

High Precision Differentiation Techniques for Data-Driven Solution of Nonlinear PDEs by Physics-Informed Neural Networks

Marat S. Mukhametzhanov

Department of Computer Science, Modeling, Electronics and Systems Engineering (DIMES),
University of Calabria, 87036 – Rende (CS), Italy
m.mukhametzhanov@dimes.unical.it

October 2022

Abstract

Time-dependent Partial Differential Equations with given initial conditions are considered in this paper. New differentiation techniques of the unknown solution with respect to time variable are proposed. It is shown that the proposed techniques allow to generate accurate higher order derivatives simultaneously for a set of spatial points. The calculated derivatives can then be used for data-driven solution in different ways. An application for Physics Informed Neural Networks by the well-known DeepXDE software solution in Python under Tensorflow background framework has been presented for three real-life PDEs: Burgers', Allen-Cahn and Schrodinger equations.

1 Introduction

Time-dependent Partial Differential Equations (PDEs) arise frequently in real-life applications: e.g., diffusion process of liquid flows (see, e.g., [18]), heat distribution in time (see, e.g., [21]), simulations of nonlinear wave dynamics (see, e.g., [28]), groundwater flow dynamics (see, e.g., [4]), quantum dynamics (see, e.g., [8]), computational mechanics (see, e.g., [15]), etc. These applications are very important from both theoretical and practical points of view. For instance, groundwater flow simulations can be used to predict hydro-geological risks, which are crucial for infrastructures located in seismic or unstable regions (see, e.g., [1, 14]). High precision efficient simulations and modeling in this case can be used to predict different risks arising in this field. In this case, numerical models can be used to describe fluid dynamics: e.g., diffusion equation or Burgers' equations (see, e.g., [2]).

In order to solve difficult nonlinear PDEs, there exist different approaches: e.g., finite element method (FEM, see, e.g., [13]) or Isogeometric analysis (IGA,

see, e.g., [5, 19]). Recently, a wide interest to deep learning has led to another approach for solving challenging PDEs: data-driven solution by Physics-Informed Neural Networks (PINNs, see, e.g., [6, 12, 20]).

Neural networks are usually called as universal approximators allowing one to approximate any function with any given accuracy if there is sufficient data. For this reason, they can be used for the approximation of the solution to differential equations if there is enough information or data about the solution (e.g., sufficiently large data set on boundary and initial conditions and sufficiently large data set of collocation points, i.e., the points inside the domain, which are used to maintain the PDE dynamics minimizing the loss function defined through the equation without initial or boundary conditions, see, e.g., [20]).

However, in practice the information about the solution of the PDE can be very low: only initial and boundary conditions on a limited (e.g., onedimensional) domain. In this case, too large set of collocation points can lead to a bias of the solution into the dynamics of the PDE without taking into account real values of the solution and, thus, to a small accuracy with respect to the exact solution. Thus, deterministic methods can be used with the PINNs to obtain additional high precision data to train or tune the deep learning models (see, e.g., [20]). As it has been shown in [16], high precision derivatives for PDEs can also improve the performance of the simulation without necessity of elaboration of complex analytical formulae. In this paper, to overcome the above mentioned issues related to lack of data, new differentiation techniques are proposed for the time-dependent PDEs. The proposed schemes allow to generate much more data about the solution of the PDE giving so much more additional information to the neural networks for data-driven solution.

PINNs are used usually to solve two kinds of problems related to differential equations: data-driven solution and data-driven discovery (see, e.g., [20]). More specifically, there can be considered the following nonlinear Partial Differential Equation (PDE):

$$U_t = N[U, \lambda], \quad U = U(t, x), \quad t \in [t_0, t_{end}], \quad x \in D, \quad (1)$$

where the data-driven solution consists of approximation of the solution $U(t, x)$ using the data points on the initial and boundary conditions given all the parameters λ of the model, while the discovery problem consists of approximation of the model parameters λ given the values of $U(t, x)$ at some points. In this paper, the data-driven solution problem is considered.

Data-driven solution consists of approximation of the unknown solution $U(t, x)$ by neural networks using the (noisy) observations of the function $U(t, x)$. The loss function, usually, is defined by combinations of the given data and collocations points: e.g., as $L(t, x) = \|V(t, x) - U(t, x)\|_{\Gamma_u} + \|f_V(t, x)\|_{\Gamma_f}$, where $U(t, x)$ and $V(t, x)$ are the real and approximated solutions, respectively, $f_V(t, x) = V_t - N[V, \lambda]$, while $\|\cdot\|_{\Gamma}$ is some norm (e.g., Euclidean) over the set Γ , which consists of the respective values (t, x) used to define the loss function. Collocation points $(t, x) \in [t_0, t_{end}] \times D$ can be chosen as many as needed, since they are used only to guarantee the PDE (1) for the approximated solution $V(t, x)$ without taking into account real values of $U(t, x)$. Differentiation of

$V(t, x)$ can be performed, e.g., using Automatic Differentiation (for instance, by Tensorflow). Known observations of $U(t, x)$ can be given, frequently, by initial condition and/or boundary conditions (in the latter case, the respective loss function is also defined). Sometimes, the known observations of the solution $U(t, x)$ can be given as a set of (noisy) measurements without any analytical and/or software code. In this paper, we consider the case when the initial value condition is given by a function $g(x)$, $x \in D$, which can be also given by computational procedures, e.g., as a “black box” without analytical formulae.

It becomes clear, for the above mentioned issues, that the availability of the high precision data is crucial for the accuracy of the solution to PDEs using PINNs. However, in the case when there is no additional data, except the initial and boundary conditions, it can be difficult to tune the neural network to produce the results with high precision. In that case, the available data is limited only by the values at $t = t_0$ and, probably, by boundary conditions, which cannot give all necessary information about the solution (e.g., its dynamics around the initial time at $t > t_0$), even if some additional information can be obtained using PINNs along with deterministic algorithms for PDEs (see, e.g., [20]). Derivatives of $U(t, x)$ at the initial time t_0 can provide an important information, which can be used to tune better the neural networks: e.g., adding another loss function on the derivatives $L_d(t, x) = \sum_{i=1}^n \|\frac{\partial^i V}{\partial t^i}(t_0, x) - \frac{\partial^i U}{\partial t^i}(t_0, x)\|_{\Gamma_x}$, or trying to obtain a higher precision approximation of $U(t, x)$ around t_0 (e.g., by the Taylor expansion) to use it for fine-tuning the network.

However, since the initial condition is given only at the fixed time t_0 , it is impossible to differentiate it directly, e.g., by Automatic Differentiation, to obtain high precision derivatives. In the papers [10, 11], several algorithms for computation of the higher order derivatives with respect to the time variable have been proposed for Ordinary Differential Equations (ODEs). There are two most important drawbacks regarding the algorithms presented there. The first one is related to the using of the right-side function of the ODE for computation of the derivatives. In particular, the function $f(y)$ from the (autonomous) ODE $y'(t) = F(y)$ should be available to be calculated at any concrete fixed numerical value of y . However, this is not the case of PDEs, since the right-side function $N[U, \lambda]$ can depend also on the derivatives U_x and U_{xx} of the (unknown) solution $U(t, x)$ and cannot be calculated for a fixed numerical value of U . For this reason, the algorithms proposed in [10, 11] cannot be used directly to calculate the higher order derivatives of U with respect to t .

The second issue in the above mentioned algorithms consists in the using of a software for working with infinite and infinitesimal quantities (e.g., Levi-Civita field [7, 25], the Infinity Computer [23, 24], Nonstandard analysis [22], etc.). The Infinity Computer was used in the above mentioned papers for dealing with infinite and infinitesimal quantities. It can be described briefly as the computer system, which is based on the positional numeral system with the infinite radix $\textcircled{1}$ (defined as the number of elements of the set of natural numbers, i.e., $\textcircled{1} = |\mathbb{N}|$, see, e.g., [24], section 4.1). A number C on the Infinity Computer is represented

as follows (see [24], section 4.3):

$$C = c_0 \mathbb{1}^{p_0} + c_1 \mathbb{1}^{p_1} + \dots + c_n \mathbb{1}^{p_n}, \quad (2)$$

where $c_i \neq 0$, $i = 0, \dots, n$, are finite floating-point numbers called grossdigits, while p_i , $i = 0, \dots, n$, are called grosspowers and can be finite, infinite¹, and infinitesimal of the same form (2) presented in the decreasing order: $p_0 > p_1 > \dots > p_n$. However, according to its description in the papers [11, 24] and patents [23], it can work only under the standard floating-point arithmetic, i.e., only with finite floating-point grossdigits. For this reason, it is difficult to use this methodology to work with neural networks, e.g., using Tensorflow, where the approximations are made using tensors instead of single points x .

In this paper, a new technique is proposed to calculate the derivatives of the unknown solution $U(t, x)$ at the initial time t_0 . The algorithms require using of the software for working with infinitesimal quantities. The required software solution, which can be used not only with floating-point numbers, but with tensors and/or even symbolic expressions, is also presented. The proposed techniques are incorporated with data-driven solution to PDEs using PINNs by Tensorflow.

The rest of the paper is structured as follows. Section 2 describes the software solution used to deal with infinitesimal numbers and the respective arithmetic. Section 3 presents new differentiation techniques embedded in data-driven solution of PDEs. Section 4 presents the results of the proposed techniques for several real-life problems with known solutions and derivatives. Section 5 presents the results of application of the proposed schemes for data-driven solution by PINNs defined through the DeepXDE software for several real-life problems. Finally, Section 6 concludes the paper.

2 Developed software solution for infinitesimal quantities

There exist several different approaches to work numerically with infinite and infinitesimal numbers: Non-Archimedean fields (e.g., Levi-Civita field, see [7, 25]), the Infinity Computer (see, e.g., [24]), Dual and Hyper-Dual numbers (see, e.g., [27]), Labeled Sets and Numerosities (see, e.g., [3]), etc. All these methodologies are successfully used in different fields, in particular, in differentiation (see, e.g., [7, 11, 16]). Each of them has its own advantages and limitations, which are briefly described in [16], and their detailed comparison is out of scope of the present paper.

All the approaches listed above can be briefly described as follows. A number C representable in a framework for working numerically with infinite and

¹It should be noted that only finite grosspowers are used so far at all sources related to numerical computations, since there is not defined a procedure to compare different numbers with infinite and/or infinitesimal grosspowers as it has been shown in [9], Fact 6.11. For this reason, the versions of the Infinity Computer described in the references cited above are similar to the Levi-Civita field implementations, see, e.g., [25] and [11]

infinitesimal numbers can be defined as

$$C = c_0 H_{p_0} + c_1 H_{p_1} + \dots c_n H_{p_n}, \quad (3)$$

where c_i , $i = 0, \dots, n$, are standard floating-point numbers, while H_{p_i} , $i = 0, \dots, n$, are finite, infinite, or infinitesimal numbers in the form $H_p = h_1 h_2 \dots h_k$, where k is the number of “axis” of the approach (e.g., $k = 1$ and $H_p = \mathbb{1}^p$ for the Infinity Computer, $k = 1$ and $H_p = \varepsilon^p$ for the Levi-Civita field, $k > 1$ and $H_p = \varepsilon_1^{q_1} \varepsilon_2^{q_2} \dots \varepsilon_k^{q_k}$, $q_1 + q_2 + \dots q_k = p$, for hyper-dual numbers). For instance, a number C can be representable in the Levi-Civita field as follows:

$$C = c_0 \varepsilon^{p_0} + c_1 \varepsilon^{p_1} + \dots + c_n \varepsilon^{p_n}, \quad (4)$$

where $c_i \neq 0$ and p_i , $i = 0, 1, \dots, n$, are finite floating-point numbers, p_i are ordered in increasing order: $p_0 < p_1 < \dots < p_n$ (see, e.g., [26]). Since in this paper, only derivatives with respect to t are considered, then there is no sense to use hyper-dual numbers, which can be useful for a mixed differentiation due to different “axis” of the infinitesimal base. The Infinity Computer with finite powers of $\mathbb{1}$ can also be considered as a particular case of Non-Archimedean fields, e.g., of the Levi-Civita field, since it is easy to see that one can obtain the number in the Infinity Computing form (2) just by fixing $\varepsilon = \mathbb{1}^{-1}$ in (4). However, it should be noted that, in general, the Infinity Computer and the Levi-Civita field are two different approaches (see, e.g., [24], footnote 10). For this reason, in this paper, the Levi-Civita field is considered as the main used approach for dealing with infinite and infinitesimal numbers and no further considerations refer to the Infinity Computer methodology.

However, since, usually, in data-driven solution, the main data types are tensors instead of single floating-point numbers, then it can be useful to adapt the methodology to work efficiently with tensors. In particular, it is required to represent a tensor G in the following form:

$$G = G_0 \varepsilon^{p_0} + G_1 \varepsilon^{p_1} + \dots + G_n \varepsilon^{p_n}, \quad (5)$$

where G_i , $i = 0, \dots, n$, are tensors of the same dimensions, while p_i , $i = 0, \dots, n$, are floating-point numbers written down in a strictly increasing order. Obviously, the tensor G can be also written down in different formats, e.g., as

$$G = G_{\varepsilon,0} + G_{\varepsilon,1} + \dots + G_{\varepsilon,n},$$

where $G_{\varepsilon,i}$ is a tensor with every element of the form (4). However, in this case, it can be difficult to use efficiently modern frameworks for working with tensors (e.g., Tensorflow <https://www.tensorflow.org/> or Pytorch <https://pytorch.org/>), since the data types within each tensor are re-defined, while the form (5) allows to use the advantages of such the frameworks within the Levi-Civita software solution, because it is much simpler to re-define just four main arithmetic operations, than to create another efficient framework for dealing with tensors.

In this paper, a simplified software solution for working with finite and infinitesimal numbers similar to the Levi-Civita field has been developed. It is

based on the Python list comprehensions² and every quantity C in this framework is represented as follows:

$$C = C_0\varepsilon^0 + C_1\varepsilon^1 + \dots + C_n\varepsilon^n, \quad (6)$$

where C_i , $i = 0, \dots, n$, are quantities of the same type and dimensions as C (and can be equal to zero of the same type). Standard finite tensors C_{finite} can be represented in this form simply by $C_0 = C_{finite}$ and $C_i = 0$ for all $i > 0$. In other words, the number C can be represented simply by the vector of the coefficients $[C_0, C_1, \dots, C_n]$. Arithmetic operations between the quantities can be simply defined as follows.

Let A and B be two quantities of the same dimensions in the form (6) represented by the vectors $[A_0, \dots, A_n]$ and $[B_0, \dots, B_n]$, respectively. Then, $A+B$ is another number of the form (6) represented by the vector $[A_0 + B_0, A_1 + B_1, \dots, A_n + B_n]$. The subtraction is defined in the same way, thus, its definition is omitted.

The result of multiplication $A \times B$ can also be obtained and represented by the following vector:

$$[A_0 \times B_0, A_0 \times B_1 + A_1 \times B_0, A_0 \times B_2 + A_1 \times B_1 + A_2 \times B_0, \dots, A_n \times B_n],$$

where $A_i \times B_k$ is the element-wise multiplication in the case of tensors. Division can also be defined in a similar way, so its definition is omitted. It should be noted that both multiplication and division should truncate the results after a given number of elements (e.g., after n -th element) to preserve the same number of elements within the vector.

Two important and necessary operations can also be added: multiplication and division by the infinitesimal ε^k , $k > 0$. These operations just result in shifting the coefficients to the right (multiplication) or to the left (division). In particular, if A is represented by $[A_0, A_1, \dots, A_n]$, then $A \times \varepsilon^k$ is represented by $[0, \dots, 0, A_0, A_1, \dots, A_{n-k}]$, where the first k elements are equal to zero of the same type as A (e.g., zero tensor), and the last elements are truncated to preserve the same dimension of the array. Similarly, A/ε^k is represented by $[A_k, A_{k+1}, \dots, A_n, 0, \dots, 0]$, where the first k elements have been truncated, while the last k are set to 0. It should be noted that, similarly to fixed-point arithmetic, the presented methodology cannot be used for general purpose problems due to truncations resulting in above mentioned multiplications and divisions. However, this methodology is sufficient for illustration purposes to deal with differentiation problems, where there is no necessity of working with infinite quantities. For more complex problems, there should be constructed a software, e.g., similar to the floating-power simulator from [11], adapted to work with tensors.

It is important to note that there should not arise infinite values in differentiation, if the functions under consideration are differentiable. However, there can

²It is clear that faster tools can be used instead of lists, e.g., Numba. Python lists are used for illustration purposes only.

be necessary to hold intermediate infinite values: for example, if $g(x) = 1.5x^2$, then $g(\varepsilon) = 1.5\varepsilon^2$, which is infinitesimal. If $g(x)$ is constructed through complex computational procedures, there can be a situation when $g(x)$ is defined as, e.g., $g(x) = (3x^2/2x^4)/x^{-4}$, where $g(\varepsilon) = (3\varepsilon^2/2\varepsilon^4)/\varepsilon^{-4} = 1.5\varepsilon^{-2}/\varepsilon^{-4} = 1.5\varepsilon^2$, i.e., it is necessary to work with infinite values ε^{-4} and ε^{-2} as well. In this case, it is possible to add also several negative powers of ε in (6) and to reserve several elements in the respective vector form. In this paper, however, it is supposed that all functions are well-defined and there won't arise situations of this kind just for simplicity.

3 Proposed differentiation methods for PDEs

Let us consider the following nonlinear PDE:

$$U_t = F(U, U_x, U_{xx}, t, x), \quad U = U(t, x), \quad t \in [t_0, t_{end}], \quad x \in D \subset \mathbb{R}^N, \quad (7)$$

with the following initial conditions:

$$U(t_0, x) = g(x), \quad x \in D, \quad (8)$$

and some boundary conditions. Here, $U(t, x)$ is the unknown solution of the PDE (possibly, multidimensional vector function $U : \mathbb{R}_+ \times \mathbb{R}^N \rightarrow \mathbb{R}^M$, $M \geq 1$), t and x are independent variables denoted as time and spatial variables, respectively, $t_0 < t_{end}$ are two real-valued numbers, D is the spatial domain, e.g., a hyperinterval $D = [a, b] \subset \mathbb{R}^N$. Without loss of generality, let us consider $t_0 = 0$ and $t_{end} = T$, where $T \in \mathbb{R}$, $T > 0$. Let us also consider the univariate case $N = 1$, i.e., the variable x is univariate, just for simplicity and without loss of generality (it will be easy to see that the algorithms are easily generalizable to the case $N > 1$).

Note also that the equation of the higher than the first order with respect to time can also be transformed into (7) by increasing the dimension and substituting $U_t = V$, $V_t = F(U, V, U_x, V_x, U_{xx}, V_{xx}, t, x)$ (for the second order equation $U_{tt} = F(U, U_x, U_{xx}, t, x)$), adding so additional functions and equations (see the Wave equation as an example in the next section).

In the present paper, the problem of numerical computation of the higher order derivatives $\frac{\partial^k U}{\partial t^k}$, $k = 1, 2, 3, \dots$, of $U(t, x)$ at the initial time $t = 0$ is considered. The boundary conditions are not used in the presented algorithms, thus they can be given in any format (e.g., Dirichlet, Neumann, Cauchy, etc.).

The main idea of the proposed algorithm consists of using the initial solution $U(0, x) = g(x)$ for reconstruction of the Taylor expansion with respect to t of $U(t, x)$ around $t = 0$ simultaneously for all available (fixed) values $x = X$ (e.g., generating k values $x_1, x_2, \dots, x_k$ randomly within D and then sending them to $g(x)$ in tensor form X using Tensorflow). The equation (7) at $t = 0$ becomes:

$$U_t|_{t=0, x=X} = F(g(X), g'(X), g''(X), 0, X), \quad (9)$$

If the explicit formulae of $g(x)$ are available, then it is possible to calculate its derivatives with respect to x by any available method (e.g., analytically,

symbolically, or by automatic differentiation). Otherwise, if the analytical formulae are not available, the derivatives of $g(x)$ can be obtained by automatic differentiation or by other techniques, e.g., using hyper-dual numbers.

The equation (9) allows to obtain the first derivative U_t at $t = 0$ and $x = X$. This additional information allows to generate the first-order Taylor expansion of $U(t, X)$ for small t . In particular, given the infinitesimal value h , the first-order approximation u_1 of $U(h, X)$ can be obtained as follows

$$u_1 = U(0, X) + U_t|_{t=0, x=X} \cdot h = g(X) + F(g(X), g'(X), g''(X), 0, X) \cdot h. \quad (10)$$

After computation of u_1 , one can also calculate the value $F(u_1, u'_1, u''_1, h, X)$, where u'_1 and u''_1 denote, respectively, the first and the second derivatives of u_1 with respect to x at $t = 0$ and $x = X$. These two derivatives can be again calculated using, e.g., automatic differentiation of u_1 , since it is calculated from $F(g(X), g'(X), g''(X), 0, X)$, which, in its turn, is calculated from the function $g(X)$, for which the automatic differentiation can be applied.

Since the value h is represented in the form (6) (or, more generally, in the form (4)), then u_1 and $F(u_1, u'_1, u''_1, h, X)$ are also written down in the same form. Just for simplicity, hereinafter, the value $h = \varepsilon$ from (6) represented by the vector $[0, 1, 0, \dots, 0]$ is used. In this case, u_1 is already defined in this form with the tensors $C_0 = g(X)$, $C_1 = F(g(X), g'(X), g''(X), 0, X)$ and all C_i , $i > 1$, are zero tensors, while the value $F(u_1, u'_1, u''_1, h, X)$ will be also written down in powers of ε :

$$F(u_1, u'_1, u''_1, h, X) = F_{1,0}\varepsilon^0 + F_{1,1}\varepsilon^1 + \dots + F_{1,n}\varepsilon^n, \quad (11)$$

where $F_{1,i}$ are tensors of the same dimension. It is easy to extract the value $F_{1,1}$ from (11), that is the first-order Lie derivative of $F(\dots)$ with respect to t at $t = 0$ and $x = X$. Since the function F defines the first-order derivative of $U(t, x)$ at $t = 0$, $x = X$, then its first-order Lie derivative defines the second-order time derivative of U . For this reason, it is possible to re-construct the Taylor expansion of U up to the second derivative obtaining so the value u_2 as follows:

$$u_2 = U(0, X) + U_t|_{t=0, x=X} \cdot h + \frac{1}{2!}F_{1,1} \cdot h^2 = u_1 + \frac{1}{2!}F_{1,1} \cdot h^2, \quad (12)$$

Since $h = \varepsilon$ and represented by $[0, 1, 0, \dots, 0]$, then $h^2 = \varepsilon^2$ is represented by the vector $[0, 0, 1, 0, \dots, 0]$. Thus, the respective vector representation of u_2 can be also obtained: $[g(X), F(g(X), g'(X), g''(X), 0, X), \frac{1}{2}F_{1,1}, 0, \dots, 0]$, i.e., the first two coefficients C_0 and C_1 are the same as in u_1 , the coefficient C_2 is equal to $\frac{1}{2}F_{1,1}$ and the rest is zero. It should be noted that in (10) and (12), the value h is used instead of explicit ε just to distinguish the infinitesimal *number* h represented by the vector $[0, 1, 0, \dots, 0]$ in the form (6) and the infinitesimal base ε , which is used in (6) and is not explicitly stored in the computer memory.

Given the value u_2 , again, the value $F(u_2, u'_2, u''_2, h, X)$ can be calculated:

$$F(u_2, u'_2, u''_2, h, X) = F_{2,0}\varepsilon^0 + F_{2,1}\varepsilon^1 + \dots + F_{2,n}\varepsilon^n, \quad (13)$$

from where, similarly, the coefficient $F_{2,2}$ can also be extracted, which represents the second-order Lie derivative of F divided by $2!$, i.e., the third-order time derivative of U at $t = 0$ and $x = X$. Extracting $F_{2,2}$ from (13) and multiplying it by $2!$, one can obtain the third-order approximation u_3 : $u_3 = u_2 + \frac{1}{3!} \cdot 2! \cdot F_{2,2} \cdot h^3 = u_2 + \frac{1}{3} F_{2,2}$.

It is easy to see that the coefficients $F_{2,0}$ and $F_{2,1}$ coincide with the values $F_{1,0}$ and $f_{1,1}$, respectively, since the first two coefficients in u_1 and u_2 coincide. Every iteration of the procedure described above improves the order of the approximation of $U(h, X)$ and allows to obtain the required derivatives of $U(t, x)$ at $t = 0$ and $x = X$, so it becomes easy to define the Algorithm 1.

Algorithm 1 Differentiation of $U(t, x)$ from (7) w.r.t. time t

Input

- 1: The problem (7) with the initial condition (8) linked to an automatic differentiation technique (or any other similar) with respect to x .
- 2: The set (e.g., in tensor form) $X \subset D$ of fixed points within the domain D .
- 3: The software for working with infinitesimal numbers (e.g., the presented Levi-Civita field solution for the numbers from (6)) and the respective number n of infinitesimal parts to work with from (6) or from (4) in general.
- 4: The infinitesimal $h = 1\varepsilon^1$ in the respective vector form $[0, 1, 0, \dots, 0]$.

Result

- 5: The tensor of the derivatives $\frac{\partial^i U}{\partial t^i}(t, x) |_{t=0, x=X}$, $i = 0, 1, \dots, K$, $K \leq n$.

Main procedure

- 6: $u \leftarrow g(X) + F(g(X), g'(X), g''(X), 0, X) \cdot h$
 - 7: $u' \leftarrow g'(X)$, $u'' \leftarrow g''(X)$
 - 8: **for** $i = 2, 3, \dots, K$ **do**
 - 9: Calculate the first and second order derivatives u'_i and u''_i of the coefficient of ε^{i-1} at u with respect to x at $x = X$:
 - 10: $u' = u' + u'_i \varepsilon^i$,
 - 11: $u'' = u'' + u''_i \varepsilon^i$
 - 12: Calculate $F(u, u', u'', h, X)$ up to the coefficient of ε^{i-1} :
 - 13: $F(u_i, u'_i, u''_i, h, X) = F_0 \varepsilon^0 + F_1 \varepsilon^1 + \dots + F_{i-1} \varepsilon^{i-1}$.
 - 14: Save the derivative $\frac{\partial^i U}{\partial t^i}(t, x) |_{t=0, x=X} \leftarrow (i-1)! \cdot F_{i-1}$.
 - 15: Update $u \leftarrow u + \frac{1}{i} \cdot F_{i-1} \cdot h^i$.
 - 16: **end for**
 - Return** the obtained derivatives $\frac{\partial^i U}{\partial t^i}(t, x) |_{t=0, x=X}$ or the Taylor expansion u of $U(t, x)$ at $t = h$ and $x = X$.
-

Remark 1. Since the values in the form (6) can be represented by the vectors, then it is not necessary to re-calculate the derivatives of the coefficients with respect to x of the coefficients of ε^l in u for $l \leq i$: they are already calculated at the previous steps. The vector form of the values u , u' and u'' is very useful here for this reason (only the last component is differentiated and updated every time).

Remark 2. There is also produced the Taylor expansion $u = C_0 \varepsilon^0 + C_1 \varepsilon^1 +$

$\dots + C_K \varepsilon^K$ of $U(h, X)$, where $C_0 = g(X)$ and $C_i = \frac{(i-1)!}{i!} F_{i-1}$, $i = 1, \dots, K$, which can also be useful in practice for several reasons. Since, u is represented by the vector, then it is easy to substitute the value ε by any other finite or infinitesimal value s obtaining so different approximations of $U(s, X)$ for different s close to 0. For instance, having the Taylor expansion u up to the 10-th derivative, it is possible to calculate the value $U(0.1, X)$ just multiplying element-wise the vector $[C_0, C_1, \dots, C_K]$ by the vector $[1, 0.1, 0.1^2, \dots, 0.1^K]$. This issue allows to obtain high precision approximations of U for $t > 0$ generating so additional data values and improving the performance of the respective PINNs. In the next sections, there will be presented several examples of such the improvements.

Remark 3. The accuracy can be lower for the highest derivatives (e.g., of the order 10), if the absolute value of the derivative is high (see, e.g., the Wave equation in the next section with large values of the parameter C). This occurs due to the limitations of the floating-point arithmetic and due to the use of factorials in the computation of the derivatives: the i -th derivative of U is obtained multiplying the coefficient F_{i-1} by $(i-1)!$ (see the lines 11-12 of the Algorithm 1). This issue will be studied in the future work in order to develop a more efficient algorithm of computations without involving the factorials and, thus, without losing the accuracy even for the highest derivatives.

It should be noted that the proposed techniques are referenced with the initial condition $U(0, x) = g(x)$. However, in data-driven solution, the initial condition function $g(x)$ can be unknown. The set of values $U(0, x_i) = g_i$ can be given instead. In this case, the algorithms presented in this paper can also be used, e.g., as follows. First, the function $g(x)$ can be approximated by another differentiable neural network using both initial and boundary conditions. Then, the derivatives with respect to x (required to calculate the derivatives with respect to t) can be calculated, e.g., using Tensorflow Autodiff, allowing so to calculate the time derivatives as well (time derivatives cannot be calculated in the same way by Autodiff, since the initial conditions are defined for only one time value $t = 0$).

4 Numerical experiments on benchmarks with known solutions

Numerical experiments have been carried out in Jupyter notebooks using Google Colab Pro. The software has been written in Python 3.7.14. Autodiff of Tensorflow (version 2.8.2) has been used for the computation of the derivatives of U with respect to x .

For each test problem, the derivatives up to the order 10 have been calculated $\frac{\partial^i U}{\partial t^i}(t, x)$, $i = 0, 1, \dots, 10$, at the time $t_0 = 0$ on 50 different points x_i chosen uniformly within the interval D excluding the points, where the solution $U(t_0, x)$ is too small, just to avoid divisions by 0 in the relative error computations. All computations have been performed in tensor form for all 50 points simultaneously, i.e., $X = \text{tensorflow.Variable}(x, dtype = 'float64')$, where x is

the array of 50 different points.

Once all 10 derivatives have been obtained, the Normalized Root Mean Square Error (NRMSE) has been calculated for each derivative with respect to the exact derivatives (known analytically for each test problem):

$$NRMSE(y_{true}, y_{approx}) = \frac{1}{N} \frac{\|y_{true} - y_{approx}\|_2}{y_{true}^{MAX} - y_{true}^{MIN} + 1}, \quad (14)$$

As it has been already mentioned before, the error of the computation of the highest derivatives cannot be a unique metric to determine the quality of the obtained information. For this reason, the Taylor expansion at $t = 0.01, 0.05, 0.1$, has been also calculated using the obtained value u being the Taylor approximation of $U(\varepsilon, X)$ around $t = 0$ and $x = X$. The NRMSE has been also calculated with respect to the exact value of the solution (the exact solution $U(t, x)$ is also known for each test problem).

4.1 Heat equation

Let us consider the following PDE from [21]:

$$U_t = \alpha U_{xx}, \quad x \in [0, L], t \in [0, 1], \quad (15)$$

with the periodic initial condition:

$$U(0, x) = \sin\left(\frac{n\pi x}{L}\right), \quad x \in [0, L], \quad n = 1, 2, \dots, \quad (16)$$

where $\alpha = 0.4$ is the thermal diffusivity constant, $L = 1$ is the length of the bar, $n = 1$ is the frequency of the sinusoidal initial conditions. The Dirichlet boundary conditions are given as follows:

$$U(t, 0) = U(t, L) = 0, \quad t \geq 0. \quad (17)$$

The exact solution of (15) is known:

$$U(t, x) = e^{\frac{-n^2\pi^2\alpha t}{L^2}} \cdot \sin\left(\frac{n\pi x}{L}\right), \quad (18)$$

from where it is also possible to calculate the exact derivatives of U with respect to time:

$$\frac{\partial^i U}{\partial t^i}(t, x) = \left(\frac{-n^2\pi^2\alpha}{L^2}\right)^i \cdot U(t, x). \quad (19)$$

The Heat equation (15) is quite simple, but allows to study well the proposed algorithm. The respective values of the NRMSE for the first 10 derivatives are presented in Table 1 (column Heat). As it can be seen, the obtained errors are of the same order as the machine precision, giving so a lot of additional information about the dynamics of the model.

The NRMSE values of the Taylor expansion at the time $t_1 = 0.01, 0.05, 0.1$ and the same X have been also calculated with respect to the exact solution.

Diff. order	Heat	Diffusion Derivatives	Diffusion Taylor coeff.	Wave
0	0	0	9.23e-30	0
1	1.70e-16	2.33e-16	2.33e-16	0
2	3.00e-16	2.89e-15	1.45e-15	2.37e-16
3	3.55e-16	4.69e-14	7.82e-15	0
4	3.98e-16	4.37e-13	1.82e-14	3.10e-16
5	4.23e-16	3.97e-12	3.31e-14	0
6	9.77e-16	2.46e-11	3.42e-14	3.32e-16
7	6.88e-16	4.20e-10	8.34e-14	0
8	6.47e-16	2.65e-09	6.57e-148	4.01e-16
9	8.05e-16	3.06e-08	8.42e-14	0
10	6.61e-16	3.61e-07	9.94e-14	4.31e-16

Table 1: The obtained NRMSE values for the first 10 derivatives of the Heat equation, Diffusion Equation (including the Taylor coefficients NRMSE) and Wave Equation. All errors are of the same order than the machine precision ($\approx 2.2 \cdot 10^{-16}$), except the Diffusion equation, for which the Taylor coefficients are accurate instead.

One can see from Table 2 (column Heat) that the results have been calculated with a high accuracy (e.g., the NRMSE at $t_1 = 0.01$ is smaller than the machine precision, while the one at $t_1 = 0.05$ is of the same order than the machine precision) giving so additional information about the solution $U(t, x)$, which was unknown a priori. The graph of both the obtained Taylor solution and the exact analytical one at $t = 0.1$ for 500 uniform points $x \in D$ is presented in Figure 1(a).

t_1	Heat	Diffusion	Wave
0.01	7.75e-17	3.03e-17	1.08e-16
0.05	3.63e-16	3.25e-17	1.01e-16
0.1	6.63e-13	4.22e-17	2.16e-15

Table 2: The obtained NRMSE values for the Taylor expansion of the Heat, Diffusion and Wave equations at $t_1 = 0.01, 0.05, 0.1$. The obtained approximations have a high accuracy being so useful candidates for an additional data for PINNs.

4.2 Diffusion equation

Let us consider the following PDE from [21]:

$$U_t = U_{xx} - e^{-t}(\sin(\pi x) - \pi^2 \sin(\pi x)), \quad x \in [-1, 1], t \in [0, 1], \quad (20)$$

with the initial condition:

$$U(0, x) = \sin(\pi x), \quad x \in [-1, 1]. \quad (21)$$

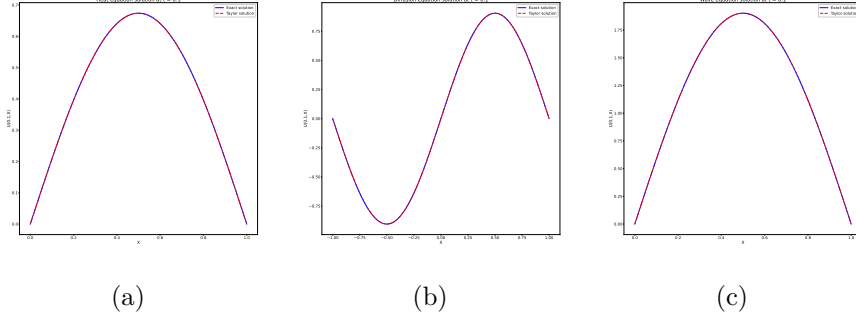


Figure 1: The obtained Taylor approximation using 10 derivatives (red) and the exact solution (blue) of the Heat, Diffusion and Wave equations at $t = 0.1$. Both the solutions coincide up to a very small error for all three test problems.

The Dirichlet boundary conditions are given as follows:

$$U(t, -1) = U(t, 1) = 0, \quad t \geq 0. \quad (22)$$

The exact solution of (15) is known:

$$U(t, x) = e^{-t} \cdot \sin(\pi x), \quad (23)$$

from where it is also possible to calculate the exact derivatives of U with respect to time:

$$\frac{\partial^i U}{\partial t^i}(t, x) = (-1)^i \cdot U(t, x). \quad (24)$$

The equation (20) is a bit more complicated, than the previous one, due to the presence of the arguments t and x explicitly in the function F . For this reason, it also becomes an important illustrative example for the proposed algorithm. The respective values of the NRMSE for the first 10 derivatives are presented in Table 1 (columns Diffusion Derivatives and Diffusion Taylor coeff.). As it can be seen, the accuracy decreases with the order of the derivatives. This can be easily explained by the inefficient use of the factorials and by the limitations of the floating-point arithmetic.

The latter fact can be substantiated by the following computations. Let us study directly the coefficients C_i of the Taylor expansion u from the Algorithm 1 without multiplying them by the respective factorials, i.e., for $h = \varepsilon$ in Algorithm 1: $u = C_0\varepsilon^0 + C_1\varepsilon^1 + \dots + C_n\varepsilon^n$, where C_i is the tensor of the respective dimension (equal to the i -th time derivative divided by $i!$ at $t = 0$ and $x = X$). Let us then calculate the NRMSE for each coefficient C_i with respect to the exact values $\frac{1}{i!} \frac{\partial^i U}{\partial t^i}(0, X)$ (all these computations are made using decimal type to maintain the precision). The results are presented in the last column of Table 1.

One can see that the accuracy in computation of the derivatives was lost just due to the limitations of the floating-point numbers, while the accuracy

in direct coefficients of the Taylor expansion is much higher. The latter fact can also be substantiated by Table 2 (column Diffusion), where the NRMSE values of the Taylor expansion at the time $t_1 = 0.01, 0.05, 0.1$ and the same X have been also calculated with respect to the exact solution. One can see from Table 2 that the results have been calculated up to machine precision at all three points giving so a lot of additional information about the solution $U(t, x)$, which was unknown a priori also in this case. The graph of both the obtained Taylor solution and the exact analytical one at $t = 0.1$ for 500 uniform points $x \in D$ is presented in Figure 1(b).

4.3 Wave propagation equation

Let us consider the following second-order PDE from [28]:

$$U_{tt} = C^2 \cdot U_{xx}, \quad x \in [0, 1], t \in [0, 1], \quad C > 0, \quad (25)$$

with the initial condition:

$$\begin{aligned} U(0, x) &= \sin(\pi x) + \sin(A\pi x), \quad x \in [0, 1], \quad A > 0, \\ U_t(0, x) &= 0. \end{aligned} \quad (26)$$

The Dirichlet boundary conditions are given as follows:

$$U(t, 0) = U(t, 1) = 0, \quad t \geq 0. \quad (27)$$

The exact solution of (15) is known:

$$U(t, x) = \sin(\pi x) \cos(C\pi t) + \sin(A\pi x) \cos(AC\pi t), \quad (28)$$

from where it is also possible to calculate the exact derivatives of U with respect to time:

$$\frac{\partial^i U}{\partial t^i}(t, x) = (C\pi)^i \sin(\pi x) f_1(t) + (AC\pi)^i \sin(A\pi x) f_2(t), \quad (29)$$

where

$$\begin{cases} f_1(t) = \cos(C\pi t), \quad f_2(t) = \cos(AC\pi t), & \text{if } i \% 4 = 0, \\ f_1(t) = -\sin(C\pi t), \quad f_2(t) = -\sin(AC\pi t), & \text{if } i \% 4 = 1, \\ f_1(t) = -\cos(C\pi t), \quad f_2(t) = -\cos(AC\pi t), & \text{if } i \% 4 = 2, \\ f_1(t) = \sin(C\pi t), \quad f_2(t) = \sin(AC\pi t), & \text{if } i \% 4 = 3, \end{cases} \quad (30)$$

The main difference of this problem with respect to the previous ones consists of the order of the equation (25). Since the Algorithm 1 has been proposed for the first order equations only, then it should be transformed increasing so the dimension of the problem $Y_t = F(Y, Y_x, Y_{xx}, t, x)$, where $Y = (U, V)$, $V = U_t$, i.e. $Y : \mathbb{R}_+ \times \mathbb{R}^N \longrightarrow \mathbb{R}^2$:

$$\begin{cases} U_t = V, \\ V_t = C^2 \cdot U_{xx}, \end{cases} \quad (31)$$

and the respective initial and boundary conditions.

Here, the values of the parameters A and C are fixed equal both to 1 just for simplicity. The obtained results are presented in Table 1 (column Wave). One can see that the derivatives have been calculated with a high accuracy (of the same order than the machine precision). It should be noted that the errors for the odd orders are equal to zero, since $f_1(t)$ and $f_2(t)$ in (25) are both equal to zero at $t = 0$.

The NRMSE values of the Taylor expansion at the time $t_1 = 0.01, 0.05, 0.1$ and the same X have been also calculated with respect to the exact solution. One can see from Table 2 that the results have been again calculated with a high accuracy giving so a lot of additional information about the unknown solution $U(t, x)$. The graph of both the obtained Taylor solution and the exact analytical one at $t = 0.1$ for 500 uniform points $x \in D$ is presented in Figure 1(c).

5 Applications for PINNs

As it can be seen from the previous section, the proposed algorithm allows to obtain a new information about the dynamics of the model described by the PDE. Let us study how this new information allows to improve the quality of the PINNs for data-driven solution of the PDEs. The schemes proposed in this section are not unique, but just the simplest ones allowing already to increase the accuracy of the solution.

First of all, the PINNs from the well-known DeepXDE library written in Python with Tensorflow (see, <https://github.com/lululxvi/deepxde> and [17]) have been used to solve the PDEs. The library DeepXDE has been chosen for its simplicity, since it allows to define different types of the initial and boundary conditions without changing the code of the PINNs. However, the proposed techniques can also be used for other libraries, e.g., from [12].

For each test problem, the first N derivatives at $t = 0$ for 100 randomly chosen points x have been calculated. Then, the Taylor approximations using the obtained derivatives have been calculated at $t = 0.01, 0.02, 0.03, 0.04$ and 0.05 for the same values x obtaining so 500 additional approximations of $U(t, x)$. The obtained additional values have been added as the Point set conditions into the DeepXDE solver. Comparison with two versions with and without the additional conditions has been then performed.

For each test problem, forward PINNs have been generated with the following parameters: 5 inner layers with 64 neurons each; 8000 collocation (domain) points for minimizing the loss function on the PDE, 400 points on the boundary conditions, 800 points at $t = 0$ by initial conditions. First, each model has been compiled using Adam optimizer with 15000 iterations (epochs). Then, the training continued with L-BFGS algorithm until stopping criterion. The L_2 norm has been calculated on 50 random points $x \in D$ at some time t_1 (chosen similarly for all test problems). All remaining parameters and options have been set to their default values. Loss functions have been set directly using the internal DeepXDE options (deepxde.icbc classes for the initial and boundary

conditions) without setting the formulae explicitly.

For each test problem, the PINN with the above mentioned parameters has been compiled and trained. Then, in order to study the usefulness of the proposed techniques, the PINN with the same parameters and additional conditions has been also defined. In particular, N first derivatives $\frac{\partial^i U}{\partial t^i}(0, X)$ have been calculated at 100 randomly chosen points X . Then, the Taylor expansions at $t = 0.01, 0.02, \dots, 0.05$, have been constructed using the calculated derivatives. The obtained 500 Taylor values have been added as additional conditions to the model as `deepxde.icbc.PointSetBC`. After that, the model has been compiled and trained in the same way with the same parameters. The results are presented for each test problems in the following subsections. The Python code for the main PINN implementation and visualizations has been taken from https://deepxde.readthedocs.io/en/latest/demos/pinn_forward.html. The obtained results and models configurations can be also found in https://github.com/maratsmuk/DiffPDE_PINN for transparency and illustration aims only.

5.1 Burgers Equation

The first problem under consideration was the well-known Burgers' equation (see, e.g., [2]).

$$\frac{\partial U}{\partial t} = -U \frac{\partial U}{\partial x} + \nu \frac{\partial^2 U}{\partial x^2}, \quad (32)$$

where $\nu = \frac{0.01}{\pi}$ is the constant and with the following initial and boundary conditions:

$$\begin{aligned} U(t, -1) &= U(t, 1) = 0, \\ U(0, x) &= -\sin(\pi x), \\ x \in [-1, 1], \quad t \in [0, 1], \quad U &= U(t, x) \in \mathbb{R}. \end{aligned} \quad (33)$$

First, the Dirichlet boundary conditions and standard initial conditions have been generated. The best train loss of the standard model has been obtained: $8.13e - 08$. The L_2 norm of the obtained solution at $t = 0.1$ and 50 random points x has also been calculated: $4.183e - 05$ with the mean residual 0.0002. The plot of the obtained solution is presented in Figure 2(a).

Then, the first 7 derivatives have been calculated at $t = 0$. The obtained Taylor approximations have been calculated for 100 random points x at $t = 0.01, 0.02, \dots, 0.05$. The obtained 500 values have been added as additional conditions to the model, which was re-initialized, compiled and trained with the same parameters. The final obtained loss has become $8.38e - 08$. The obtained L_2 norm at $t = 0.1$ has become better: $3.612e - 05$ with the mean residual 0.000227. The plot of the obtained solution is also presented in Figure 2(b). The obtained L_2 norm has become better with almost the same number of epochs (31321 epochs against 31347 for the standard scheme) showing so a promising behavior of the proposed scheme.

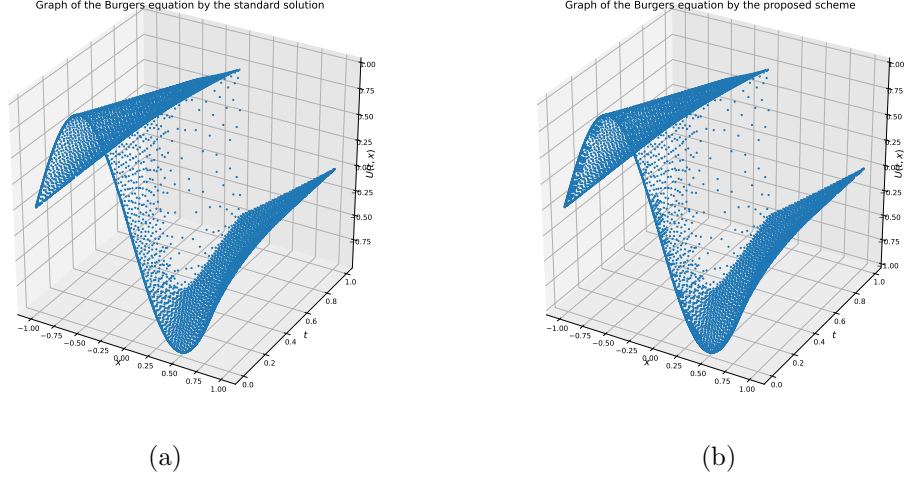


Figure 2: The obtained solutions by PINNs using the standard model (left) and with additional conditions by the proposed scheme (right) of the Burgers' equation.

5.2 Allen-Cahn Equation

The second problem under consideration was the well-known Allen-Cahn equation (see, e.g., [20]):

$$\frac{\partial U}{\partial t} = d \frac{\partial^2 U}{\partial x^2} + 5(U - U^3), \quad (34)$$

where $d = 0.0001$ and with the following initial and boundary conditions:

$$\begin{aligned} U(t, -1) &= U(t, 1), \\ U_x(t, -1) &= U_x(t, 1), \\ U(0, x) &= x^2 \cos(\pi x), \\ x &\in [-1, 1], \quad t \in [0, 1], \quad U = U(t, x) \in \mathbb{R}, \end{aligned} \quad (35)$$

First, the Periodic boundary conditions for both U and U_x and standard initial conditions have been generated. The best train loss of the standard model has been obtained: $6.45e - 04$. The L_2 norm of the obtained solution at $t = 0.1$ and 50 random points x has also been calculated: 0.0584 with the mean residual 0.0073. The plot of the obtained solution is presented in Figure 3(a).

Then, the first 7 derivatives have been calculated at $t = 0$. The obtained Taylor approximations have been calculated for 100 random points x at $t = 0.01, 0.02, \dots, 0.05$. The obtained 500 values have been added as additional conditions to the model, which was re-initialized, compiled and trained with the same parameters. The final obtained loss has become $7.37e - 06$. The obtained L_2 norm at $t = 0.1$ has become much better: 0.0017 with the mean residual

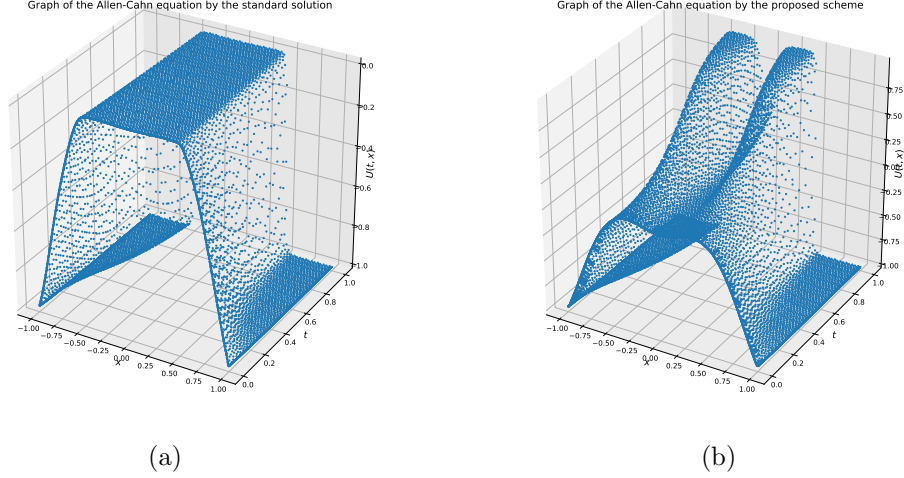


Figure 3: The obtained solutions by PINNs using the standard model (left) and with additional conditions by the proposed scheme (right) of the Allen-Cahn equation. One can see that the standard model failed to approximate the solution correctly, while additional data by the proposed scheme allowed to improve the accuracy a lot.

0.00146. The plot of the obtained solution is also presented in Figure 3(b). One can see from Figure 3 that the standard model was not able to approximate the solution $U(t, x)$ correctly within the given iterations. Moreover, the overfitting has clearly arisen around the initial solution. The proposed scheme, instead, has allowed to obtain a much better solution without overfitting, due to additional data obtained from the calculated derivatives.

5.3 Nonlinear Schrodinger Equation

Finally, the well-known Nonlinear Schrodinger equation has also been studied (see, e.g., [20]):

$$i \frac{\partial h}{\partial t} = -0.5 \frac{\partial^2 h}{\partial x^2} - |h|^2 h, \quad (36)$$

where h is a complex valued function and with the following initial and boundary conditions:

$$\begin{aligned} h(0, x) &= 2 \operatorname{sech}(x), \\ h(t, -5) &= h(t, 5), \\ h_x(t, -5) &= h_x(t, 5), \\ x \in [-5, 5], \quad t \in [0, \pi/2], \quad h(t, x) &= U(t, x) + iV(t, x) \in \mathbb{C}. \end{aligned} \quad (37)$$

In order to train the model, the function h has been considered as a two-dimensional function $h = (U, V)$, which was represented by the neural network with two outputs. The equation (36) has been defined as a system of two PDEs for the real and imaginary parts, respectively, allowing one to calculate the derivatives $\frac{\partial^i h}{\partial t^i}(0, X)$ and to compile and train the respective models.

First, the Periodic boundary conditions for both h and h_x and standard initial conditions have been generated separately for the real and imaginary parts of h . The best train loss of the standard model has been obtained: $3.05e - 06$. The L_2 norm of the obtained solution at $t = 0.1021$ (the closest to 0.1 time value available in the data set) and 50 random points x has also been calculated: 0.00145, 0.00405 and 0.0013 for the real part, imaginary one and for the absolute function $|h|$, respectively. The plots of the obtained solution is presented in Figure 4(a) and (b).

Then, the first 5 derivatives have been calculated at $t = 0$. The obtained Taylor approximations have been calculated for 100 random points x at $t = 0.01, 0.02, \dots, 0.05$. The obtained 500 values for real and imaginary parts of h have been added as additional conditions to the model, which was re-initialized, compiled and trained with the same parameters. The final obtained loss has become $9.55e - 06$ (a bit bigger than for the standard model due to many additional conditions). The obtained L_2 norm at $t = 0.1021$ has become better: 0.000696, 0.00356 and 0.000641 for the real part, imaginary one and for the absolute function $|h|$, respectively. The plot of the obtained solution is also presented in Figure 4(c) and (d). Again, the proposed scheme allowed to obtain a better solution within the same number of iterations (≈ 31000 iterations for both the models).

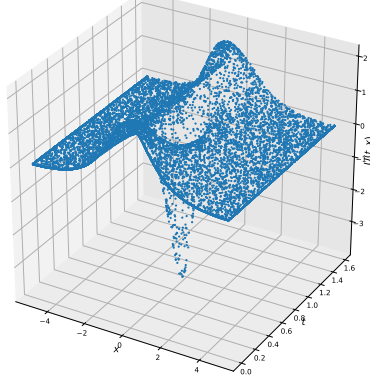
6 Conclusion

A new higher order differentiation method has been proposed for time-dependent PDEs. The proposed technique allows to obtain the derivatives with a high precision allowing so to obtain a lot of additional data that can be used for data-driven solution by PINNs. This additional information can be used in different ways, e.g., defining the additional loss functions directly on the derivatives or by adding a high precision Taylor expansion to generate data points inside the domain.

A well-known DeepXDE software has been used for data-driven solution of several real-life problems under Tensorflow background framework. A way to add new additional data points to the PINN model has been presented in this context. The results have shown that the performance of the deep learning models can be improved in this case.

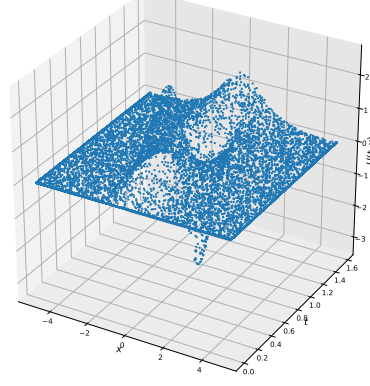
It should be noted that the proposed applications of the differentiation techniques presented in this paper have only illustrative purposes. It is already clear that the proposed methods allow to obtain much more data on the dynamics of the system. Since, data is the key value for the neural networks and deep learning, this additional data can be used in different ways in order to improve

Graph of the Nonlinear Schrodinger equation by the standard solution Real



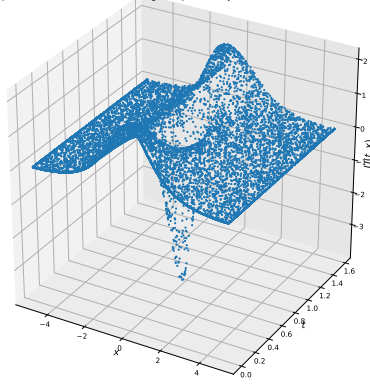
(a)

Graph of the Nonlinear Schrodinger equation by the standard solution Imaginary



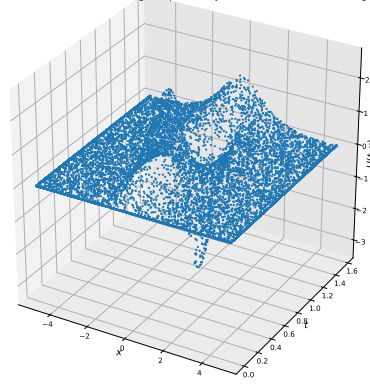
(b)

Graph of the Nonlinear Schrodinger equation by the standard solution Real



(c)

Graph of the Nonlinear Schrodinger equation by the standard solution Imaginary



(d)

Figure 4: The obtained solutions by PINNs using the standard model (top, figures (a) and (b)) and with additional conditions by the proposed scheme (bottom, figures (a) and (b)) of the nonlinear Schrodinger equation.

both the performance and accuracy. Other, more sophisticated models, loss functions and PINNs using the proposed differentiation schemes as well as more efficient differentiation method is the subject of the future works.

References

- [1] Mary P. Anderson, William W. Woessner, and Randall J. Hunt. *Applied groundwater modeling: simulation of flow and advective transport*. Academic Press, London, 2015. 2nd ed.
- [2] C. Basdevant, M. Deville, P. Haldenwang, J.M. Lacroix, J. Ouazzani, R. Peyret, P. Orlandi, and A.T. Patera. Spectral and finite difference solutions of the burgers equation. *Computers & Fluids*, 14(1):23–41, 1986.
- [3] V. Benci and M. Di Nasso. Numerosities of labeled sets: a new way of counting. *Advances in Mathematics*, 173:50–67, 2003.
- [4] Oliver Lopez Corona, Pablo Padilla, Oscar Escolero, Tomas Gonzalez, Eric Morales-Casique, and Luis Osorio-Olvera. Complex groundwater flow systems as traveling agent models. *PeerJ*, 2:e557, 2014.
- [5] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. 2009.
- [6] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what’s next. *Journal of Scientific Computing*, 92, 2022.
- [7] Darren Flynn and Khodr Shamseddine. On computational applications of the Levi-Civita field. *Journal of Computational and Applied Mathematics*, 382:113041, 2020.
- [8] David J. Griffiths. *Introduction to Quantum Mechanics*. Prentice Hall, New Jersey, 2004. 2nd ed.
- [9] A.E. Gutman, M.G. Katz, T.S. Kudryk, and S.S. Kutateladze. The mathematical intelligencer flunks the olympics. *Foundations of Science*, 22:539–555, 2017.
- [10] F. Iavernaro, F. Mazzia, M. S. Mukhametzhanov, and Y.D. Sergeyev. Conjugate-symplecticity properties of Euler-Maclaurin methods and their implementation on the Infinity Computer. *Applied Numerical Mathematics*, 155:58–72, 2020.
- [11] F. Iavernaro, F. Mazzia, M. S. Mukhametzhanov, and Y.D. Sergeyev. Computation of higher order Lie derivatives on the Infinity Computer. *Journal of Computational and Applied Mathematics*, 383:113135, 2021.

- [12] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3:422–440, 2021.
- [13] Jozse Korelc and Peter Wriggers. *Automation of Finite Element Methods*. 2016.
- [14] N. Kresic and A. Mikszewski. *hydrogeological conceptual site models data analysis and visualization*. CRC Press, Boca Raton, 2007.
- [15] L. Leonetti, F. Liguori, D. Magisano, and G. Garcea. An efficient isogeometric solid-shell formulation for geometrically nonlinear analysis of elastic shells. *Computer Methods in Applied Mechanics and Engineering*, 331:159 – 183, 2018.
- [16] Leonardo Leonetti and Mukhametzhanov Marat. Novel first and second order numerical differentiation techniques and their application to nonlinear analysis of Kirchhoff–Love shells. *Computational Mechanics*, 70:29–47, 2022.
- [17] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.
- [18] Karol Makuch, Robert Hołyst, Tomasz Kalwarczyk, Piotr Garstecki, and John F. Brady. Diffusion and flow in complex liquids. *Soft Matter*, 16:114–124, 2020.
- [19] V. P. Nguyen, C. Anitescu, S. P. A. Bordas, and T. Rabczuk. Isogeometric analysis: An overview and computer implementation aspects. *Mathematics and Computers in Simulation*, 117:89 – 116, 2015.
- [20] 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.
- [21] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Numerical gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 40(1):A172–A198, 2018.
- [22] A. Robinson. *Non-standard Analysis*. Princeton University Press, Princeton, 1996.
- [23] Y.D. Sergeyev. *Computer system for storing infinite, infinitesimal, and finite quantities and executing arithmetical operations with them*. USA patent 7,860,914 (2010), EU patent 1728149 (2009), RF patent 2395111 (2010).
- [24] Y.D. Sergeyev. Numerical infinities and infinitesimals: Methodology, applications, and repercussions on two Hilbert problems. *EMS Surveys in Mathematical Sciences*, 4:219–320, 2017.

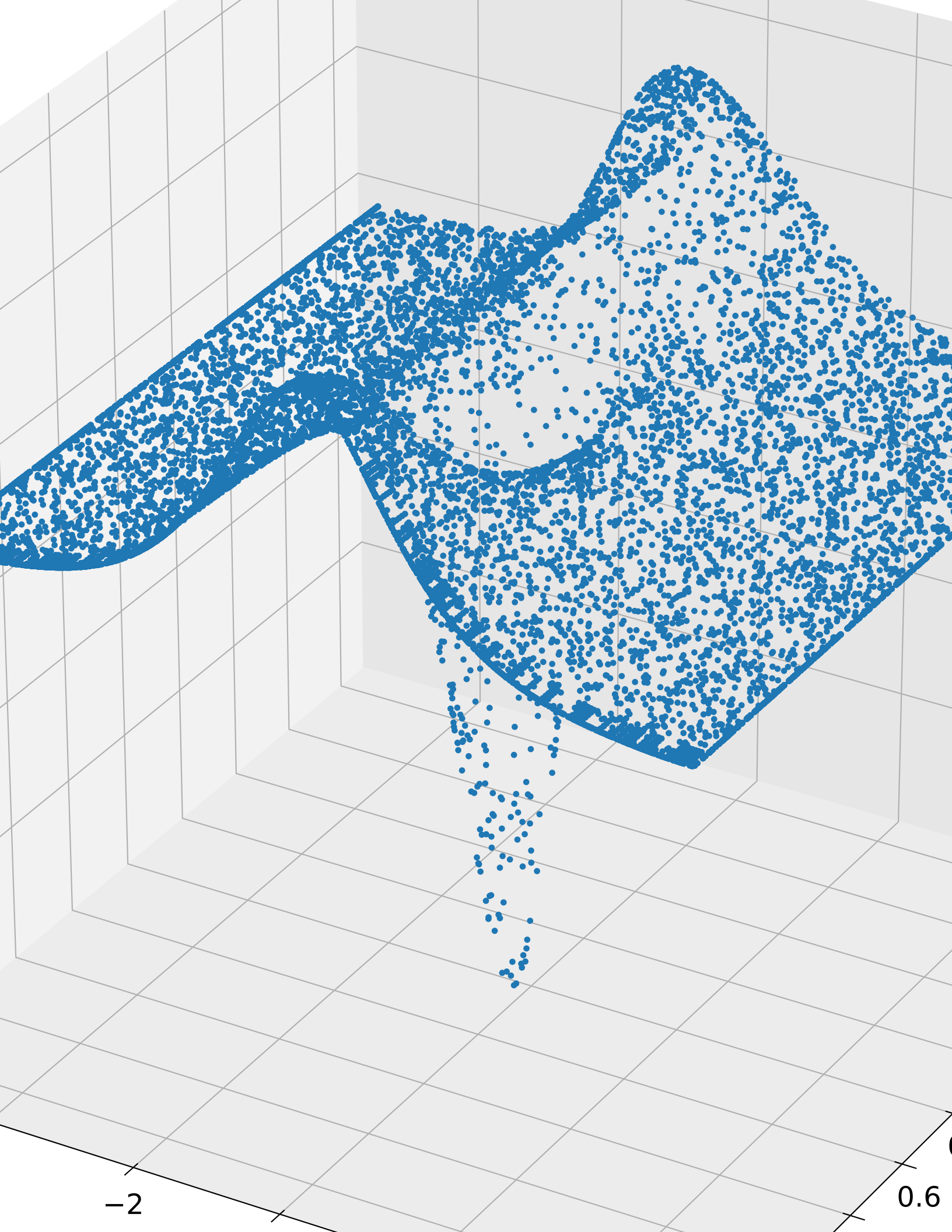
- [25] K. Shamseddine and M. Berz. Exception handling in derivative computation with non-archimedean calculus. In *Computational Differentiation: Techniques, Applications, and Tools*, pages 37–51. SIAM, Philadelphia, 1996.
- [26] Khodr Shamseddine and Martin Berz. Analysis of the levi-civita field, a brief overview. In Martin Berz and Khodr Shamseddine, editors, *Contemporary Mathematics, Advances in p-Adic and Non-Archimedean Analysis*, volume 508, pages 215–237. The American Mathematical Society, Providence, 2010.
- [27] Masato Tanaka, Takashi Sasagawa, Ryuji Omote, Masaki Fujikawa, Daniel Balzani, and Jörg Schröder. A highly accurate 1st- and 2nd-order differentiation scheme for hyperelastic material models based on hyper-dual numbers. *Computer Methods in Applied Mechanics and Engineering*, 283:22–45, 2015.
- [28] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021.

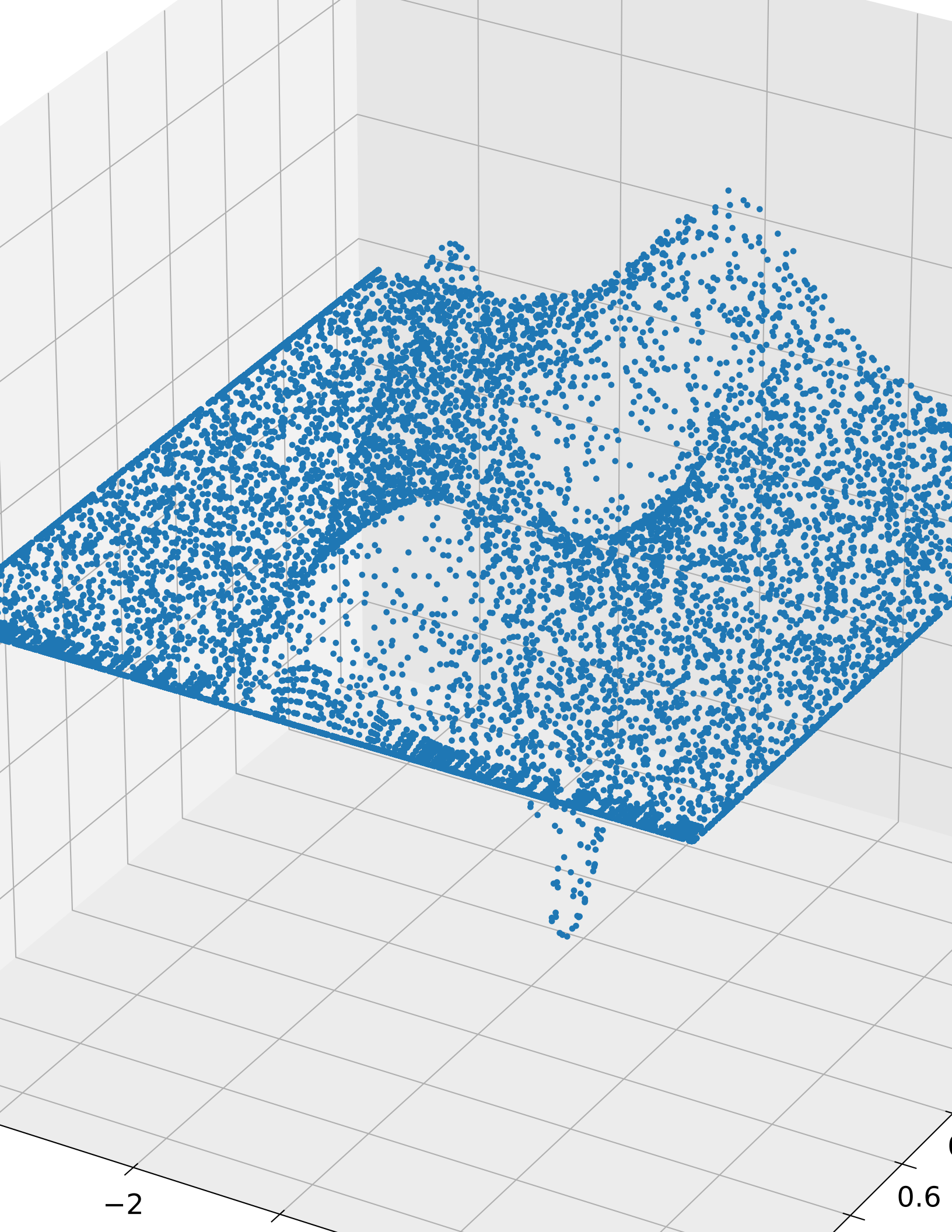
This figure "Diffusion.png" is available in "png" format from:

<http://arxiv.org/ps/2210.00518v2>

This figure "Heat.png" is available in "png" format from:

<http://arxiv.org/ps/2210.00518v2>





This figure "Wave.png" is available in "png" format from:

<http://arxiv.org/ps/2210.00518v2>