Journal of Computational Physics 442 (2021) 110493

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Beyond the Courant-Friedrichs-Lewy condition: Numerical methods for the wave problem using deep learning

Oded Ovadia *, Adar Kahana, Eli Turkel, Shai Dekel

Department of Applied Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel

ARTICLE INFO

Article history:
Available online 6 June 2021

Keywords:
Numerical methods
Stability
Explicit and implicit schemes
Spatio-temporal
Physically-informed

ABSTRACT

We investigate a numerical method for approximating the solution of the one dimensional acoustic wave problem, when violating the numerical stability condition. We use deep learning to create an explicit non-linear scheme that remains stable for larger time steps and produces better accuracy than the reference implicit method. The proposed spatio-temporal neural-network architecture is additionally enhanced during training with a physically-informed term, adapting it to the physical problem it is approximating and thus more accurate.

1. Introduction

We propose a method to solve the wave propagation problem in a homogeneous domain while violating the stability conditions. We focus on the one-dimensional problem in the time domain (one spatial and one time coordinate). We set reflecting boundary conditions (e.g., a string held at both sides). Hyperbolic problems have been investigated by many authors [1–3] and in many fields, such as fluid dynamics, acoustics, electromagnetic problems, etc. Most studies discuss solutions that do not involve instabilities. Others focus on calculating a stability condition limit, e.g., if the condition is met, the solution is numerically stable. We propose a method, based on a physically-informed deep-learning approach, producing stable solutions, even if the Courant-Friedrichs-Lewy (CFL) stability condition is not met.

When approximating a physical problem numerically one chooses a grid on which the solution will be calculated. When the number of grid points is immensely large, the amount of calculations might exceed the hardware performance limit (for example, calculating the propagation of an electromagnetic wave that was sent from a satellite and propagates over an entire continent with 1 meter precision). On the other hand, facing a difficult problem (high order equation, complex numerical model, singularities etc.) even a small number of points can be too much to handle. According to the CFL condition, using a fine grid requires also a fine temporal discretization which makes the solution even more complex. For example:

- Modeling long and large processes, e.g., a cosmic phenomenon that takes years to develop.
- Modeling a very short phenomenon that occurs once in a very large time-span, e.g., a volcanic eruption that takes minutes but happens randomly once a decade.

* Corresponding author.
E-mail address: odedovadia@mail.tau.ac.il (O. Ovadia).

https://doi.org/10.1016/j.jcp.2021.110493
0021-9991/

- Modeling a wave problem with a very large propagation speed, e.g., a shock wave hitting a rod.

We model the propagation of underwater acoustic waves by the acoustic wave equation [13,14,4] with parameters of the medium (such as wave propagation velocity, boundary conditions etc.). We numerically approximate the solution using a simple central difference Finite-Difference (FD) scheme in both space and time (2). The CFL condition is a necessary condition for the stability of this method [6], fixing a ratio between the spatial and temporal discretization sizes. When the CFL condition is not met, the numeric solution explodes, resulting in an exponentially growing error over time. The larger the space stencil the larger is the allowed time step. However, using a compact stencil limits the time step. It is possible to violate the CFL condition by using properties of the specific system to be solved. For example, Turkel and Zwas [11] use coarse mesh information for the shallow water equations. Liu and Dong [12], based on previous work, use a wave splitting for conservation laws. We shall consider the acoustic wave equation using only standard finite difference schemes. Future work will extend it to other wave-like equations e.g. elasticity and electromagnetics.

In an attempt to develop a numerically stable scheme while violating the CFL condition, we train a neural-network to learn the elements of the scheme. We generate solutions synthetically from randomly chosen initial conditions using a stable scheme and sub-sample the solutions in such a way that they are unstable (the CFL condition is not satisfied). These sub-samples are the training set for the neural-network which is then tested on different synthetically generated unstable problems. Although the training part takes hours, using the non-linear scheme produced by the network takes milliseconds. In addition, other explicit numerical methods cannot produce a solution under violation of the CFL condition while the proposed method does. Therefore, we compare the solution to both an explicit and an implicit method. We also compare the method for a wave problem with a simple analytic solution, measuring the difference between the method and the analytic solutions over time to check the methods performance over many time steps.

We improve the method by adding a physically-informed [7,8] loss element to the neural network during training. We exploit the fact that the network solves the acoustic wave problem, and create a loss term that compares the network's learned solution with another solution produced by the FD approximation. This makes the network physically aware of the problem it is trying to approximate and therefore ensures better convergence resulting in higher precision of the numerical method.

The outline for the remainder of this paper is as follows. Section 3 discusses the physical problem and the mathematical formulation used to model the problem. Section 4 reviews key aspects of learning methods and present the deep-learning model used. Section 5 presents the numerical experiments conducted and the findings using the proposed methods. Concluding remarks are found in section 6.

2. Background

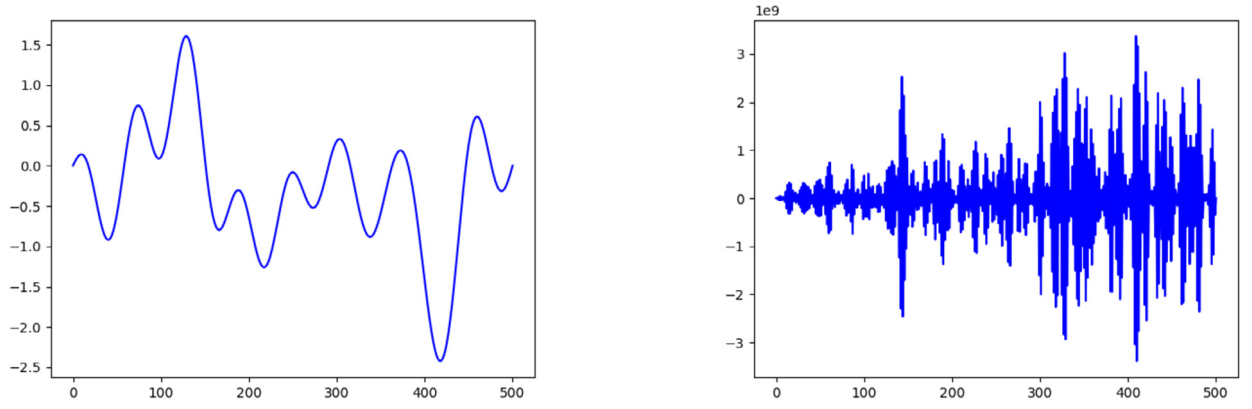
Given a system of differential equations there exists two different approaches for approximating the solution for the system - explicit and implicit methods [9,10]. Implicit methods usually require large computational effort. Explicit methods are conditionally stable, meaning that if the stability condition is not satisfied the solution explodes.

Some studies focus on developing conditionally-stable explicit methods with less limiting conditions. Some methods introduce relaxation coefficients, while others use augmentation of the problem to create a dual condition that is less demanding [17]. These methods usually have lower accuracy compared to the implicit models, but they are able to solve problems that implicit solvers cannot due to the lower computational effort. Usually, these methods are tailored to the physical problem they are trying to approximate (in terms of the parameters and even the algorithm that defines the method).

A common issue with numerical methods is that the higher the accuracy we require, the higher the computational effort we need. This trade-off is a main concern when developing a numerical method, and it motivates researchers to search for either finding the most accurate solution or the most efficient solution. There are several studies that try to find a model that will be sufficiently accurate and the least complex for a given problem. However, there is no perfect model that is both high-order, accurate and efficient and also suitable for every problem.

We propose an explicit method that is more accurate than most implicit methods for large time steps. The approach is to create a neural-network and train it with different solutions for PDEs so that the network can learn a numerical method. The weights of the network represent the coefficients of the method, and after training is done and the coefficients are learned, we use the network to predict the later state of the system based on the previous ones which makes it an explicit method. Although training is a complex procedure, it is only done once and using the trained model is a very efficient procedure. We compare the performance of this method to an implicit method in terms of accuracy over time, and observe promising results.

The field of machine learning is growing very rapidly and many other data-driven fields use methods in machine learning to approximate models. Specifically, in numerical analysis, the use of physically-informed neural-networks [7] is growing. Such methods make the network physically aware of the problem it is solving. Here, we demonstrate another use of the physically-informed idea - constructing a loss function for the network that is based on the wave problem. We have achieved improved results using such a method.



(a) $\alpha = 0.875$ (b) $\alpha = 8.75$

Fig. 1. Approximation of a solution to the wave problem after 10 iterations using the FD method with different CFL numbers.

3. Numerical modeling

3.1. Mathematical model

The general formulation of the wave problem is given by -

$$\begin{cases} \ddot{u}(\overrightarrow{x},t) = \nabla \cdot (c^2(\overrightarrow{x})\nabla u(\overrightarrow{x},t)) & \overrightarrow{x} \in \Omega,\ t \in (0,T], \\ u(\overrightarrow{x},0) = u_0(\overrightarrow{x}) & \overrightarrow{x} \in \Omega, \\ \dot{u}(\overrightarrow{x},0) = v_0(\overrightarrow{x}) & \overrightarrow{x} \in \Omega, \\ u(\overrightarrow{x},t) = f(\overrightarrow{x},t) & \overrightarrow{x} \in \partial\Omega_1,\ t \in [0,T], \\ \nabla u(\overrightarrow{x},t) = g(\overrightarrow{x},t) & \overrightarrow{x} \in \partial\Omega_2,\ t \in [0,T], \qquad \partial\Omega_1 \cup \partial\Omega_2 = \partial\Omega, \end{cases} \tag{1}$$

where $u(\overrightarrow{x},t)$ is the wave amplitude or acoustic pressure, c is the wave propagation speed, u_0 and v_0 are the initial pressures and velocities respectively, and f and g are boundary condition of types Dirichlet and Neumann respectively. In this work we investigate the one-dimensional case. Therefore, throughout the paper x is treated as a scalar so $x \in \Omega = [a,b]$. We also assume that c is constant. The specific system we solve is given in section 5.1 by (5). Problem (1) is well-posed and thus small changes in the problem conditions result in small changes in the solution and there exists a unique continuous solution to the problem inside the domain.

3.2. Numerical approximation methods

We approximate the spatial domain Ω using N_x nodes. We also approximate the temporal direction $[0,T]$ using N_t nodes. Both are uniform divisions. We denote Δx and Δt as the grid spacing. We use the notation u_i^n for the solution at point $(i \cdot \Delta x, n \cdot \Delta t)$ where $i = 0, \ldots N_x - 1$ and $n = 0, \ldots N_t - 1$. A commonly used method to approximate the solution is the FD method [5]. We use it to approximate both in time and space as follows:

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} = c^2 \left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \right). \tag{2}$$

3.3. Numerical stability

Denote $\alpha = c\frac{\Delta t}{\Delta x}$. This is the Courant-Friedrichs-Lewy (CFL) number. The CFL criterion $\alpha = c\frac{\Delta t}{\Delta x} \leq 1$ is a necessary and sufficient condition for the convergence of FD method [6]. When we choose a grid (selection of $N_x, N_t, \Delta x$ and Δt) that does not satisfy the CFL condition, $\alpha > 1$, after a few iterations in time the solution "blows up" as shown in Fig. 1.

3.4. The data-driven problem

We formulate the computational problem as a supervised-learning problem. In supervised learning, each sample has a label and a chosen learning algorithm learns the connection between each data sample and its label. Typically in deep learning, the size of the training set is large. This allows the model to learn how to generalize to unseen samples with

sufficient accuracy. In addition, designing the right architecture of the machine learning model to fit the physical nature of the problem has a significant impact on the performance.

We create the data-set by generating random initial conditions from a truncated orthonormal basis in $L^2(\Omega)$. In addition, this space is large enough to choose many initial conditions with a high variance between them. A specific choice of basis functions is given in section 5.1.

We use a truncated Taylor series at $t = 0$ to approximate the second time step:

$$u(x, 0 + \Delta t) \approx u(x, 0) + \Delta t u_t(x, 0) + \frac{\Delta t^2}{2} u_{tt}(x, 0).$$

Substituting the initial condition for the pressure (u_0 in (1)):

$$u(x, 0 + \Delta t) \approx u_0(x) + \Delta t v_0(x) + \frac{(\Delta t)^2}{2} c^2 u_{xx}(x, 0).$$

Substituting $v_0(x) \equiv 0$ and switching notation we get a formula for u^1:

$$u_i^1 = u_i^0 + \frac{\Delta t^2}{2} c^2 \frac{u_{i+1}^0 - 2u_i^0 + u_{i-1}^0}{\Delta x^2} \quad \forall i = 0, ..., N_x.$$

To ease the notation, consider the vector:

$$u^n = \begin{pmatrix} u(x_0, t_n) \\ u(x_1, t_n) \\ \vdots \\ u(x_{N_x}, t_n) \end{pmatrix},$$

so u^n denotes the solution at time t_n over the entire spatial grid.

We use the FD method to create the data samples. Unfortunately, violation of the CFL condition results in an unstable solution. Therefore, we create the data using a fine grid, e.g., using $\frac{\Delta t}{m}$ where m is a natural number larger than 1. We then choose only each m^{th} sample. The data set includes input samples with $u^{(n-1)m}$ and u^{nm} and labels $u^{(n+1)m}$ for a randomly chosen $n = 0, ..., \frac{N_t}{m}$ (using a randomly generated initial condition). We want the learning algorithm to train a non-linear scheme $\hat{u}^{(n+1)m} = S(u^{(n-1)m}, u^{nm}) \approx u^{(n+1)m}$ that explicitly approximates the solution such that the coarse time discretization (using Δt) does not satisfy the CFL condition.

4. Deep learning approach

We use a deep-learning approach to solve the data-driven problem described in section 3.4. Specifically, we train a neural-network composed of convolutional and fully connected (FC or dense) layers whose coefficients (weights and bias terms) are initially unknown. After each layer we use a component-wise non-linear activation. To learn the coefficients we iterate using the Stochastic Gradient Descent (SGD) algorithm to minimize a predefined target function. We experimented with many network architectures (number and sizes of convolution and dense layers) until we obtained one that has good convergence of the SGD algorithm, resulting in good accuracy for the solution of the samples in a predefined testing set.

The fully connected layers try to learn all the connections between the input of the layer and its output using matrix multiplication by unknown weights. For an input matrix A of size $N_{train} \times N_{features}$ (number of training samples times the number of numerical values representing each sample, i.e., features) and an unknown matrix W of size $N_{features} \times N_{layersize}$ the layer output is $L(A) = W \cdot A + b$ where b is a bias vector of unknown weights as well. The unknown elements are then approximated during the training process using the SGD method, with respect to a target function.

The convolution layers try to learn local patterns in the data. They consist of a set of filters of unknown weights w_i. Convolving the input of the layer with each of the filters produces the feature map of the layer: $L(A)_i = w_i * A + b$. The filter weights are also computed by the training process using the SGD algorithm similar the fully connected layer weights. The difference between the fully connected and the convolutional layers is finding global connections versus local patterns.

After designing the network architecture (number of layers, type and size of each layer etc.) we use a gradient descent algorithm to determine all the weights. We define a loss (target) function (specified in sections 4.1 and 4.3 which the SGD algorithm tries to minimize with respect to all the weights. If the network is designed correctly, the gradient descent algorithm converges. Therefore, after a certain number of iterations (called epochs) we expect the network model, with its set of discovered weights, to make an accurate prediction $\hat{u}^{(n+1)m}$ for $u^{(n+1)m}$, from unseen testing data ($u^{(n-1)m}$ and u^{nm}).

During the training phase, we use a validation set consisting of samples that are not in the training set. The validation set is similar to the testing set. The difference is that we do not allow using the testing set during the training step. The validation set is used during the training of the model for monitoring the prediction accuracy over samples that are not in the training set. If the metrics show that the loss value of the training samples keeps decreasing but the validation loss

Table 1
Network layers types, sizes and number of weights.

Layer type	Number of weights	Number of biases	Activation	Layer output shape	Number of activation weights
Convolution	32	16	PReLU	$N_x \times 1 \times 16$	8016
Convolution	6,144	128	PReLU	$N_x \times 128$	64,128
Convolution	16,384	128	PReLU	$N_x \times 128$	64,128
Convolution	128	1	-	$N_x \times 1$	0
Dense	251,001	501	-	N_x	0

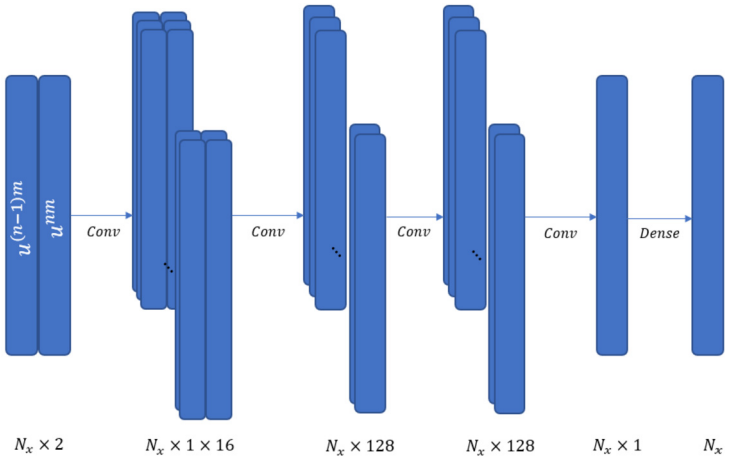


Fig. 2. The network architecture.

increases, the model is over-fitting the training data. This means that the model will accurately predict the training data but will fail with new samples, losing the ability to generalize. We terminate the training phase before the model over-fits.

Finally, using the trained model we test its ability to make predictions on a testing data-set. The samples chosen for testing are different from the training and validation samples. We examine the performance of the network on this testing set as shown in section 4.4.

4.1. Network architecture

We use a spatio-temporal network architecture. Recall that the data-set is formed using solutions of the wave problem that are dependent on space and time. We design the network accordingly, so that the first layers learn the temporal elements of the data and the layers that come after learn the spatial elements. This architecture outperformed all other architectures. We first use a convolution layer to learn the time elements between the two input vectors. We use 16 filters of size 2 for this layer to extract only the temporal information. Afterwards, we reshape the output and use two convolution layers with 128 filters of size 3 and 1 followed by a convolution layer with 1 filter of size 1. These learn the local spatial connections in the data. The architecture design of the network concludes with a dense layer. Table 1 presents the types of the layers, sizes and number of weights. Fig. 2 presents a sketch of the network architecture.

The loss (target) function we use for training the network is the Mean Squared Error (MSE). We define $MSE_{k,(n+1)m} = \frac{1}{N_x}\sum_{i=1}^{N_x}\left(\hat{u}_i^{(n+1)m} - u_i^{(n+1)m}\right)^2$ for each randomly generated initial condition where $1 \le k \le N_{ic}$ and N_{ic} is the number of initial conditions used to create the training set. The MSE loss is then $\frac{m}{N_{ic}N_t}\sum_{k=1}^{N_{ic}}\sum_{n=1}^{\frac{N_t}{m}} MSE_{k,(n+1)m}$. Low MSE means that the predictions are close to the true values. The gradient descent algorithm tries to minimize the MSE with respect to the weights of the network.

4.2. Activation function

Most layers are followed by non-linear activation functions. The non-linearity provides an infrastructure for the network to act as a more powerful approximation algorithm. We use a rectifier function called Parametric Rectified Linear Unit (PReLU, [18]). The PReLU activation is defined as follows:

$$f(x) = \begin{cases} ax, & x < 0 \\ x, & else \end{cases} \tag{3}$$

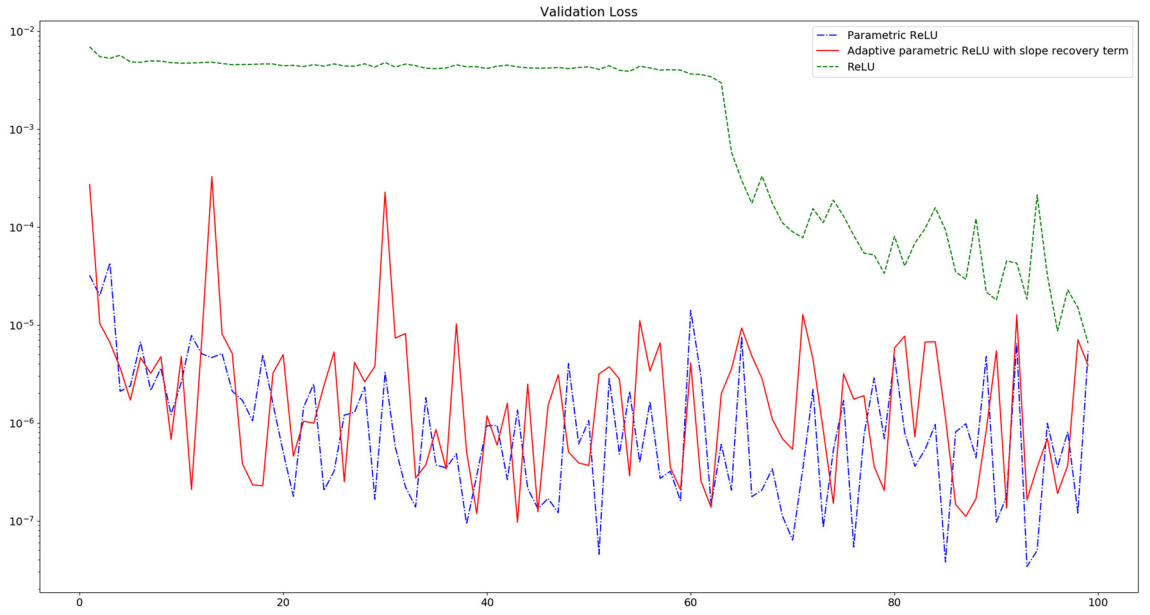


Fig. 3. Comparison of activation functions validation losses over the first 100 epochs in log-scale.

where a is a learned array with the same shape as x. The PReLU function adjusts the output in a way that is learned through the training phase and is not necessarily linear.

We experimented with the adaptive activation functions, developed by Jagtap et al. [19], [20]. This class of optimized adaptive activation functions (global, layer-wise, and neuron-wise) aims to accelerate the loss convergence rate. Fig. 3 presents the validation losses of two neuron-wise-like locally adaptive activation functions, compared to a non-adaptive ReLU activation function. Observe that both adaptive activation functions outperform ReLU. We tested both global and layer-wise adaptive activation functions, which resulted in significantly slower convergence compared to their neuron-wise counterparts and thus are not shown in the figure. The PReLU activation converged to the lowest local minimum (compared to the other activation functions). Therefore, we used PReLU (3) for the results in the remaining paper.

4.3. Physically-informed loss

We propose to enhance to the network described above by adding a loss term that makes the network "aware" of the specific physical problem it is trying to solve. We recall that the training dataset was formed by using a FD scheme with a fine spatial discretization and picking a subset of every m^{th} step. We exploit this knowledge to construct a physically-informed loss term. During the training phase, the network uses gradient descent to minimize the physically-informed loss and is expected to perform better with the additional knowledge of the problem.

When training the model with the physically-informed loss we first create the training data-set differently. The input-output mapping remains the same, but we also add $u^{(n+1)m+1}$ as an auxiliary term. Hence, when training the model, the true values of $u^{(n+1)m}$ and $u^{(n+1)m+1}$ are usable inside the loss function. We remark that the output of the network remains just $\hat{u}^{(n+1)m}$; we achieve this by designing an architecture where the last layer outputs a single vector and the loss terms handle two vectors. A sketch of the physically-informed loss calculation process is given in Fig. 4.

In each step of the training session, when calculating the loss for each training sample, we have access to the true values of the vectors $u^{(n-1)m}, u^{nm}, u^{(n+1)m}$ and $u^{(n+1)m+1}$. We first use $u^{(n-1)m}$ and u^{nm} to predict $u^{(n+1)m}$ as shown above and we denote $\hat{u}^{(n+1)m}$ as the predicted value. We calculate the MSE and set $l_{MSE} = \frac{m}{N_{ic}N_t}\sum_{k=1}^{N_{ic}}\sum_{n=1}^{\frac{N_t}{m}} MSE_{k,(n+1)m}$. We then use the FD method to calculate $u^{(n+1)m+2} = FD(u^{(n+1)m}, u^{(n+1)m+1})$ and $\hat{u}^{(n+1)m+2} = FD(\hat{u}^{(n+1)m}, u^{(n+1)m+1})$. We can apply the FD method j times to get $u^{(n+1)m+j}$ and $\hat{u}^{(n+1)m+j}$ such that $2 \le j \le m$. We then calculate the MSE between these vectors and set $l_{PI} = \frac{m}{N_{ic}N_t}\sum_{k=1}^{N_{ic}}\sum_{n=1}^{\frac{N_t}{m}} MSE_{k,(n+1)m+j}$. The combined loss term we use is then $loss = l_{MSE} + l_{PI}$. Note, that the case $j = 1$ is equivalent to calculating l_{MSE}:

$$\forall i = 1, ..., N_x,\ k = 1, ..., N_{ic}: \quad u_i^{(n+1)m+2} - \hat{u}_i^{(n+1)m+2} = FD(u_i^{(n+1)m}, u_i^{(n+1)m+1}) - FD(\hat{u}_i^{(n+1)m}, u_i^{(n+1)m+1}) =$$

$$\cancel{2 \cdot u_i^{(n+1)m+1}} - u_i^{(n+1)m} + c^2 \cdot \frac{\Delta t^2}{\Delta x^2}\left(u_{i+1}^{(n+1)m+1} \cancel{- 2 \cdot u_i^{(n+1)m+1}} + u_{i-1}^{(n+1)m+1}\right) -$$

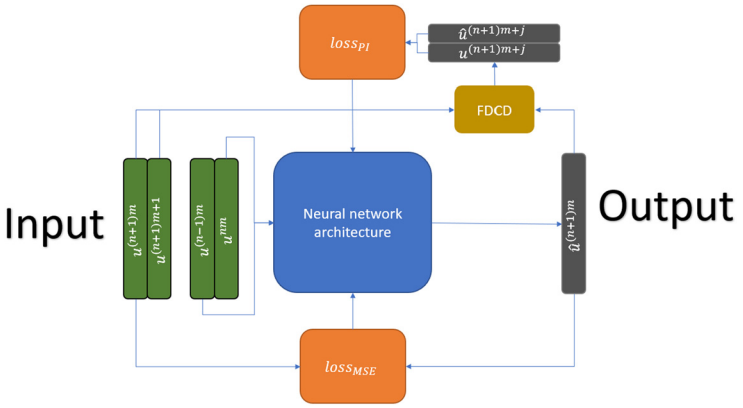


Fig. 4. Physically-informed loss calculation process.

$$-\left(\cancel{2\cdot u_i^{(n+1)m+1}}-\hat{u}_i^{(n+1)m}+c^2\cdot\frac{\Delta t^2}{\Delta x^2}\left(u_{i+1}^{(n+1)m+1}\cancel{-2\cdot u_i^{(n+1)m+1}}+u_{i-1}^{(n+1)m+1}\right)\right)=$$
$$u_i^{(n+1)m}-\hat{u}_i^{(n+1)m}.$$

Therefore, we get $MSE_{k,(n+1)m+2}=\sum_{i=1}^{N_x}\left(\hat{u}_i^{(n+1)m+2}-u_i^{(n+1)m+2}\right)^2=\sum_{i=1}^{N_x}\left(\hat{u}_i^{(n+1)m}-u_i^{(n+1)m}\right)^2=MSE_{k,(n+1)m}$. We tested several choices for j and found that the choice of $j=m$ performed the best. When choosing $j=m$, we compute using the FD method, all of the time steps $(n+1)m+2,\ldots,(n+2)m$ and therefore we penalize their accumulated loss. The results for this method are given in section 5.2.

A concern when using this method is that initially (at the first stages of the training step) the network produces poor predictions for $\hat{u}^{(n+1)m}$, confusing the FD that might deny the networks convergence. To solve this issue, we start training the network with only the MSE loss term and later apply the additional physically-informed term. This way we let the network learn the problem first so it could make relatively accurate predictions for $\hat{u}^{(n+1)m}$ and use them in the physically loss term. Results show that after switching to the physically-informed loss, the calculated MSE loss drops 10 times lower.

The physically-informed loss term uses the FD method (2) and thus is clearly a differentiable loss in its input of two time step vectors. This enables the application of the automatic differentiation mechanism of Keras [16] for the gradient descent method.

4.4. Model evaluation

There are several methods to evaluate the performance of the trained model. The model predicts the solution of the PDE so we chose to use comparison methods that are commonly used for evaluating numerical schemes. We first define an initial condition which is a linear combination of sine functions, such that we can calculate its analytic solution over time. We can then compare the numerical methods to the analytic solution.

The numerical methods we used for comparison are the trained models (with and without the physically-informed loss term) and an implicit method (Crank-Nicolson) [9,10]. In this work we solve only a one dimensional problem so we could calculate with a small Δt and so we show the finer version along with the other comparisons. Note, the fine FD discretization gives much better results. However, with a small Δt, in a real problem it is expensive to calculate the FD iterations. We compare the methods over time using the norm:

$$\begin{matrix}\forall n=0,\ldots,\frac{N_t}{m}\\ \forall k=0,\ldots,N_{ic}\end{matrix}\qquad E_{k,(n+1)m}=\frac{\sqrt{\sum_{i=0}^{N_x}\left|u_i^{(n+1)m}-\tilde{u}_i^{(n+1)m}\right|^2}}{\sqrt{\sum_{i=0}^{N_x}\left|u_i^{(n+1)m}\right|^2}}\tag{4}$$

where $\tilde{u}$ is the solution produced by the compared method.

Two evaluations are of interest:

- **Single-step prediction**: We use the true values of $u^{(n-1)m}$ and u^{nm} to predict $u^{(n+1)m}$. We calculate $E_{k,(n+1)m}$ as in (4) for each sample in the testing set (varying initial condition and time stamps). We average the error over all time steps to obtain $\overline{E}=\frac{m}{N_{ic}N_t}\sum_{k=1}^{N_{ic}}\sum_{n=1}^{\frac{N_t}{m}}E_{k,(n+1)m}$ and compare this error term between all methods. A smaller average means results closer to the analytic solution.

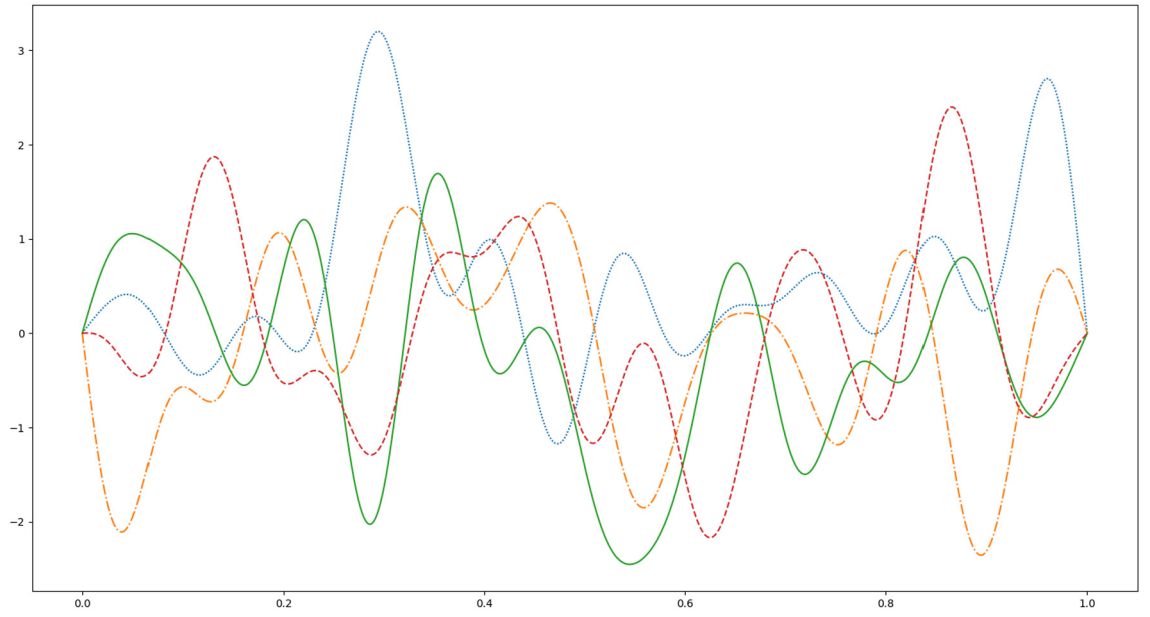


Fig. 5. 5 Randomly sampled initial conditions that were created for the data-set.

- **Multi-step prediction**: We start by using u^0 and u^m to predict $\hat{u}^{2m}$. We use u^m and the predicted $\hat{u}^{2m}$ to predict $\hat{u}^{3m}$. In general, we use $\hat{u}^{(n-1)m}$ and $\hat{u}^{nm}$ to predict $\hat{u}^{(n+1)m}$. We compare the accumulative error over time between all methods. For each initial condition we calculate $\overline{E_k} = \frac{m}{N_t}\sum_{n=1}^{\frac{N_t}{m}} E_{k,(n+1)m}$ from the multi-step predictions. We then take the mean of all $\overline{E_k}$ in the testing set (Mean of $\overline{E_k}$ in Table 3). In addition, we calculate the maximal and median of values of $E_{k,(n+1)m}$ over m and average them over k. Smaller metrics (mean, maximum and median) mean results closer to the analytic solution.

5. Numerical tests and results

5.1. Experiment setup

We tested the method on problem (1) setting $\Omega = [0,1]$, $T = 1$, $c = 7$ and $v_0(x) = 0$. Problem (1) then becomes:

$$\begin{cases} \ddot{u}(x,t) = c^2 u_{xx}(x,t) & 0 \le x \le 1,\ 0 \le t \le T, \\ u(x,0) = u_0(x) & 0 \le x \le 1, \\ \dot{u}(x,0) = 0 & 0 \le x \le 1, \\ u(0,t) = u(1,t) = 0 & 0 \le t \le T. \end{cases} \tag{5}$$

For the numerical grid we chose $N_t = 4000$ and $N_x = 500$ on the domain $[0,1]$. For these conditions the CFL number is:

$$\alpha = c\frac{\Delta t}{\Delta x} = 7 \times \frac{\frac{1}{400}}{\frac{1}{500}} = 8.75 > 1,$$

which violates the CFL condition. Multiplying Δt by $m = 10$ we get a CFL number of 0.875 which satisfies the stability condition.

The orthonormal basis functions we use are $\{sin(\pi kx)\}_{k=1}^{20}$. We use these basis functions to create the initial conditions that form the training, validation and testing data-sets. Each initial condition is a linear combination of the basis functions $\sum_{n=1}^{20} a_n sin(\pi nx)$. The coefficients are randomly generated and satisfy $\sum_{n=1}^{20} |a_n|^2 = 1$. Fig. 5 presents 5 examples of randomly generated initial conditions.

We generate $1,250$ initial conditions and use the FD method with the fine discretization $(m\Delta t)$ as described in section 3.4 to create the data-set consisting of $1,250 \cdot 397 = 496,250$ samples. Each sample is of the form:

$$Input:\ (u^{n-10}, u^n) \quad \rightarrow \quad Label:\ (u^{n+10}, u^{n+11}), \quad n = 10, 20, ..., 3980$$

The samples are split into training, validation and testing sets with ratios of 80%, 10% and 10% respectively (total size of 5 GB of data). The stochastic gradient descent optimizer we used is the ADAM optimizer [15] with batch size of 16 and the

Table 2
Single-step comparison of the various methods over the testing data.

Method	$\overline{E}$
Model trained without physically-informed component	0.012139
Model trained with physically-informed component	**0.012067**
Explicit FD	0.012091
Implicit	1.23709

Fig. 6. Error over time comparing four different approximation methods.

code was written using Keras [16]. We trained for 50 epochs without the MSE loss term and 350 more with the combined (MSE and physically-informed) loss term. The model was saved based on best performance on the validation set. We used a nVidia Tesla T4 GPU to train the model for approximately 2 days. The trained model size is approximately 5 MB. This model is used for the following calculations.

5.2. Results

We compare the performance of the method as discussed in section 4.4. The evaluations are performed on the testing set (which does not include samples from the training or validation sets). The single-step prediction average errors are shown in Table 2. As shown in the table, the model performs better than the implicit method. Recall that the FD method is calculated using the fine discretization $\frac{\Delta t}{m}$. It is used only as a reference solution (with coarse discretization Δt the error "blows up").

The training set used for training the model contains solutions computed using the finite-difference method. We observe that the errors of the model (with and without the physically-informed component) and the explicit finite-differences method are similar, which shows that the model trained well for the single step prediction. Using the model for multi-step prediction is different since it is not what the model was trained for, and therefore it is a more difficult task. We expect the error values to vary between the methods and showcase the contribution of the physically-informed component in the multi-step results.

The multi-step prediction over time error for a specific initial condition is given in Fig. 6. The FD result is shown for reference. We observe that although the implicit method remains stable, the error is much larger than the one of the model. In addition, the improvement of the physically-informed element is clear from the figure. The best performance for this initial condition is by using the neural-network model with the physically-informed loss term. We tested the model with several different initial conditions constructed using the defined basis functions and achieved the same result. The error over time is oscillatory due to the nature of the sine functions we used to create the data.

The multi-step metrics discussed in section 4.4 are given in Table 3. We observe higher error values due to the accumulative error in the multi-step process. As explained in 4.4, the explicit method is calculated with a finer time discretization and the performance is better as expected, it is shown only for reference. The model with the physically-informed loss term is performing the best.

Table 3
Multi-step comparison of the various methods over the testing data.

Method	Mean of $\overline{E}$	Mean of $max(E_{ic}(t_n))$	Mean of $median(E_{ic}(t_n))$
Model trained without physically-informed component	0.197140	0.665867	0.153857
Model trained with physically-informed component	**0.021158**	**0.062016**	**0.018769**
Explicit FD	0.011912	0.036059	0.010889
Implicit	1.122231	1.791727	1.155944

Fig. 7. Error over time comparing the implicit method and the model with physically-informed term, with an initial condition of the type (6).

5.3. Generalization

Using sines as basis functions for training and testing the model raises the question whether the model can perform well given an initial condition that was not formed using a finite linear combination of sine functions. We explore the ability of the model to generalize - predict solutions of initial conditions it was not trained on with high accuracy. We define a set of testing random initial conditions using:

$$u(x,0) = \sum_{k=1}^{20} a_k e^{-(x-b_k)^2+c_k} sin(\pi k x), \tag{6}$$

where $\sum_{k=1}^{20} |a_k|^2 = \sum_{k=1}^{20} |b_k|^2 = \sum_{k=1}^{20} |c_k|^2 = 1$, $a_k, b_k, c_k \in [-1,1]$ are generated randomly. Using separation of variables, we find the analytic solution of the wave problem given such an initial condition by finding its Fourier coefficients. The coefficients are calculated using:

$$A_l = \int_0^1 \left(\sum_{j=1}^{20} a_j e^{-(x-b_j)^2+c_j} sin(\pi j x) sin(\pi l x) \right) dx, \quad \forall l: \ A_l \neq 0 \ almost \ surely,$$

and the solution is given by:

$$u(x,t) = \sum_{l=1}^{\infty} A_l sin(\pi l x) cos(c \pi l t).$$

We compare the methods to the explicit FD approximation. We calculate the norms as in (4.4) but with the fine FD approximation instead of the analytic solution. We compare the model with and without the physically-informed component and the implicit method in terms of multi-step prediction performance as defined in section 4.4. An example for the error over time for a randomly generated initial condition as in (6) is given in Fig. 7. Table 4 presents the metrics discussed in section 4.4 for a testing set generated with initial conditions as in (6).

Table 4
Multi-step comparison of the various methods over the testing data.

Method	Mean $\overline{E_k}$	Mean $\max_n(E_{k,(n+1)m})$	Mean $median_n(E_{k,(n+1)m})$
Model trained without physically-informed component	0.175957	0.597869	0.145254
Model trained with physically-informed component	**0.049573**	**0.104450**	**0.048273**
Implicit	1.131291	2.050409	1.149680

5.4. Stiff source terms

Up to this point, we have demonstrated the capabilities of our model in scenarios with highly smooth initial conditions which were resolved faithfully by the simulation parameters. However, real life problems in fields such as computational fluid dynamics or underwater acoustics often operate under different sets of simulation parameters that face other numerical difficulties which are not directly related to the CFL condition. One such obstacle is the inclusion of a *stiff source term*.

Consider equation (5). We transform it into an inhomogeneous wave equation by introducing a source term $s(x,t)$:

$$\begin{cases} \ddot{u}(x,t) = c^2 u_{xx}(x,t) + s(x,t) & 0 \le x \le 1,\ 0 \le t \le T, \\ u(x,0) = u_0(x) & 0 \le x \le 1, \\ \dot{u}(x,0) = 0 & 0 \le x \le 1, \\ u(0,t) = u(1,t) = 0 & 0 \le t \le T. \end{cases} \tag{7}$$

The inclusion of a source term does not constitute a problem in and of itself. However, by choosing a source term whose behavior requires finer simulation discretization than the other components of the equation we might encounter the phenomenon of *stiffness*, and consequently get inaccurate results.

We add a source function with the following form:

$$s(x,t) = \begin{cases} Asin(rc\pi t)sin(\pi x), & t \in [t', t''] \\ 0, & else \end{cases} \tag{8}$$

where $r \in \mathbb{N}$, $A \in \mathbb{R}$, and $0 \le t' < t'' \le T$.

Using the same initial conditions as in 5.1, we get the following analytic solution:

$$u(x,t) = \begin{cases} u_h(x,t) + A\frac{(-r\cos(c\pi t) - \sin(c\pi rt))\sin(\pi x)}{c^2\pi^2(r^2-1)}, & t \in [t', t''] \\ u_h(x,t), & else \end{cases} \tag{9}$$

where $u_h(x,t)$ is the analytic solution of the homogeneous problem (5).

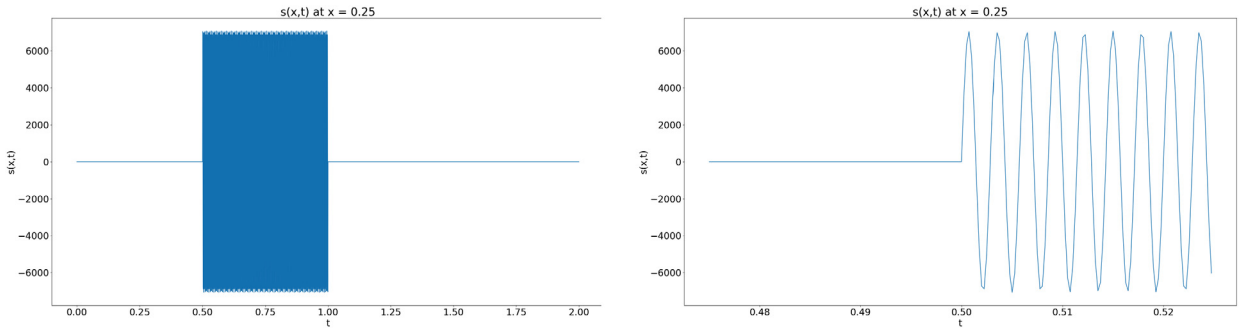
The spatial oscillations of (9) are roughly of the same order of magnitude as those of the initial condition. However, the temporal oscillations are dependent on r. Hence, by choosing r to be larger than the temporal oscillations of the homogeneous analytic solution, we might encounter stiffness. Note that A must be at least the same order of magnitude as r^2 in order to offset the division by $r^2 - 1$ in (9); otherwise we'd get an insignificant contribution to the analytic solution. Lastly, we chose t' and t'' at which the term added due to the inclusion of a source in (9) zeros out to avoid discontinuities, and clearly see the influence of a stiff source.

In our computation we chose: $r = 100$, $A = 10,000$ and $[t', t''] = [0.5, 1]$. We also increased the propagation time to $T = 2$ to observe the model's accuracy after the end of the pulse. Consequently, we doubled N_t to keep the CFL number the same as in 5.1.

In Fig. 8 we see the highly oscillatory behavior of the source function, and the discrepancy between its values and those of the initial conditions Fig. 5. Due to insufficient sampling, we miss important aspects that affect the values of the numerical solution. The consequences of this problem are evident in Fig. 9. Both the FD method and the model perform well, until a noticeable jump that occurs at $t = 0.5$, which is the beginning of the pulse of the source function. However, there is one major difference between the model's performance and that of the FD method. In the FD method, the stiffness in $[0.5, 1]$ leaves a lingering effect on future simulations after $t = 1$. This is in stark contrast to the model's error, where there's a small "bump" around the pulse area, after which it recovers without accumulating. Once more we see that the physically informed loss adds a valuable contribution to our model. We also tested the implicit Crank-Nicolson with a CFL number of approximately 1.5. It remained stable, but the results were highly inaccurate. Consequently, the large error dominated the plot scaling, and thus we excluded it from Fig. 9.

6. Conclusion

In this work we introduced a method to solve PDEs explicitly while violating of the CFL condition. We used deep-learning to overcome the stability issue such that the trained neural network is used as an explicit numerical scheme. Additionally,



(a) The source over the full propagation time.

(b) A zoomed in view of the beginning of the pulse.

Fig. 8. The source $s(x,t)$ at $x = 0.25$.

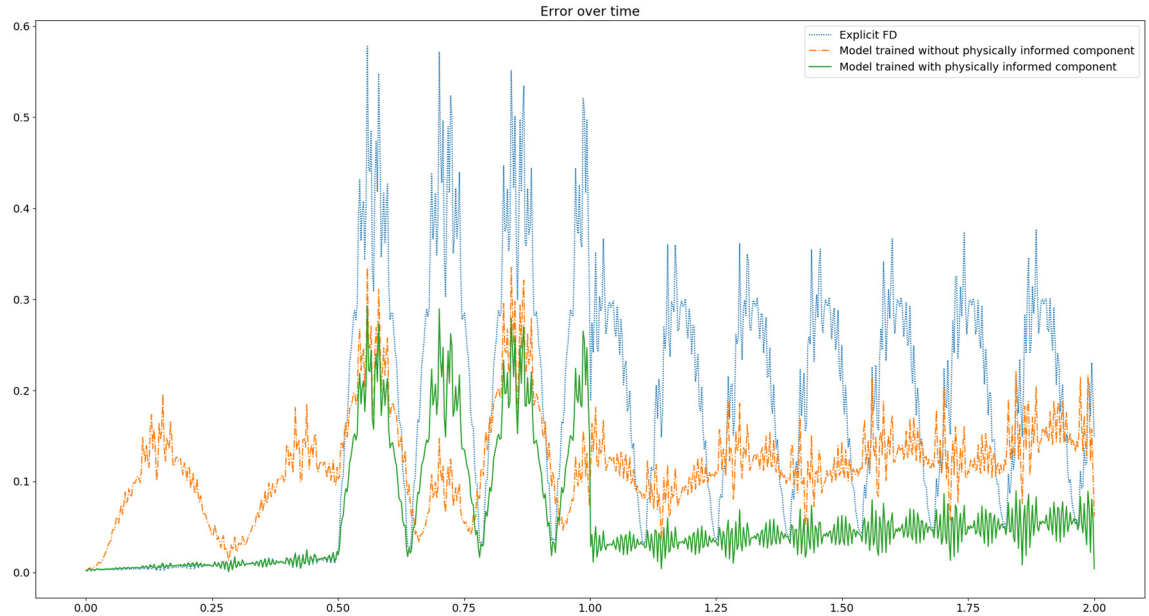


Fig. 9. Error over time comparing the explicit FD method with $N_t = 8,000$ and the model with $N_t = 800$. All simulations have the same source term (8).

we improved the method by training the model with a physically-informed loss term that makes the network aware of the physical problem it is trying to approximate. We tested the model on the one dimensional wave equation and examined the results. We observed better accuracy than the reference implicit model while preserving the fast calculation times. The presented approach can be used for many other PDE related problems.

We plan on expanding the model to more difficult problems. For example, stiff problems with both large and small physical processes such as the weather equations that model both wind propagation (low propagation speed) and influence of acoustic waves on the wind (high propagation speed) on the same grid. We also intend to check the robustness of the model with respect to noise in the data and develop accurate models in the presence of noise. Also, further research will be done on making the network more physically aware (in addition to the physically-informed loss term). For example, designing network layers that mimic the state-of-the-art numerical schemes while including a set of learned weights to optimize for the specific data-set used.

CRediT authorship contribution statement

Oded Ovadia: Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Adar Kahana:** Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Eli Turkel:** Conceptualization, Methodology, Supervision. **Shai Dekel:** Conceptualization, Methodology, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. Abgrall, C.W. Shu, Handbook of Numerical Methods for Hyperbolic Problems: Basic and Fundamental Issues, 1st edition, North Holland, 2016.
[2] R.J. LeVeque, Finite Volume Methods for Hyperbolic Problems, Cambridge Texts in Applied Mathematics, 2002.
[3] B. Gustafsson, H.O. Kreiss, J. Oliger, Time-Dependent Problems and Difference Methods, 2nd edition, Pure and Applied Mathematics, Wiley, 2013.
[4] A. Kahana, E. Turkel, D. Givoli, Convective wave equation and time reversal process for source refocusing, J. Comput. Acoust. 26 (02) (2018) 1850016.
[5] D. Richtmeyer, K.W. Morton, Difference Methods for Initial Value Problems, 2nd edition, Wiley, New York, 1967.
[6] R. Courant, K. Friedrichs, H. Lewy, Über die partiellen Differenzengleichungen der mathematischen Physik, Math. Ann. 100 (1928) 32–74.
[7] M. Raissi, P. Perdikaris, G.E. Karniadakis, 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).
[8] A. Kahana, E. Turkel, S. Dekel, D. Givoli, Obstacle segmentation based on the wave equation and deep learning, J. Comput. Phys. 413 (2019).
[9] U.M. Ascher, S.J. Ruuth, R.J. Spiteri, Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations, Appl. Numer. Math. 25 (2–3) (1997).
[10] L. Pareschi, G. Russo, Implicit-explicit Runge-Kutta schemes for stiff systems of differential equations, Recent Trends Numer. Anal. 3 (269–289) (2000).
[11] E. Turkel, G. Zwas, Explicit large time step schemes for the shallow water equations, in: Advances in Computer Methods for Partial Differential Equations III, 1979, pp. 65–69.
[12] F.J. Liu, H.T. Dong, Second-order large time step wave adding scheme for hyperbolic conservation laws, J. Comput. Phys. 408 (2020) 1–36.
[13] L.C. Evans, Partial Differential Equations, American Mathematical Society, Providence, 1998.
[14] J. Jost, Partial Differential Equations, Springer-Verlag, New York, 2002.
[15] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv:1412.6980, 2014.
[16] F. Chollet, Keras, https://keras.io/.
[17] N.J. Higham, Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics, 1996, xxxviii (595-663).
[18] H. Kaiming, Z. Xiangyu, R. Shaoqing, S. Jian, Delving deep into rectifiers: surpassing human-level performance on ImageNet classification, arXiv:1502.01852.
[19] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2019).
[20] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks, Proc. R. Soc. A 476 (2020).