) with $n = 2$ and $T = 0.2$. (b) Equation (5.1) with $n = 3$ and $T = 0.2$.

Figure 5: Numerical solution of (5.1) with $T = 10$ and $\nu = 1$.

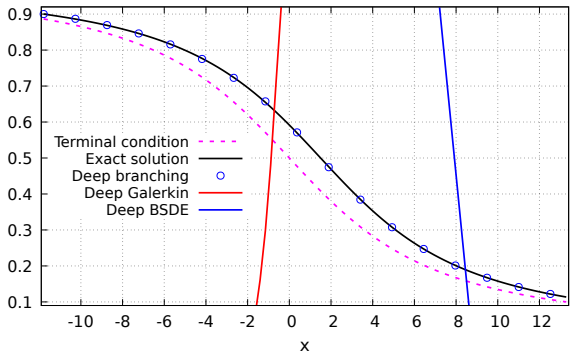
When $n = 1/2$, we have

$$h(u) = \frac{\nu^2}{(u_2 - u_1)^2} \left( \frac{u_1 + u_2 - 2u}{(u - u_1)(u_2 - u)} + \frac{2}{u_2 - u_1} \log \frac{u_2 - u}{u - u_1} \right),$$

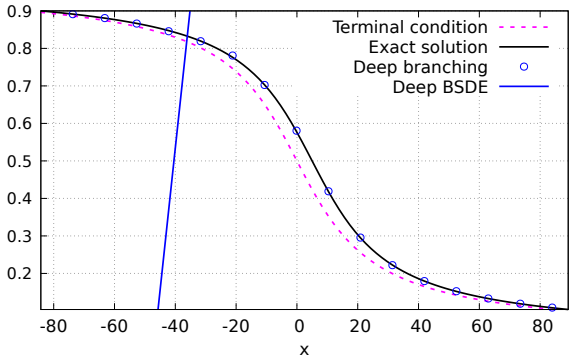
and when $n = 1/3$, we find

$$h(u) = \nu^3 \left( \frac{(u_2 - u_1)^{-3}}{2} \left( \frac{1}{(u - u_1)^2} - \frac{1}{(u_2 - u)^2} \right) \right.$$
$$\left. + 3(u_2 - u_1)^{-4} \left( \frac{1}{u - u_1} - \frac{1}{u_2 - u} \right) + 6(u_2 - u_1)^{-5} \log \left( \frac{u_2 - u}{u - u_1} \right) \right), \quad u_1 < u < u_2.$$

Figure 6 presents estimates of the solution of the non-Newtonian Burgers equation (5.1) with explicit solution given by (5.3) with parameters $n = 1/2$ and $n = 1/3$. We note that in this fractional order setting, the deep branching method is more stable than the deep Galerkin and deep BSDE methods at comparable runtimes.




(a) Equation (5.1) with $n = 1/2$ and $T = 3$.




(b) Equation (5.1) with $n = 1/3$ and $T = 10$.

Figure 6: Numerical solution of the non-Newtonian equation (5.1) with $T = 10$ and $\nu = 1$.

No data or graph are available for the deep Galerkin method in Figure 6$-b$) as the numerical output is NaN.

# 6 Domains with boundary conditions

In this section, we add a Dirichlet domain condition of the form

$$u(t,x) = 0, \qquad (t,x) \in [0,T] \times \partial\Omega, \tag{6.1}$$

to the system (2.1), on the boundary $\partial\Omega$ of a domain $\Omega \subset \mathbb{R}^d$. In this case, the recursive Relation (2.7) is modified into

$$\begin{aligned} c(u)(t,x) = \mathbb{E}\Bigg[ & \mathbf{1}_{\{\tau_1 > T-t\}} \mathbf{1}_{\{(x+W_u)_{u\in[0,\nu(T-t)]} \subset \Omega\}} \frac{c(u)(T, x + W_{\nu(T-t)})}{\overline{F}(T-t)} \\ & + \mathbf{1}_{\{\tau_1 \leq T-t\}} \sum_{Z \in \mathcal{M}(c)} \int_t^T \mathbf{1}_{\{I_c = Z\}} \mathbf{1}_{\{x+W_u \in \Omega,\ \forall 0 \leq u \leq \nu(s-t)\}} \frac{\prod_{z\in Z} z(u)(s, x + W_{\nu(s-t)})}{q(I_c)\rho(\tau_1)} ds \Bigg], \end{aligned} \tag{6.2}$$

where the indicators $\mathbf{1}_{\{(x+W_u)_{u\in[0,\nu(T-t)]}}$ and $\mathbf{1}_{\{x+W_u\in\Omega,\ \forall 0\leq u\leq \nu(s-t)\}}$ account for the presence of the boundary condition (6.1).

### Path discretization approach

The first approach to implementing the indicator $\mathbf{1}_{\{x+W_u\in\Omega,\ \forall 0\leq u\leq\nu(s-t)\}}$ in (6.2) is to discretize the Brownian path over $m \geq 1$ time steps as

$$\{x + W_u : u = 0,\ \nu(s-t)/m,\ 2\nu(s-t)/m, \ldots, \nu(s-t)\},$$

and to check the indicator conditions at discrete times. This can be achieved by passing the parameter `bm_discretization_steps=m` to the deep branching solver. Such an approach allows us to deal with arbitrary domains $\Omega$, however it may require a longer runtime and may lose accuracy close the boundary $\partial\Omega$.

### Survival probability approach

The second approach is to estimate explicitly the conditional survival probability

$$\mathbb{P}\big(\{x + W_u \in \Omega,\ \forall 0 \le u < \nu(s-t)\} \cap \{x + W_{\nu(s-t)} \in \Omega\} \mid W_{\nu(s-t)}\big) \tag{6.3}$$

and to pass the corresponding formula to the deep branching solver via the parameter `conditional_probability_to_survive`. This approach results into a shorter runtime and a higher accuracy over the whole domain. Its drawback is that a closed-form expression may not be available for arbitrary domains.

In case $\Omega$ is the hyperrectangle $\Omega = \prod_{i=0}^{d_{in}-1} [a_i, b_i]$, the probability (6.3) can be written in closed form as

$$\prod_{i=0}^{d_{in}-1} \frac{1}{\sqrt{2\pi\nu(s-t)}} \sum_{k=-\infty}^{\infty} \left( \exp\left( \frac{\big(W^{(i)}_{\nu(s-t)}\big)^2 - \big(W^{(i)}_{\nu(s-t)} + 2k(b_i - a_i)\big)^2}{2\nu(s-t)} \right) - \exp\left( \frac{\big(W^{(i)}_{\nu(s-t)}\big)^2 - \big(2x + W^{(i)}_{\nu(s-t)} - 2a_i + 2k(b_i - a_i)\big)^2}{2\nu(s-t)} \right) \right), \tag{6.4}$$

see e.g. Relation (4.44) in [Bor17].

### Example - Heat equation with boundary condition

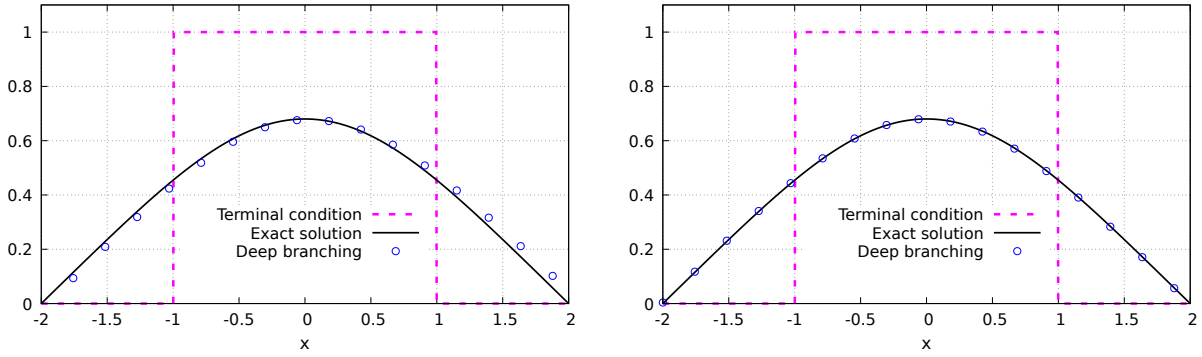
As an example, letting $\Omega := [\mathrm{lo}, \mathrm{hi}]$, we consider the heat equation

$$\begin{cases} \partial_t u_A(t,x) + \dfrac{\nu}{2} \Delta u_A(t,x) = 0, & (t,x) \in [0,T] \times \Omega, \\ u_A(t,x) = 0, & (t,x) \in [0,T] \times \partial\Omega, \\ u_A(T,x) = \mathbf{1}_{\{x \in A\}}, & x \in \Omega, \end{cases} \tag{6.5}$$

in dimension $d = 1$, where $A = [a,b] \subset \Omega$, which admits the closed-form solution

$$\begin{aligned} u_A(t,x) \ = \ & \sum_{k=-\infty}^{\infty} \Phi\left( \frac{b - x + 2k(\mathrm{hi} - \mathrm{lo})}{\sqrt{\nu(T-t)}} \right) - \sum_{k=-\infty}^{\infty} \Phi\left( \frac{a - x + 2k(\mathrm{hi} - \mathrm{lo})}{\sqrt{\nu(T-t)}} \right) \\ & - \sum_{k=-\infty}^{\infty} \Phi\left( \frac{b + x - 2\mathrm{lo} + 2k(\mathrm{hi} - \mathrm{lo})}{\sqrt{\nu(T-t)}} \right) + \sum_{k=-\infty}^{\infty} \Phi\left( \frac{a + x - 2\mathrm{lo} + 2k(\mathrm{hi} - \mathrm{lo})}{\sqrt{\nu(T-t)}} \right), \end{aligned}$$

$(t,x) \in [0,T] \times \partial\Omega$. Figure 7 presents the output of the deep branching solver for the equation (6.5) with $\Omega = [-2,2]$ and $A = [-1,1]$, using either path discretization or (6.4).




(a) Heat equation (5.1) by discretization.

(b) Heat equation (5.1) by (6.4).

Figure 7: Numerical solution of the heat equation (6.5) with boundary.

**Example - Poiseuille flow**

Next, we consider the time-independent and time-dependent Poiseuille flows modeled by the two-dimensional Navier-Stokes equation on the square $[-L,L]^2$, i.e.

$$\partial_t u(t,x,y) + \frac{\nu}{2}\Delta u(t,x,y) = \nabla p(t,x,y) + u(t,x,y)\cdot\nabla u(t,x,y), \tag{6.6}$$

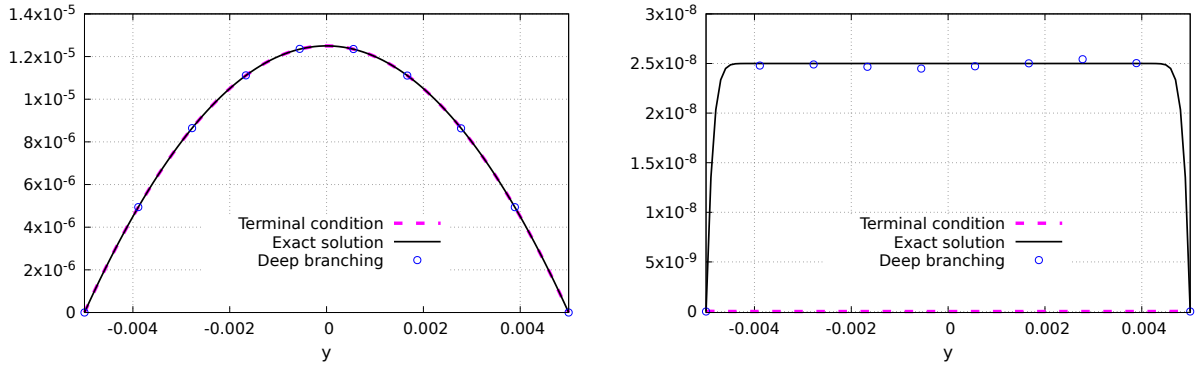
$(x,y) \in [-L,L]^2$, $t \in [0,T]$, with boundary conditions on the walls $y = L$ and $y = L$, and an inlet at $x = -L$ flowing to the direction $x = L$, with the divergence-free condition $\operatorname{div} u = 0$. Using the pressure function

$$p(t,x,y) := \alpha v_0^2 - \frac{\nu v_0}{L^2}x, \quad (x,y) \in [-L,L]^2, \quad t \in [0,T],$$

it can be checked that the velocity flow $u(t,x,y) := (g(t,y),0)$ with

$$g(t,y) := v_0\left(1 - \frac{y^2}{L^2}\right) + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}32v_0}{\pi^3(2n+1)^3} e^{-\frac{(2n+1)^2\pi^2\nu(T-t)}{8L^2}} \cos\frac{(2n+1)\pi y}{2L}, \tag{6.7}$$

solves the Navier-Stokes equation (6.6), see e.g. Eq. (16) in [SKS+03]. Figure 8 presents the output of the deep branching solver using (6.4) for the equation (6.6), for the stationary solution $u(t,x,y) = (v_0(1 - y^2/L^2),0)$ and for the time-dependent solution series (6.7) truncated with 100 terms.




(a) Stationary solution $(v_0(1-y^2/L^2),0)$. (b) Time-dependent solution (6.7) with $t=10$.

Figure 8: Numerical solutions $u(t,x,y)$ of the Poiseuille equation (6.6).

# A Appendix

In this appendix, we show how the system of equations (2.6) can be obtained as a consequence of the Duhamel identity (2.2) and the Faà di Bruno formula. For simplicity of exposition, we restrict ourselves to the case where (2.1) takes the form

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x) + \dfrac{\nu}{2}\Delta u(t,x) + f\left(u(t,x), \dfrac{\partial u}{\partial x_1}(t,x), \dots, \dfrac{\partial u}{\partial x_d}(t,x)\right) = 0, \\ u(T,x) = \phi(x), \qquad (t,x) \in [0,T]\times\mathbb{R}^d, \end{cases} \tag{A.1}$$

and refer to [NPP23a] for the general case. In the setting of (A.1), the writing of $\mathcal{M}(g^*)$ in (2.4) simplifies to

$$\begin{aligned} \mathcal{M}(g^*) &:= \left\{\left(f^*, \left(\frac{\partial g}{\partial z_0}\right)^*\right)\right\} \\ &\bigcup_{p=1}^{d}\bigcup_{i=0}^{d}\left\{\left(\left(\frac{\partial g}{\partial x_p}\right)^*, \left(\frac{\partial f}{\partial x_i}\right)^*, \frac{\partial^{1+\lambda_i}}{\partial y_p^{1+\lambda_i}}\right)\right\} \bigcup \bigcup_{\substack{i,j=0,\dots,d\\ k=1,\dots,d}} \left\{\left(-\frac{\nu}{2}(\partial_{z_i}\partial_{z_j} g)^*, \frac{\partial^{\lambda_i+1}u}{\partial y_i^{\lambda_i}\partial y_k}, \frac{\partial^{\lambda_j+1}u}{\partial y_j^{\lambda_j}\partial y_k}\right)\right\}, \end{aligned} \tag{A.2}$$

$g^* \in \mathfrak{C}$, where $(\lambda_0,\dots,\lambda_d) := (0,1,\dots,1)$, and using the notation (2.3) we have

$$\begin{aligned} &\partial_t g^*(u) + \frac{\nu}{2}\Delta g^*(u) \\ &= \sum_{p=0}^{d} \frac{\partial^{\lambda_p}}{\partial y_p^{\lambda_p}}\left(\partial_t u + \frac{\nu}{2}\Delta u\right)\left(\frac{\partial g}{\partial z_p}\right)^*(u) + \frac{\nu}{2}\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=1}^{d}\left(\frac{\partial^{\lambda_i+1}u}{\partial y_i^{\lambda_i}\partial y_k}\right)\left(\frac{\partial^{\lambda_j+1}u}{\partial y_j^{\lambda_j}\partial y_k}\right)\left(\frac{\partial^2 g}{\partial z_i \partial z_j}\right)^*(u) \\ &= -\sum_{p=0}^{d}\left(\frac{\partial g}{\partial z_p}\right)^*(u)\frac{\partial^{\lambda_p}}{\partial y_p^{\lambda_p}} f^*(u) + \frac{\nu}{2}\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=1}^{d}\left(\frac{\partial^{\lambda_i+1}u}{\partial y_i^{\lambda_i}\partial y_k}\right)\left(\frac{\partial^{\lambda_j+1}u}{\partial y_j^{\lambda_j}\partial y_k}\right)\left(\frac{\partial^2 g}{\partial z_i \partial z_j}\right)^*(u) \end{aligned}$$

$$
\begin{aligned}
&= -\left(\frac{\partial g}{\partial z_0}\right)^*(u) f^*(u) - \sum_{p=1}^{d} \left(\frac{\partial g}{\partial z_p}\right)^*(u) \frac{\partial}{\partial y_p} f^*(u) \\
&\quad + \frac{\nu}{2} \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=1}^{d} \left(\frac{\partial^{\lambda_i+1} u}{\partial y_i^{\lambda_i} \partial y_k}\right) \left(\frac{\partial^{\lambda_j+1} u}{\partial y_j^{\lambda_j} \partial y_k}\right) \left(\frac{\partial^2 g}{\partial z_i \partial z_j}\right)^*(u) \\
&= -\left(\frac{\partial g}{\partial z_0}\right)^*(u) f^*(u) - \sum_{p=1}^{d} \sum_{i=0}^{d} \left(\frac{\partial g}{\partial z_p}\right)^*(u) \frac{\partial^{1+\lambda_i} u}{\partial y_p^{1+\lambda_i}} \frac{\partial f}{\partial z_i} \left(u(t,y), \frac{\partial u}{\partial y_1}(t,y), \ldots, \frac{\partial u}{\partial y_d}(t,y)\right) \\
&\quad + \frac{\nu}{2} \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=1}^{d} \left(\frac{\partial^{\lambda_i+1} u}{\partial y_i^{\lambda_i} \partial y_k}\right) \left(\frac{\partial^{\lambda_j+1} u}{\partial y_j^{\lambda_j} \partial y_k}\right) \left(\frac{\partial^2 g}{\partial z_i \partial z_j}\right)^*(u).
\end{aligned}
$$

Rewriting the above equation in the Duhamel integral form yields

$$
\begin{aligned}
g^*(u)(t,x) &= \int_{\mathbb{R}^d} \varphi(T-t, y-x) g(\phi(y)) dy \\
&+ \int_t^T \int_{\mathbb{R}^d} \varphi(s-t, y-x) \Bigg( -\frac{\nu}{2} \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=1}^{d} \frac{\partial^{\lambda_i+1} u}{\partial y_i^{\lambda_i} \partial y_k} \frac{\partial^{\lambda_j+1} u}{\partial y_j^{\lambda_j} \partial y_k} \left(\frac{\partial^2 g}{\partial z_i \partial z_j}\right)^*(u) \\
&+ \left(\frac{\partial g}{\partial x_0}\right)^*(u) f^*(u) + \sum_{p=1}^{d} \sum_{i=0}^{d} \left(\frac{\partial g}{\partial x_p}\right)^*(u) \frac{\partial^{1+\lambda_i} u}{\partial y_p^{1+\lambda_i}} \frac{\partial f}{\partial z_i} \left(u(s,y), \frac{\partial u}{\partial y_1}(s,y), \ldots, \frac{\partial u}{\partial y_d}(s,y)\right) \Bigg) dy ds,
\end{aligned}
$$

which shows (2.6) for codes $c \in \mathfrak{C}$ of the form $c = g^*$, $g \in \mathcal{C}^{0,\infty}([0,T] \times \mathbb{R}^d)$, from the definition (A.2) of $\mathcal{M}$. Similarly, for $\mu \in \mathbb{N}^d$, using the Faà di Bruno formula, see Theorem 2.1 in [CS96], denoting by $\mathbf{1}_q$ the vector made of 1 at the position $q \in \{1, \ldots, d\}$ and 0 elsewhere, we have

$$
\begin{aligned}
\partial_\mu u(t,x) &= \int_{\mathbb{R}^d} \varphi(T-t, y-x) \partial_\mu u(T,y) dy \\
&+ \int_t^T \int_{\mathbb{R}^d} \sum_{\substack{1 \le \eta_0 + \cdots + \eta_d \le |\mu| \\ 1 \le s \le |\mu|}} \sum_{\substack{1 \le |k_1|, \ldots, |k_s|,\ 0 \prec l^1 \prec \cdots \prec l^s \\ k_1^i + \cdots + k_s^i = \eta_i,\ i=0,\ldots,d \\ |k_1| l_j^1 + \cdots + |k_s| l_j^s = \mu_j,\ j=1,\ldots,d}} \frac{\prod_{i=1}^{d} \mu_i!}{\prod_{\substack{1 \le r \le s \\ 1 \le q \le d}} k_r^q! \, (l_1^r! \cdots l_d^r!)^{k_r^q}} (\partial_\eta f)^* \sum_{\substack{1 \le r \le s \\ 1 \le q \le d}} (\partial_{l^r + \mathbf{1}_q} u(s,y))^{k_r^q} dy ds,
\end{aligned}
$$

which yields (2.6) for codes $c \in \mathfrak{C}$ of the form $c = \partial_\mu = \partial_{x_1}^{\mu_1} \cdots \partial_{x_d}^{\mu_d}$, $\mu \in \mathbb{N}^d$, from the definition (2.5) of $\mathcal{M}$. The dimension-free argument of Theorem 3.2 in [NPP23b] then shows that (2.8) holds provided that $\mathcal{H}(t,x,\mathrm{Id})$ is integrable and the solution of (2) is unique.

# References


[BBC+21] C. Beck, S. Becker, P. Cheridito, A. Jentzen, and A. Neufeld. Deep splitting method for parabolic PDEs. *SIAM J. Sci. Comput.*, 43(5):A3135–A3154, 2021.

[BBH+20] S. Becker, R. Braunwarth, M. Hutzenthaler, A. Jentzen, and Ph. von Wurstemberger. Numerical simulations for full history recursive multilevel Picard approximations for systems of high-dimensional partial differential equations. *Commun. Comput. Phys.*, 28(5):2109–2138, 2020.

[BEJ19] C. Beck, W. E, and A. Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *J. Nonlinear Sci.*, 29(4):1563–1619, 2019.

[BHZ19] Y. Bruned, M. Hairer, and L. Zambotti. Algebraic renormalisation of regularity structures. *Invent. Math.*, 215:1039–1156, 2019.

[BLS18] M. Badgaish, J.E. Lin, and P. Seshaiyer. Mathematical analysis and simulation of a coupled nonlinear fluid structure interaction model with application to aneurysms. *Communications in Applied Analysis*, 22:637–661, 2018.

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

[Bra10] A.G. Bratsos. A fourth-order numerical scheme for solving the modified Burgers equation. *Comput. Math. Appl.*, 60(5):1393–1400, 2010.

[But10] J.C. Butcher. Trees and numerical methods for ordinary differential equations. *Numerical Algorithms*, 53:153–170, 2010.

[Cha13] J.F. Chassagneux. Linear multi-step schemes for BSDEs. Preprint arXiv:1306.5548v1, 2013.

[CKSY23] J. Chessari, R. Kawai, Y. Shinozaki, and T. Yamada. Numerical methods for backward stochastic differential equations: A survey. *Probab. Surv.*, 20, 2023.

[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.

[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.

[DB02] P. Deuflhard and F. Bornemann. *Scientific Computing with Ordinary Differential Equations*, volume 42 of *Texts in Applied Mathematics*. Springer-Verlag, New York, 2002.

[Deb12] L. Debnath. *Nonlinear partial differential equations for scientists and engineers.* Basel: Birkhäuser, 3rd revised edition, 2012.

[EHJK19] W. E, M. Hutzenthaler, A. Jentzen, and T. Kruse. On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *Journal of Scientific Computing*, 79:1534–1571, 2019.

[EHJK21] W. E, M. Hutzenthaler, A. Jentzen, and T. Kruse. Multilevel Picard iterations for solving smooth semilinear parabolic heat equations. *Partial Differential Equations and Applications*, 2, 2021.

[Gub10] M. Gubinelli. Ramification of rough paths. *J. Differential Equations*, 248(4):693–721, 2010.

[Har96] S.E. Harris. Sonic shocks governed by the modified Burgers' equation. *European Journal of Applied Mathematics*, 7:201–222, 1996.

[HJE17] J. Han, A. Jentzen, and W. E. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Preprint arXiv:1706.04702, 39 pages, 2017.

[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.

[HJK22] M. Hutzenthaler, A. Jentzen, and T. Kruse. Overcoming the curse of dimensionality in the numerical approximation of parabolic partial differential equations with gradient-dependent nonlinearities. *Found. Comput. Math.*, 22:905–966, 2022.

[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.

[HLTT14] P. Henry-Labordère, X. Tan, and N. Touzi. A numerical algorithm for a class of BSDEs via the branching process. *Stochastic Processes and their Applications*, 124(2):1112–1140, 2014.

[HLW06] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration*, volume 31 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 2006.

[HPW20] C. Huré, H. Pham, and X. Warin. Deep backward schemes for high-dimensional nonlinear PDEs. *Math. Comp.*, 89(324):1547–1579, 2020.

[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.

[LLP23] W. Lefebvre, G. Loeper, and H. Pham. Differential learning methods for solving fully nonlinear PDEs. *Digital Finance*, 5:189–229, 2023.

[LZCC22] L. Lyu, Z. Zhang, M. Chen, and J. Chen. MIM: A deep mixed residual method for solving high-order partial differential equations. *Journal of Computational Physics*, 452(1):110930, 2022.

[NPP23a] J.Y. Nguwi, G. Penent, and N. Privault. A deep branching solver for fully nonlinear partial differential equations. Preprint arXiv:2203.03234, 17 pages, to appear in Journal of Computational Physics, 2023.

[NPP23b] J.Y. Nguwi, G. Penent, and N. Privault. A fully nonlinear Feynman-Kac formula with derivatives of arbitrary orders. *Journal of Evolution Equations*, 23:Paper No. 22, 29pp., 2023.

[NPP23c] J.Y. Nguwi, G. Penent, and N. Privault. Numerical solution of the incompressible Navier-Stokes equation by a deep branching algorithm. *Communications on Computational Physics*, 34:261–289, 2023.

[Pen91] S. Peng. Probabilistic interpretation for systems of quasilinear parabolic partial differential equations. *Stochastics Stochastics Rep.*, 37(1-2):61–74, 1991.

[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.

[PWG21] H. Pham, X. Warin, and M. Germain. Neural networks-based backward scheme for fully nonlinear PDEs. *Partial Differ. Equ. Appl.*, 2(1):Paper No. 16, 24, 2021.

[RED05] M.A. Ramadan and T.S. El-Danaf. Numerical treatment for the modified Burgers equation. *Mathematics and computers in simulation*, 70:90–98, 2005.

[SD08] B. Saka and İ Dağ. A numerical study of the Burgers' equation. *Journal of the Franklin Institute*, 345:328–348, 2008.

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

[SKS+03] L. Di G. Sigalotti, J. Klapp, E. Sira, Y. Meleán, and A. Hasmy. SPH simulations of time-dependent Poiseuille flow at low Reynolds numbers. *Journal of Computational Physics*, 191(2):622–638, 2003.

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

[WB11] D. Wei and H. Borden. Traveling wave solution of Burger's equation for power-law non-Newtonian flows. *Appl. Math. E-Notes*, 11:133–138, 2011.

[WH12] D. Wei and K. Holladay. Travelling wave solutions of Burgers' equation for Gee-Lyon fluid flows. *Appl. Math. E-Notes*, 12:129–135, 2012.