Advances in Continuous and Discrete Models

RESEARCH Open Access

Adaptive neural-domain refinement for solving time-dependent differential equations

Toni Schneidereit[1]* and Michael Breuß

*Correspondence:
Toni.Schneidereit@b-tu.de
[1]Applied Mathematics Group, Brandenburg University of Technology Cottbus-Senftenberg, Platz der Deutschen Einheit 1, 03046 Cottbus, Germany

Abstract

A classic approach for solving differential equations with neural networks builds upon neural forms, which employ the differential equation with a discretisation of the solution domain. Making use of neural forms for time-dependent differential equations, one can apply the recently developed method of domain segmentation. That is, the domain may be split into several subdomains, on which the optimisation problem is solved.

In classic adaptive numerical methods, the mesh as well as the domain may be refined or decomposed, in order to improve the accuracy. Also, the degree of approximation accuracy may be adapted. Therefore, it is desirable to transfer such important and successful strategies to the field of neural-network-based solutions. In the presented work, we propose a novel adaptive neural approach to meet this aim for solving time-dependent problems.

To this end, each subdomain is reduced in size until the optimisation is resolved up to a predefined training accuracy. In addition, while the neural networks employed are by default small, we propose a means to adjust also the number of neurons in an adaptive way. We introduce conditions to automatically confirm the solution reliability and optimise computational parameters whenever it is necessary. Results are provided for several initial-value problems that illustrate important computational properties of the method.

Keywords: Neural forms; Differential equations; Physics-informed neural networks; Adaptive neural refinement; Domain decomposition

1 Introduction

Differential equations (DEs) are important models in many areas of science and engineering, as they often represent real-world phenomena [1]. A special class of DEs are initial-value problems (IVPs), describing the time evolution of a system. The variety of neural-network approaches for solving DEs has increased over the last years and decades [2–4]. They mostly focus on obtaining a loss function out of the DE structure and given initial or boundary conditions. The loss function in this context has the characteristic of connecting the DE with the neural-network framework [5, 6]. This may be achieved with so-called neural forms (NFs), which are in fact trial solutions satisfying the given conditions [7–9].

The NF approach results in an unsupervised learning framework. In the end, the NF represents the solution of the DE.

Other neural approaches combine unsupervised and supervised training, where the neural-network outcome is compared to true (known) data corresponding to certain domain data [10–13]. Typically, the unsupervised part arises from the DE structure, while given initial or boundary conditions are directly added to the loss function and are treated in a supervised way [14]. The resulting difference, is then used for learning the adjustable neural-network parameters. Therefore, the neural network itself represents the solution of the DE after training in such approaches.

Turning to classical numerical methods for solving all kinds of DEs, one may consider, e.g. Runge–Kutta methods [15, 16] for time integration or the finite-element method (FEM) [17]. In order to obtain high accuracy and robustness, many numerical schemes feature adaptive mechanisms regarding, e.g. step-size control [1, 15] or mesh refinement [18–20]. That is, certain areas of the solution domain may require more elements or grid points, in other words a refined mesh, for improved reliability and accuracy. Such adaptive mesh-refinement techniques enable the mesh to be locally refined based on a suitable error estimate.

Several works offer neural-network-based strategies and approaches to generate optimal meshes or mesh refinements for use with the finite-element method [21, 22]. Predicting areas that are of interest in the sense of a required mesh refinement using neural networks is the objective of [23]. Their time-series analysis is employed to predict element-wise solution gradient values. The used neural network yields an indicator based on local gradient values in space and time. This indicator is then used to predict whether a mesh refinement or a coarsening may be suitable. While in this method the mesh-refinement indicator is realised by a neural network, the FEM is used for solving the partial differential equation (PDEs) examples. Complementary to the latter approach, in [24] a learning strategy is developed that keeps the mesh fixed but selects the numerical scheme that gives locally high accuracy based on local gradients.

The most relevant related article in the context of our work may be the adaptive neural approach in [25], so let us discuss this work more in detail. It features a feedforward neural network in a framework combining both supervised and unsupervised terms, similar to [10, 12, 13]. The training process includes several evolution steps, each consisting of the optimisation over the training points combined with an evaluation of results at a finer grid. The latter is realised with the same set of neural-network parameters obtained from the training step. It is proposed to start with a coarse grid and to perform local grid refinement whenever the resulting network loss differs. The method is developed for boundary-value problems arising with stationary PDEs, like, e.g. the Poisson equation. Results indicate that more complex neural-network architectures (w.r.t. number of layers and neurons) or more training points may increase the accuracy.

Let us stress that in the discussed work [25], the mesh is refined but treated in a global fashion and not decomposed into subdomains. However, we also find the combination of domain decomposition with neural networks [12, 13], which is also related to our method. The so-called physics-informed neural networks may be independent (e.g. in size and initialisation) in predefined and discrete subdomains. At the subdomain interfaces, a physical continuity condition holds and the average solution of the two corresponding neural networks at each interface is enforced. Several neural networks learn the governing DE locally

instead of using a single one for the entire domain, which results overall in a small network loss.

Problem statement and contribution. In our previous work, we investigated the computational characteristics of a small feedforward neural network with only one hidden layer [26]. It is well known that the parameters within a neural network are not independent of each other. That is, changing one component of the setup may require a change to other components to improve or at least to maintain the results. Computationally, larger domain sizes appear to be challenging for the NF approach [7] together with the studied neural-network setup. Turning to weight initialisation, the use of random initial weights allows us to achieve results that can be considered as reliable, while they lead to variances in repeated computations, based on the initially generated values. The use of the same initial weight value for each neuron, e.g. zero, typically results in less-accurate approximations. However, for testing frameworks, this deterministic initialisation leads to identical results in repeated computations and creates a baseline. Based on these investigations, we proposed a collocation polynomial extension for the NF and a subdomain-division approach, which splits the solution domain into equidistant subdomains [5]. Since the NF adopted from [7] directly incorporate the initial condition in its construction, each temporal subdomain generates a new initial condition for the subsequent subdomain. As it turns out, both extensions to the original NF approach were able to improve the computational results with respect to weight initialisation and larger domain sizes. However, equidistant subdomains may not be the optimal choice in regions where the solution is easy or difficult to learn.

Therefore, we now propose the adaptive neural-domain refinement (ANDRe), which makes use of the subdomain collocation (polynomial) neural forms (SCNF). These are optimised repeatedly over the domain. The domain itself is allowed to split into subdomains, which may locally decrease in size, whenever the network loss is not sufficiently small. Therefore, we combine the advantageous characteristics from domain decomposition and adaptive mesh refinement. Furthermore, we embed into the described process a means to adapt the number of neurons used for optimisation in each subdomain. This is done with the aim to increase reliability and accuracy of the approximation.

Thus, we also combine adaptive refinement of the domain with adaptivity in the neural sense. In addition, the results provide an opportunity to discuss the relation between different measurement metrics like, e.g. the numerical error and the neural-network loss.

The following section introduces the subdomain collocation neural form (SCNF), as well as the incorporated neural networks and the optimisation. Based on the SCNF, we continue to propose the adaptive domain refinement (ANDRe) algorithm. Later, this approach is applied to four IVPs, each representing a different type. The results are discussed in detail and we finish the paper by a conclusion with an outlook to possible future work.

2 The methods

The overall aim is to solve IVPs in form of

$$G\big(t, u(t), \dot{u}(t)\big) = 0, \qquad u(t_0) = u_0, \quad t \in D \subset \mathbb{R}, \tag{1}$$

with given initial values $u(t_0) = u_0$. We identify $\dot{u}(t)$ as the time derivative of $u(t)$. Let us note at this point, that G in Eq. (1) may also denote a system of IVPs. In the following, we

will first focus on IVPs with only one equation and later provide the necessary information in order to extend the approach to systems of IVPs.

2.1 The subdomain collocation neural form (SCNF)

The NF approach [7] seeks to replace the solution function with a differentiable trial solution

$$\tilde{u}(t,\mathbf{p}) = A(t) + F(t,\mathbf{p}), \tag{2}$$

which connects the given IVP with a neural-network term, incorporating the weight vector $\mathbf{p}$. In Eq. (2), both $A(t)$ and $F(t,\mathbf{p})$ are problem specific and have to be constructed under the requirement of fulfilling the initial condition. in addition to replacing the solution function, its time derivative is expressed as well by differentiating the constructed NF.

One of the possible NF constructions, to which we will refer as classic NF and that has been proposed in [7], is to set $A(t) = u_0$ and $F(t,\mathbf{p}) = N(t,\mathbf{p})(t - t_0)$. The neural-network term $N(t,\mathbf{p})$ receives given (later discretised) points from the domain and depends on the neural-network weights $\mathbf{p}$. Further details are provided in the upcoming subsection. This configuration ($\tilde{u}(t,\mathbf{p}) = u_0 + N(t,\mathbf{p})(t - t_0)$) ensures removal of the impact of $N(t,\mathbf{p})$ at the initial point t_0, whereas then $\tilde{u}(t,\mathbf{p})$ equals the initial value u_0.

The subdomain collocation neural-form (SCNF) approach [5] now opts to extend the classic NF (Eq. (2)) in two directions, *(i)* by increasing the polynomial degree of the term $F(t,\mathbf{p})$ as explained below and *(ii)* by introducing domain segmentation, which is a novel approach to solve the IVP on subdomains in order to increase the numerical accuracy.

Since u_0 is a constant value and $N(t,\mathbf{p})$ is multiplied by $(t - t_0)$, we find

$$\tilde{u}(t,\mathbf{p}) = u_0 + N(t,\mathbf{p})(t - t_0) \tag{3}$$

to resemble a first-order polynomial in $(t - t_0)$ (in detail: $\tilde{u}(t,\mathbf{p}) = u_0(t - t_0)^0 + N(t,\mathbf{p})(t - t_0)^1$). Hence, we propose to extend the polynomial order of the classic NF, inspired by collocation polynomials [1]. Therefore, we transform $\tilde{u}(t,\mathbf{p}) \to \tilde{u}_C(t,\mathbf{P}_m)$ with

$$F(t,\mathbf{p}) \to F(t,\mathbf{P}_m) = \sum_{k=1}^{m} N_k(t,\mathbf{p}_k)(t - t_0)^k \tag{4}$$

to find

$$\tilde{u}_C(t,\mathbf{P}_m) = u_0 + \sum_{k=1}^{m} N_k(t,\mathbf{p}_k)(t - t_0)^k. \tag{5}$$

Here, m represents the SCNF order. This polynomial extension adds more flexibility to the approach. However, this is achieved in a different way from just increasing the number of hidden-layer neurons in a single neural network, since additional networks are connected to the factors $(t - t_0)^k$. We expected a higher accuracy from the use of higher orders, which has partially been confirmed, see [5]. Since we are testing our framework with deterministic initial weights (zero), let us mention that each neural network effectively acts as having

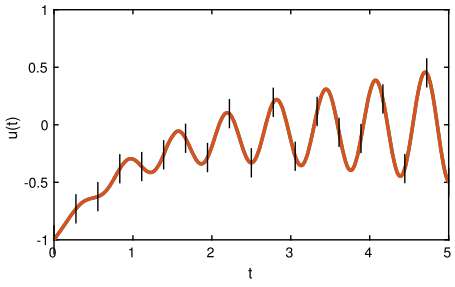


Figure 1 SCNF domain-segmentation example with fixed and equidistant subdomains, (orange) analytical IVP solution (cf. IVP in Eq. (29)), (black/marked) subdomain boundaries, see [5] for details

a single neuron in the hidden layer. Although the corresponding values and weight are split between the neurons. Therefore, the monomials $(t - t_0)^k$ do have a certain impact on the NF and the flexibility increases.

The weight matrix $\mathbf{P}_m$ in Eq. (5) stores each weight vector $\mathbf{p}_k$, $k = 1, \dots, m$ as column vectors. We discretise the domain D by the collocation method employing a uniform grid with $n + 1$ grid points t_i ($t_0 < t_1 < \cdots < t_n$), so that our novel collocation NF approach for the IVP (1) leads to the ℓ_2 loss function formulation

$$E[\mathbf{P}_m] = \frac{1}{2(n+1)} \sum_{i=0}^{n} \left\{ G\big(t_i, \tilde{u}_C(t_i, \mathbf{P}_m), \dot{\tilde{u}}_C(t_i, \mathbf{P}_m)\big) \right\}^2. \tag{6}$$

Since the domain variable in $(t_i - t_0)^k$ effectively acts as a scaling of $N_k(t_i, \mathbf{p}_k)$, we conjecture that a large domain-size variation may introduce the need for a larger number of training points or the use of a more complex neural-network architecture. Having this in mind, it appears very natural to couple the collocation NF with a technique that refines the computational domain. To this end we will consider the non-adaptive version of domain segmentation that opts to split the domain into a fixed number of equidistant subdomains.

That said, we split the solution domain D into subdomains D_l, $l = 1, \dots, h$, with $n + 1$ grid points $t_{i,l}$ per domain segment. Now, the NF is resolved separately in each subdomain. The interfacing grid points overlap, e.g. the computed value $\tilde{u}_C(t_{n,l-1}, \mathbf{P}_{m,l-1})$ at the last grid point of any subdomain D_{l-1} is set to be the new initial value $\tilde{u}_C(t_{0,l}, \mathbf{P}_{m,l})$ for the next subdomain D_l. The general idea is visualised in Fig. 1, where the black/vertical marks represent the equidistantly distributed subdomain boundaries for the solution of an example IVP.

Summarising the construction up to now, the SCNF satisfies the new initial values in each domain segment, namely

$$\tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}) = \tilde{u}_C(t_{0,l}, \mathbf{P}_{m,l}) + \sum_{k=1}^{m} N_k(t_{i,l}, \mathbf{p}_{k,l})(t_{i,l} - t_{0,l})^k. \tag{7}$$

The neural-network terms are now scaled by $(t_{i,l} - t_{0,l})^k$. Depending on the subdomain size, those factors may up- or downscale $N_k(t_{i,l}, \mathbf{p}_{k,l})$. Incorporated in Eq. (6), the SCNF time derivative reads

$$\begin{aligned} \dot{\tilde{u}}_C(t_{i,l}, \mathbf{P}_{m,l}) = \sum_{k=1}^{m} \big[& \dot{N}_k(t_{i,l}, \mathbf{p}_k)(t_{i,l} - t_{0,l})^k \\ & + N_k(t_{i,l}, \mathbf{p}_k) k (t_{i,l} - t_{0,l})^{k-1} \big]. \end{aligned} \tag{8}$$

In order to keep the overview of all terms and indices, we sum them up again: The ith grid point in the lth subdomain is denoted by $t_{i,l}$, while $t_{0,l}$ is the initial point in the subdomain D_l with the initial value $\tilde{u}_C(t_{0,l}, \mathbf{P}_{m,l})$. That is, $t_{n,l-1}$ and $t_{0,l}$ are overlapping grid points. In D_1, $\tilde{u}_C(t_{0,1}, \mathbf{P}_{m,1}) = u(t_0)$ holds. The matrix $\mathbf{P}_{m,l}$ contains the set of the m neural-network weight vectors $\mathbf{p}_k, k = 1, \ldots, m$ in the corresponding subdomain. Finally, $N_k(t_{i,l}, \mathbf{p}_{k,l})$ denotes the kth neural network in D_l.

The loss function employing the SCNF aims to minimise for each subdomain D_l the energy

$$E_l[\mathbf{P}_{m,l}] = \frac{1}{2(n+1)} \sum_{i=0}^{n} \left\{ G\big(t_{i,l}, \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}), \dot{\tilde{u}}_C(t_{i,l}, \mathbf{P}_{m,l})\big) \right\}^2. \tag{9}$$

Similar to Eq. (6), the loss function now incorporates $\tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l})$ instead of $u(t)$. Let us mention at this point that the $(n+1)$ grid points in Eq. (9) are used for the neural-network training and are referred to training points. Afterwards, the neural networks are used to verify the result with the learned weights and so-called verification points (VP). The latter are also grid points, but differently distributed from the training points (TP). Therefore, the corresponding grid points will later be referred to as n_{TP} with loss-function notation $E_l^{\mathrm{TP}}[\mathbf{P}_{m,l}]$ and n_{VP} with $E_l^{\mathrm{VP}}[\mathbf{P}_{m,l}]$, respectively.

If G in Eq. (1) represents a system of IVPs, each solution function requires its own SCNF and the loss function derives from the sum over all the separate ℓ_2-norm terms, i.e. one for each equation involved. We will address this extension in detail in the corresponding example later.

2.2 Neural-network architecture and optimisation

Let us start with an overview on the kth neural-network architecture as displayed in Fig. 2. The term N_k, cf. Eq. (7), represents in general a feedforward neural network with one input-layer neuron for the discretised domain data $t_{i,l}$, H hidden-layer neurons and one output-layer neuron. In addition, both input layer and hidden layer incorporate one bias neuron. In general, the number of neurons in the hidden layer directly impacts the number of adjustable network weights, labelled as $v_{j,k}$ (input-layer neuron), $\eta_{j,k}$ (input-layer bias neuron), $\rho_{j,k}$ (hidden-layer neurons) and γ_k (hidden-layer bias neuron) in Fig. 2. These weights are stored in the weight vector $\mathbf{p}_k$. The neural-network output reads

$$N_k(t_{i,l}, \mathbf{p}_k) = \sum_{j=1}^{H} \rho_{j,k} \sigma(z_{j,k}) + \gamma_k. \tag{10}$$

Here, $\sigma_{j,k} = \sigma(z_{j,k}) = 1/(1 + e^{-z_{j,k}})$ represents the sigmoid activation function, with the weighted sum $z_{j,k} = v_{j,k} t_{i,l} + \eta_{j,k}$. Therefore, H hidden-layer neurons result in $(3H + 1)$ neural-network weights in our framework. The input layer passes the domain data $t_{i,l}$, weighted by $v_{j,k}$ and $\eta_{j,k}$, to the hidden layer for processing. The neural-network output $N_k(t_{i,l}, \mathbf{p}_k)$ is again a weighted sum of the values $\rho_{j,k}\sigma(z_{j,k})$ and γ_k.

The neural-network training is the process of minimising the loss function, cf. Eq. (9), with respect to the neural-network weights $\mathbf{p}_k$. In addition, and in contrast to the $(3H + 1)$ weights for the kth neural network, the minimisation of $E_l[\mathbf{P}_{m,l}]$ requires $m(3H + 1)$ weights to be adjusted, due to the SCNF order m.

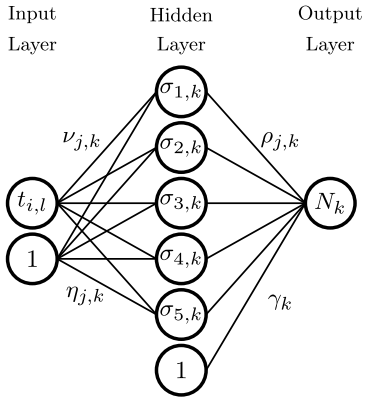


Figure 2 Architecture of the *k*th neural network in the *l*th subdomain

In practice, the goal is to find a local minimum in the weight space (or energy landscape), which perhaps consists of many extreme points. This may be realised by first- or second-order optimisation techniques, which use the first or second loss-function derivatives, respectively. One epoch of training includes the computation of the loss-function gradient (first-order derivatives) $\nabla E_l[\mathbf{P}_{m,l}]$ w.r.t. the adjustable network weights, averaged over all training points (grid points). We will refer to this learning procedure as full batch training, since the neural-network weights are updated only once per epoch. That is, the loss function and its gradient are computed and averaged with respect to all grid points in a subdomain. One subdomain may consist of, e.g. ten training points. Afterwards, the averaged (over all grid points of one subdomain) loss-function gradient is used to update the weights. This usually takes several epochs for a successful training. In this paper, we use Adam optimisation [27] in order to update the neural-network weights. With full batch training, the loss function also returns an (averaged) scalar value for each epoch. It provides information about the training status and whether the minimisation of $E_l[\mathbf{P}_{m,l}]$ can be considered as accomplished or not. Let us recall, that each $\mathbf{p}_k$ is separately optimised.

Details on the optimisation Let us consider the example IVP

$$\dot{u}(t) = t\sin(10t) - u(t), \qquad u(0) = -1 \tag{11}$$

for which we find G, as a part of the loss function, in Eq. (9) as

$$G = \dot{\tilde{u}}_C(t_{i,l}, \mathbf{P}_{m,l}) + \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}) - t_{i,l}\sin(10t_{i,l}) = 0. \tag{12}$$

The minimisation of Eq. (9) aims to obtain G in Eq. (12) as close to zero as possible. That is, the expression of interest actually reads

$$\dot{\tilde{u}}_C(t_{i,l}, \mathbf{P}_{m,l}) + \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}) \approx t_{i,l}\sin(10t_{i,l}). \tag{13}$$

Here, the values of $t_{i,l}\sin(10t_{i,l})$ are predetermined by the domain grid points, whereas the SCNF and its time derivative additionally depend on the neural-network weights and their optimisation. Hence, Eq. (13) can be considered as satisfied for various combinations of $\dot{\tilde{u}}_C(t_{i,l}, \mathbf{P}_{m,l}) + \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l})$. Hence, the results may highly depend on the final location in the weight space.

One reason for this circumstance may relate to the complexity of the energy landscape, which has multiple (local) minima that can lead to several combinations of the left-hand side in Eq. (13). Not all of these combinations must be real or useful solutions for the given domain-training points. This issue may occur, e.g. when the initial weights are far away from a suitable minimum for a helpful approximation. Also, when there is an unfavourable local minimum close to the initialisation, the optimiser may only find this one. However, fine tuning all the incorporated computational parameters is a challenging task since some of these are not independent of each other [9, 26].

3 The novel adaptive neural-domain refinement (ANDRe)

We propose in this section the embedding of the previously introduced SCNF approach into an adaptive algorithm. The resulting refinement strategy features two components, *(i)* verification of the SCNF training status arising from the loss function (Eq. (9)) in each subdomain serving as an error indicator and *(ii)* an algorithmic component to perform the domain refinement.

Algorithm summary In Fig. 3, we consider an artificial example to sketch the concept behind ANDRe in a visual way. The basic idea is to optimise the loss function $E_l[\mathbf{P}_{m,l}]$ for a given number of equidistant training points (n_{TP}) in each subdomain and to evaluate the results at equidistant verification points (n_{VP}), intermediate to n_{TP}. To obtain the subdomains, the algorithm starts with the loss-function optimisation on the entire domain (Fig. 3(1.)). If the predefined verification-loss bound $\sigma > 0$ is not fulfilled, the domain is split in half. Now the optimisation task starts again for the left half since we only know the initial value for this subdomain. When the verification loss $E_l^{\mathrm{VP}}[\mathbf{P}_{m,l}]$ (loss function evaluated for n_{VP}) again fails to go below σ, the current (left) subdomain is reduced in size (see differences in Fig. 3(2.) to (3.)), whereas a splitting is only performed when the computation takes place in the rightmost subdomain and σ is not satisfied by $E_l^{\mathrm{VP}}[\mathbf{P}_{m,l}]$, meaning that the original right domain border is always kept and not shifted during refinement. The process of comparing the verification loss to its error bound, reducing the current subdomain and starting the optimisation another time, is repeated until $E_l^{\mathrm{VP}}[\mathbf{P}_{m,l}] \leq \sigma$. Therefore, in the artificial example in Fig. 3(3.), the leftmost subdomain is now considered to be learned.

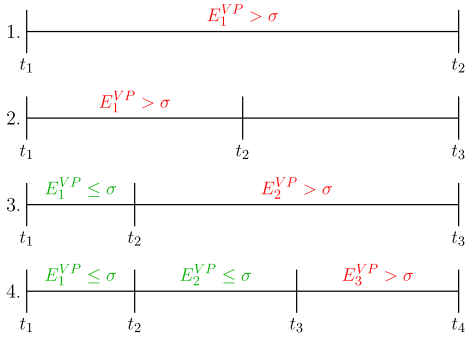


Figure 3 A visualisation of the basic idea behind ANDRe, with the error comparison and (sub-)domain split/reduction

Now, the process starts again for the rightmost subdomain (see Fig. 3(3.) and (4.)) with a new initial condition provided by the learned (left) subdomain. However, the current (new) subdomain starts at the right boundary of the first (learned) subdomain and ends at the right boundary of the entire domain. Therefore, the already learned subdomain is excluded from further computations.

If a subdomain becomes too small or if the verification loss increases after a subdomain split/reduction, the computational parameters are adjusted in a predefined, automated way. Details on the parameter adjustment will be provided in a corresponding paragraph later.

Let us now provide detailed information about ANDRe, which is shown as a flowchart in Fig. 4. The starting point is the choice of the SCNF order m and the subdomain-resize parameter δ, which acts as a size reduction whenever a decrease is necessary. For optimisation we use equidistant training points n_{TP}. An important constant is the verification-loss bound $\sigma > 0$, used to verify the SCNF solution in the corresponding subdomain. After each complete optimisation, the results are evaluated by the loss function with the previous learned weights at intermediate verification points, resulting in $E_l^{\mathrm{VP}}[\mathbf{P}_{m,l}]$. The latter (scalar value) is then compared to σ in order to determine whether the solution can be considered as reliable or not. We define l as the index of the subdomain, in which the SCNF is currently solved and h represents the total number of subdomains. The latter is not fixed and will increase throughout the algorithm. Finally, the first domain is set to be

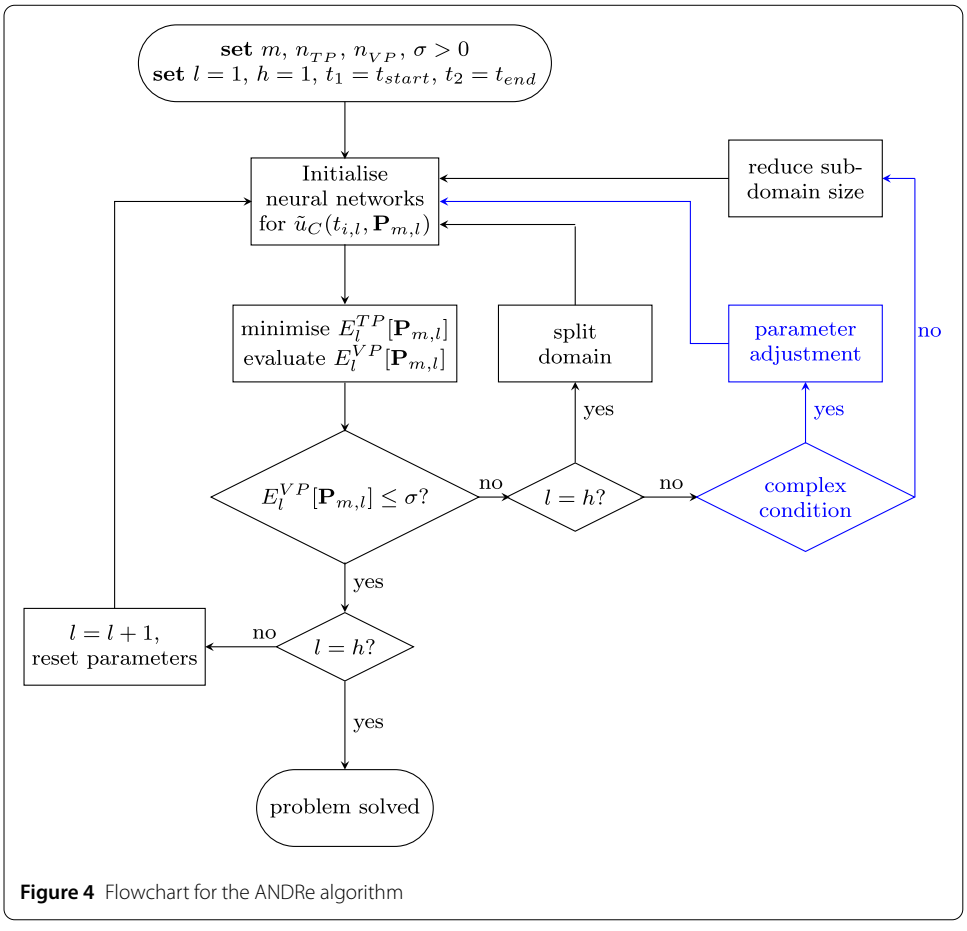


Figure 4 Flowchart for the ANDRe algorithm

the entire given domain $D_1 = [t_{\text{start}}, t_{\text{end}}] = [t_1, t_2]$. Please note, while on the computational side, the subdomains are discretised and corresponding grid points denoted by $t_{i,l}$, we only refer to subdomain boundaries by t_l in this paragraph, for simplicity.

Flowchart explanation The first processing operation

$$\text{Initialise neural networks for } \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}) \tag{14}$$

covers setting the initial architecture parameters such as number of hidden-layer neurons, Adam learning rate and initialising the weights for $\mathbf{P}_{m,l}$.

Afterwards, the optimisation problem

$$\text{minimise } E_l^{\text{TP}}[\mathbf{P}_{m,l}] \tag{15}$$

$$\text{evaluate } E_l^{\text{VP}}[\mathbf{P}_{m,l}] \tag{16}$$

is solved by training the SCNF framework for given equidistant n_{TP} over the entire domain $D_1 = [t_1, t_2]$ (cf. Fig. 3(1.)). The evaluation for equidistant and intermediate n_{VP} leads to the verification loss $E_l^{\text{VP}}[\mathbf{P}_{m,l}]$.

Then, the first decision block compares the verification loss (after the training process has ended) to the error bound σ:

$$E_l^{\text{VP}}[\mathbf{P}_{m,l}] \leq \sigma\,? \tag{17}$$

- *Eq.* (17) *NO*: If the verification loss did not go below σ, the size of the current subdomain will be reduced. However, first, another decision has to be made here. Namely, has $E_l[\mathbf{P}_{m,l}]$ been solved for the first time on the current, rightmost (sub-)domain or in other words, is the current domain index l equal to the number of total subdomains h:

$$l = h? \tag{18}$$

  - *Eq.* (18) *YES*: This means the right boundary is t_{end} and we have to split the current subdomain l first, which leads to an increase of the number of total subdomains by 1 ($h = h + 1$). The boundaries now have to be adjusted with the left one t_l to remain unchanged, while the former right boundary is now scaled by $t_{l+1} = t_l + \delta(t_{l+1} - t_l)$, after $t_{l+2} = t_{l+1}$ is set to be the right boundary of domain $l + 1$. For example, if an entire domain $D_1 = [t_1, t_2] = [0, 10]$ has to be split for the first time with $\delta = 0.5$, the resulting subdomains are $D_1 = [t_1, t_2] = [0, 5]$ and $D_2 = [t_2, t_3] = [5, 10]$. Afterwards, the algorithm leads back to Eq. (14).
  - *Eq.* (18) *NO*: In this case the current subdomain has already been split up. Now, the right boundary has to be adjusted in order to decrease the current subdomain size. However, beforehand we check for a complex condition (highlighted in blue) to ensure that a subdomain does not become too small. Additionally, we also check if the verification loss decreased compared to the prior computation on the same subdomain l. That is, the algorithm compares the verification loss from the formerly larger subdomain l to the current, size reduced subdomain l. The condition

itself may come in different shapes. We decided to check for one of the

$$\begin{aligned} &\text{complex conditions:}\\ &t_{l+1} - t_l \leq 0.1?\\ &\text{or}\\ &E_l^{\mathrm{VP}} \text{ from previous } (l \neq h) \text{ subdomain} \leq \text{current } E_l^{\mathrm{VP}}? \end{aligned} \tag{19}$$

- *Eq.* (19) *YES*: At this point we employ a

$$\text{parameter adjustment,} \tag{20}$$

  which may be realised to be problem specific and is later addressed in a corresponding paragraph. Afterwards, the algorithm leads back to Eq. (14). Basically speaking, the adjustable parameters may include the number of hidden-layer neurons, the learning rate, the number of training points and so on.
- *Eq.* (19) *NO*: In this case, the subdomain is still large enough to be reduced in size while the verification loss decays. Therefore, we resize the right subdomain boundary t_{l+1} to

$$t_{l+1} = t_l + \delta(t_{l+1} - t_l), \tag{21}$$

  where δ denotes the domain-resize parameter. Continuing the example from above, resizing D_1 of the already split domain leads to $D_1 = [t_1, t_2] = [0, 2.5]$ and $D_2 = [t_2, t_3] = [2.5, 10]$. Afterwards, the algorithm leads back to Eq. (14).

- *Eq.* (17) *YES*: In the case of the verification loss being smaller or equal compared to σ, the current subdomain l has been successfully learned by means of a sufficiently small verification loss. Now, it is necessary to determine whether we are in the last subdomain (right boundary is t_{end}) or if there is still one subdomain to solve the optimisation problem on, namely

$$l = h? \tag{22}$$

  - *Eq.* (22) *NO*: There is at least one subdomain left and therefore the current subdomain index is updated to $l = l + 1$ in order to solve the optimisation problem on the adjacent subdomain. Additionally, we reset all the possibly adjusted parameters to the initial ones. Thus, we make sure to not overuse the variable parameters in regions where the solution computes by using the initial ones. The algorithm then leads back to Eq. (14).
  - *Eq.* (22) *YES*: All subdomains have been successfully learned and the IVP is entirely solved.

We developed ANDRe in four steps, making it an adaptive neural algorithm for domain refinement. Excluding the blue part in Fig. 4, the black part represents a fully functional algorithm that can refine the domain in an adaptive way with the focus laying on the verification loss. Prior to this final version, the training loss was used as the main training status indicator. The evaluation stage (verification loss) on the other hand was later added,

in addition to the training loss. It turned out that small training losses do not necessarily result in a comparable numerical error, presumably due to possible overfitting. Therefore, we included the verification stage, to reduce the impact of overfitting on the end result. However, we later recognised that the verification loss has a much stronger relation to the numerical error. Therefore, we were able to reduce the complexity by laying the focus directly on the verification. Furthermore, in some examples we recognised that the preset neural-network architecture may not be flexible enough to learn certain subdomains. Hence, we upgraded ANDRe to incorporate an automated parameter-adjustment mechanism, highlighted in Fig. 4 as blue. Whenever a subdomain becomes too small or the verification loss in a subdomain increases compared to the previous optimisation on the same subdomain (e.g. prior to a size reduction), network- and optimisation-related parameters may be re-balanced in a predefined way. We will later provide experimental evidence to prove the capabilities of ANDRe.

4 Computational results and discussion

In this section we discuss the computational results for different IVPs, solved by ANDRe. Beforehand, the framework parameters and methods are further specified.

Details on parameters and measurement metrics The NF approach comes with plenty of parameters. We have already shown in a computational study [26], that they are not independent of each other. Changing one parameter may require another parameter to be changed as well in order to improve or maintain the reliability.

Table 1 lists the computational parameters that are initially fixed in our computational setup. Parameters marked with (*) will be separately discussed in the corresponding paragraph. The initial weight values, the SCNF order as well as the number of epochs and training points (n_{TP}) have been previously investigated and are fixed to suitable values, see [5, 26] for further details. Nonetheless, each parameter has its impact on the solution. Key in training the neural networks are the training points $t_i, i = 0, \ldots, 9$, schematically depicted in Fig. 5 as green circles. Generally speaking, Fig. 5 shows an arbitrary subdomain and the notation t_i was chosen for simplicity. From now on, the grid points in subdomain l are again referred to as $t_{i,l}$ and follow the structure in Fig. 5.

Table 1 Initial computational parameters, (*) part of parameter adjustment

Comp. parameter	Value
hidden-layer neurons*	5
initial weight values	0
initial learning rate*	1e-3
number of epochs	1e5
training points (n_{TP})	9
verification points (n_{VP})	11
SCNF order (m)	5
resize parameter (δ)	0.5

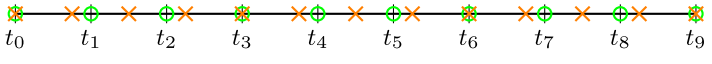


Figure 5 Visualisation of grid-point distribution in a subdomain, (green/circle) 10 equidistant training points t_i, (orange/cross) 12 equidistant verification points. Please note that, e.g. $n_{\mathrm{TP}} = 9$ refers to ten training points in Table 1

They serve as the input data and are important for the optimisation of the loss function

$$E_l^{\mathrm{TP}}[\mathbf{P}_{m,l}] = \frac{1}{2(n+1)} \sum_{i=0}^{n_{\mathrm{TP}}} \left\{ G\big(t_{i,l}, \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}), \dot{\tilde{u}}_C(t_{i,l}, \mathbf{P}_{m,l})\big) \right\}^2 \tag{23}$$

in the lth subdomain. Let us comment on an optimisation procedure in some detail, referred to as incremental learning, employed in [10]. Here, the computation includes several complete optimisations per (temporarily untouched) subdomain. That is, for example with five increments in Fig. 5, the training points are split into five sets and the first optimisation only takes t_0 and t_1 into account. Then, the second one uses t_0, t_1, t_2, t_3 with the same weights from the first (complete) optimisation. This is continued until the optimisation uses all training points. The incremental learning procedure only applies to the training process and $E_l^{\mathrm{TP}}[\mathbf{P}_{m,l}]$, not to the verification.

Speaking of that, the verification is performed with the loss function and the corresponding verification points (n_{VP}), which are differently distributed (cf. Fig. 5) than the training points. With these discrete points and after the training process, the loss function returns a scalar value named verification loss

$$E_l^{\mathrm{VP}}[\mathbf{P}_{m,l}] = \frac{1}{2(n+1)} \sum_{i=0}^{n_{\mathrm{VP}}} \left\{ G\big(t_{i,l}, \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}), \dot{\tilde{u}}_C(t_{i,l}, \mathbf{P}_{m,l})\big) \right\}^2. \tag{24}$$

As the naming suggests, this verification loss is used to evaluate and verify the training results to indicate whether the IVP has been solved sufficiently well or not. For this purpose, the verification-loss bound σ will be compared to Eq. (24).

The domain-resize parameter has also been fixed for all computations to $\delta = 0.5$. A larger value, up to $\delta = 0.9$, would find individual subdomains faster due to the larger size reduction, but perhaps result in too many subdomains. On the other hand, a smaller value, down to $\delta = 0.1$ may find the individual subdomains more carefully but would also heavily increase the computation time. We will discuss an experiment regarding the domain-resize parameter later.

Table 1 will later also be extended by problem-specific parameters, which are *(i)* verification-loss bound σ, *(ii)* computational domain size, *(iii)* initial conditions and *(iv)* learning increments. These parameters will be specified and discussed in a subsequent paragraph.

Turning to the measurement metrics for the results, we will compare ANDRe to the analytical solutions of four different IVPs. We make use of the absolute value differences between the analytical solution and ANDRe in the context of the (averaged) ℓ_1-norm $\Delta u_{l,1}$ and the ℓ_∞-norm $\Delta u_{l,\infty}$

$$\Delta u_{l,1} = \frac{1}{n+1} \sum_{i=0}^{n} \big| u(t_{i,l}) - \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}) \big|, \tag{25}$$

$$\Delta u_{l,\infty} = \max_i \big| u(t_{i,l}) - \tilde{u}_C(t_{i,l}, \mathbf{P}_{m,l}) \big|, \tag{26}$$

where Δu_1 and Δu_∞ average the numerical error over all subdomains

$$\Delta u_1 = \frac{1}{h} \sum_{l=1}^{h} \Delta u_{l,1}, \tag{27}$$

$$\Delta u_\infty = \frac{1}{h} \sum_{l=1}^{h} \Delta u_{l,\infty}. \tag{28}$$

The ℓ_∞-norm basically returns the largest numerical error value. We will later refer to the corresponding norms as ℓ_1-error and ℓ_∞-error.

Details on parameter adjustment Let us now comment on the parameter adjustment since this part of the algorithm required a lot of fine tuning. After several experiments with different parameter-adjustment methods, not documented here, the Adam learning rate and the number of hidden-layer neurons were chosen to be a part of the parameter adjustment and may change during the process. The number of hidden-layer neurons directly determines the number of adjustable weights. Furthermore, they are connected to the universal approximation theorem [28]. This basically states that one hidden layer with a finite number of sigmoidal neurons is able to approximate every continuous function on a subset of $\mathbb{R}$. Since the finite number is not known beforehand, making the number of hidden-layer neurons an adjustable parameter in this approach, seems reasonable and so does starting with a small number (five neurons).

The initial learning rate of Adam optimisation impacts how significant the location in the weight space changes after a weight update. Figuratively speaking, the larger the initial learning rate, the farther the optimiser can travel in the weight space, adding more flexibility and increasing the chance to find a suitable minimum. In this context, such a suitable minimum can be located at different positions, depending on the subdomain. It is not guaranteed by any means to find one near by the starting point. That motivates us to start the computation with a fairly small initial learning rate (values taken from [27]) and to enable ANDRe to increase this value outside the optimisation cycle. That is, the initial learning rate can increase several times before the number of hidden-layer neurons rearranges by two additional neurons. Adjusting the number of neurons resets the learning rate to its default parameter.

Once a subdomain has been successfully learned in this way, both parameters are reset to their initial values. Let us recall that the parameter adjustment does not take place during an optimisation cycle, it rather appears outside. In other words, we do not perturb the neural-network training during the optimisation process.

4.1 The evaluation of ANDRe for different initial-value problems

In [5] we have shown that the SCNF with a fixed number of subdomains is capable of solving IVPs on larger domains. Increasing this number resulted in a decreasing numerical error. Now with ANDRe, we show that by demanding the network error to become sufficiently small in each subdomain, the algorithm can automatically determine a suitable number (and distribution) of the subdomains.

The following paragraph will introduce the IVPs for our evaluation. We have chosen these examples because *(i)* each one represents a different IVP type, *(ii)* except for the last (system of IVPs) example, the analytical solutions are available and *(iii)* each of them incorporates at least one interesting behaviour. However, the difficulty is limited because of *(ii)*, but the focus of this paper does not lie on competitiveness in the first place. We rather show that the NF approach [7, 26] benefits from our extension in terms of accuracy on large domains. In addition, this paper serves as an investigation of the relation between both numerical error and neural-network loss.

Example IVPs and their analytical solutions As a first example, we take on the following IVP with constant coefficients

$$\begin{cases} \dot{\psi}(t) - t\sin(10t) + \psi(t) = 0, & \psi(0) = -1, \\ \psi(t) = \sin(10t)(\frac{99}{10{,}201} + \frac{t}{101}) + \cos(10t)(\frac{20}{10{,}201} - \frac{10t}{101}) - \frac{10{,}221}{10{,}201}e^{-t}, \end{cases} \tag{29}$$

which incorporates heavily oscillating and increasing characteristics, similar to instabilities. This example is still relatively simple and serves to demonstrate the main properties of our approach. We then proceed to an IVP with non-constant coefficients, that includes trigonometric and exponentially increasing terms:

$$\begin{cases} \dot{\phi}(t) + \frac{1+\frac{1}{1000}e^t\cos(t)}{1+t^2} + \frac{2t}{1+t^2}\phi(t) = 0, & \phi(0) = 5, \\ \phi(t) = \frac{1}{1+t^2}(-t - \frac{e^t\cos(t)}{2000} - \frac{e^t\sin(t)}{2000} + \frac{10{,}001}{2000}). \end{cases} \tag{30}$$

Furthermore, we choose to investigate the results for the non-linear IVP

$$\begin{cases} \frac{\dot{\omega}(t)}{\cos^2(\omega(t))}\frac{1}{\cos^2(2t)} - 2 = 0, & \omega(0) = \frac{\pi}{4}, \\ \omega(t) = \arctan(\frac{1}{4}\sin(4t) + t + 1), \end{cases} \tag{31}$$

which also has non-constant coefficients. Finally, we used ANDRe to solve the following non-linear system of IVPs

$$\begin{cases} \dot{\tau}(t) = A\tau(t) - B\tau(t)\kappa(t), & \tau(0) = \tau_0, \\ \dot{\kappa}(t) = -C\kappa(t) + D\tau(t)\kappa(t), & \kappa(0) = \kappa_0, \end{cases} \tag{32}$$

which is also known as the Lotka–Volterra equations [29], with parameters $A = 1.5, B = 1$, $C = 3, D = 1$. The initial values are $\tau_0 = 3, \kappa_0 = 1, \kappa_0 = 3, \kappa_0 = 5$ depending on the subsequent experiment. The chosen value for κ_0 will be explicitly addressed. Since the Lotka–Volterra equations in Eq. (32) do not have an analytical solution, we will compare the results to a numerical solution method, namely Runge–Kutta 4.

We take the coupled IVPs in Eq. (32) to demonstrate how the loss function for the NF approach reads. It is obtained as the sum of ℓ_2-norms of each equation, cf. Eq. (9). We use $\tilde{\tau}_C = \tilde{\tau}_C(t_{i,l}, \mathbf{P}_{m,l})$ and $\tilde{\kappa}_C = \tilde{\kappa}_C(t_{i,l}, \mathbf{P}_{m,l})$ as shortcuts:

$$\begin{aligned} E_l[\mathbf{P}_{m,l}] = \frac{1}{2(n+1)}\sum_{i=0}^{n}\big[&\{\dot{\tilde{\tau}}_C - A\tilde{\tau}_C + B\tilde{\tau}_C\tilde{\kappa}_C\}^2 \\ &+ \{\dot{\tilde{\kappa}}_C + C\tilde{\kappa}_C - D\tilde{\tau}_C\tilde{\kappa}_C\}^2\big]. \end{aligned} \tag{33}$$

This equation is then subject to optimisation/training and verification.

ANDRe and the analytical solutions In this section we demonstrate the results for applying ANDRe to the previously introduced example IVPs. We discuss the contrast to the analytical solutions and in the case of Lotka–Volterra, to the numerical results provided by Runge–Kutta 4. In addition to the already given computational parameters in Table 1,

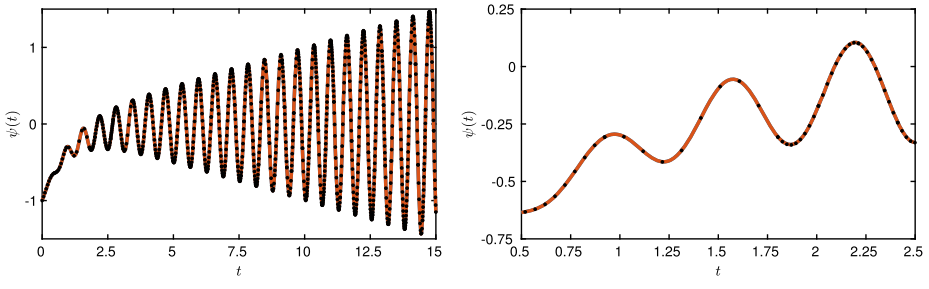
the problem-specific parameters are listed in Table 2. The corresponding initial conditions are given with the examples above.

The domain sizes are chosen in this way, so that interesting parts in the analytical solution are visible and as challenges available for ANDRe. During the experimental testing, we recognised that the neural-network loss and especially the verification loss $E_l^{\mathrm{VP}}[\mathbf{P}_{m,l}]$ were not becoming arbitrarily small. In addition, the experiments revealed the problem-specific dependencies of (local) minima locations in the weight space. Therefore, we had to find (in an experimental way) the verification-loss bounds σ for each example IVP.

In Fig. 6, both the analytical solution (orange/solid) and ANDRe solution (black/dotted) are shown for the IVP in Eq. (29). Table 3 shows that 113 subdomains were necessary in order to satisfy the chosen verification-loss bound. The corresponding (averaged) ℓ_1-error indicates a decent behaviour, which we consider to represent a reliable solution to the IVP. It also compares to the results from our SCNF approach [5] (predefined equidistant subdomain distribution). In comparison to ANDRe, a computation with the same number of subdomains (113) using the non-adaptive SCNF approach, returned an ℓ_1-error of 3.9294e-4. Therefore, ANDRe maintains the solution accuracy and comes with an advanced measurement metric to non-equidistantly distribute the training points (see Fig. 6b). We expected the density of this distribution to be higher at peaks and dips than in between. However, the subdomain $D_3 = [0.9229, 1.8027]$ is fairly large with certain peaks

Table 2 Problem-specific parameters, (σ) represents the verification-loss bound, (inc) is short for increments and refers to the learning procedure discussed in the context of Fig. 5

Example	Domain	σ	Inc.
IVP in Eq. (29)	$t \in [0, 15]$	1e-5	5
IVP in Eq. (30)	$t \in [0, 25]$	1e-4	5
IVP in Eq. (31)	$t \in [0, 20]$	1e0	2
IVP in Eq. (32)	$t \in [0, 30]$	1e-3	5



(a) entire domain (b) clipped domain

Figure 6 *IVP in Eq.* (29). Comparison between (orange/solid) analytical solution and (black/dotted) ANDRe solution

Table 3 Overview of the numerical results for the example IVPs in Eq. (29)–Eq. (32), where ℓ_1- and ℓ_∞-error are shown for IVPs with an exact solution given, (h) total number of learned subdomains

Example	Domain	h	ℓ_1-error	ℓ_∞-error
IVP in Eq. (29)	$t \in [0, 15]$	113	1.4499e-4	1.9268e-4
IVP in Eq. (30)	$t \in [0, 25]$	50	6.8152e-4	9.8980e-4
IVP in Eq. (31)	$t \in [0, 20]$	32	4.6545e-3	4.9861e-3
IVP in Eq. (32)	$t \in [0, 30]$	51	–	–

and dips. Compared to its adjacent subdomain $D_4 = [1.8027, 2.0089]$, the size of D_3 is unique, but also has a lower numerical error assigned. Hence, the (local) numerical error in one subdomain, as well as the subdomain size itself do not necessarily share the global behaviour, while a larger number of subdomains in the non-adaptive domain segmentation approach leads to a decreasing numerical error [5].

Figure 7 shows the subdomain distribution related to Fig. 6 in the beginning for $D = [0, 5]$. We find the domain-size adjustment parameter δ to show a significant influence here. It appears to be very important where one subdomain ends, as this may cause the adjacent one to be more difficult to solve. Please note that this statement holds under the consideration of the chosen neural-network parameters.

In contrast to the previous example, the IVP in Eq. (30) is solved on an even larger domain with extensively increasing values. The results are shown in Fig. 8 and aim to show that ANDRe is capable of solving time-integration problems on large domains with small neural networks. This is of particular importance as the domain size has been identified as an intricate parameter of the underlying problem, see also the detailed study in [26].

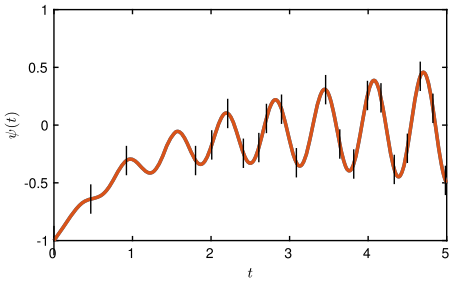


Figure 7 *IVP in Eq.* (29). ANDRe subdomain distribution for a cut-out of Fig. 6a, (orange/solid) analytical solution, (black/marked) subdomain boundaries, cf. Fig. 1

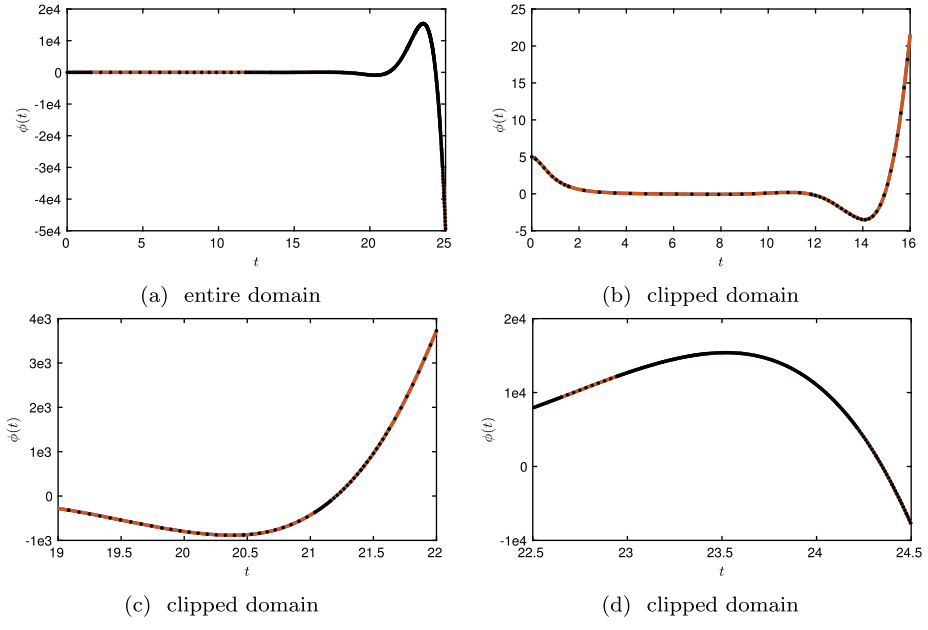
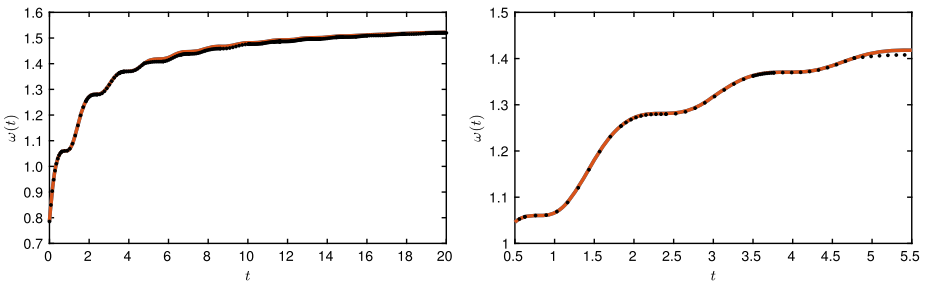


Figure 8 *IVP in Eq.* (30). Comparison between (orange/solid) analytical solution and (black/dotted) ANDRe solution



(a) entire domain (b) clipped domain

Figure 9 *IVP in Eq.* (31). Comparison between (orange/solid) analytical solution and (black/dotted) ANDRe solution

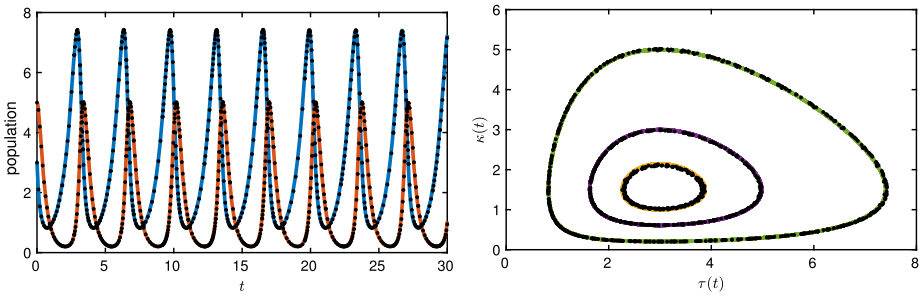
As displayed in Fig. 8, the ANDRe solution fits the analytical solution (orange) again on a qualitative and useful level. In total, the algorithm has finished after splitting the solution domain into 50 subdomains with the averaged ℓ_1-error of $\Delta\phi_1$ = 6.8152e-4 (cf. Table 3). In comparison, the non-adaptive SCNF approach did solve the IVP for the 50 subdomains ANDRe required.

Results for the IVP in Eq. (31) are displayed in Fig. 9. We decided to investigate this example because of the saddle points, which repeatedly occur. We observe a reliable solution approximation in the beginning of Fig. 9a. However, from subdomain D_7 and t = 4.7760 on, we can see that ANDRe starts to differ from the analytical solution. Although it keeps the general trend, and seems to converge against the analytical solution again in the end, the differences in this region are remarkable. This also marks a turning point computational-wise, which we will discuss more in detail in the corresponding experimental paragraph. Nonetheless, we had to decide to limit the verification-loss bound to σ = 1e0, since the computation with a lower error bound always became stuck around this area. This means both the Adam learning rate and the number of hidden-layer neurons started to increase heavily. Although one would suggest, based on the universal approximation theorem, that at some point ANDRe would move on, we cancelled the time-consuming computation at this point. A computation with 32 subdomains using the non-adaptive SCNF approach returned an ℓ_1-error of $\Delta\omega_1$ = 2.1308e-3 and therefore compares to the results from the ANDRe solution.

In Fig. 10, both the Runge–Kutta 4 solutions and the ANDRe solutions are shown. For a fair comparison on the quantitative side, both methods should use an equal number of training points, which in this case would arise from ANDRe solution. However, we are more interested in a qualitative comparison, since the Runge–Kutta 4 is known to provide very good results. ANDRe found a useful solution for the Lotka–Volterra equations in Fig. 10a, since there are only minor differences from the qualitative perspective.

Figure 10b shows the solution related to three different initial values for the predators. Let us note that although Fig. 10b only displays the solution at the training points (the same holds for the previous example IVPs), the trained SCNF is capable of evaluating the solution at every arbitrary discrete grid point over the entire domain, which is an advantage over numerical integration methods.

Numerical error and neural-network loss The measurement metrics (numerical error and verification loss) are highly relevant to discuss for ANDRe. In the subsequent diagrams



(a) Population over time, $\tau(0) = 3$, $\kappa(0) = 5$ (b) Comparison between $\tau(t)$ and $\kappa(t)$ in phase space with $\tau(0) = 3$ and (yellow) $\kappa(0) = 1$, (purple) $\kappa(0) = 3$, (green) $\kappa(0) = 5$.

Figure 10 *IVP in Eq.* (32). Comparison between (coloured/solid) Runge–Kutta 4 solution with 1e3 grid points and (black/dotted) ANDRe solution

we show the ℓ_1-error (blue/solid), the ℓ_∞-error (black/dotted), as well as the verification loss (green/solid) and the training loss (orange/marked) over the successfully learned subdomains.

Commenting on the relation between the verification and the training loss in Fig. 11a for the IVP in Eq. (29) (cf. Eqs. (23) and (24)), we see that both are mostly equal. This implies that the corresponding subdomains have been effectively learned up to the desired state. Since the main goal of ANDRe is to make use of the verification loss as a measurement metric for the numerical error, finding a relation between both metrics is desirable. In Fig. 11a, there are some regions that may indicate such a relation. However, in other regions almost no consistent relation is visible, which makes it difficult to find a clear statement regarding a possible relation.

Let us comment on the behaviour of both verification and training loss, displayed in Fig. 11b. While both match, they undergo the preset of σ = 1e-4 in some cases by several orders. Two adjacent subdomains may have a verification/training loss with significant differences. In addition to local differences between subdomains, all of the results displayed in this paragraph show unique trends, solidifying the difficulty of finding a relation between the neural-network loss and the numerical error.

Commenting on Fig. 11c for the IVP in Eq. (31), the possible local relations between the numerical error and neural-network loss seem to have disappeared. For the corresponding verification-loss bound σ = 1e0, even the training and verification loss are most of the time no longer equal. We find that even when the training/verification loss indicates a shallow (local) minimum in the weight space, the numerical error can still be useful. These findings are interesting, since in some parts one may not consider the neural network to have learned the underlying structure, based on the loss.

The results for the Lotka–Volterra equations in Fig. 11d also seem to indicate an arbitrary behaviour. That is, the local minima of orders around ≈1e-8 relate to arbitrary subdomains, that are not connected to, e.g. the periodic extreme points of the solution.

In the context of using neural networks, the initialisation plays an important role. Although we have decided to test ANDRe with a deterministic initialisation (all weights are zero), to avoid stochastic influences, a random initialisation is supposed to be more suitable in general. Therefore, we also want to provide results regarding different random initialisation. Table 4 shows five different initialisations with random values between $[-1, 1]$

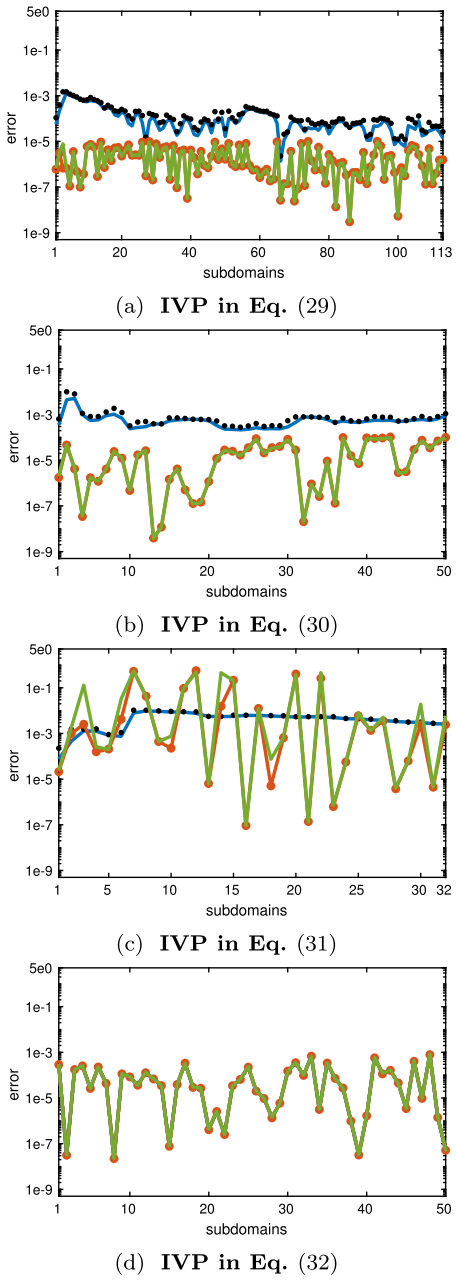


Figure 11 Error comparison, (blue/solid) numerical error, (black/dotted) infinity norm, (orange/marked) training loss, (green/solid) verification loss

Table 4 *IVP in Eq.* (29). Results for different (random) initialisations

Number	h	$\Delta\psi_1$
1	105	1.2793e-4
2	112	1.7026e-4
3	104	1.2479e-4
4	111	7.4877e-5
5	108	8.4929e-5

for the IVP in Eq. (29). Let us recall that ANDRe with a deterministic initialisation required 113 subdomains with an ℓ_1-error of $\Delta\psi_1$ = 1.4499e-4. We find that the results using ran-

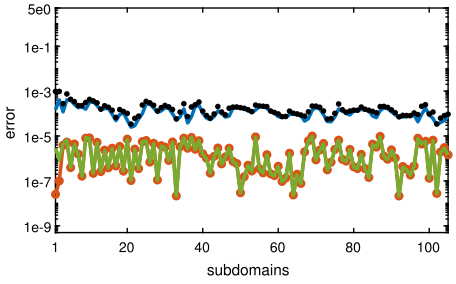
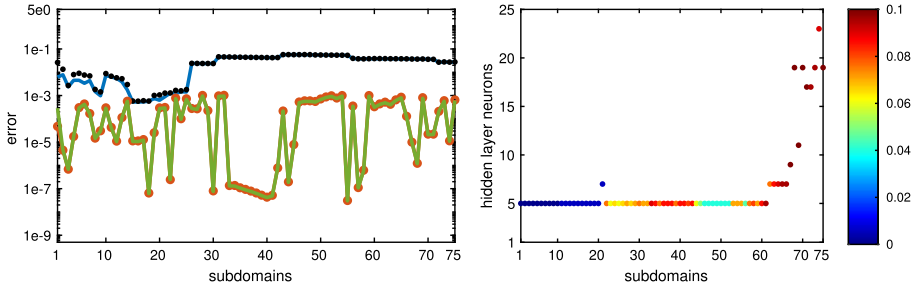


Figure 12 *IVP in Eq.* (29). Solved with ANDRe and random initialisation, (blue/solid) ℓ_1-error, (black/dotted) ℓ_∞-error, (orange/marked) training loss, (green/solid) verification loss



(a) Error comparison, (blue/solid) ℓ_1-error, (black/dotted) ℓ_∞-error, (orange/marked) training loss, (green/solid) verification loss.

(b) Visualisation of the automatic parameter adjustment (hidden-layer neurons and learning rate) over the subdomains, (coloured) Adam learning rate α.

Figure 13 *IVP in Eq.* (30). Solved with ANDRe and the alternative SCNF approach in Eq. (34)

dom initial weights are similar, also regarding the number of subdomains that are required to find a suitable solution. However, the range of $\Delta\psi_1$ clearly shows how sensitive a neural-network approach is in terms of numerical accuracy, where some random initial weights lead to a better or a less good ℓ_1-error. For initialisation number 1 in Table 4, we display the numerical errors and the training/verification loss in Fig. 12. The general trend here is also comparable to its deterministic counterpart in Fig. 11a. We find this result to be a justification of using deterministic initial weights for testing, although random initialisations are preferable for future work.

The diagram in Fig. 13a shows the results for a different SCNF approach [5, 10], combined with ANDRe. In contrast to the NF approach described in Sect. 2.1 (using the initial condition to construct the NF), now we directly combine with neural networks with the polynomial ansatz [5, 26]:

$$\tilde{\phi}_C(t_{i,l}, \mathbf{P}_{m,l}) = N_1(t_{i,l}, \mathbf{p}_{1,l}) + \sum_{k=2}^{m} N_k(t_{i,l}, \mathbf{p}_{k,l})(t_{i,l} - t_{0,l})^{k-1}. \tag{34}$$

Since the initial condition is not included in Eq. (34), it appears as an additional term directly in the loss function. Here, we use

$$g(t) = \frac{1 + \frac{1}{1000}e^t \cos(t)}{1 + t^2} \tag{35}$$

as a shortcut for:

$$E_l[\mathbf{P}_{m,l}] = \frac{1}{2(n+1)} \sum_{i=0}^{n} \left\{ \dot{\tilde{\phi}}_C(t_{i,l}, \mathbf{P}_{m,l}) + g(t) + \frac{2t}{1+t^2} \tilde{\phi}_C(t_{i,l}, \mathbf{P}_{m,l}) \right\}^2 + \frac{1}{2} \left\{ N_1(t_{0,l}, \mathbf{p}_{1,l}) - \tilde{\phi}_C(t_{0,l}, \mathbf{P}_{m,l}) \right\}^2. \tag{36}$$

Hence, the initial condition is learned by the first neural network. The loss-function construction concept is very similar to physics-informed neural networks [12, 13]. However, the polynomial approach is different in this context. Let us recall that the initial values follow $\tilde{\phi}_C(t_{0,1}, \mathbf{P}_{m,1}) = \phi(0)$ in the leftmost subdomain and $\tilde{\phi}_C(t_{0,l}, \mathbf{P}_{m,l}) = \tilde{\phi}_C(t_{n,l-1}, \mathbf{P}_{m,l-1})$ elsewhere. Since both approaches (Eq. (7) and Eq. (34)) only differ in their loss-function construction, it appears natural to compare them. That is, the results in Fig. 13a compare to Fig. 11b. The behaviour of both verification/training loss does not seem to be connected to the numerical error, for both methods. Figure 13a shows less-accurate results for the ℓ_1-error. The loss of accuracy possibly relates to the fact that here the new initial condition for the next subdomain is not fixed by adding it to the NF. It rather has to be learned again, which in practice may harm the usefulness of this approach. The gap between both ℓ_1-error and ℓ_∞-error closes at a certain point.

When turning to Fig. 13b, we observe that the parameter adjustment brought the Adam learning rate (coloured) up to various values in order to finish learning the subdomains. Additionally, the necessary number of hidden-layer neurons also heavily increases towards higher subdomains.

Although the results in terms of the numerical error are not better than in Fig. 11b, we see here the benefits of the automatic parameter adjustment. With this feature, ANDRe was able to solve the IVP. That is, we see our approach to enable the parameter adjustment when necessary, to be justified by the results. However, this does not support the overall usage of this alternative SCNF approach in the context of ANDRe.

Method and parameter evaluation In this paragraph we investigate and evaluate different parts of the method.

In Fig. 14, the sizes of the learned subdomains for the IVP in Eq. (29) are shown. As the general trend points towards smaller subdomains, we witness that there are local differences. Let us compare both the numerical error in Fig. 11a and the subdomain sizes in Fig. 14. In the first ten subdomains there seems to be a certain correlation, a larger size in this range results in a larger numerical error. A smaller verification-loss bound σ to deal with the discrepancy between verification and training loss in Fig. 11a may have resulted

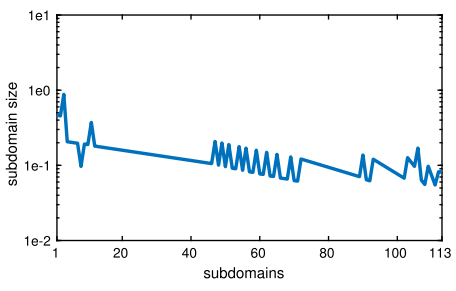


Figure 14 *IVP in Eq. (29).* Visualisation of the learned subdomain sizes

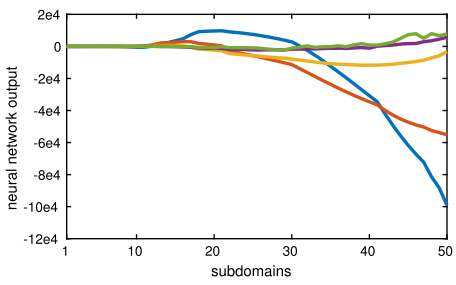


Figure 15 *IVP in Eq.* (30). Output of the incorporated SCNF neural networks, (blue) N_1, (orange) N_2, (yellow) N_3, (purple) N_4, (green) N_5

Table 5 *IVP in Eq.* (29). Results for a complete learning procedure for one subdomain

t_{25}	t_{26}	$\Delta\psi_1$	E_{25}^{TP}	E_{25}^{VP}	α
5.5953	15.000	1.3557	22.888	23.598	1e-3
5.5953	10.298	2.3780	1.7622	22.695	1e-3
5.5953	7.9465	1.3774	6.8529	34.448	1e-3
5.5953	7.9465	0.3868	10.640	9.9730	6e-3
5.5953	6.7709	3.4447e-2	3.2209e-2	3.9242e-2	6e-3
5.5953	6.1831	1.1881e-2	3.4791e-2	3.5682e-2	6e-3
5.5953	5.8892	3.3488e-4	1.0875e-4	1.0779e-4	6e-3
5.5953	5.7423	1.6882e-4	2.4565e-6	2.3747e-6	6e-3

in another size reduction with better results. However, the statement that a smaller (local) subdomain size implies a better numerical error does not hold here. Since the lower limit for the subdomain size is set to 0.1 in ANDRe, in practice, at a certain point, the subdomain size does not become smaller than this preset value.

Turning to Fig. 15, the (learned) neural-network outputs are displayed for the incorporated SCNF (cf. Eq. (7)) order $m = 5$ of the IVP in Eq. (30) (cf. Fig. 8). That is, the displayed graphs represent each of the five neural networks $N_k(t_{i,l}, \mathbf{p}_k)$, $k = 1, \ldots, 5$, without the scaling factors $(t_{i,l} - t_{0,l})^k$, over the subdomains. We see the first and second SCNF orders dominate the results in the later subdomains. A subdomain size below 1 implies a smaller influence of the higher SCNF orders. However, since there are the raw network outputs displayed, the scaling factors appear to have a direct impact on the training of the corresponding network.

In Table 5, quantitative results for the entire learning process of one subdomain of the IVP in Eq. (29) are displayed. The left subdomain boundary t_{25} remains constant, while the right subdomain boundary t_{26} is adjusted as in Eq. (21). The verification-loss values E_{25}^{VP} demonstrate the appearance of non-uniform learning during the solution process and show how important the verification loss and the parameter adjustment are. While E_{25}^{TP} decreases (as intended) for the first two subdomain-size reductions, it increases for the third one, which leads to a growth of the initial learning rate α. Now, for the same subdomain size, E_{25}^{VP} decreased significantly (while E_{25}^{TP} has increased again). That circumstance enables ANDRe to continue reducing the subdomain size until it is sufficiently small.

From the perspective of employing a condition for minimising the loss function, the question arises as to how the algorithm outcome is affected by different error-bound values σ. Table 6 shows the overall ℓ_1-error, verification loss and training loss for different σ regarding the IVP in Eq. (30). The choice of σ has a direct impact on each error value, as they all decrease the smaller σ becomes. However, an experiment for σ = 1e-7 did not finish learning the subdomains. We terminated the computation after the num-

Table 6 *IVP in Eq.* (30). Results for different σ, on domain $t \in [0, 25]$

σ	h	$\Delta\phi_1$	$E^{\mathrm{VP}}[\mathbf{P}_{m,l}]$	$E^{\mathrm{TP}}[\mathbf{P}_{m,l}]$
1e-1	37	5.5512e-2	1.8907e-2	1.8537e-2
1e-2	39	1.0749e-2	1.5714e-3	1.6255e-3
1e-3	47	6.2122e-3	1.1582e-4	1.1917e-4
1e-4	50	6.8152e-4	2.6581e-5	2.7788e-5
1e-5	59	3.0005e-4	1.9260e-6	2.2057e-6
1e-6	74	1.1870e-4	2.1157e-7	2.2076e-7

Table 7 *IVP in Eq.* (31). Results for different domain-size reduction parameter values δ, on domain $t \in [0, 5]$

δ	h	$\Delta\omega_1$
0.9	5	6.2398e-3
0.8	5	7.0250e-4
0.7	4	3.4629e-3
0.6	5	9.1571e-2
0.5	4	6.9541e-4
0.4	5	6.2569e-3
0.3	5	4.4258e-3
0.2	7	1.1200e-3
0.1	13	2.7217e-4

Table 8 *IVP in Eq.* (31). Results for different numbers of training points n_{TP}, on domain $t \in [0, 10]$

n_{TP}	h	$\Delta\omega_1$
10	10	1.4801e-2
20	12	4.5183e-3
40	13	3.4629e-3
80	10	4.6137e-2

ber of hidden-layer neurons crossed fifty one. In this subdomain, the smallest verification loss was 1.9881e-7 but the optimisation did not manage to go below σ = 1e-7. This phenomenon may again relate to the complexity of the loss-function energy landscape. Either such a (local) minimum could not be found by the Adam optimiser for various reasons, or even the global minimum is still too shallow for that error bound. The results in Table 6 also confirm the results from [5], where an increasing number of subdomains shows a decreasing numerical error.

In the following, we want to discuss experimental results for different domain-resize parameter values δ in Table 7 for the IVP in Eq. (31). The higher this value is set, the more aggressive each subdomain is reduced in size. On the other hand, the smaller δ is, the finer the subdomains are reduced in size. However, one would expect the necessary number of subdomains to increase, the larger the resize parameter is. This assumption is based on the fact that a larger δ possibly reduces the size of each subdomain much more than necessary, making them smaller than required. However, in reality the results and the amount of subdomains are comparable for all δ, if we exclude $\delta = 0.1$. On the ℓ_1-error side, except for $\delta = 0.6$, all the results are comparable. Although the results are highly problem specific and may change with a larger domain size, we find $\delta = 0.5$ to provide the best mix with $h = 4$ and $\Delta\omega_1$ = 6.9541e-4. This domain-size parameter was used for all the computations.

For the last parameter-evaluation experiment we decided to investigate different numbers of training points n_{TP}. For each computation, see Table 8, we doubled the number

of training points. There are always 20% more grid points used for verification (n_{VP}) than for training. The results indicate, at least for this particular IVP and computational setup, that doubling the number of grid points from initially $n_{\mathrm{TP}} = 10$ to $n_{\mathrm{TP}} = 20$ is beneficial. The ℓ_1-error is almost one order better in this case. However, the verification-loss bound is still $\sigma = 1e0$ as even with more grid points the computation did not finish for smaller σ. Continuing the research on ANDRe, further investigations are required to determine why the first-order optimisation is sometimes not able to minimise the training/verification loss up to a desired bound (σ).

5 Conclusion and future work

The proposed ANDRe is based on two components. First, the resulting verification loss arising from the inbound subdomain collocation neural form (SCNF) acts as a measurement metric and refinement indicator. The second component is the proposed algorithm that refines the solution domain in an adaptive way. In this paper, we have shown that the approach is capable of solving time-dependent DEs of different types, incorporating various interesting characteristics, in particular including large domains and extensive variations of solution values.

A significant advantage of ANDRe is the verification step to make sure that the solution is also useful outside of the chosen training points. All this makes ANDRe a unique and conceptually useful framework.

However, several questions remain open for future work. While there seems to be a certain and natural relation between the neural-network loss and the numerical error, in reality this relation appears to be sometimes a sensitive issue. It is unclear yet, whether some minima in the loss-function energy landscape contribute better to the numerical error, or not. However, we find the verification loss to already serve as a useful error indicator in ADNRe. In addition, we would like the numerical error to proportionally correspond to the neural-network verification. If we could manage to achieve an improvement in the correlation between both errors or understand the relation more in detail on the theoretical level, we think that the ANDRe approach can perform even better in the future.

We also find it relevant to further investigate the computational parameters and fine tuning the parameter-adjustment part of ANDRe. The verification step may be considered as a part in the optimisation process, to predict early, whether a further optimisation in the corresponding subdomain is useful or a size reduction is mandatory. This could lower the computational cost but has to be incorporated and tested carefully so as not to lose any information during the optimisation process.

Since ANDRe represents an additional discretisation in time, the approach should also work for PDEs with both time and spatial components and it appears natural to extend in future work the method to multidimensional DEs.

Acknowledgements

The authors want to thank the unknown reviewer for his or her excellent job on carefully reading this manuscript and providing highly useful feedback and suggestions for further improving this work.

Funding

Open Access funding enabled and organized by Projekt DEAL. This publication was funded by the Graduate Research School (GRS) of the Brandenburg University of Technology Cottbus-Senftenberg and the Federal Ministry for Education and Research (Germany) as part of the project KI@MINT.

Abbreviations
DE, differential equation; NF, neural form; FEM, finite-element method; PDE, partial differential equation; ANDRe, adaptive neural-domain refinement; SCNF, subdomain collocation polynomial neural form; TP, training points; VP, verification points..

Availability of data and materials
The authors declare that upon reasonable request, the code is available from the corresponding author.

Declarations

Competing interests
The authors declare no competing interests.

Author contributions
The authors declare that the main idea and experiments were proposed by TS and the manuscript was written in cooperation with MB. All authors read and approved the final manuscript.

Received: 18 January 2023 Accepted: 11 October 2023 Published online: 25 October 2023

References

1. Antia, H.M.: Numerical Methods for Scientists and Engineers, 1st edn. Hindustan Book Agency, New Delhi (2012)
2. Maede, A.J. Jr, Fernandez, A.A.: The numerical solution of linear ordinary differential equations by feedforward neural networks. Math. Comput. Model. **19**(12), 1–25 (1994). https://doi.org/10.1016/0895-7177(94)90095-7
3. Yadav, N., Yadav, A., Kumar, M.: An Introduction to Neural Network Methods for Differential Equations. SpringerBriefs in Applied Sciences and Technology. Springer, Dordrecht (2015). https://doi.org/10.1007/978-94-017-9816-7
4. Dissanayake, M.W.M.G., Phan-Thien, N.: Neural-network-based approximations for solving partial differential equations. Commun. Numer. Methods Eng. **10**(3), 195–201 (1994). https://doi.org/10.1002/cnm.1640100303
5. Schneidereit, T., Breuß, M.: Collocation polynomial neural forms and domain fragmentation for initial value problems. Neural Comput. Appl. **34**, 7141–7156 (2022). https://doi.org/10.1007/s00521-021-06860-4
6. Mall, S., Chakraverty, S.: Application of Legendre neural network for solving ordinary differential equations. Appl. Soft Comput. **43**, 347–356 (2016). https://doi.org/10.1016/j.asoc.2015.10.069
7. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. **9**(5), 987–1000 (1998). https://doi.org/10.1109/72.712178
8. Lagari, P.L., Tsoukalas, L.H., Safarkhani, S., Lagaris, I.E.: Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions. Int. J. Artif. Intell. Tools **29**(5), 2050009 (2020). https://doi.org/10.1142/S0218213020500098
9. Schneidereit, T., Breuß, M.: Solving ordinary differential equations using artificial neural networks—a study on the solution variance. In: Proceedings of the Conference Algoritmy, pp. 21–30 (2020)
10. Piscopo, M.L., Spannowsky, M., Waite, P.: Solving differential equations with neural networks: applications to the calculation of cosmological phase transitions. Phys. Rev. D **100**(1), 016002 (2019). https://doi.org/10.1103/PhysRevD.100.016002
11. Lagaris, I.E., Likas, A., Papageorgiou, D.G.: Neural-network methods for boundary value problems with irregular boundaries. IEEE Trans. Neural Netw. **11**(5), 1041–1049 (2000). https://doi.org/10.1109/72.870037
12. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045
13. Jagtap, A.D., Kharazmi, E., Karniadakis, G.E.: Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems. Comput. Methods Appl. Mech. Eng. **365**, 113028 (2020). https://doi.org/10.1016/j.cma.2020.113028
14. Blechschmidt, J., Ernst, O.G.: Three ways to solve partial differential equations with neural networks—a review. GAMM-Mitt. **44**(2), e202100006 (2021). https://doi.org/10.1002/gamm.202100006
15. Hairer, E., Nørsett, S.P., Wanner, G.: Solving Ordinary Differential Equations 1: Nonstiff Problems, 2nd edn. Springer Series in Computational Mathematics. Springer, Berlin (1993). https://doi.org/10.1007/978-3-540-78862-1
16. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations 2: Stiff and Differential-Algebraic Problems, 2nd edn. Springer Series in Computational Mathematics. Springer, Berlin (1996). https://doi.org/10.1007/978-3-642-05221-7
17. Zienkiewicz, O.C., Taylor, R.L., Zhu, J.Z.: The Finite Element Method: Its Basis and Fundamentals, 7th edn. Elsevier Butterworth-Heinemann, Oxford (2013). https://doi.org/10.1016/B978-1-85617-633-0.00019-8
18. Bangerth, W., Rannacher, R.: Adaptive Finite Element Methods for Differential Equations. Lectures in Mathematics. Springer, Basel (2003). https://doi.org/10.1007/978-3-0348-7605-6
19. Berger, M.J., Oliger, J.: Adaptive mesh refinement for hyperbolic partial differential equations. J. Comput. Phys. **53**(3), 484–512 (1984). https://doi.org/10.1016/0021-9991(84)90073-1
20. Verfürth, R.: A posteriori error estimation and adaptive mesh-refinement techniques. J. Comput. Appl. Math. **50**(1), 67–83 (1994). https://doi.org/10.1016/0377-0427(94)90290-9
21. Alfonzetti, S.: A finite element mesh generator based on adaptive neural network. IEEE Trans. Magn. **34**(5), 3363–3366 (1998). https://doi.org/10.1109/20.717791
22. Bohn, J., Feischl, M.: Recurrent neural networks as optimal mesh refinement strategies. Comput. Math. Appl. **97**, 61–76 (2021). https://doi.org/10.1016/j.camwa.2021.05.018
23. Manevitz, L., Bitar, A., Givoli, D.: Neural network time series forecasting of finite-element mesh adaptation. Neurocomputing **63**, 447–463 (2005). https://doi.org/10.1016/j.neucom.2004.06.009
24. Breuß, M., Dietrich, D.: Fuzzy numerical schemes for hyperbolic differential equations. In: KI 2009: Advances in Artificial Intelligence. Lecture Notes in Computer Science, vol. 5803, pp. 419–426. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-04617-9_53

25. Anitescu, C., Atroshchenko, E., Alajlan, N., Rabczuk, T.: Artificial neural network methods for the solution of second order boundary value problems. Comput. Mater. Continua **59**(1), 345–359 (2019). https://doi.org/10.32604/cmc.2019.06641
26. Schneidereit, T., Breuß, M.: Computational characteristics of feedforward neural networks for solving a stiff differential equation. Neural Comput. Appl. **34**, 7975–7989 (2022). https://doi.org/10.1007/s00521-022-06901-6
27. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization (2017). arXiv:1412.6980
28. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989). https://doi.org/10.1007/BF02551274
29. Anisiu, M.C.: Lotka, Volterra and their model. Didáct. Math. **32**, 9–17 (2014)

Publisher's Note