Electronic Research Archive

https://www.aimspress.com/journal/era

ERA, 33(2): 890–906.
DOI: 10.3934/era.2025040
Received: 4 September 2024
Revised: 02 January 2025
Accepted: 15 January 2025
Published: 14 February 2025

Research article

Constructing hidden differential equations using a data-driven approach with the alternating direction method of multipliers (ADMM)

Jye Ying Sia[1,2,*], Yong Kheng Goh[2], How Hui Liew[2] and Yun Fah Chang[3]

[1] School of Mathematical Sciences, Sunway University, Bandar Sunway, 47500 Selangor, Malaysia

[2] Lee Kong Chian Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Bandar Sungai Long, Cheras, 43000 Kajang, Selangor, Malaysia

[3] School of Accounting and Finance, Taylor's University, 47500 Subang Jaya, Selangor, Malaysia

* **Correspondence:** Email: jyeyings@sunway.edu.my.

Abstract: This paper adopted the alternating direction method of multipliers (ADMM) which aims to delve into data-driven differential equations. ADMM is an optimization method designed to solve convex optimization problems. This paper attempted to illustrate the conceptual ideas and parameter discovery of the linear coupled first-order ODE. The estimation of the coefficients of the underlying equation utilized a combination of algorithms between physics-informed neural networks (PINNs) and sparse optimization. Both methods underwent a sufficient amount of looping during the search for the best combinations of coefficients. The PINNs method took charge of updating weights and biases. The updated trainable variables were then fetched to the sparse optimization method. During the sparse optimization process, ADMM was used to restructure the constrained optimization problems into unconstrained optimization problems. The unconstrained optimization problem usually consists of smooth (differentiable) and non-smooth (non-differentiable) components. By using the augmented Lagrangian method, both smooth and non-smooth components of the equations can be optimized to suggest the best combinations of coefficients. ADMM has found applications in various fields, such as signal processing, machine learning, and image reconstruction, which involve decomposable structures. The proposed algorithm provides a way to discover sparse approximations of differential equations from data. This data-driven approach provides insights and a step-by-step algorithm guide to allow more research opportunities to explore the possibility of representing any physical phenomenon with differential equations.

Keywords: data-driven differential equations; alternating direction method of multipliers (ADMM); physics-informed neural networks (PINNs); sparse optimization; coupled first-order ODE

1. Introduction

Data-driven differential equations aim to formulate and study various types of differential equations using simulated or available data without explicit knowledge of the underlying physical principles governing the system. As we know, well-known differential equations are often derived from known physical laws or theories. In data-driven approaches that solely rely on data, it is believed that the data carries information and properties representing the physical system. Applying machine learning mechanisms with the help of statistical methods, the underlying differential equations can be estimated directly from the observed data.

In order to demonstrate the accuracy of the parameter estimation, a set of coupled first-order ODEs is solved using both the numerical solution, the explicit Runge-Kutta method of order 5(4) (RK45), and the analytical method, PINNs. As a general note, any selection of a suitable numerical method should work to compare with the trained solutions obtained from the PINNs method. Generally, quoted from one of the chapters in [1], the RK method is an effective and widely used method for solving the initial-value problems of differential equations. In Python Scipy library documentation, the numeric 5(4) indicates that the error is controlled assuming the accuracy of the fourth-order under the classification of accuracy method, but steps are taken using the fifth-order accuracy formula. The analytical solutions have been verified (see [2]) that they are similar to the numerical solutions. In the same paper, it was illustrated that the universal approximation theorem (UAT) stated that neural networks (NN) have universality, i.e., any function can be approximated by approaching the result with NNs. NNs can achieve this by adjusting weight and bias parameters in the hidden layer to derive the staircase fitting of the respective function. The weights determine the width of the function, whereas the biases determine the position of the function.

The beauty of the data-driven approach is shown using the set of experimental data points in the coefficient search. Simply put, the set of experimental data points portrays the features of the ODE, specifying the parameter estimation of the unknown underlying equations. The parameter estimation can be achieved with the combination of the PINNs method and the sparse optimization method. The use of the PINNs method is due to its ability to provide reasoning on physical phenomena that model the dynamic development of the system of equations, explained by the derivatives computed from the PINNs method. In this paper, in order to achieve the purpose of parameter estimation, the known coupled first-order ODE is kept aside, and then the analytical solution is employed as the experimental data. Notation-wise, the experimental data for a coupled first-order ODE is denoted as $u[x, y]$ during the algorithm setting. Knowing the values of $u[x, y]$, the derivatives, u_t, can be computed by using the PINNs method. To kickstart the estimation process, a column vector array, ϕ, consisting of all potential variables, is formed. For example, $\phi = \begin{bmatrix} 1 & x & x^2 & x^3 & y & \dots \end{bmatrix}$. If there are in total n number of potential variables listed, then the dimension of the ϕ matrix follows $n \times 1$. Since all the potential variables are generated from the variables x and y, based on the given data, expressed in u in terms of x and y, the numerical values of each variable listed in the ϕ array will be computed accordingly. The desired coefficient matrix is denoted as Λ. This coefficient matrix, Λ, generally is defined to follow dimension $m \times n$, depending on the definition of the functions to be optimized. Hence, similar to the concept of solving a system of linear equations, $Ax = b$, the setting to search for the best coefficient combination is set to solve $\Lambda^T \phi = u_t$. The Λ matrix will undergo a series of training processes to optimize the coefficients. It is expected that the outcome of the optimization should suggest the best-fit combination

of the coefficients that represent the underlying equations. The detailed process of optimization is illustrated in Section 3.2.

To the best of our knowledge, the most frequently used method in solving inverse problems with the aim to uncover the coefficient of the governing equations can be achieved by optimization methods, Bayesian inference, and regression techniques. Some of the examples are sparse modeling (demonstrated in [3]) and unconstrained mathematical optimization such as the gradient descent algorithm involving iterations in training machine learning models. Recently, Kamyab et al. [4] have conducted a survey on current deep learning methods used for solving inverse problems. They categorized the existing analytical methods into analytic inversion, iterative methods, discretization as regularization, and variational methods, which often make use of the concept to regularize and constrain the problem to obtain numerically stable solutions. In their paper, the alternating direction method of multipliers (ADMM) is mentioned which can deal with a huge collection of images that need to be optimized with a regularizer. This is because the ADMM algorithm suggests that by combining the features of dual decomposition and the augmented Lagrangian method, it could efficiently handle large-scale problems. ADMM uses the augmented Lagrangian method as the optimization catalyst to search for the best combinations of coefficients. Chen et al. [5] have further described the usage of ADMM in discovering the coefficient of the underlying equation. In [6], they outlined the idea of ADMM in low-rank matrix recovery. In fact, ADMM is an optimization algorithm being used to solve convex optimization problems, particularly those involving separable objectives or constraints. The intention of this paper is to apply ADMM for parameter estimation. The optimization problem setting in this paper contains a combination of differentiable parts and non-differentiable parts. In this context, the concept of alternating in ADMM refers to alternating between optimizing the differentiable part and non-differentiable part while updating the Lagrange multipliers. The PINNs method is used to optimize the differentiable part. Then, the sparse optimization method is applied to optimize the non-differentiable part which could then suggest the best combination of parameters which leads to the simplest equations. Hence, PINNs and sparse optimization are applied alternatively in a loop. This is a process to keep the trainable variables (weights and biases) updated and at the same time search for optimized sparse solutions. Both the PINNs and sparse optimization methods attempt to extract the underlying properties of the differential equations that describe the relationships between variables within a system based solely on experimental data. This invention allows for the discovery or modeling of complex systems where the underlying governing equations might be unknown or difficult to ascertain through numerical methods.

2. Significance

It is commonly known, at least to the best of our literature review, that dealing with the inverse problem of unknown underlying equations is difficult to recover. This means that given a set of data from any physical phenomenon, it is a great challenge to search for the underlying equations that govern the phenomena because the underlying equations are unknown. Hence, understanding the features of the given data set is important. Since the experimental data is trained using the proposed PINNs method, it could speed up the modeling process as the coupled first-order ODE is well studied beforehand. The specialty of the PINNs method, which allows the training process to be done with reasoning, has indirectly assisted the understanding of the dynamics of the system.

The aim is to uncover the equations governing a physical phenomenon directly from data, which

can be done in the spirit of machine learning or statistical learning. This work develops a novel framework to discover the underlying governing equations in a dynamical system. The proposed combinations of algorithms provide a way to discover sparse approximations of differential equations from data. This data-driven approach provides insights and a step-by-step algorithm guide to allow more research opportunities to explore the possibility of representing any physical phenomenon with differential equations. The illustration showing how the proposed method works on the simple linear coupled first-order ODE can serve as a ground truth of the reliability and effectiveness of the method. Then, the mechanism could be extended to deal with complex systems.

3. Materials and methods

This section will outline the details of the PINNs method and the sparse optimization method.

3.1. The physics-informed neural networks (PINNs) method

In every neural-network training process, trainable variables are initiated. Similar to the setting of conventional neural networks (CNNs) and multi-layer perceptrons (MLPs) as described in [7], the PINNs method requires the setting of hyperparameters. Since the setting of hyperparameters is done before the training process, these parameters do not learn from the data. Instead, these parameters are set to determine the training behavior and model architecture, specifying how models can be learned or formed. Initially, it is required to set up the model architecture by deciding the number of input, hidden, and output layers, number of neurons, and activation function in use. The number of neurons serves as the communication platform between the input layer and hidden layers and then between hidden layers and output layer during the training process. All the layers of neurons are connected by weights. Hyperparameters such as the learning rate, batch size, number of epochs, and types of optimizers are initialized for model compilation in the PINNs method. Every epoch of the training process will lead to the trainable variables, i.e., weight and bias updates with backpropagation, particularly taking place in the hidden layers. All CNNs, MLPs, and PINNs are also required to employ an activation function. During the training process, the activation function is selected as a catalyst to increase the accuracy of the model training. Some of the commonly used activation functions are ReLU, sigmoid, and tanh. Lastly, the output layer provides the trained solutions. Unlike CNNs and MLPs, the PINNs method is designed to approximate solutions to partial differential equations (PDEs) or ordinary differential equations (ODEs). This can be achieved because the PINNs method incorporates the underlying physics of a problem by embedding differential equation constraints into the loss function. The uniqueness of the PINNs method which incorporates the data and derivative loss in the model is classified as the regularization step in neural networks. This regularization step can ensure the neural network learns solutions consistent with the underlying physical laws and generalizes well to unseen scenarios. The entire training process of PINNs is illustrated in Figure 1. With this, model compilation is initiated to build the model so that the loss function embedded can assist in ensuring that optimization can be achieved for data consistency, and at the same time complying with physical laws.

Usually, the loss function is formed based on the mean square difference from the data values part and derivatives (gradients) part, as described in [8]. During this optimization process, the optimizer algorithm is specified as well. The optimizer selected is Adam with a default learning rate of 0.001. The Adam optimizer is chosen because Adam could adapt to the learning rate for each parameter

individually, which is especially useful in solving differential equations. The default learning rate of 0.001 in Adam has been empirically tested and works well in many deep learning applications (see [9]). The learning rate of 0.001 is also adopted in Scikit-learn from the Python library which provides tools for machine learning, data mining, and data analysis. Once the model is compiled, the next training step is to fine-tune and seek the best-trained solution with the smallest loss value. This step is where the model fitting is executed. Model fitting is where the actual training process begins using training data with forward and backpropagation processes to update the trainable variables in each epoch.

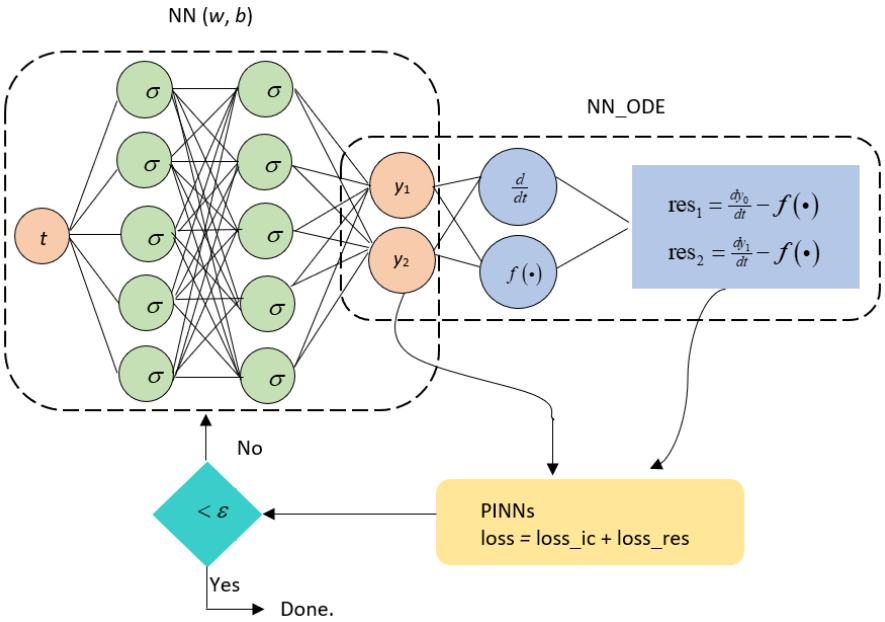

Figure 1. Procedure of PINNs.

In this paper, the PINNs method is used to obtain the experimental data sets to pass over to the sparse optimization part to retrieve the coefficients of the underlying equations. The PINNs method is one type of machine learning method that combines neural networks with physical characteristics to solve differential equations and model physical systems. Hence, unlike ordinary neural networks, the PINNs method is able to train the data and suggest a solution with reasoning. This statement is supported by the paper written by [10]. In the same paper, some common activation functions were listed such as sigmoid, hyperbolic tangent, rectified linear unit (ReLU), Gaussian error linear unit (GELU), etc. Usually, the activation function is commonly assumed to be sufficiently smooth in practice. The selection of an activation function and optimizer are subject to the suitability of the fitting problem and there is no strict guide on which is better. Raissi et al. [11] are some of the pioneers who used the PINNs method to solve forward and inverse problems for partial differential equations. Mishra et al. [12] were also able to demonstrate that the PINNs method could efficiently approximate inverse problems for PDEs. As stated in the paper by Stiasny et al. [13], the PINNs method can universally approximate any continuous function with an arbitrary degree of accuracy. This means that the PINNs method is not only able to ensure that the accuracy of the suggested solution can be achieved, but it also makes sure that the gradients of the equation can be satisfied.

The computation of the gradients of the equation is performed by gradientTape (see [14]), which is

a built-in command in TensorFlow. Sometimes, gradientTape is referred to as an automatic differentiation mechanism. TensorFlow uses a "tape" to record the operations and input values from the forward pass to compute the gradients using backpropagation. After the computation, all objects temporarily stored in the tape will be discarded, and TensorFlow releases any associated resources. The computation involved in automatic differentiation uses the concept of the dual numbers algorithm. Dual numbers encode both the function value and its derivative simultaneously for efficient computation of derivatives through the chain rule.

3.2. Sparse optimization method

Sparse optimization optimizes the solutions to optimization problems while encouraging sparsity in the solution, meaning that the solution is expected to have many zero or close-to-zero elements. This is because the sparse solutions aim to identify only a few parameters or variables that contribute significantly, while the rest are close to zero. In the paper written by Schaeffer [15], it is stated that sparsity plays a key role in optimization and data sciences. In particular, the regularization term, l_1 norm, is often used as a proxy for sparsity. The paper stated that l_1 norm is used to penalize the number of nonzero coefficients in order to promote coefficient sparsity. Therefore, sparse optimization often involves the use of regularization terms to be in use as a "penalty" to fine-tune the magnitude of coefficients or parameters in optimization problems. In the survey paper written by Li et al. [16], they reviewed and made comparisons on some sparse optimization methods used for modeling such as LASSO, elastic net and others. In their paper, sparse modeling is introduced for feature selection. It is emphasized that features with nonzero estimated coefficients are selected. Hence, with the same principle, in this paper, the best coefficients are obtained based on nonzero estimated coefficients.

In situations where the underlying data or parameters have a sparse structure or where simplicity and interpretability of the solution are desirable, sparse optimization appears to be particularly useful. In machine learning, sparse optimization is employed for feature selection. Especially while dealing with massive data (see [17]), sparse optimization can achieve the goal of identification of the most relevant features or variables that contribute to a predictive model. As indicated in their paper, while searching for the relevant patterns, aiming to strike a balance between accuracy and simplicity, sparse optimization could efficiently select the crucial and essential variables, ignoring the less important ones.

Any non-smooth function is non-differentiable and hence has limitations to be solved using well-known optimization methods, such as steepest descent, conjugate gradient, etc. For a convex function, it is not necessary that there exists a local or global minimum point. To tackle similar problems, as stated in the paper written by Huang et al. [18], it is possible to make use of ADMM which could efficiently decompose the complex problem into smaller pieces so that each piece will be easier to handle by speeding up the optimization process. The construction of ADMM involves the minimization of a sum of functions. Each component in the function is possibly subject to some constraints. As described in [19], recently ADMM became a well-known optimization framework for many conventional machine learning problems. As stated, ADMM serves as a gradient-free optimizer that can efficiently overcome the gradient vanishing problem and poor-conditioning optimization problems. In summary, ADMM restructures the constrained objective function into smaller parts. It is believed that by doing so, the smaller parts of the functions will be less complicated and comparatively able to be solved more easily. The technique applied for restructuring the objective function in ADMM uses the augmented

Lagrangian method so that the resulting objective function will be unconstrained and can be solved term by term in the function.

Generally, to illustrate the algorithm, recall that our intention is to uncover the underlying equations. Hence, it is assumed that $f(x)$ denotes the smooth convex function of the underlying equations, which is done by solving $\Lambda^T\phi - u_t = 0$ which is required to be minimized, written as $\underset{x}{\text{minimize}}\ f(x)$. Due to the convexity of the coupled first-order ODE, another function $g(x)$ is introduced to represent the non-smooth part. The objective function has now become $\underset{x}{\text{minimize}}\ f(x) + g(x)$. This construction of the objective function tallies with what has been recommended by Nishihara et al. [20]. In order to avoid confusion of the terms, the objective function can be re-written as

$$\begin{aligned} &\underset{x,\,z}{\text{minimize}} \quad f(x) + g(z), \\ &\text{subject to} \quad h(x - z) = 0. \end{aligned} \tag{3.1}$$

The constraint $h(x, z) = h(x - z) = 0$ is an artificially created constraint. During the process of minimizing the function $f(x)$, an l_1 regularization term is imposed, contributed by the function $g(z) = \lambda\|z\|_1$ to ensure the suggested outcome (a combination of coefficients of the model) is at its simplest form. The constraint, $h(x - z) = 0$, compares the difference between the smooth (x) and non-smooth (z) terms, which are assumed to be approximately zero. Ideally, no dissimilarity between x and z should appear, whereby x and z should match exactly. As suggested by Yuan et al. [21], the formation of Eq (3.1) is a nonconvex sparsity constrained/ sparse-regularized optimization problem, which is difficult to solve. Therefore, by applying ADMM, the constrained optimization problem can be transformed into unconstrained optimization, and hence, the general augmented Lagrangian form of the objective function is:

$$\underset{x,\,z}{\text{minimize}} \quad f(x) + g(z) + y^T h(x,\ z) + \frac{\rho}{2}\|h(x,\ z)\|_2^2. \tag{3.2}$$

Looking at the objective function written in the form of Eq (3.2), the notations $f(x)$ and $g(z)$ are represented in the same context as in Eq (3.1). The term y^T is introduced as the Lagrange multiplier for the constraint $h(x, z)$; whereas ρ is the augmentation parameter controlling the penalty on the constraint violation. Usually, ρ is assigned to the value of one being the natural scaling of the baseline result. Specifically, for the coupled first-order ODE in this paper, the augmented Lagrangian form is:

$$\underset{\Lambda,\,z}{\text{minimize}} \quad \tfrac{1}{2}\|\Lambda^T\phi - u_t\|_2^2 + \lambda\|z\|_1 + \sum_{i,j}(y_{ij}\Lambda^T - y_{ij}z^T) + \frac{\rho}{2}\|\Lambda^T - z^T\|_2^2. \tag{3.3}$$

Referring to Eq (3.3), the dimension of $\Lambda^T\phi - u_t$, z, $y_{ij}\Lambda^T - y_{ij}z^T$ follows $m \times n$. Particularly, the smooth function $\frac{1}{2}\|\Lambda^T\phi - u_t\|_2^2$ usually will be solved by the least squares estimator method. The least squares estimator method minimizes the sum of the squares of the difference between the entries of $\Lambda^T\phi$ and u_t. As for the non-smooth function, it is observed that the non-smooth function $g(z) = \lambda\|z\|_1$ utilizes the l_1 norm which tries to retain z to zero. The accompanied coefficient λ is a penalty set to control how sparse the non-smooth function should be. The larger the value for λ, the higher the sparsity of the outcome will be, leading to more zeros in the coefficients. In [22] on the details of ADMM in optimization, the sparsity can be achieved in the suggested outcome, i.e., the sparse matrix because of the regularization norm 1 term. In order to deal with the non-smooth function, z, the soft thresholding denoted as S, will be introduced to "smoothen" the term $\|z\|_1$. Then, the notation y here

serves as an indicator of minimization on the function $\Lambda^T - z^T$. Since $\Lambda^T - z^T$ approaches zero, the term $\sum_{i,j}(y_{ij}\Lambda^T - y_{ij}z^T)$ will converge to zero. The last term with notation $\Lambda^T - z^T$ in the equation is accompanied by another penalty, ρ. This penalty term controls how strictly the condition of $\Lambda^T - z^T$ should be satisfied. The larger ρ is, the more strict it is to ensure $\Lambda^T - z^T$ must be zero to satisfy the condition.

Solving for Eq (3.3) under sparse optimization involves an iteration update of the terms Λ, z, and y, which is

$$\Lambda_{k+1}^T\left(\phi\phi^T + \rho I\right) = \left(u_t\phi^T - y_{ij}^k + \rho z_k^T\right), \tag{3.4}$$

$$z_{k+1}^T = S\left(\frac{\lambda}{\rho}, \frac{y_{ij}^k}{\rho} + \Lambda_k^T\right), \tag{3.5}$$

$$y_{ij}^{k+1} = y_{ij}^k + \rho\left(\Lambda_{k+1}^T - z_{k+1}^T\right). \tag{3.6}$$

During iteration updates, components Λ, z, and y are updated to suggest the best combinations of coefficients for the coefficient matrix, Λ.

3.3. PINNs-sparse method

As mentioned earlier, with the intention of building the ground truth of the problem, this paper combined the PINNs method and sparse optimization approach to uncover the coefficients of the underlying equations. In short, the combination of methods is named the PINNs-sparse method. The PINNs method will be specifically applied to obtain the analytical solutions of the coupled first-order ODE. First, given an initial-value coupled first-order ODE problem with $\left[\frac{dx}{dt}, \frac{dy}{dt}\right]$, the analytical solutions are estimated using the PINNs method with time t as the input data and u as the output trained solutions in terms of x and y, i.e., $u[x, y]$. Then, the ϕ array is constructed consisting of the guessing of potential variables and is computed numerically, with the trained experimental data x and y.

The data-driven parameter estimation process kickstarts with the PINNs method when the experimental data, $u[x, y]$, are being passed over to the model setting and compilation process to set up the model and obtain the derivative, u_t. The coefficient estimation process starts by alternating the PINNs algorithm and sparse optimization iterations in a loop. Similar to passing u_t, after the initiated Λ^T coefficient matrix with all ones is passed to the sparse optimization part, the process begins with the updates of the Λ^T, z, and y components. Then, the new updated Λ^T coefficient matrix will pass back to the PINNs algorithm to repeat the training process. The PINNs part will always be in charge of updates of trainable variables resulting in the changes in u_t to pass to the sparse optimization part. This alternating process is repeated until the loss value from PINNs and sparse optimization are both at a minimum. In other words, once the Λ coefficient matrix is stable, loss values from both PINNs and sparse optimization remain unchanged at minimum, and then the coefficient estimation process is complete. Ideally, the training process shall not continue when the values of u_t and Λ^T remain unchanged. For checking purposes, the product of the outcome obtained from sparse optimization, Λ^T, and the experimental data array, ϕ, is expected to be approximately equal to u_t, obtained from the PINNs method, i.e., $\Lambda^T\phi = u_t$. Therefore, upon completion of the estimating process, the accuracy of the coefficient

matrix can be determined by finding the mean square error between $\Lambda^T\phi$ and u_t. Summarized below is the step-by-step PINNs-sparse algorithm.

Algorithm 1 PINNs-Sparse Algorithm

Require: Initialization of hyperparameters, variables, and the model architecture setting
Ensure: Data: Analytical solutions, $u[x, y]$, from the PINNs method.

1. Compute the derivatives, u_t, on PINNs and pass over to Sparse. ▷ First PINNs
2. Find the numerical values of u_t and Λ^T ((Eq 3.3)).
3. Solve Eq (3.3) by performing iteration updates on Λ^T (3.4), z (3.5), and y (3.6). ▷ First Sparse
4. Pass back Λ^T from sparse optimization to the PINNs method.
5. Update u_t in PINNs then pass to Sparse. ▷ Second PINNs
6. Repeat Steps 3 – 5 until u_t and Λ^T remain unchanged. ▷ Looping between PINNs and Sparse
7. Computation of the loss function and accuracy checking between $\Lambda^T\phi$ and u_t.
8. Graphical representation based on the estimated coefficient model.

In short, the complexity of neural networks falls under the set up of the model architecture as so far there has been no specific guide on the ideal number of hidden layers needed in the neural-network training process. However, the computation time and the optimization process of the PINNs method are short and efficient. The training process can be done in just a few iterations, as illustrated in the examples in the Results section. Generally, some commonly used optimization methods such as gradient descent and conjugate gradient are not suitable for large-scale, non-smooth convex optimization problems because these methods are sensitive to the step size whereby gradients are not well-defined everywhere causing the solutions to be hard or fail to converge. As a comparison, ADMM in the PINNs-sparse algorithm can excel in large-scale, non-smooth convex optimization problems because it can decompose the smooth and non-smooth components into smaller parts to solve, with the assistance of the augmented Lagrangian formation and l_1 norm regularization introduced.

4. Results

In this section, the derivation of the results in the searching coefficient for a linear coupled first-order ODE using the PINNs-sparse method is demonstrated.

4.1. Linear coupled first-order ODE

The linear coupled first-order ODE illustrated in this paper is

$$\begin{cases} \frac{dx}{dt} = 1 + 0.2x - 0.3y, \\ \frac{dy}{dt} = 2 - 0.4x + 0.5y, \end{cases} \tag{4.1}$$

with initial condition $u[t = 0] = [0, 1]$. From the coupled first-order ODE, it can be observed that this is similar to solving a system of two linear equations. Hence, the dimension of the solution, u, will be 2×1. Letting the dimension of the variable array, ϕ, be 3×1, and by convention, the dimension of the coefficient matrix follows 3×2, then, represented in vector form, the equation is written as $u_t = \Lambda^T\phi$. In this problem, the hyperparameters setting is with seven hidden layers with the number of

neurons being 8, 16, 32, 64, 32, 16, and 8, and the activation function is GELU, running on 100 epochs with a batch size of 4. GELU was selected because it has a significantly smoother gradient transition than the sharp and abrupt ReLU. One of the advantages for both ReLU and GELU activation functions is that both of them do not activate all the neurons at the same time.

4.1.1. Validation of results

The linear coupled first-order ODE is solved using both the numerical method, RK45, and the analytical method, PINNs. The intention of having it solved by the numerical method is to make sure the PINNs method can produce the same solutions. The trained solutions, $u[x, y]$, obtained from the PINNs method are compared with the numerical solutions from RK45. The comparison of the results is shown below. It can be observed from Figure 2 that the solutions trajectory based on the PINNs method is able to nearly overlap with the solutions trajectory of the numerical method, RK45.

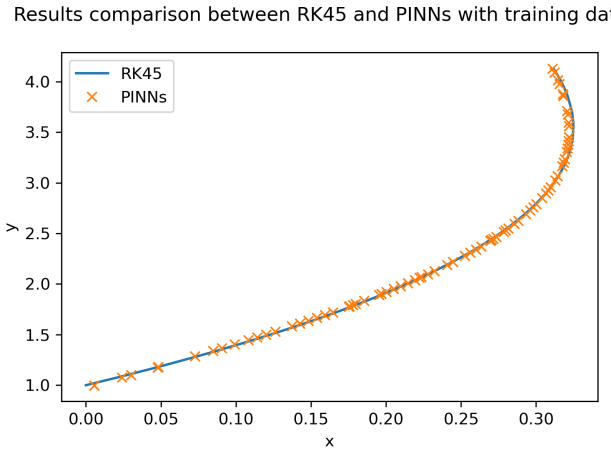

Figure 2. Results between RK45 and PINNs for the linear coupled first-order ODE.

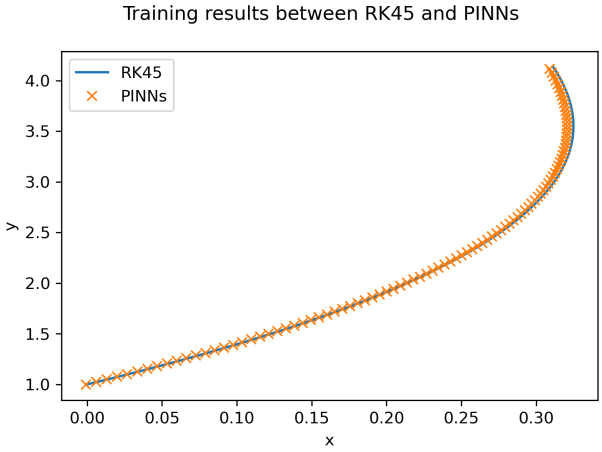

Figure 3. PINNs part training results between RK45 and PINNs for the linear coupled first-order ODE.

Putting aside the coupled first-order ODE, the trained solutions, $u[x, y]$, are referred to in the search for best combinations of coefficients to uncover the underlying equations. The known coupled first-order ODE will not be used for any computation until the end of the coefficient search because it is used as a reference of comparison to verify the performance of the parameter estimation of the PINNs-sparse method.

In this paper, to demonstrate the ability of the PINNs-sparse method in coefficient search, the ϕ array is set to include only the definite potential variable, i.e., $[1\ x\ y]$. The initialization of Λ and y are formed with a matrix of all ones, and then a random number initializes the z matrix. By performing the PINNs-sparse procedure steps listed under Section 3.3, the coefficient search process is stopped after observing unchanged values of u_t and Λ. Figure 3 shows the PINNs part of the PINNs-sparse method, verified by comparison with the trajectory solutions of the numerical method, RK45.

The outcome obtained from the PINNs-sparse method on the suggested parameters for the coupled first-order ODE is shown in Eq (4.2). It is compared with the known linear coupled first-order ODE earlier, shown in Eq (4.1). Observing the coefficients, it can be concluded that they are quite similar. In particular, this set of combinations of coefficients is generated based on 10 iterations, with a switch between PINNs and sparse optimization being the best fit. The learning rate is not applicable in this problem due to setting the maximum likelihood of the function equal to zero, which is optimal.

$$\begin{cases} \frac{dx}{dt} = 0.9993 - 1.1657x - 0.1711y, \\ \frac{dy}{dt} = 1.9588 - 3.1502x + 0.7653y. \end{cases} \tag{4.2}$$

In order to visualize the performance of the suggested parameters for the linear coupled first-order ODE, the estimation of the coefficients obtained from the PINNs-sparse method is substituted to be solved using the numerical method, RK45. The trajectory solutions, based on the estimated coefficients, are compared with the trajectory solutions of the known coupled first-order ODE, solved by the numerical method, RK45. As a graphical result of the outcome of the PINNs-sparse method, shown in Figure 4, the parameters fit well. As an additional remark, the loss value contributed by the PINNs part is 0.4736 whereas the loss value contributed by the sparse part is 0.01011. Hence, the total loss value of the PINNs-sparse method is 0.4837.

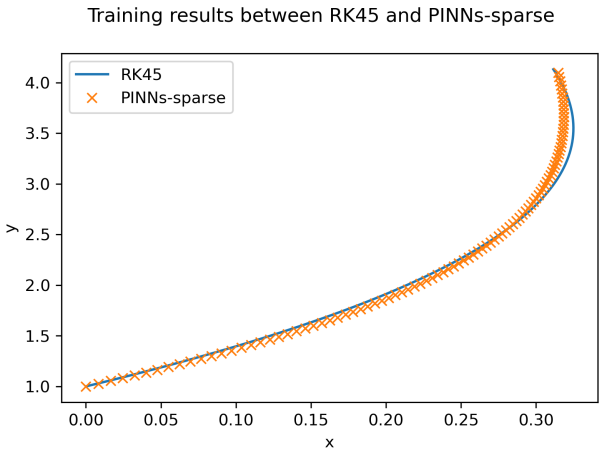

Figure 4. Training results between RK45 and the PINNs-sparse method for the linear coupled first-order ODE.

As a side note, the convergence rate of PINNs in solving direct (4.1) and inverse (4.2) problems is determined based on the indication from the log-loss of PINNs, as shown in Figure 5. It can be observed that in the direct problem, the PINNs method is able to converge after approximately 10 epochs; the PINNs part from the inverse problem (4.2) converges after 2 epochs.

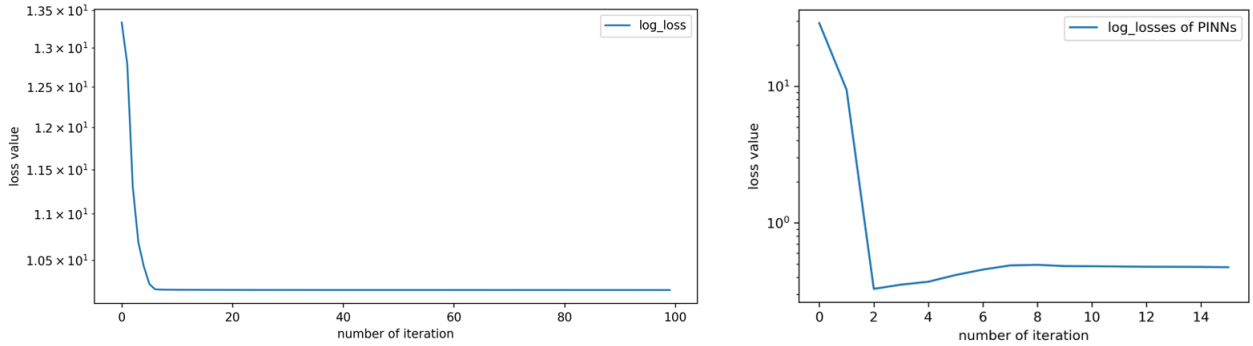

Figure 5. Log-loss of PINNs between Figures 2 and 3.

4.2. Non-linear coupled ODE

The non-linear coupled ODE showcases how the proposed method could uncover the underlying equations of its simplest form, verified by the exact trajectory match. The proposed problem is

$$\begin{cases} \frac{dx}{dt} = 1 + 0.2x^2, \\ \frac{dy}{dt} = 2 - 0.4xy - 0.3y, \end{cases} \tag{4.3}$$

with initial conditions $u[t = 0] = [0, 1]$. In this problem, hyperparameters are set to seven hidden layers with a number of neurons of 8, 16, 32, 64, 32, 16, and 8, and the activation function is GELU, with 100 epochs and a batch size of 4. The optimizer selected is Adam with a default learning rate of 0.001.

4.2.1. Validation of results

With a similar approach, the nonlinear coupled ODE is solved using both the numerical method, RK45, and the analytical method, PINNs. It is verified that the PINNs method works well in nonlinear coupled equations too because both trajectories match exactly, as shown in Figure 6.

Putting aside the equations, the trained solutions, $u[x, y]$, obtained from the PINNs method are referred to in the search for best combinations of coefficients to uncover the underlying equations. The known equations will not be used for any computation until the end of the coefficient search because they are used as a reference of comparison to verify the performance of the parameter estimation of the PINNs-sparse method.

In this problem, to demonstrate the ability of the PINNs-sparse method in coefficient search, the ϕ array is set to include the simplest form of the potential variable, i.e., $[1\ x\ y]$. The initialization of Λ and y are formed with a matrix of all ones, and then a random number initializes the z matrix. By performing the PINNs-sparse procedure steps listed under Section 3.3, the coefficient search process is stopped after observing unchanged values of u_t and Λ. Figure 7 shows the PINNs part of the PINNs-sparse method, verified by comparison with the trajectory solutions of the numerical method, RK45.

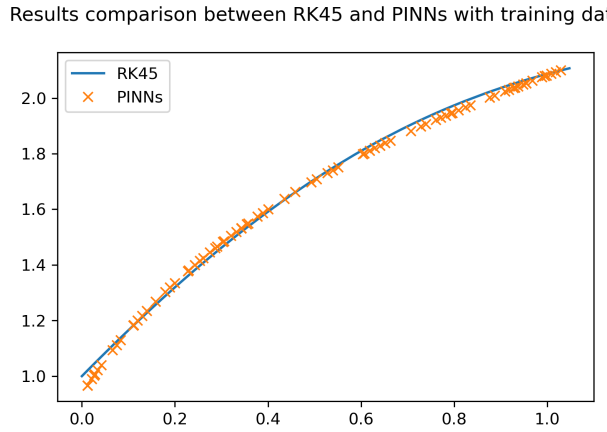

Figure 6. Results between RK45 and PINNs for the non-linear coupled ODE.

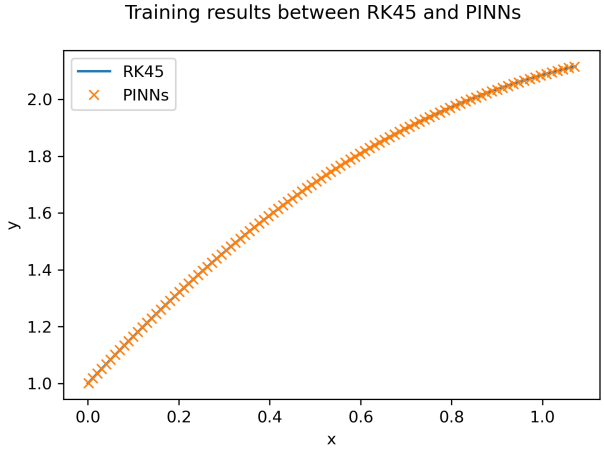

Figure 7. PINNs part training results between RK45 and PINNs for the non-linear coupled ODE.

The outcome obtained from the PINNs-sparse method on the suggested parameters for the coupled first-order ODE is shown in Eq (4.4). It is compared with the known non-linear coupled first-order ODE earlier, shown in Eq (4.3).

$$\begin{cases} \frac{dx}{dt} = -0.4210 - 1.01478x + 1.2057y, \\ \frac{dy}{dt} = 0.8545 - 1.8863x + 0.7413y. \end{cases} \tag{4.4}$$

This set of combinations of coefficients provides an estimation of best fit. Although the terms vary from the original equations, the trajectory for both estimated solutions and the known non-linear coupled first-order ODE fit well, as shown in Figure 8. This is often the case and reasonable because under the situation whereby the underlying equations are not known, the estimated form of the coefficients can be freely in any form. As an additional remark, the loss value contributed by the PINNs part is 0.2636 whereas the loss value contributed by the sparse part is 0.004619. Hence, the total loss value

of the PINNs-sparse method in this case results in value of 0.2682.

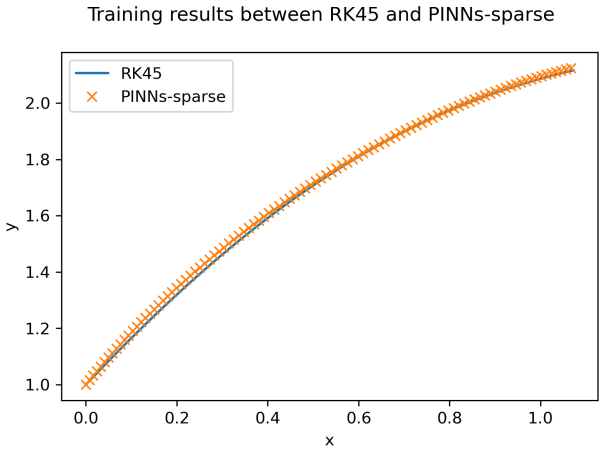

Figure 8. Training results between RK45 and the PINNs-sparse for the non-linear coupled ODE.

Similarly, the convergence rate of PINNs in solving direct (4.3) and inverse (4.4) problems is determined based on the indication from the log-loss of PINNs, as shown in Figure 9. It can be observed that in the direct problem, the PINNs method is able to converge after approximately 2 epochs; the PINNs part from the inverse problem converges after 3 epochs.

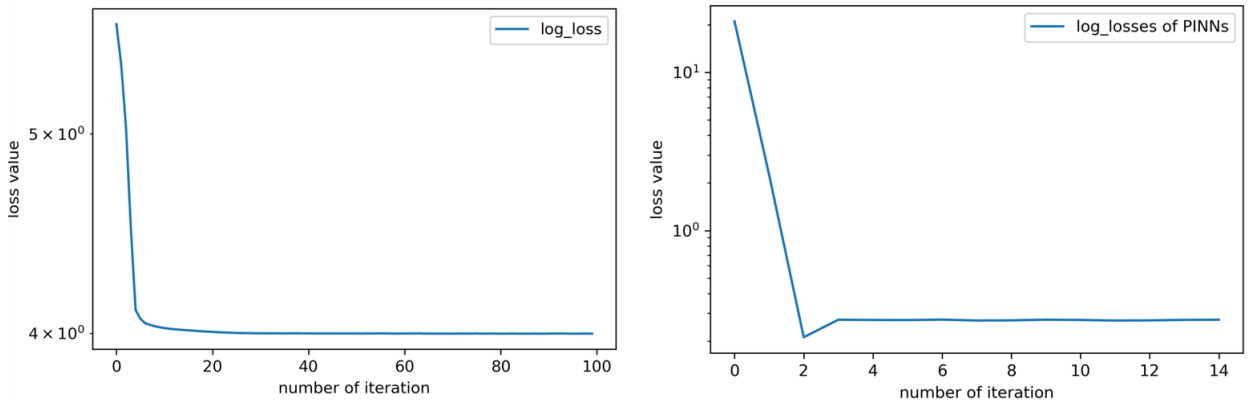

Figure 9. Log-loss of PINNs between Figures 6 and 7.

5. Conclusions and future work

Regarding the comparisons of computational resource consumption and computational accuracy between the PINNs method and any numerical method (here referring to the RK45 method), numerical methods are well-known for their efficiency in computing power and accuracy when dealing with well-defined optimization problems. However, when dealing with situations where equations are unknown and only experimental and derivative data are known, here is where the PINNs method could be applied to estimate the coefficients of the unknown equations. Generally, numerical methods are good at esti-

mating unique solutions with specific initial and boundary conditions. However, when the optimization problem arises with multiple conditions needing to be incorporated, then the PINNs method could be used to optimize the parameter estimation by satisfying all the conditions as well as being able to meet the gradient of the solution trajectory. This is because the PINNs method can naturally incorporate boundary conditions, initial conditions, and any additional physical constraints directly into the loss function with regularization.

As a conclusion, this paper has made use of ADMM to investigate data-driven differential equations, particularly focusing on solving convex optimization problems. The iterative process, involving multiple loops of updating weights and biases with PINNs and employing ADMM for constrained optimization restructuring, underscores the complexity and efficacy of the proposed approach. This paper has provided insights and shown how the novel method, PINNs-sparse, could uncover the governing differential equations. This initiative could provide an alternative option to deal with similar research problems on how to recover the underlying differential equations. This simple approach demonstrates the potential of how the underlying differential equations can be retrieved, solely based on data. During the search for the governing differential equations, optimization techniques are applied to ensure that the combination of coefficients is the most ideal case with minimum error. This could also increase the reliability of the research.

In this paper, we have demonstrated the proposed method for solving linear and non-linear coupled ODEs for direct and inverse problems. In real applications, the linear coupled ODEs have several applications in finance and tomography, particularly in modeling systems where multiple variables influence each other dynamically over time. The research direction is still continuously being carried out to extend the concept of parameter estimation using the PINNs-sparse method in nonlinear or more complex systems. It is believed that this study could contribute to the broader understanding and application of this versatile optimization method in solving complex and decomposable problems across various fields. For example, it is under our radar to make use of the comprehensive collection of machine learning data sets encompassing 15 terabytes of numerical simulations across various spatiotemporal physical systems (see [23]) with the proposed method to model complex physical systems. The beauty of unveiling the hidden equations from given data is encouraging in finding out how the world works and discovering previously unknown aspects of physical phenomena. It is particularly useful when an extreme physical phenomenon comes to our attention, such as a flood, extreme weather, traffic problems, etc., where we could educationally apply the method proposed in this paper in order to explain the scenario with functions or equations.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work is supported and funded by Sunway University under the Kick-Start Grant Scheme (GRTIN-KSGS(02)-DAPS-2022) and the Publication Support Scheme. The authors would also like to thank the reviewers for the constructive feedback provided.

Conflict of interest

The authors declare that they have no competing interests.

References

1. L. Zheng, X. Zhang, *Modelling and Analysis of Modern Fluid Problems*, Elsevier INC., 2017.
2. J. Y. Sia, Y. K. Goh, H. H. Liew, Y. F. Chang, Error analysis of physics-informed neural networks (PINNs) in typical dynamical systems, *J. Fiz. Malays.*, **44** (2023), 10044–10051.
3. H. R. Samuel, L. B. Steven, L. P. Joshua, K. J. Nathan, Data-driven discovery of partial differential equations, *Sci. Adv.*, **3** (2017), e1602614. https://doi.org/10.1126/sciadv.1602614
4. S. Kamyab, Z. Azimifar, R. Sabzi, P. Fieguth, Survey of deep learning methods for inverse problems, preprint, arXiv:2111.04731v2. https://doi.org/10.48550/arXiv.2111.04731
5. Z. Chen, Y. Liu, H. Sun, Physics-informed learning of governing equations from scarce data, *Nat. Commun.*, **12** (2021), 1–13. https://doi.org/10.1038/s41467-021-26434-1
6. J. F. Cai, K. S. Wei, Exploiting the structure effectively and efficiently in low rank matrix recovery, in *Handbook of Numerical Analysis*, **19** (2018), 21–51. https://doi.org/10.1016/bs.hna.2018.09.001
7. A. Haya, Deep learning-based model architecture for time-frequency images analysis, *Int. J. Adv. Comput. Sci. Appl.*, **9** (2018). https://doi.org/10.14569/IJACSA.2018.091268
8. I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE Trans. Neural Networks*, **9** (1998), 987–1000. https://doi.org/10.1109/72.712178
9. A. Ahmad, T. M. Umadevi, J. C. Y. Ng, J. E. Toh, S. Koo, A. A. S. Zailan, A comparison review of optimizers and activation functions for convolutional neural networks, *J. Appl. Technol. Innovation*, **7** (2023), 37–45.
10. Y. Shin, J. Darbon, G. E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, *Commun. Comput. Phys.*, **28** (2020), 2042–2074. https://doi.org/10.4208/cicp.oa-2020-0193
11. 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** (2019), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
12. S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs, *IMA J. Numer. Anal.*, **42** (2021), 981–1022. https://doi.org/10.1093/imanum/drab032
13. J. Stiasny, S. Chevalier, S. Chatzivasileiadis, Learning without Data: Physics-Informed Neural Networks for fast time-domain simulation, in *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, (2021), 438–443. https://doi.org/10.1109/SmartGridComm51999.2021.9631995
14. P. H. W. Hoffmann, A Hitchhiker's guide to automatic differentiation, *Numer. Algorithms*, **72** (2016), 775–811. https://doi.org/10.1007/s11075-015-0067-6

15. H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, *Proc. R. Soc. A: Math., Phys. Eng. Sci.*, **473** (2017). https://doi.org/10.1098/rspa.2016.0446

16. X. Li, Y. Wang, R. Ruiz, A survey on sparse learning models for feature selection, *IEEE Trans. Cybern.*, **52** (2022), 1642–1660. https://doi.org/10.1109/TCYB.2020.2982445

17. K. Kampa, S. Mehta, C. A. Chou, W. A. Chaovalitwongse, T. J. Grabowski, Sparse optimization in feature selection: application in neuroimaging, *J. Global Optim.*, **59** (2014), 439–457. https://doi.org/10.1007/s10898-013-0134-2

18. K. Huang, H. Samani, C. Yang, J. Chen, Alternating direction method of multipliers for convex optimization in machine learning - interpretation and implementation, in *2022 2nd International Conference on Image Processing and Robotics (ICIPRob)*, (2022), 1–5. https://doi.org/10.1109/ICIPRob54042.2022.9798720

19. J. Wang, H. Li, L. Zhao, A convergent ADMM framework for efficient neural network training, preprint, arXiv:2112.11619. https://doi.org/10.48550/arXiv.2112.11619

20. R. Nishihara, L. Lessard, B. Recht, A. Packard, M. Jordan, A general analysis of the convergence of ADMM, in *Proceedings of the 32nd International Conference on Machine Learning*, **37** (2015), 343–352.

21. G. Y. Yuan, S. Li, W. S. Zheng, A block decomposition algorithm for sparse optimization, in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining*, (2020), 275–285. https://doi.org/10.1145/3394486.3403070

22. B. Stephen, P. Neal, E. Chu, P. Borja, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, *Found. Trends Mach. Learn.*, **3** (2020), 1–122. https://doi.org/10.1561/2200000016

23. O. Ruben, M. Michael, M. Lucas, M. Rudy, J. A. Fruzsina, B. Miguel, et al., The Well: a large-scale collection of diverse physics simulations for machine learning, preprint, arXiv:2412.00568. https://doi.org/10.48550/arXiv.2412.00568

AIMS Press

© 2025 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0)