
DEEP EULER METHOD: SOLVING ODES BY APPROXIMATING THE LOCAL TRUNCATION ERROR OF THE EULER METHOD

A PREPRINT

Xing Shen

School of Mathematical Sciences
Zhejiang University
Hangzhou, Zhejiang, China
shenxingsx@zju.edu.cn

Xiaoliang Cheng

School of Mathematical Sciences
Zhejiang University
Hangzhou, Zhejiang, China
xiaoliangcheng@zju.edu.cn

Kewei Liang*

School of Mathematical Sciences
Zhejiang University
Hangzhou, Zhejiang, China
mat1kw@zju.edu.cn

March 24, 2020

ABSTRACT

In this paper, we propose a deep learning-based method, deep Euler method (DEM) to solve ordinary differential equations. DEM significantly improves the accuracy of the Euler method by approximating the local truncation error with deep neural networks which could obtain a high precision solution with a large step size. The deep neural network in DEM is mesh-free during training and shows good generalization in unmeasured regions. DEM could be easily combined with other schemes of numerical methods, such as Runge-Kutta method to obtain better solutions. Furthermore, the error bound and stability of DEM is discussed.

Keywords Deep Euler Method, Deep neural network, Ordinary differential equation

1 Introduction

Many problems in science and engineering can be modeled into a set of ordinary differential equations (ODEs)

$$G(x, y, y', y'', \dots) = 0, \quad x \in [a, b] \subset \mathbb{R}.$$

In most cases, it can not be easy to obtain the analytic solution and so one must typically rely on a numerical scheme to accurately approximate the solution. The important issues confronting the numerical study appear in the initial value problems since higher-order ODEs can be converted into the system of the first-order ODEs. Basic methods for initial value problems are the extremely popular Euler method or the Runge-Kutta method. However, numerical methods have often to balance the discretization step size and computation time. Furthermore, the class of stiff ordinary differential equations may still present a more serious challenge to numerical computation.

In recent years, there has been a growing interest in solving the differential equations and the inverse problems by deep learning. The works include numerical solutions of ODEs and PDEs ([18], [14], [21], [3]), recovery of the involving systems ([2], [7], [8]), overcoming the curse of dimension of high-dimensional PDEs ([4], [6]), uncertainty quantification ([22], [26]) etc. Besides, several works have focused on the combination of traditional numerical methods and deep neural networks. ([20]) proposed a merger of Galerkin methods and deep neural networks (DNNs) to solve high-dimensional partial differential equations (PDEs). They trained DNNs to satisfy the differential operator, initial condition, and boundary conditions. ([15]) introduced physics-informed neural networks (PINNs), which is a deep learning framework for the synergistic combination of mathematical models and data. Following the physical laws of the control dynamics system, PINNs can deduce the solution of PDE and obtain the surrogate model. ([25]) presented a deep Ritz method for the numerical solution of variational problems based on the Ritz method. ([5]) theoretically analyzed the relationship between DNN and finite element method(FEM). They explored the ReLU DNN representation of a continuous piecewise linear basis function in the finite element method. ([11]) proposed PDE-Net

*

to predict the dynamics of complex systems. The underlying PDEs can be discovered from the observation data by establishing the connections between convolution kernels in CNNs and differential operators. Based on the integral form of the underlying dynamical system, ([13]) considered ResNet block as a one-step method and recurrent ResNet and recursive ResNet as multi-step methods. ([27]) approximated the evolution operator by a residual network to solve and recover unknown time-dependent PDEs. ([16]) blended the multi-step schemes with deep neural networks to identify and forecast nonlinear dynamical systems from data. ([17]) proposed neural networks based Model Order Reduction technique to solve dynamical systems arising from differential equations. ([24]) used reinforcement learning to empower Weighted Essentially Non-Oscillatory Schemes(WENO) for solving 1D scalar conservation laws.

It is well known that the forward Euler method is very easy to implement but it can't give accurate solutions. The main reason is that the Euler method has only one order approximation accuracy, which requires a very small step size for any meaningful result. This makes the Euler method rarely used in practical applications and motivates us to propose a new Euler method combined with DNNs. We call the new method as deep Euler method (DEM). DEM only has the most general structure of a fully connected neural network, without any special designs in its structure, such as residual connections. As with some other deep neural network models, DEM also learns its representation using supervised pre-training. After the neural network gets trained satisfactorily, we post-process it to predict the solution of the ODE. The key difference is that in DEM, we explicitly capture information of the local truncation error of the Euler method instead of directly approaching the solution of the ODE. DEM has achieved state-of-the-art performance in solving ODEs, which is much better than the conventional numerical method, especially than the classical Euler method. This success can be attributed to the ability of the deep neural network in learning very strong hierarchical nonlinear representation. In particular, breakthroughs in supervised learning training are essential for deep neural networks to effectively and robustly predict.

The paper is organized as follows. We introduce the main idea of DEM in section 2 and give theoretical results of DEM in section 3. Based on DEM, we also derive other schemes of the single-step method for solving ODEs in section 4. In Section 5, numerical examples are given to demonstrate the capability and effectiveness of DEM. Finally, we conclude the paper in section 6.

2 Deep Euler Method

2.1 Formulation

Considering the following initial value ordinary differential equation:

$$\begin{cases} \frac{dy}{dx} = f(x, y), & x \in I = [a, b], \\ y(a) = c, \end{cases} \quad (1)$$

where the solution $y(x) : I \rightarrow \Omega \subset \mathbb{R}^n$ and f satisfies the Lipschitz condition in y , i.e.,

$$\|f(x, y_1) - f(x, y_2)\| < L\|y_1 - y_2\|.$$

We introduce the discretization mesh (or sampling points) in x ,

$$a = x_0 < x_1 < \dots < x_M = b.$$

Let $h_m = x_{m+1} - x_m$ be the mesh size and y_m be the numerical approximation of $y(x_m)$. The forward Euler method for (1) is

$$\begin{cases} y_m = y_{m-1} + h_{m-1}f(x_{m-1}, y_{m-1}), & m = 1, \dots, M \\ y_0 = c. \end{cases} \quad (2)$$

Note that in most cases, we always adopt the uniform mesh for the forward Euler method, $h = h_m$ and $x_m = a + mh$, $m = 0, 1, \dots, M - 1$. The local truncation error and the global error are defined as

$$R_m = y(x_{m+1}) - y(x_m) - hf(x_m, y(x_m)) = \int_{x_m}^{x_{m+1}} f(x, y(s))ds - hf(x_m, y(x_m)). \quad (3)$$

and

$$e = |y(x_m) - y_m|,$$

respectively. It is well known that $R_m = \mathcal{O}(h^2)$ and $e = \mathcal{O}(h)$ ([9]).

To obtain higher accuracy than the Euler method, a direct scheme is to separate a part of R_m to improve the Euler step y_{m+1} , so as to reduce the local truncation error and the global error of Euler method. To this end, we introduce a feedforward neural network in DEM that infers the update of an Euler step. As universal approximators, multilayer

fully connected feedforward neural networks can approximate any continuous function arbitrarily ([10]). From (3), we could consider R_m as a continuous function of variables x_m, x_{m+1}, y_m . Thus, we utilize the fully connected neural network trained with enough measured data to approximate $\frac{1}{h_m^2} R_m$.

Let $\mathcal{N}(x_i, x_j, y_i; \theta) : \mathbb{R}^{n+2} \rightarrow \mathbb{R}^n$ be the nonlinear operator defined by a multilayer fully connected neural network. The parameter θ includes all the weights and the biases in the neural network. DEM for (1) can be written as

$$\begin{cases} y_{m+1} = y_m + h_m f(x_m, y_m) + h_m^2 \mathcal{N}(x_m, x_{m+1}, y_m; \theta), & m = 0, \dots, M-1, \\ y_0 = c. \end{cases} \quad (4)$$

Formula (4) consists of Euler approximation and neural network approximation. The first part makes full use of the information of f to express the linearity of ODE. The latter corrects the results of Euler approximation to obtain higher accuracy and express nonlinear features. Abstractly, \mathcal{N} can be thought of as a parametric function that learns how to represent the local truncation error so that their most salient characteristics can be reconstructed from its inputs and outputs. The output of $\mathcal{N}(x_m, x_{m+1}, y_m; \theta)$ contains all features extracted in training to update the formula of the Euler method. Compared with using a neural network to approximate the solution of ODE directly, DEM separates the nonlinear part from the numerical scheme and then makes full use of neural networks to approximate the local truncation error of Euler method. Moreover, \mathcal{N} has the same function as the nonlinear denoising process. This provides a very powerful and flexible method for solving ODEs because we can impose high order error correction and reduce the constrain of the step size h in the Euler method. Hence, DEM can either improve the accuracy of the Euler method or speed up the computations of ODEs.

There are underlying principles for designing neural network architecture. In fact, we have another design of neural network $\mathcal{N}(x_i, x_j; \theta) : \mathbb{R}^2 \rightarrow \mathbb{R}^n$. The output of the neural network is still an approximation of the local truncation error, while the input only has x_m and x_{m+1} . In this case, if $n \gg 2$, the neural network becomes very difficult to train. Because the dimension of the output is much larger than that of the input, it is almost impossible for the neural network to predict the sophisticated target with such few features.

2.2 Details of DEM

DEM is only a standard multilayer fully connected neural network, without any special designs in its structure, such as residual connections. With the input $\mathbf{x} = (x_i, x_j, y_i) \in \mathbb{R}^{n+2}$, the neural network in DEM can be written as

$$\mathcal{N}(\mathbf{x}; \theta) = L_K \circ \sigma \circ L_{K-1} \cdots \sigma \circ L_1(\mathbf{x}).$$

The nonlinear activation function $\sigma(t) = \max\{0, t\}$ is rectified linear units (ReLU) function. The k -th hidden layer has the following form

$$L_k(z) = \mathbf{W}_k z + \mathbf{b}_k, \quad 1 \leq k \leq K,$$

where the weight matrix $\mathbf{W}_k \in \mathbb{R}^{p_k \times p_{k-1}}$, the bias $\mathbf{b}_k \in \mathbb{R}^{p_k}$, p_k is the number of neurons in the k -th layer.

We assume that the measurement data is contaminated by noise, so that the training dataset $\mathbf{D} = \{(x_j, z_j)\}_{j=1}^N$ has the form $z_j = y(x_j) + \delta_j$ and satisfies

$$\frac{1}{N} \sum_{j=0}^N \delta_j^2 \leq \delta^2,$$

where the scalar δ is called the noise level.

For any pair of measurements $\{(x_i, z_i), (x_j, z_j)\}$ ($x_i < x_j$), we introduce the local truncation error function

$$R(x_i, x_j, z_i, z_j) = \frac{1}{(\Delta x)^2} [z_j - z_i - \Delta x f(x_i, z_i)], \quad (5)$$

where $\Delta x = x_j - x_i$. With the following supervised loss

$$J(\theta) = \frac{2}{N(N-1)} \sum_{1 \leq i, j \leq N} \|\mathcal{N}(x_i, x_j, z_i; \theta) - R(x_i, x_j, z_i, z_j)\|_{L^1}, \quad (6)$$

DEM learns to approximate the local truncation error of the Euler method. The coefficient comes from $C_N^2 = \frac{N(N-1)}{2}$ the number of pairs in dataset D .

Note that for any input pair $\{x_i, x_j\}$, each of the training captures the features in the local truncation error, which have close relations with f . Once features extractors corresponding to all pairs are trained and the strong hierarchical non-linear representations are generated, any new $y(x)$ ($x \neq x_i \in D$) is then represented by (4). On the other hand,

DEM is mesh-free because all training data can be generated randomly and not necessary to locate at mesh points. Moreover, recalling the local truncation error of the Euler method is proportional to the square of the step size, we have $\frac{1}{h^2}R_m = \mathcal{O}(1)$. Hence, the neural network of DEM approaches a non-linear continuous function of $\mathcal{O}(1)$, which is much easier than directly approximating the solution of the ODE. This makes DEM easier to train and requires fewer data in training.

3 Theoretical Analysis

3.1 Error Bound

Lemma 1. *Assume that the trained neural network \mathcal{N} satisfies*

$$|\mathcal{N}(x_m, x_{m+1}, z_m; \theta) - R(x_m, x_{m+1}, z_m, z_{m+1})| < \mathcal{O}(\eta).$$

If $\delta < \eta$ and $h > \sqrt{\frac{\delta}{\eta}}$, then

$$\left| \mathcal{N}(x_m, x_{m+1}, y(x_m); \theta) - \frac{1}{h^2}R_m \right| < \mathcal{O}(\eta).$$

Proof. From Lemma 4 in [28], we have the conclusion that the neural network in DEM is Lipschitz continuous. That is, for any $\mathbf{x}_1, \mathbf{x}_2$,

$$|\mathcal{N}(\mathbf{x}_1; \theta) - \mathcal{N}(\mathbf{x}_2; \theta)| \leq L_{\mathcal{N}} \|\mathbf{x}_1 - \mathbf{x}_2\|,$$

where $L_{\mathcal{N}} = \alpha^K \beta^K$, $\alpha = \max_{1 \leq k \leq K} \|W_k\|_{\infty}$, $\beta = \max_{a, b \in \mathbb{R}, a \neq b} \frac{|\sigma(a) - \sigma(b)|}{|a - b|}$, K is the number of layers of the neural network. From

$$\begin{aligned} |\mathcal{N}(x_m, x_{m+1}, y(x_m); \theta) - \mathcal{N}(x_m, x_{m+1}, z_m; \theta)| &\leq L_{\mathcal{N}} |y(x_m) - z_m| \\ &\leq CL_{\mathcal{N}} \delta < \mathcal{O}(\eta), \quad (C \text{ is a constant}) \end{aligned}$$

and

$$\begin{aligned} &\left| R(x_m, x_{m+1}, z_m, z_{m+1}) - \frac{1}{h^2}R_m \right| \\ &= \frac{1}{h^2} \left| [z_{m+1} - y(x_{m+1})] - [z_m - y(x_m)] - h[f(x_m, z_m) - f(x_m, y(x_m))] \right| \\ &\leq \frac{2C\delta}{h^2} + \frac{L\delta}{h} \\ &\leq \mathcal{O}(\eta), \end{aligned}$$

we have

$$\begin{aligned} \left| \mathcal{N}(x_m, x_{m+1}, y(x_m); \theta) - \frac{1}{h^2}R_m \right| &< |\mathcal{N}(x_m, x_{m+1}, y(x_m); \theta) - \mathcal{N}(x_m, x_{m+1}, z_m; \theta)| \\ &+ \left| R(x_m, x_{m+1}, z_m, z_{m+1}) - \frac{1}{h^2}R_m \right| \\ &+ |\mathcal{N}(x_m, x_{m+1}, z_m; \theta) - R(x_m, x_{m+1}, z_m, z_{m+1})| \\ &\leq \mathcal{O}(\eta) \end{aligned}$$

□

For each pair of measurements $\{(x_i, z_i), (x_j, z_j)\}$, we use (5) to construct the training samples of the neural network. In the proof of Lemma 1, we have known that $\left| R(x_m, x_{m+1}, z_m, z_{m+1}) - \frac{1}{h^2}R_m \right|$ is smaller than a quantity which contains a factor $\frac{1}{h}$. This indicates the big h is a good choice. Therefore, we will only select the measurement pair with the large $h = x_j - x_i$ to construct training samples.

Theorem 1. *Under the assumptions of Lemma 1, the local truncation error of DEM is $\mathcal{O}(\eta h^2)$ and the global truncation error is $\mathcal{O}(\eta h)$.*

Proof. From (4), the local truncation error (LTE) of DEM is

$$\begin{aligned} LTE &= |y(x_{m+1}) - y(x_m) - hf(x_m, y(x_m)) - h^2\mathcal{N}(x_m, x_{m+1}, y(x_m); \theta)| \\ &\leq h^2 \left| \mathcal{N}(x_m, x_{m+1}, y(x_m); \theta) - \frac{1}{h^2} R_m \right| \\ &< \mathcal{O}(\eta h^2). \end{aligned}$$

Hence, we can also conclude that the global truncation error is $\mathcal{O}(\eta h)$. \square

Compared with the Euler method, the errors of DEM are reduced by η times. The solution with high accuracy can be obtained. Besides, the size constrains of h in the Euler method can be relaxed to speed up the computation of the solutions. For example, if the global error should be $\mathcal{O}(10^{-6})$, the step size in the Euler method is at most 10^{-6} . Starting from the initial $y(0)$, it takes 10^6 Euler steps to get the solution $y(1)$. If $\eta = 10^{-4}$, the step size in DEM can be 10^{-2} and the number of steps can be reduced to 10^2 , which is much smaller than the Euler method.

3.2 Numerical stability

Theorem 2. *Under the assumptions of Theorem 1, DEM is stable.*

Proof. For the initial values y_0 and z_0 ($y_0 \neq z_0$), DEM generates two approaches $\{y_m\}$ and $\{z_m\}$ with

$$\begin{aligned} y_m &= y_{m-1} + h_{m-1}f(x_{m-1}, y_{m-1}) + h_{m-1}^2\mathcal{N}(x_{m-1}, x_m, y_{m-1}; \theta), \\ z_m &= z_{m-1} + h_{m-1}f(x_{m-1}, z_{m-1}) + h_{m-1}^2\mathcal{N}(x_{m-1}, x_m, z_{m-1}; \theta). \end{aligned} \quad (7)$$

Since $\sum_{m=0}^{M-1} h_m = b - a$ and $|\mathcal{N}(x_{m-1}, x_m, y_{m-1}; \theta) - \mathcal{N}(x_{m-1}, x_m, z_{m-1}; \theta)| \leq L_{\mathcal{N}}|y_{m-1} - z_{m-1}|$, we have

$$\begin{aligned} |y_m - z_m| &\leq |y_{m-1} - z_{m-1}| + h_{m-1}|f(x_{m-1}, y_{m-1}) - f(x_{m-1}, z_{m-1})| \\ &\quad + h_{m-1}^2 L_{\mathcal{N}} |y_{m-1} - z_{m-1}| \\ &\leq (1 + h_{m-1}L + h_{m-1}^2 L_{\mathcal{N}}) |y_{m-1} - z_{m-1}| \\ &\leq \prod_{n=0}^{m-1} (1 + h_n L + h_n^2 L_{\mathcal{N}}) |y_0 - z_0| \\ &\leq C |y(0) - z(0)| \end{aligned}$$

where C is constant. \square

Considering the stiff equation $\frac{dy}{dx} = \lambda y$, where $\lambda < 0$ and the initial value $y(a) = c$. The stability domain of DEM is $\{h \in \mathbb{C} \mid |1 + h\lambda + h^2 L_{\mathcal{N}}| \leq 1\}$, while the Euler method is $\{h \in \mathbb{C} \mid |1 + h\lambda| \leq 1\}$. Although it can not be proved theoretically that the former must be larger than the latter, DEM can use a large step in numerical experiments. Under such a step size, the forward Euler method is certainly unstable. On the other hand side, a large stability domain can be obtained by adjusting $L_{\mathcal{N}}$. Recall that $L_{\mathcal{N}} = \alpha^k \beta^K$, where β is determined by the activation function. If ReLU activation function is adopted then $\beta = 1$. We can change the norm of weight matrix α by using the techniques of weight clipping ([19]) and weight normalization ([1]), to adjust $L_{\mathcal{N}}$. For example, when $\lambda = -5$ and $L_{\mathcal{N}} = 6$, The stability domain of DEM is $0 < h \leq \frac{5}{6}$, while the Euler method is $0 < h \leq \frac{2}{5}$. Hence we can use a larger step size to solve the equation than the Euler method.

4 Single Step methods

Based on the idea of approximating the local truncation error with a deep neural network, DEM could be generalized to other linear single-step methods. For instance, Heun's method

$$y_{m+1} = y_m + \frac{h}{2} [f(x_m, y_m) + f(x_{m+1}, y_m + hf(x_m, y_m))]$$

is a second-order Runge-Kutta method. We can also add a deep neural network in it and get

$$y_{m+1} = y_m + \frac{h}{2} [f(x_m, y_m) + f(x_{m+1}, y_m + hf(x_m, y_m))] + h_{m-1}^3 \mathcal{N}(x_m, x_{m+1}, y_m; \theta). \quad (8)$$

More generally, a p order single-step method of (1) can be written as:

$$\begin{cases} y_{m+1} = y_m + h\psi(x_m, y_m, h), \\ y_0 = c. \end{cases} \quad (9)$$

The local truncation error is

$$R_m = y_{m+1} - y_m - h\psi(x_m, y_m, h) = \mathcal{O}(h^{p+1}).$$

The method (9) can also be modified as

$$\begin{cases} y_{m+1} = y_m + h\psi(x_m, y_m, h) + h^{p+1}\mathcal{N}(x_m, x_{m+1}, y_m; \theta), \\ y_0 = c. \end{cases} \quad (10)$$

For (8) and (10), the local truncation errors are $\mathcal{O}(\eta h^3)$ and $\mathcal{O}(\eta h^{p+1})$, respectively. The corresponding global truncation errors are $\mathcal{O}(\eta h^2)$ and $\mathcal{O}(\eta h^p)$.

5 Numerical Example

5.1 Example 1

Considering the following initial value problem:

$$\begin{cases} \frac{dy}{dx} = \frac{3}{2} \frac{y}{x+1} + \sqrt{x+1}, x \in [0, 10] \\ y(0) = 0 \end{cases} \quad (11)$$

where the exact solution is $y = (x+1)^{3/2} \log(x+1)$.

At first, we highlight the performance of DEM with different step sizes with noise-free measured data. We generate 200 random noise-free measured data $\{(t_i, y(t_i))\}_{i=1}^{200}$, by sampling from a uniform distribution $\mathcal{F}_t = U(0, 5)$, then we train the deep neural network \mathcal{N} by minimizing the loss function of (6). All norms used in this paper are \mathcal{L}_1 norm. The neural network, with 8 layers, 80 neurons and the ReLU activation function in each layer, is trained for 50 epochs, optimized with Adam. All the learning rate in this paper is set to be 5×10^{-3} . The same neural network architecture is used in Deep Heun's method (8) for comparison. Figure 1 shows the evolution of the trained $\mathcal{N}(x_m, x_{m+1}, y_m; \theta)$ in DEM and the local truncation error function $R(x_m, x_{m+1}, y_m, y_{m+1})$ in the Euler method. The four different step sizes, that is $h = 0.01, 0.1, 1.0$ and 2.0 are displayed. Since it is trained in $(0, 5)$, $\mathcal{N}(x_m, x_{m+1}, y_m; \theta)$ almost coincides with $R(x_m, x_{m+1}, y_m, y_{m+1})$ in $(0, 5)$.

Figure 2 shows the exact solution and four approximations of (11). The four different step sizes are also displayed. It is observed that only in the small step h can the Euler method and the Huen obtain a more accurate approximation of the solution. We also note the fact that \mathcal{N} is only trained in $(0, 5)$. However, in $(5, 10)$, DEM and DHM can get the accurate approximation of the solution even for the bigger step size $h = 2.0$. This indicates the efficient prediction of the deep neural network.

In Table 1, we discuss the results of the comparison among four methods for the different step sizes. The Euler method, the Heun's method, DEM and DHM are adopted for solving (11). The first four columns of Table 1 show the prediction errors between the exact solution and the approximation in L_1 norm, i.e. $e = \max_m |y(x_m) - y_m|$. Since the global truncation error of Deep Euler Method and Deep Heun's method are $\mathcal{O}(\eta h)$ and $\mathcal{O}(\eta h^2)$. That is the reason that when $h \geq 1$, DEM and DHM get more accuracy than the Euler method and the Heun method. Our goal is to get the estimates of η . In view of Lemma 1, $\left| \mathcal{N}(x_m, x_{m+1}, z_m; \theta) - \frac{1}{h^2} R_m \right| \approx |\mathcal{N}(x_m, x_{m+1}, z_m; \theta) - R(x_m, x_{m+1}, z_m, z_{m+1})| = \mathcal{O}(\eta)$. In the column of ε_{mean} , we present the mean of the difference between \mathcal{N} and R , which is $\varepsilon_{mean} = \frac{1}{M} \sum_m |\mathcal{N} - R| = \mathcal{O}(\eta)$. In the last column, we present the results of DEM (the fourth column) divided by the result of the Euler method (the second column), i.e., $e_{DEM}/e_{Euler} = \mathcal{O}(\eta)$. From the last two columns, we can conclude that $\eta = \mathcal{O}(0.001)$, which also indicates the efficiency of DEM.

Table 2 shows ε_{mean} for different network architectures (the number of hidden layers and neurons) and the different number of random measured points. We evolution the neural networks with $h = 0.1$. To avoid uncertainty during the training process, we simulate each case ten times and take the average value of them. It can be observed that the prediction accuracy increase with the number of measured points. If the networks with too small layers and neurons per layer (such as 2 layers and 20 neurons per layer), it is not suitable in the case of a small number of measured data since it has a high bias in the training region $[0, 5]$ and a high variance in the testing region $(5, 10]$. If the networks with the more layers and neurons per layer (such as 16 layers and 160 neurons per layer), it may be overfitted in the case

of the small number of measured data since it has low bias and high variance. More measured data can reduce the variance. We choose the networks with 8 layers and 80 neurons per layer for all experiments. Because no matter in a small amount or a large number of measured data, it has low variance and low bias than others.

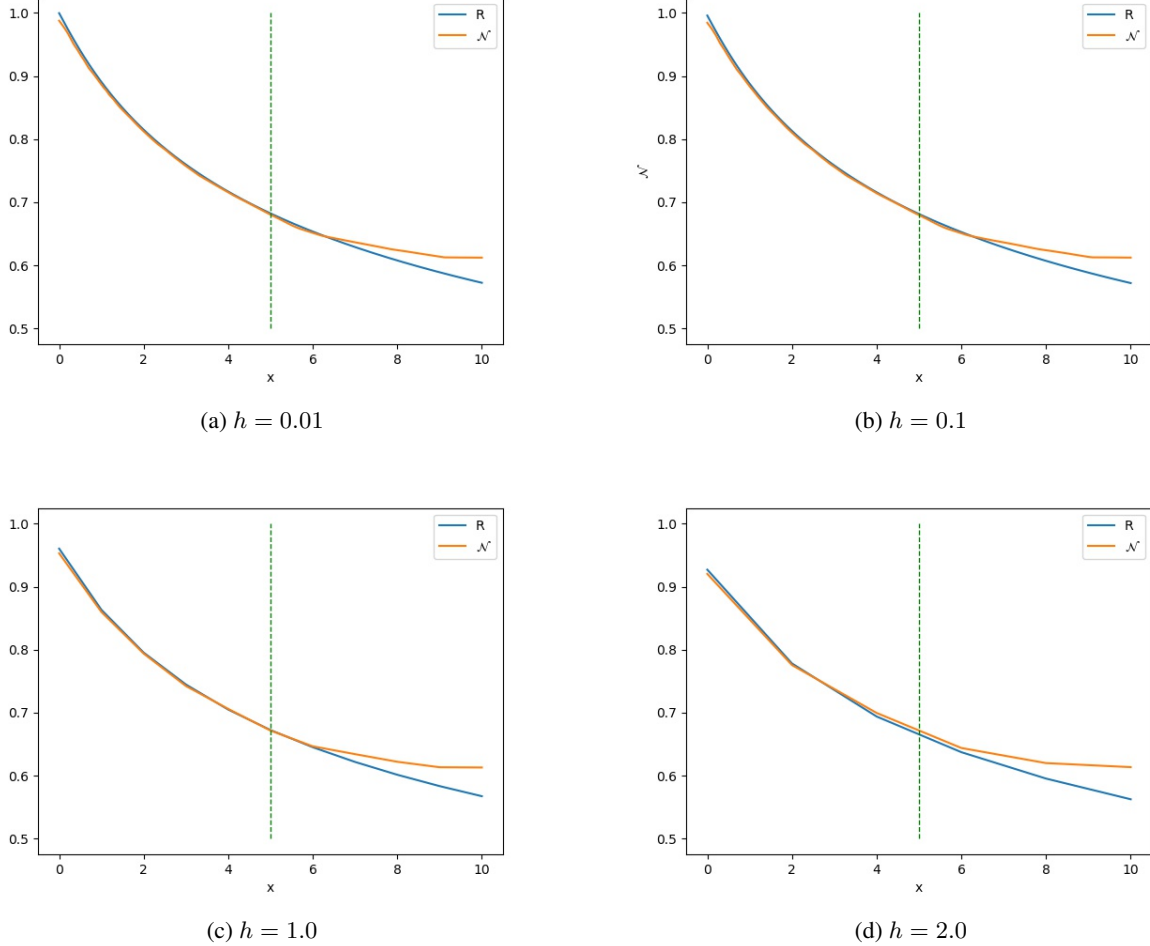


Figure 1: The evolution of the trained neural network $\mathcal{N}(x_m, x_{m+1}, y_m; \theta)$ in DEM and of the local truncation error function $R(x_m, x_{m+1}, y_m, y_{m+1})$ in the Euler method. The four different step sizes are 0.01, 0.1, 1.0 and 2.0. The red line is the result of \mathcal{N} . The blue line is the result of R .

Table 1: The results of the comparison among four methods for different step sizes. Prediction errors between the exact solution and the approximation in L_1 norm are listed, i.e., $e = \max_m |y(x_m) - y_m|$. The column of ε_{mean} is $\sum_m |\mathcal{N} - R| / M$, where M is the number of the steps. The last column is the result of DEM (the fourth column) divided by the result of the Euler method (the second column).

| step size | Euler method | Heun's method | DEM | DHM | ε_{mean} | e_{DEM}/e_{Euler} |
|-----------|--------------|---------------|--------|----------|----------------------|---------------------|
| 0.01 | 0.42 | 0.0017 | 0.0014 | 0.000053 | 0.0086 | 0.0033 |
| 0.1 | 4.05 | 0.15 | 0.013 | 0.0051 | 0.0089 | 0.0032 |
| 1 | 28.42 | 8.10 | 0.073 | 0.32 | 0.012 | 0.0026 |
| 2 | 43.16 | 18.78 | 0.083 | 1.03 | 0.016 | 0.0019 |

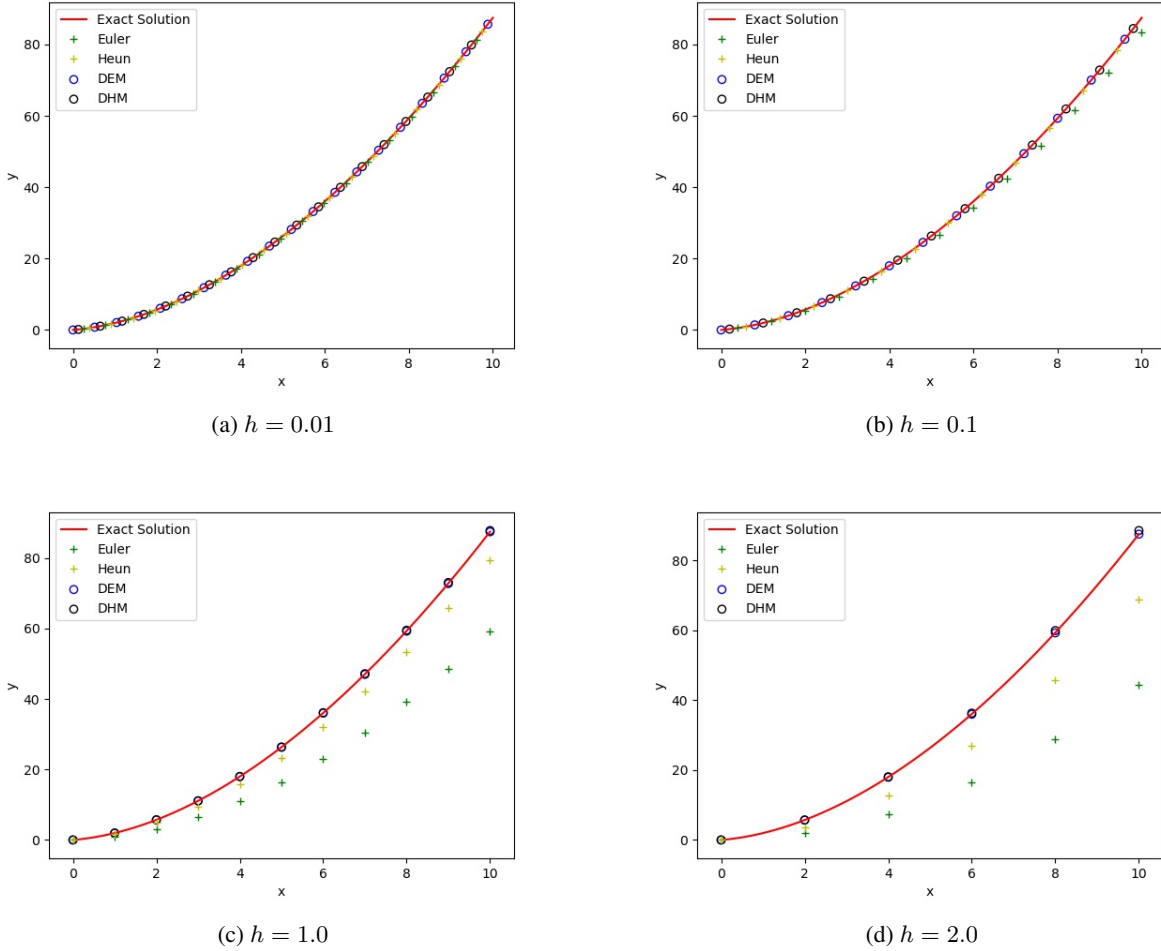


Figure 2: The exact solution and four approximations of (11). The four different step sizes are 0.01, 0.1, 1.0 and 2.0. The green plus is the result of the Euler method. The yellow plus is the result of the Heun method. The blue circle is the result of DEM. The black circle is the result of DHM.

Table 2: The results of ε_{mean} , i.e., the average error of between \mathcal{N} and R , for the different number of hidden layers and neurons per layer, as well as the different number of measured points. The number on the left is the error in the training region $[0, 5]$, and the error in the test region $(5, 10]$ is on the right. All the step size is $h = 0.1$.

| points | layers & neurons | | | | | | | |
|--------|------------------|------|---------------|-------|---------------|-------|-----------------|-------|
| | 2×20 | | 4×40 | | 8×80 | | 16×160 | |
| 10 | 0.21 | 0.69 | 0.067 | 0.36 | 0.033 | 0.11 | 0.071 | 0.17 |
| 25 | 0.20 | 0.81 | 0.03 | 0.16 | 0.014 | 0.061 | 0.075 | 0.16 |
| 50 | 0.081 | 0.41 | 0.024 | 0.36 | 0.022 | 0.073 | 0.049 | 0.12 |
| 100 | 0.017 | 0.21 | 0.0093 | 0.14 | 0.011 | 0.045 | 0.014 | 0.052 |
| 200 | 0.0096 | 0.28 | 0.0056 | 0.080 | 0.0093 | 0.030 | 0.0084 | 0.035 |
| 500 | 0.0066 | 0.22 | 0.0024 | 0.072 | 0.0028 | 0.048 | 0.0039 | 0.048 |

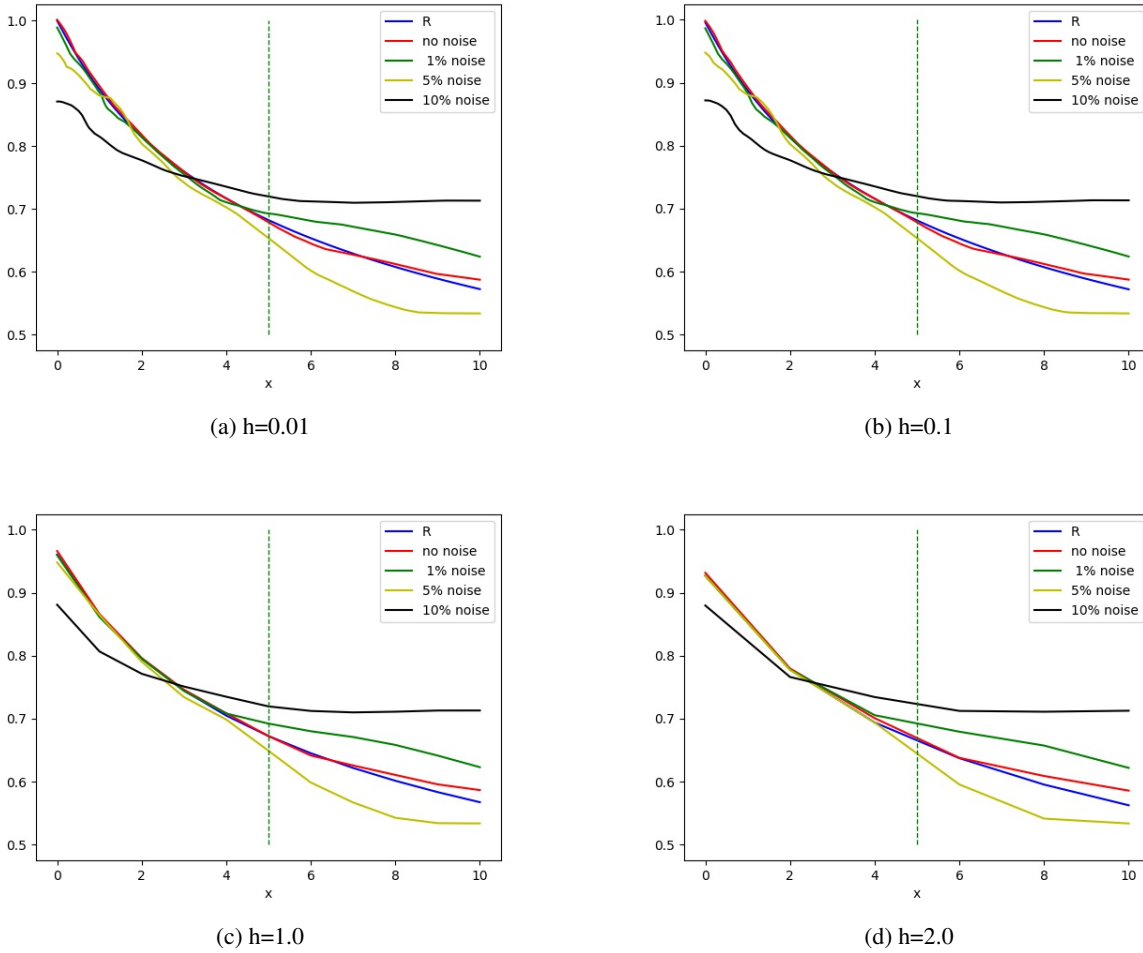


Figure 3: For comparison, the DEMs and the local truncation error function R are listed together. Each \mathcal{N} of DEM is trained for one case of noise levels ($\delta = 0\%, 1\%, 5\%, 10\%$). Four subfigures show the evolutions of the four $\mathcal{N}(x_m, x_{m+1}, y_m; \theta)$ and R for $h = 0.01, 0.1, 1.0$ and 2.0 , respectively.

Figure 3 shows the DEMs and the local truncation error function R together for comparison. Each \mathcal{N} of DEM is trained for one case of noise levels ($\delta = 0\%, 1\%, 5\%, 10\%$). The four different step sizes, that is $h = 0.01, 0.1, 1.0$ are displayed. It can be observed from the subfigures that the change of step size has little effect on the result. In fact, from Table 3, we also noted that. When the step size h changes from 0.01 to 2.0 , ϵ_{mean} under various noise levels ($\delta = 0, 1\%, 5\%, 10\%$) are almost the same. However, obviously, e_{DEM} is the smallest in the case of the smallest h and the lest noise level (noise-free).

Table 3: The performance of DEM for the different noise levels and the corresponding numerical solutions.

| | | ϵ_{mean} | | | | e_{DEM} | | | |
|-----|----------|-------------------|------|------|------|-----------|-------|------|------|
| | | 0% | 1% | 5% | 10% | 0% | 1% | 5% | 10% |
| h | δ | | | | | | | | |
| | 0.01 | 0.005 | 0.02 | 0.04 | 0.07 | 0.001 | 0.001 | 0.01 | 0.02 |
| | 0.1 | 0.005 | 0.02 | 0.03 | 0.07 | 0.01 | 0.01 | 0.01 | 0.02 |
| | 0.5 | 0.005 | 0.02 | 0.03 | 0.07 | 0.06 | 0.09 | 0.35 | 0.58 |
| | 1.0 | 0.006 | 0.03 | 0.03 | 0.07 | 0.12 | 0.27 | 0.50 | 0.71 |
| 2.0 | 0.01 | 0.03 | 0.02 | 0.07 | 0.24 | 0.55 | 0.45 | 0.50 | |

5.2 Example 2

We now consider a system of first-order nonlinear differential equations, the Lotka-Volterra equation ([12]),

$$\begin{cases} \frac{dy_1}{dx} = \alpha y_1 - \beta y_1 y_2, \\ \frac{dy_2}{dx} = -\gamma y_2 + \delta y_1 y_2. \end{cases} \quad (12)$$

This equation describes the dynamics of the populations of two species, one as a predator and the other as prey. In (12), y_1 and y_2 are the number of prey and predator, respectively. Let $\mathbf{y} = [y_1, y_2]^T$, x represent time and $\alpha, \beta, \gamma, \delta$ be the parameters describing the relationship of two species. In this work, we take $\alpha = \beta = \gamma = \delta = 1$, and the initial conditions $\mathbf{y} = [y_1(0), y_2(0)]^T = [2, 1]^T$.

For comparison, the exact solution is gotten by the numerical method of *RK45* in `scipy.integrate` [23] with 10^{-6} relative tolerances. We sample 1000 random points from a uniform distribution $\mathcal{F}_t = U(0, 15)$ to construct the noise-free training dataset $\{(t_i, y(t_i))\}_1^{1000}$. The deep neural network with 8 hidden layers and 80 neurons per layer is trained with super parameters the same as example 1.

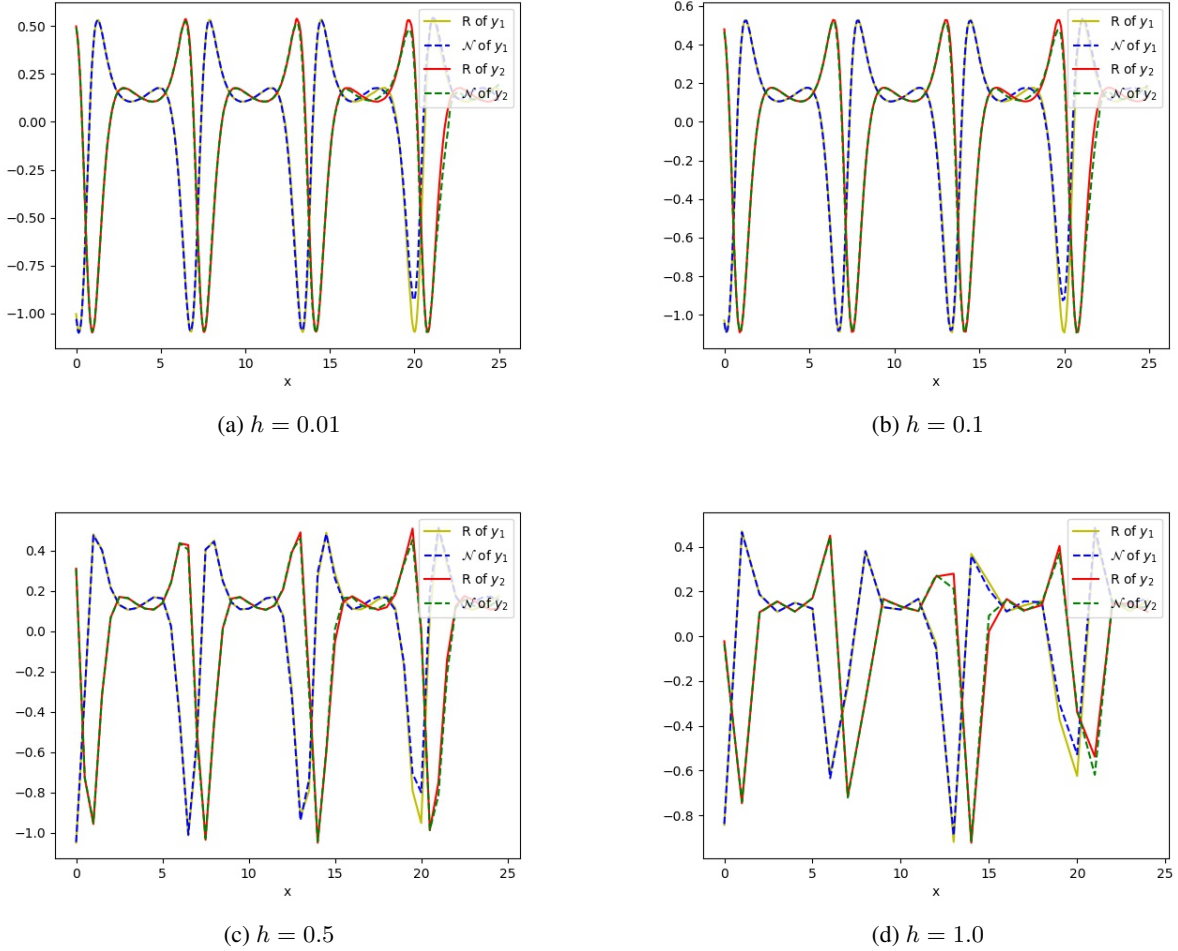
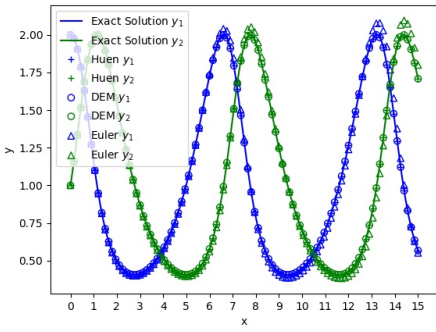
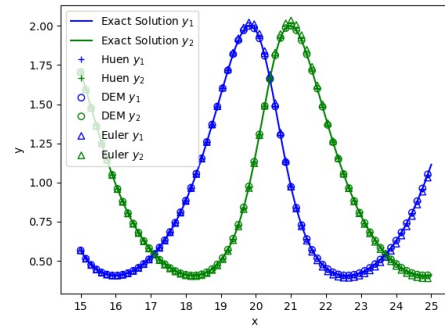


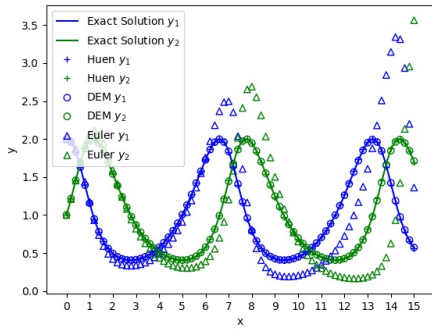
Figure 4: The evolution of the trained neural networks $\mathcal{N}(x_m, x_{m+1}, \mathbf{y}_m)$ in DEM and the local truncation error function $R(x_m, x_{m+1}, \mathbf{y}_m, \mathbf{y}_{m+1})$ for different step sizes $h = 0.01, 0.1, 0.5, 1.0$



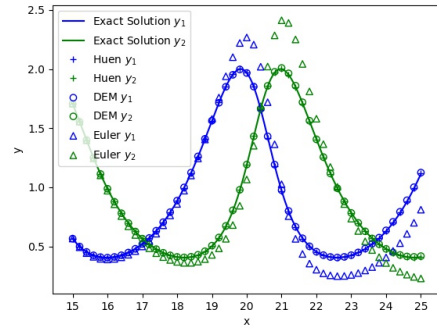
(a) $h = 0.01$ in $[0,15]$



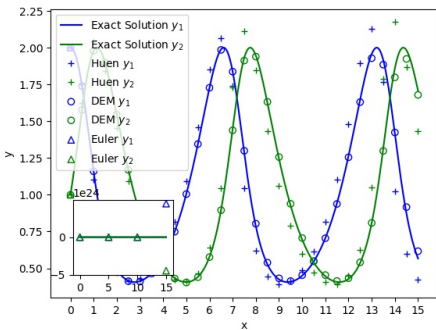
(b) $h = 0.01$ in $[15,25]$



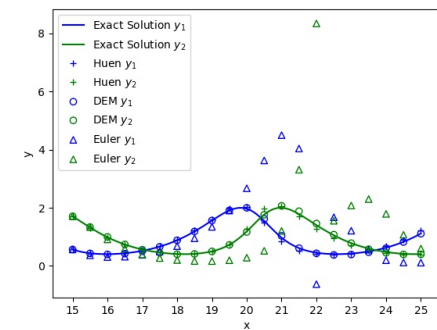
(c) $h = 0.1$ in $[0,15]$



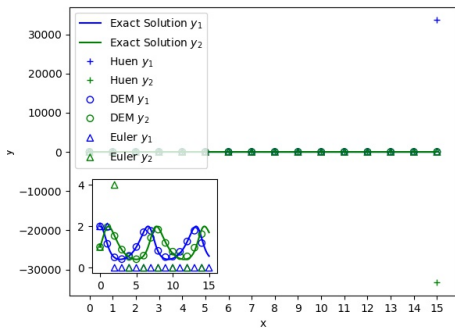
(d) $h = 0.1$ in $[15,25]$



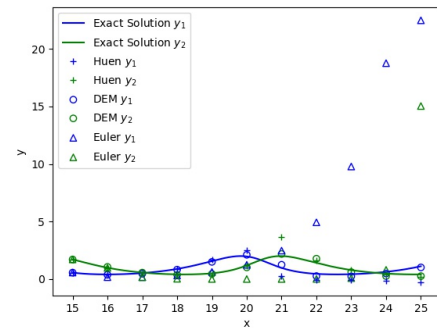
(e) $h = 0.5$ in $[0,15]$



(f) $h = 0.5$ in $[15,25]$



(g) $h = 1.0$ in $[0,15]$



(h) $h = 1.0$ in $[15,25]$

Figure 5: The evolutions of the Euler method, the Huen method and DEM are displayed for noise-free measured data. The four different step sizes ($h = 0.01, 0.1, 0.5$ and 1.0) are considered.

Figure 4 shows the neural network \mathcal{N} of DEM and the local truncation error function R in both the training region $[0, 15]$ and the testing region $(15, 25]$. The four different cases of $h = 0.01, 0.1, 0.5, 1.0$ are displayed. It shows that DEM can accurately approximate the solution in the whole region $(15, 25]$ even for $h = 1.0$. In Figure 5, we show the evolutions of the Euler method, the Huen method and DEM for comparison. For the cases of $h = 0.5$ and $h = 1.0$, the results of the Euler method over $[0, 15]$ are displayed in the left sub-figure. For the case of $h = 0.5$, the solutions of the Euler method diverge from the exact value very much near $x = 15$. When $h = 1.0$, the solution y_2 of the Euler method is close to 4, but the solution of y_1 is near 0, both of which are far away from the exact value. The Huen method has similar defects. When $h = 1.0$, the solutions of the Huen method are close to 30000, which is far away from the exact value. It can be observed that no matter the small h or the big h , the results of DEM and the curve of the exact solution almost coincide.

5.3 Example 3

Considering the Kepler problem :

$$\begin{cases} \frac{dy_1}{dx} = y_3, \\ \frac{dy_2}{dx} = y_4, \\ \frac{dy_3}{dx} = -\frac{y_1}{(y_1^2 + y_2^2)^{3/2}}, \\ \frac{dy_4}{dx} = -\frac{y_2}{(y_1^2 + y_2^2)^{3/2}}. \end{cases} \quad (13)$$

It describes the motion of the sun and a single planet which is a special case of the two-body problem. We denote the time by x . Let $(y_1(x), y_2(x))$ be the positions of the planet in rectangular coordinates centered at the sun and $y_3(x), y_4(x)$ be the velocity components in the y_1 and y_2 directions. The initial value are $y(0) = [1, 0, 0, 1]^T$, and the exact solution is $y(x) = [\cos(x), \sin(x), -\sin(x), \cos(x)]^T$.

Similar to example 2, we use the uniform distribution to generate 1000 noise-free data points and select the same values of the super parameters as in Example 1. In Figure 6, four different cases of h are displayed. It can be observed that $\mathcal{N}(x_m, x_{m+1}, \mathbf{y}_m)$ of DEM well approximates $R(x_m, x_{m+1}, \mathbf{y}_m, \mathbf{y}_{m+1})$ for all of h . We can also note that the approximations of DEM (circles in the figure) almost coincides with the curve of the exact solution in Figure 7, even for $h = 1.0$.

6 Conclusion

In this work, we proposed a Deep Euler Method with the idea of approximating the truncation error in the Euler method via deep learning. When deep neural network \mathcal{N} is trained to approximate the local truncation error function with accuracy $\mathcal{O}(\eta)$, the global truncation error of DEM with the step size h would be $\mathcal{O}(\eta h)$ while the Euler method is only $\mathcal{O}(h)$. Since η can be small enough, it could achieve high accuracy solutions even with a big step size ($h \geq 1$). DEM significantly improves the accuracy of the Euler method and reduces the constrain of the step size in the Euler method. On the other hand, since the training objective function of the deep neural network in DEM is always $\mathcal{O}(1)$, the deep neural network can be easily trained and fast to converge, even if only the simplest architecture of the fully connected network and only a few training data are used. Moreover, DEM shows good robustness with the noise of the measured data.

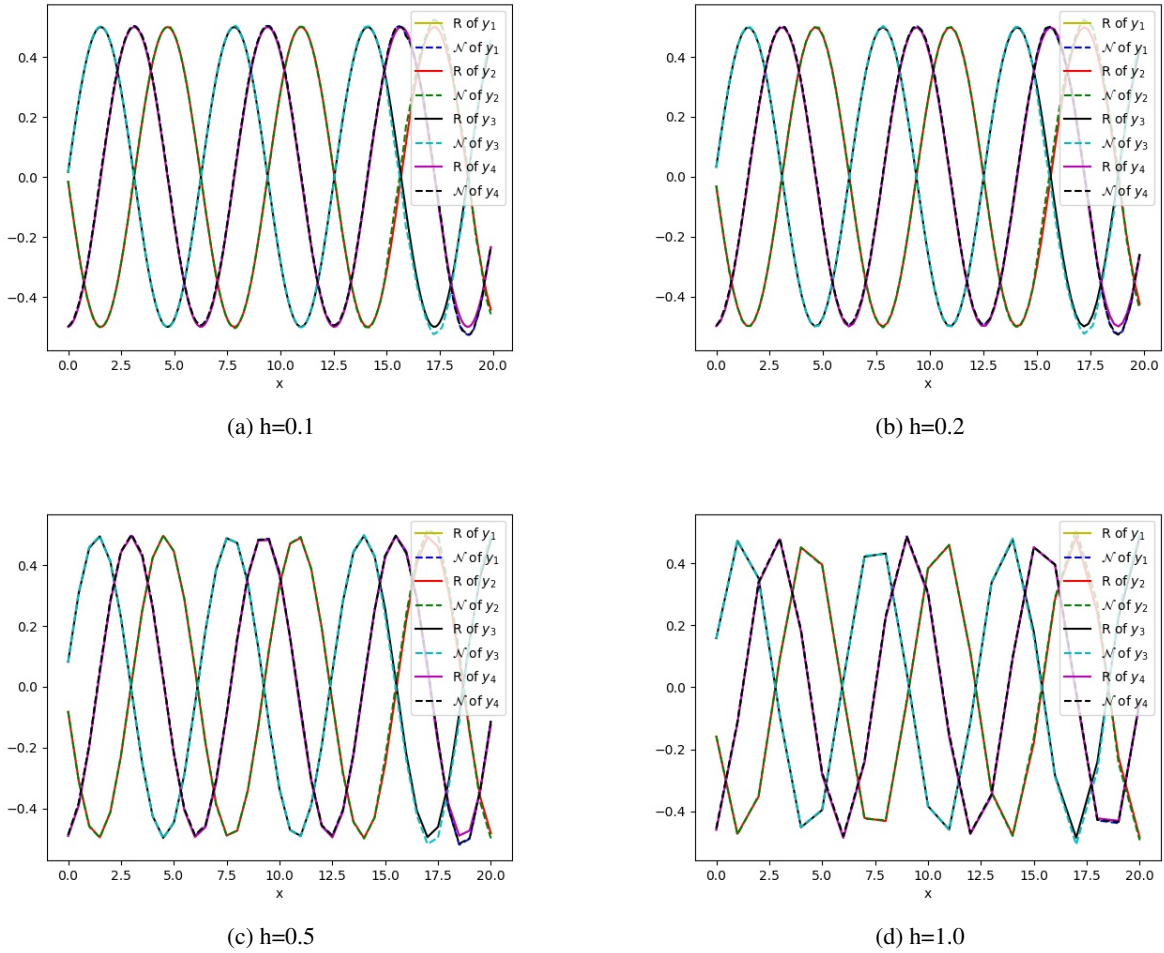
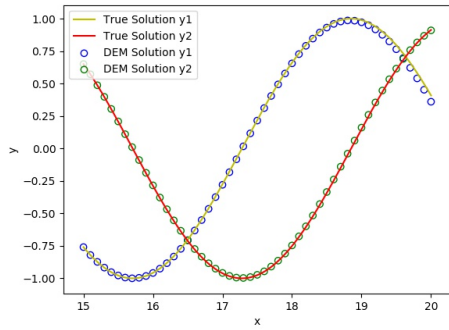
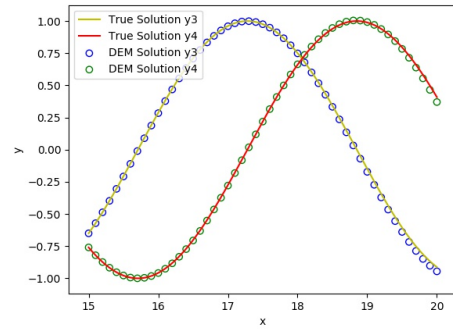


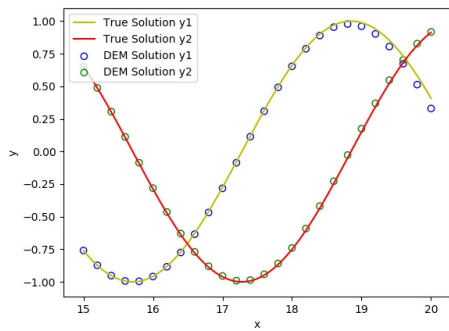
Figure 6: The evolutions of the neural network $\mathcal{N}(x_m, x_{m+1}, \mathbf{y}_m)$ and the local truncation error function $R(x_m, x_{m+1}, \mathbf{y}_m, \mathbf{y}_{m+1})$ of the equation (13) for $h = 0.1, 0.2, 0.5, 1.0$, respectively.



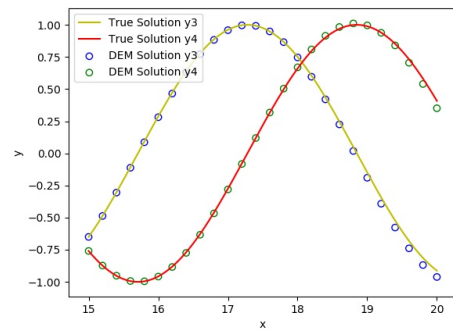
(a) The components of y_1, y_2 with $h = 0.1$



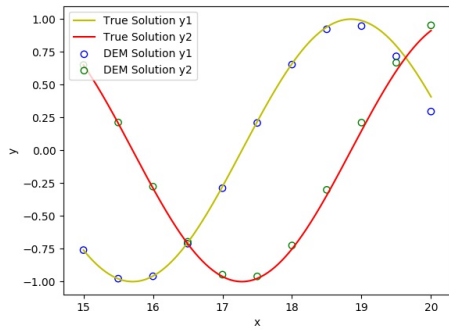
(b) The components of y_3, y_4 with $h = 0.1$



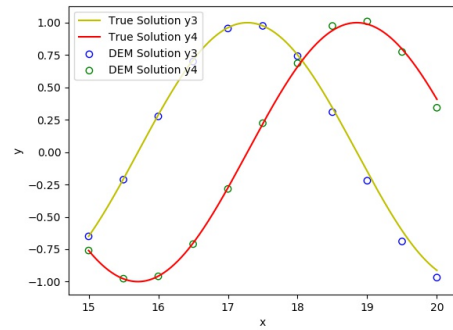
(c) The components of y_1, y_2 with $h = 0.2$



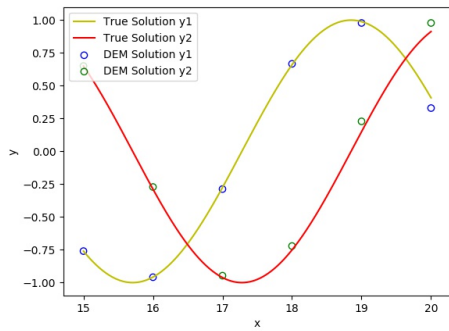
(d) The components of y_3, y_4 with $h = 0.2$



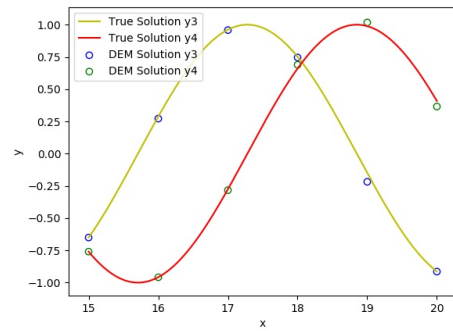
(e) The components of y_1, y_2 with $h = 0.5$



(f) The components of y_3, y_4 with $h = 0.5$



(g) The components of y_1, y_2 with $h = 1.0$



(h) The components of y_3, y_4 with $h = 1.0$

Figure 7: The exact solution and the approximation of DEM of the equation (13) on region $(15, 20]$ for $h = 0.1, 0.2, 0.5, 1.0$, respectively.

References

- [1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein gan”. In: *arXiv preprint arXiv:1701.07875* (2017).
- [2] Gert-Jan Both et al. “DeepMoD: Deep learning for Model Discovery in noisy data”. In: *arXiv preprint arXiv:1904.09406* (2019).
- [3] Amir Barati Farimani, Joseph Gomes, and Vijay S Pande. “Deep learning the physics of transport phenomena”. In: *arXiv preprint arXiv:1709.02432* (2017).
- [4] Jiequn Han, Arnulf Jentzen, and E Weinan. “Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning”. In: *arXiv preprint arXiv:1707.02568* (2017), pp. 1–13.
- [5] Juncai He et al. “Relu deep neural networks and linear finite elements”. In: *arXiv preprint arXiv:1807.03973* (2018).
- [6] Martin Hutzenthaler et al. “Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations”. In: *arXiv preprint arXiv:1807.01212* (2018).
- [7] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. “Solving parametric PDE problems with artificial neural networks”. In: *arXiv preprint arXiv:1707.03351* (2017).
- [8] Yuehaw Khoo and Lexing Ying. “SwitchNet: a neural network model for forward and inverse scattering problems”. In: *SIAM Journal on Scientific Computing* 41.5 (2019), A3182–A3201.
- [9] Stig Larsson and Vidar Thomee. *Partial Differential Equations with Numerical Methods (Texts in Applied Mathematics vol. 45)*. Springer-Verlag, Berlin, 2008.
- [10] Moshe Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6 (1993), pp. 861–867.
- [11] Zichao Long, Yiping Lu, and Bin Dong. “PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network”. In: *Journal of Computational Physics* 399 (2019), p. 108925.
- [12] AJ Lotka. “Elements of physical biology. Williams and Wilkins”. In: *Baltimore, Md* (1925).
- [13] Tong Qin, Kailiang Wu, and Dongbin Xiu. “Data driven governing equations approximation using deep neural networks”. In: *Journal of Computational Physics* 395 (2019), pp. 620–635.
- [14] Maziar Raissi. “Deep hidden physics models: Deep learning of nonlinear partial differential equations”. In: *The Journal of Machine Learning Research* 19.1 (2018), pp. 932–955.
- [15] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707.
- [16] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Multistep neural networks for data-driven discovery of nonlinear dynamical systems”. In: *arXiv preprint arXiv:1801.01236* (2018).
- [17] F Regazzoni, L Dedè, and A Quarteroni. “Machine learning for fast and reliable solution of time-dependent differential equations”. In: *Journal of Computational Physics* 397 (2019), p. 108852.
- [18] Samuel H Rudy, J Nathan Kutz, and Steven L Brunton. “Deep learning of dynamics and signal-noise decomposition with time-stepping constraints”. In: *Journal of Computational Physics* 396 (2019), pp. 483–506.
- [19] Tim Salimans and Durk P Kingma. “Weight normalization: A simple reparameterization to accelerate training of deep neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 901–909.
- [20] Justin Sirignano and Konstantinos Spiliopoulos. “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of Computational Physics* 375 (2018), pp. 1339–1364.
- [21] Yifan Sun, Linan Zhang, and Hayden Schaeffer. “Neupde: Neural network based ordinary and partial differential equations for modeling time-dependent data”. In: *arXiv preprint arXiv:1908.03190* (2019).
- [22] Rohit K Tripathy and Ilias Bilonis. “Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification”. In: *Journal of computational physics* 375 (2018), pp. 565–588.
- [23] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: <https://doi.org/10.1038/s41592-019-0686-2>.
- [24] Yufei Wang et al. “Learning to Discretize: Solving 1D Scalar Conservation Laws via Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1905.11079* (2019).
- [25] E Weinan and Bing Yu. “The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems”. In: *Communications in Mathematics and Statistics* 6.1 (2018), pp. 1–12.
- [26] Nick Winovich, Karthik Ramani, and Guang Lin. “ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains”. In: *Journal of Computational Physics* 394 (2019), pp. 263–279.

- [27] Kailiang Wu and Dongbin Xiu. “Data-driven deep learning of partial differential equations in modal space”. In: *Journal of Computational Physics* (2020), p. 109307.
- [28] Huan Xu and Shie Mannor. “Robustness and generalization”. In: *Machine learning* 86.3 (2012), pp. 391–423.