Greedy Sampling for Parameter Estimation in Partial Differential Equations

Ali Forootani* Second Author†

*Address of first author.
Email: f.author@email.com, ORCID: 0000-0000-0000-0000
†Address of second author.
Email: s.author@email.com, ORCID: 0000-0000-0000-0000

Abstract: Partial differential equation parameter estimation is a mathematical and computational process used to estimate the unknown parameters in a partial differential equation model from observational data. This paper employs a greedy sampling approach based on the Discrete Empirical Interpolation Method to identify the most informative samples in a dataset associated with a partial differential equation to estimate its parameters. Greedy samples are used to train a physics-informed neural network architecture which maps the nonlinear relation between spatio-temporal data and the measured values. To prove the impact of greedy samples on the training of the physics-informed neural network for parameter estimation of a partial differential equation, their performance is compared with random samples taken from the given dataset. Our simulation results show that for all considered partial differential equations, greedy samples outperform random samples, i.e., we can estimate parameters with a significantly lower number of samples while simultaneously reducing the relative estimation error. A `Python` package is also prepared to support different phases of the proposed algorithm, including data prepossessing, greedy sampling, neural network training, and comparison.

Keywords: Discrete Empirical Interpolation Method (`DEIM`), Physics Informed Neural Network (`PINN`), Parameter Estimation

Mathematics subject classification: MSC1, MSC2, MSC3

Novelty statement: Within this study, we employ the discrete empirical interpolation method (`DEIM`), specifically the QR-factorization-based variant known as `Q-DEIM`, as a strategic sampling technique for mitigating the computational complexities and time demands associated with parameter estimation for partial differential equations (`PDE`s) using neural networks. Our methodology involves the judicious pre-selection of spatio-temporal data, thereby constructing a reduced dataset for training a neural network to estimate the coefficients of the underlying `PDE` governing the data. We establish that our proposed `Q-DEIM`-based sampling approach not only reduces the required training data for the neural network but also yields a commendable approximation of `PDE` coefficients in fewer training iterations.

1. Introduction

Partial differential equations (`PDE`s) are fundamental mathematical tools used to describe the behavior of physical systems in various scientific and engineering disciplines, such as fluid dynamics,

heat transfer, and quantum mechanics. Accurately identifying the underlying PDE governing a given system is a crucial step in understanding its behavior and making predictions. Traditional methods for identifying PDEs often rely on domain knowledge, mathematical derivations, and experimental data, which can be labor-intensive and may not be applicable in complex or poorly understood systems.

In the literature, most attempts have been made to solve the PDEs analytically or numerically. Analytical methods are based on finding a change of variable to transform the equation into something soluble or on finding an integral form of the solution [1]. These methods are often used for simple PDEs, but they can be difficult to apply to more complex equations. Numerical methods approximate the solution of a PDE by discretizing the domain of the equation and solving a system of algebraic equations, such as the finite difference method (FDM) [2] and the finite element method (FEM) [3]. These methods are more general than their analytical counterparts, but they can be computationally expensive.

Besides solving the PDEs with conventional methods, recent advances in machine learning techniques have proved their potential to address PDE problems in scenarios with limited data. This implies having access solely to the PDE problem data, rather than an extensive set of value pairs for the independent and dependent variables [4]. Taking advantage of modern machine learning software environments has provided automatic differentiation capabilities for functions realized by deep neural networks (DNN) which is a mesh-free approach and can break the curse of dimensionality of the conventional methods [5]. This approach was introduced in [6], where the term physics-informed neural networks (PINNs) was coined.

PINNs combine the expressive power of neural networks with the physical knowledge of governing equations. PINNs can be used to solve a wide variety of problems, including (i) Forward problems: to solve nonlinear PDEs on arbitrary complex-geometry domains [7,8], including fractional differential equations (FDEs) [9], and stochastic differential equations (SDEs [10–13], (ii) Inverse problems: to retrieve the unknown parameters in the PDE [6,7], (iii) Control problems: designing control inputs to achieve a desired state [14], and (iv) Optimization problems: finding the optimal solution to a problem subject to constraints [15].

Unlike the previous literature that focused more on solving PDEs, in this work we consider solving the inverse problem, that is, obtaining the underlying PDEs based solely on time series data. One of the advantages of using PINN for inverse problem is that it requires minimum modifications of the Deep Neural Network (DNN) architecture to recover the PDE coefficients and underlying governing equations of the given time series dataset.

When it comes to the challenge of discovering the underlying PDE governing a given set of spatio-temporal data, choosing an appropriate collocation/sampling strategy is crucial. As the amount of data increases, there is a need for sampling strategies that require fewer collocation points to recover the underlying dynamical system. Indeed, the location and distribution of these sampling points are highly important to the performance of PINNs. In literature, a few simple residual point sampling methods have mainly been employed, among them we can report non-adaptive uniform sampling such as (i) equispaced uniform grid, (ii) uniform random sampling, (iii) Latin hypercube sampling [6], (iv) Halton sequence [16], (v) Hammersley sequence [17], (vi) Sobol sequence [18]; and adaptive non-uniform sampling such as residual-based adaptive distribution (RAD) and residual-based adaptive refinement with distribution (RAR-D) [19]. Even though these methods seem promising, they are highly problem-dependent and are usually tedious and time-consuming. In particular, the case of adaptive sampling such as RAD or RAR-D requires re-sampling of the dataset within the training loop of the Neural Network (NN), which adds significant computational cost to the corresponding algorithm. For further details regarding non-adaptive and residual-based adaptive sampling strategies and their comparisons in PINN training, we refer to the work reported in [19].

In this paper, we make use of the discrete empirical interpolation method (DEIM) as a sampling technique to reduce the computational complexity and time consumption of PDE parameter estimation using neural networks. In a nutshell, our work consists of (a) pre-selecting a portion of the available spatio-temporal data for a PDE and (b) training a neural network using this reduced dataset in order to estimate the coefficients of the underlying PDE governing the data. We demonstrate that our proposed sampling approach not only reduces the data needed to train the neural network but also recovers a good approximation of the PDE coefficients in fewer iterations

of training.

The approach we use to perform the sampling, viz., DEIM was originally proposed in the context of model order reduction to significantly reduce the computational complexity of the popular POD method for constructing reduced-order models for time-dependent and/or parametrized nonlinear PDEs [20]. It is worth highlighting that DEIM is a discrete variant of the empirical interpolation method (EIM) for constructing an approximation of a non-affine parameterized function with a spatial variable defined in a continuous bounded domain with associated error bound on the quality of approximation [21]. In this paper, we particularly employ the QR-factorization-based DEIM procedure, i.e., Q-DEIM, which has a better upper bound error with respect to the original DEIM algorithm and has numerically robust high-performance procedures, already available in software packages such as Python, LAPACK, ScaLAPACK, and MATLAB [22]. More specifically, we methodically investigate the impact of the most informative samples acquired via Q-DEIM on PDE snapshot matrix for estimating the parameters associated with the PDE. We also compare the result of Q-DEIM sampling in PDE parameter estimation with the ones that we computed with the random sampling approach. Our findings prove the significant impact of most informative samples on the training of PINN architecture for the PDE parameter estimation. In our setting, the PINN architecture which takes advantage of Q-DEIM is named greedy sampling based PINN (GS-PINN) while in contrast the one that is fed with random samples will be called random sampling based PINN (RS-PINN). A Python package is prepared to support different implementation phases of GS-PINN and its comparison with RS-PINN on estimating the PDE parameters corresponding to the Allen-Cahn equation, Burgers' equation and Korteweg-de Vries equation.

This article is organised as follows: In Section 2, an overview of PDE parameter estimation will be discussed. We explain our greedy sampling approach and its usage for selecting most informative samples from the dataset corresponding to PDEs in Section 3. In Section 4 we introduce the core algorithm for the PDE parameter estimation. Section 5 is devoted to the simulation and comparison of the GS-PINN with RS-PINN. The paper is concluded in Section 6.

2. An Overview of PDE parameter estimation

In this section, we provide a brief background regarding the PDE parameter estimation problem. Our aim is not to provide an in-depth discussion of this concept but rather to present a brief overview of the concepts narratively.

2.1. Recap on PDE parameters estimation

The estimation of parameters for PDEs holds significance in various domains, including geophysical exploration and medical images [23]. This process is often framed as an optimization challenge constrained by PDEs, typically addressed iteratively through gradient-based optimization techniques [24]. Although significant progress has been made over the last two decades involving high-order schemes for PDEs, automatic differentiation (AD), PDE-constrained optimization, and optimization under uncertainty, parameter estimation in large-scale problem remains a significant challenge [25].

Consider a nonlinear PDE of the form

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f}_p(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x), \tag{1}$$

where $t \in [0, t_{\texttt{max}}]$ is the time variable, $x \in [x_{\texttt{min}}, x_{\texttt{max}}]$ is the space variable, $\mathbf{u}(x,t)$ is the solution, and $\mathbf{f}_p$ is generally a non-linear function of the solution and its derivatives. Since the function $\mathbf{f}_p(\cdot)$ depends on the parameter vector $\mathbf{p} \in \mathbb{R}^k$, the solution $\mathbf{u}(\cdot)$ is also a function of $\mathbf{p}$. Moreover, in this work, we consider that the parameter vector $\mathbf{p}$ is a priori unknown, and we seek to estimate it.

The main assumption behind our parameter estimation is that the function $\mathbf{f}_p(\cdot)$ is composed of only a few terms with respect to a large space of possible contributing terms. For instance, consider the non-linear function appearing in the Allen-Cahn equation

$$\mathbf{f}_p = p_1\mathbf{u} + p_2\mathbf{u}^3 + p_3\mathbf{u}_{xx}, \tag{2}$$

where $\mathbf{p} = [5,\ -5,\ 0.0001]^\top$. Similarly, the non-linear function appearing in the Korteweg-de Vries equation is,

$$\mathbf{f}_p = p_1 \mathbf{u}\mathbf{u}_x + p_2 \mathbf{u}_{xxx}, \tag{3}$$

where $\mathbf{p} = [-6,\ -1]^\top$.

The components of the parameter vector $\mathbf{p}$ can be computed via a least squares minimization formulation. This procedure often requires measuring $\mathbf{u}$ at m different time points, and n spatial locations. Consider that all these measurements are collected in a vector $\mathbf{U} \in \mathbb{R}^{n \cdot m}$. Furthermore, if we assume that the function $\mathbf{f}_p(\cdot)$ have k known linear, nonlinear, and partial derivatives terms, then it is possible to compute the following matrix:

$$\Theta(\mathbf{U}) = \big[\mathbf{1},\ \mathbf{U},\ \mathbf{U}^2,\ \ldots,\ \mathbf{U}_x,\ \mathbf{U}\mathbf{U}_x,\ \ldots\big], \quad \Theta \in \mathbb{R}^{nm \times k}, \tag{4}$$

where each column of the matrix Θ contains all of the values of a particular term that constructs the right-hand side of (1), across all of the $n \cdot m$ space-time grid points where the data is collected. For example, if we consider measurements at 200 spatial locations and 300 time points with $\mathbf{f}$ comprising 5 terms, then $\Theta \in \mathbb{R}^{200 \cdot 300 \times 5}$.

It is straightforward to compute the time derivative of $\mathbf{U}$ denoted by $\mathbf{U}_t$, which is often implemented numerically. Having $\mathbf{U}_t$ and other ingredients we can write the system of equation (1) in the following form:

$$\mathbf{U}_t = \Theta(\mathbf{U})\mathbf{p}, \tag{5}$$

where $\mathbf{p} \in \mathbb{R}^k$ is an unknown parameter vector that has to be computed by a proper algorithm and its elements are coefficients corresponding to each term in the matrix $\Theta(\cdot)$ that describe the evolution of the dynamic system in time. The i^{th} element of vector $\mathbf{p}$ is denoted by p_i.

The last step is to compute the parameter vector $\mathbf{p}$ through a least squares optimization procedure. A classical approach to solve (5) comprise solving a system of linear equations known as the normal equations. This can be concretely expressed in the following fashion:

$$\mathbf{p} = (\Theta^\top \Theta)^{-1} \Theta^\top \mathbf{U}_t, \tag{6}$$

which requires matrix inversion and multiplication of large size and it is not a good idea to explicitly form the inverse of a matrix, especially when dealing with large or ill-conditioned matrices [26]. Therefore, in literature, authors often use regularized least squares (`RLS`) which is a family of methods for solving the least squares problem while using regularization to further constrain the resulting solution. Among `RLS` approaches, we can name `LASSO` [27, 28], ridge regression [29], and elastic net [30]. Note that in this article we make use of `QR` factorization together with stochastic gradient descent technique to solve (5). Indeed, using the stochastic gradient descent algorithm which is a modification of the gradient descent method in the training loop of the `DNN`, will alleviate the complexity of solution of least squares problem (6), since we just use a random small part of our dataset instead of all of them. Hence, working on a portion of the dataset significantly reduces the computational load. We will elaborate on these settings in the upcoming sections.

3. Greedy sampling method for `PDE` parameter estimation

Estimating the parameters of `PDE`s is essential for understanding and predicting real-world phenomena, but it often poses significant challenges due to the inherent complexity and high dimensionality of these equations. One of these challenges is choosing the appropriate number of samples from a measured dataset which has an undisputed importance for the model recovery.

In many real applications such as phase-field modeling or fluid dynamics, the state variables are often measured using sensors. The number of sensors is frequently constrained by physical or economic limitations, and the strategic placement of these sensors is crucial for achieving precise estimates. Regrettably, determining the ideal sensor locations to infer the `PDE` parameters is inherently a combinatorial challenge, and existing approximation algorithms may not consistently produce effective solutions for all relevant cases. The topic of optimal sensor placement is a focus of research interest even in fields such as control theory and signal processing [31]. Generally, five types of sensor placement approaches have been reported in the literature: (i) methods based on

proper orthogonal decomposition (POD) [32], (ii) convex optimization methods [33], (iii) greedy-based algorithms such as Frame-Sens [34], (iv) heuristic approaches such as population-based search [35], and (v) machine learning techniques [36].

Even though these approaches have shown good performance in tackling the curse of dimensionality by choosing the most informative l locations from the assumed n spatial ones, their applicability for the case of PDE parameter estimation is limited due to lack of theoretical analysis (e.g. heuristic methods), lack of simplicity in implementations (machine learning approaches), and having conservative assumptions which might not be true in many cases (e.g. convex optimization methods).

More specifically, we are interested in feeding our most informative samples taken from the dataset into a DNN structure. It is worth highlighting that a DNN architecture has already shown its performance compared to traditional methods for the case of PDE solution and PDE discovery in [37]. Unlike the traditional PDE solvers that focus more on methods such as the FDM [2] and FEM [3], the DNN based approaches (such as PINN) are meshfree and therefore highly flexible. Moreover, DNN has proven to regress along both the spatial and temporal axis for a given sample set [31]. However, as the the number of training samples increase for a DNN, the longer it takes to train the network. To alleviate this situation, it becomes crucial to choose a set of informative samples for training the network, and later, to estimate the parameters of the PDE.

In this article, we make use of Q-DEIM as the sampling approach for a given dataset associated with the PDEs. In this regard, we give a brief introduction about Q-DEIM by using its connection to a popular model order reduction approach named POD. Consider the set of snapshots $\{u_1, \ldots, u_m\} \in \mathbb{R}^n$ and an associated snapshot matrix $\mathcal{U} = [u_1, \ldots, u_m] \in \mathbb{R}^{n\times m}$ that is constructed by measuring the solution at m different time points and n different spatial locations of a PDE. In the conventional POD, we construct an orthogonal basis that can represent the dominant characteristics of the space of expected solutions that is defined as Range $\mathcal{U}$, i.e., the span of the snapshots. We compute the singular value decomposition (SVD) of the snapshot matrix $\mathcal{U}$,

$$\mathcal{U} = \mathbf{Z}\mathbf{\Sigma}\mathbf{Y}^\top, \tag{7}$$

where $\mathbf{Z} \in \mathbb{R}^{n\times n}$, $\mathbf{\Sigma} \in \mathbb{R}^{n\times m}$, and $\mathbf{Y} \in \mathbb{R}^{m\times m}$ with $\mathbf{Z}^\top\mathbf{Z} = \mathbf{I}_n$, $\mathbf{Y}^\top\mathbf{Y} = \mathbf{I}_m$, and $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \cdots, \sigma_z)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_z \geq 0$ and $z = \min\{m, n\}$. The POD will select $\mathbf{V}$ as the leading $\mathtt{r}$ left singular vectors of $\mathbf{U}$ corresponding to the r largest singular values. Using Python-Numpy array notation, we denote this as $\mathbf{V} = \mathbf{Z}[:, :\mathtt{r}]$. The basis selection via POD minimizes $\mathbf{V} := \min_{\Phi\in\mathbb{R}^{n\times r}} \|\mathcal{U} - \Phi\Phi^\top\mathcal{U}\|_F^2$, where $\|\cdot\|_F$ is the Frobenius norm, over all $\Phi \in \mathbb{R}^{n\times r}$ with orthonormal columns. In this regard, we can say $\mathcal{U} = \mathbf{Z}\mathbf{\Sigma}\mathbf{Y}^\top \approx \mathbf{Z}_\mathtt{r}\mathbf{\Sigma}_\mathtt{r}\mathbf{Y}_\mathtt{r}^\top$, where matrices $\mathbf{Z}_\mathtt{r}$ and $\mathbf{Y}_r^\top$ contain the first $\mathtt{r}$ columns of $\mathbf{Z}$ and $\mathbf{Y}^\top$, and $\mathbf{\Sigma}_\mathtt{r}$ contains the first $\mathtt{r}\times\mathtt{r}$ block of $\mathbf{\Sigma}$. More details regarding POD can be found in [38].

Although the reduced-order model lies in the $\mathtt{r}$-dimensional subspace, the conventional POD suffers the issue of lifting to the original space. Hence, in the literature, they tackle this issue with different approaches such as DEIM [20]. An interesting advantage of DEIM is its flexibility to extend its results for the case of nonlinear function approximation beyond model order reduction. Moreover, the performance of the original DEIM algorithm has been improved by using QR-factorization in two aspects: (i) less upper bound error, (ii) simplicity and robustness in implementation.

POD has been used widely to select measurements in the state space that are informative for feature space reconstruction [39]. The method then has been called Q-DEIM. Indeed, we make use of the Q-DEIM algorithm to select a set of most informative samples from a given snapshot matrix $\mathcal{U} \in \mathbb{R}^{n\times m}$ of a PDE for PINN-based model discovery. Q-DEIM takes advantage of the pivoted QR factorization and the SVD, making it a powerful sampling approach. In particular, we consider QR factorization with column pivoting of $\mathbf{Z}_\mathtt{r}^\top$ and $\mathbf{Y}_\mathtt{r}^\top$ to identify the most informative samples for the location and the time in the snapshot matrix $\mathcal{U}$, respectively. The pivoting algorithm gives an approximate greedy solution approach for feature selection which is named submatrix volume maximization since matrix volume is defined as the absolute value of the determinant. QR column pivoting increases the volume of the submatrix constructed from the pivoted columns by choosing a new pivot column with maximal two-norm and then subtracting from every other column its orthogonal projection onto the pivot column. Note that QR-factorization has been implemented and optimized in most scientific computing packages and libraries, such as MATLAB, and Python. Further details about Q-DEIM, its theoretical analysis, and applications can be found in [20, 39].

Algorithm 1: Sample selection based on a two-way `Q-DEIM` procedure

Data: $\mathcal{U}$, $\{x_k\}_{k=1}^{n}$, $\{t_k\}_{k=1}^{m}$, $\epsilon_{\texttt{thr}}$.
Result: $\texttt{ind}_t$, $\texttt{ind}_x$, domain sampled pairs (t_i, x_i) and $u(t_i, x_i)$.

$\texttt{r} = 1$;
$\mathbf{Z}, \mathbf{\Sigma}, \mathbf{Y}^\top \leftarrow \texttt{SVD}(\mathcal{U})$, ▷ computing `SVD` on snapshot matrix $\mathcal{U}$;
Find the lowest $\texttt{r}$, such that $1 - \frac{\sum_{j=1}^{\texttt{r}} \sigma_j}{\sum_{j=1}^{z} \sigma_j} \geq \epsilon_{\texttt{thr}}$;
$\mathbf{Z}_{\texttt{r}} \leftarrow \mathbf{Z}[:, :\texttt{r}]$, $\mathbf{Y}_{\texttt{r}}^\top \leftarrow \mathbf{Y}^\top[:\texttt{r}, :]$, ▷ selecting $\texttt{r}$ dominant left and right singular vectors;
$\texttt{ind}_x \leftarrow \texttt{QR}(\mathbf{Z}_{\texttt{r}}^\top, \text{pivoting} = \texttt{True})$, ▷ storing pivots from pivoted `QR` factorization of $\mathbf{Z}_{\texttt{r}}^\top$;
$\texttt{ind}_t \leftarrow \texttt{QR}(\mathbf{Y}_{\texttt{r}}, \text{pivoting} = \texttt{True})$, ▷ storing pivots from pivoted `QR` factorization of $\mathbf{Y}_{\texttt{r}}^\top$;
$x_i \leftarrow$ from $\texttt{ind}_x$, $t_i \leftarrow$ from $\texttt{ind}_t$, $u(t_i, x_i)$;

3.1. Applying `Q-DEIM` algorithm on `PDE` dataset

We apply the `Q-DEIM` algorithm on the snapshot matrix $\mathcal{U}$ to select the most informative samples in the spatiotemporal grid. To achieve this, we first perform a `SVD` of the snapshot matrix $\mathcal{U}$ and compute matrices $\mathbf{Z}$, $\mathbf{\Sigma}$, and $\mathbf{Y}^\top$. The selection of $\texttt{r}$ leading singular values can be done based on an appropriate precision value $\epsilon_{\texttt{thr}}$, which is related to the underlying dynamical system, and it can be chosen heuristically. $\epsilon_{\texttt{thr}}$ is often referred to as the energy criterion in literature [40]. In particular, the $\texttt{r}$ leading singular values are chosen such that the following quantity is satisfied:

$$1 - \frac{\sum_{j=1}^{\texttt{r}} \sigma_j}{\sum_{k=1}^{z} \sigma_k} < \epsilon_{\texttt{thr}}, \quad \texttt{r} < z, \tag{8}$$

Once the desired precision is achieved, we construct a reduced approximation of the snapshot matrix $\mathcal{U}$ by using the first $\texttt{r}$ columns of matrix $\mathbf{Z}$, the first $\texttt{r}$ singular values contained in the diagonal matrix $\mathbf{\Sigma}$, and the first $\texttt{r}$ rows of $\mathbf{Y}^\top$. Based on `Python` notation, we can represent this as $\mathbf{Z}_{\texttt{r}} = \mathbf{Z}[:, :\texttt{r}]$, $\mathbf{\Sigma}_{\texttt{r}} = \mathbf{\Sigma}[:\texttt{r}, :\texttt{r}]$, and $\mathbf{Y}_{\texttt{r}}^\top = \mathbf{Y}^\top[:\texttt{r}, :]$. To choose the important time and space indices, we apply `QR` decomposition with column pivoting on the reduced order matrices $\mathbf{Y}_{\texttt{r}}^\top$ and $\mathbf{Z}_{\texttt{r}}^\top$, which contain the first $\texttt{r}$ left and right singular vectors. We represent the chosen indices corresponding to the most informative spatio-temporal points in the snapshot matrix $\mathcal{U}$ by $\texttt{ind}_x$ and $\texttt{ind}_t$. To simplify the notation, we denote the pairs of space-time points by (t_i, x_i), and the solution associated to it by u_i. Algorithm 1 summarizes the core part of the `Q-DEIM` sampling approach which takes the snapshot matrix $\mathcal{U}$, spatio-temporal domains x, t and the precision value $\epsilon_{\texttt{thr}}$ as the inputs and returns sampled pairs (t_i, x_i) and corresponding measured value $u(t_i, x_i)$. We show by $\mathcal{N}$ the cardinality of the sampled dataset.

3.1.1. Exploiting locality in time

To capture the local dynamics of the dataset and select better points in the spatiotemporal domain, we decompose the time domain into equal intervals and apply `Q-DEIM` on each sub-domain. We show the number of divisions for the time domain by $\texttt{t}_{\texttt{div}}$. In this setting, the subdomains do not overlap with each other, therefore the total number of selected points is the union of selected samples at each subdomain. For example, by using `Python` notation, if we divide the time domain into three parts, the first part can be written as $\mathcal{U}[:, :\texttt{m}/3]$, the second part as $\mathcal{U}[:, \texttt{m}/3 : 2\texttt{m}/3]$, and finally, the third part as $\mathcal{U}[:, 2\texttt{m}/3 :]$. The reason behind this decomposition is to capture the local dynamics of the `PDE` at each subdomain and sample the most informative portion of the snapshot matrix. More specifically, the system's behavior can vary across various domains, and certain physical attributes may exhibit notable distinctions. For instance, issues related to abrupt features such as shock waves may showcase these differences. On the other hand, by first dividing a sizable domain into smaller sub-domains, and then applying `Q-DEIM` on each sub-domain independently, helps avoid the requirement of complex neural network structures for the `PDE` parameter estimation.

4. Greedy Sampling based Physics Informed Neural Network (GS-PINN) for PDE Parameter Estimation

One notable constraint associated with PINNs pertain to their extensive training costs, which can hinder their performance, particularly when addressing real-world applications that necessitate real-time execution of the PINN model. Consequently, it is vital to enhance the convergence speed of such models without compromising their efficacy. In this section, we propose our algorithm which blends Q-DEIM as the sampling approach, PINN as the function approximation, and QR-factorization for the PDE parameter estimation. In this article, we assume that we a priori know the terms that contribute to the PDE dynamics. However, we do not have any information about their portions that are shown as coefficients, and our goal is to estimate them. For instance, consider the Allen-Chan equation with $\frac{\partial u}{\partial t} = p_1 u + p_2 u^3 + p_3 u_{xx}$ and associated unknown parameter vector $p^\top = [p_1,\ p_2,\ p_3]$.

PINNs are primarily used for solving PDEs [6, 41, 42]. Unlike traditional numerical methods that discretize the domain and solve the equations on a grid, PINN treat the PDE as a constraint in an optimization problem. The neural network is trained to minimize the discrepancy between the predicted and actual values of the PDE, effectively learning the underlying physics. An attractive feature of PINNs is that it requires small modification for the case of PDE parameter estimation [5].

Another notable capability of the PINN is its Automatic Differentiation (AD). To elaborate more on this aspect, it is worth highlighting that numerical differentiation is generally performed by two primary approaches: finite differences or by using a spline interpolation method on the dataset followed by derivative calculation. Finite difference techniques work directly with the available data to compute the derivative. Although finite differences are computationally efficient and easily adaptable to higher dimensions, they are sensitive to noise and require closely spaced data points for accurate results. A more precise and commonly adopted alternative involves the application of spline interpolation for data differentiation. When employing splines for fitting, the data is approximated through a piecewise polynomial with ensured continuity at the boundaries. In practical terms, splines provide more accurate results. However, their scalability to higher dimensions, particularly when employing higher-order splines, is limited [31]. This limitation poses a challenge to model discovery, as it necessitates these higher orders to account for derivatives within the feature library. Using a fifth-order spline to approximate the data essentially translates to approximating the third-order derivative with only a second-order polynomial, thereby restricting its utility in the context of model discovery[43]. While higher-order splines are limited to interpolation in a single dimension, PINN employs a neural network to interpolate along both the spatial and temporal axes. This liberates us from the requirement of using on-grid sampling, and we take advantage of this freedom by developing an alternative sampling technique. A combination of DNN with SINDy based model discovery has been proposed in [37], and several other works extended its results for the case of noisy and scarce dataset by using integration scheme in the training loop [44, 45].

To setup our PINN formulation for the parameter estimation we first write the PDE system of equations (1) in a residual format:

$$\begin{aligned} \mathcal{R}(u) &= -\frac{\partial u}{\partial t} - \mathbf{f}_p(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x) \\ &= -\frac{\partial u}{\partial t} - \mathbf{\Theta}(u)p, \end{aligned} \tag{9}$$

and proceed to approximate $\mathbf{u}(x,t)$ by a DNN. In this regard we define a neural network $\mathcal{G}_\theta(t,x)$ which has two inputs x and t, and one output $\hat{\mathbf{u}}$, i.e. $\hat{\mathbf{u}} = \mathcal{G}_\theta(t,x)$. Moreover, we harness the capabilities of DNN, in particular, automatic differentiation (AD) to compute time derivative $\frac{\partial \hat{\mathbf{u}}}{\partial t}$ with respect to its input variables, i.e. space x, and time t. Having these ingredients we define the following loss function:

$$\mathcal{L} = \mu_1 \mathcal{L}_{\texttt{MSE}} + \mu_2 \mathcal{L}_{\texttt{deri}}, \quad \mu_1, \mu_2 \in (0, 1], \tag{10}$$

where $\mathcal{L}_{\texttt{MSE}}$ is the mean square error (MSE) of the output of the DNN $\mathcal{G}_\theta$ (denoted by $\hat{\mathbf{u}}$) with respect to its input sampled domain pairs (t_i, x_i), and corresponding $u(t_i, x_i)$. The positive constants $\{\mu_1, \mu_2\}$ determine the weight of different losses in the total loss function. It is given as

$$\mathcal{L}_{\texttt{MSE}} = \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \left\| \mathbf{u}(t_i, x_i) - \hat{\mathbf{u}}(t_i, x_i) \right\|_2^2. \tag{11}$$

$\mathcal{L}_{\texttt{MSE}}$ forces the DNN to produce output in the vicinity of the measurements, and μ_1 is its weight. $\mathcal{L}_{\texttt{deri}}$ is the residual (9) and aims to compute the parameter vector p. It is computed as follows:

$$\mathcal{L}_{\texttt{deri}} = \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \left\| \frac{\partial \hat{\mathbf{u}}(t_i, x_i)}{\partial t_i} - \boldsymbol{\Theta}\big(\hat{\mathbf{u}}(t_i, x_i)\big)\hat{p} \right\|_2^2, \tag{12}$$

where $\hat{p}$ is replaced by computing the QR-factorization of the matrix $\boldsymbol{\Theta}$ and its pseudo-inverse as follows

$$\begin{aligned} \mathbf{QR} &= \boldsymbol{\Theta}(\hat{\mathbf{U}}), \\ \hat{p} &= \mathbf{R}^{-1}\mathbf{Q}^{\top}\hat{\mathbf{U}}_t, \end{aligned} \tag{13}$$

here $\hat{\mathbf{U}}$ and $\hat{\mathbf{U}}_t$ are column vectors with components $\hat{\mathbf{u}}(t_i, x_i)$ and $\frac{\partial \hat{\mathbf{u}}(t_i, x_i)}{\partial t_i}$, respectively. It is worth highlighting that the parameter vector p will be updated alongside the weights and biases of the DNN, and the matrix Θ is calculated by (4). Algorithm 2 summarizes the procedure that has been explained in this section.

Algorithm 2: Greedy Sampling based Physics Informed Neural Network (GS-PINN) for PDE Parameter Estimation

Data: $\mathcal{U}$, x, t, $\epsilon_{\texttt{thr}}$ for the Q-DEIM algorithm, matrix $\boldsymbol{\Theta}$, a neural network $\mathcal{G}_\theta$ (parameterized by θ), maximum iterations max-iter, and parameters $\{\mu_1, \mu_2\}$.

Result: Estimated parameter vector p, defining governing equations

1 (t_i, x_i), $u(t_i, x_i) \leftarrow$ Apply Q-DEIM($\mathcal{U}$) based on Algorithm 1 ▷ selecting most informative samples ;

2 Initialize the DNN module parameters, and the parameter vector p;

3 $k = 1$;

4 **while** $k <$ *max-iter* **do**

5 Feed the domain pairs (t_i, x_i) as an input to the DNN ($\mathcal{G}_\theta$) and predict output $\hat{\mathbf{u}}(t_i, x_i)$;

6 Compute the derivative information $\frac{\partial \hat{\mathbf{u}}(t_i, x_i)}{\partial t_i}$ using automatic differentiation ;

7 Compute the cost function (10) ;

8 Update the parameters of DNN (θ) and the parameter vector p using gradient descent ;

5. Simulation Results

In this section we provide several simulation examples for the case of PDE parameter estimation with our proposed GS-PINN PDE estimator. We also provide a comparison for the case where the spatio-temporal dataset is randomly sampled. The simulation examples have different level of complexity and non-linearity. To have a quantitative comparison and evaluate the performance of GS-PINN versus random sampling, we consider the following relative error for each element of parameter vector p_i:

$$\texttt{error}_{p_i} = \frac{\left| p_i^{\texttt{truth}} - p_i^{\texttt{est}} \right|}{\left| p_i^{\texttt{truth}} \right|} \tag{14}$$

where $|\cdot|$ denotes the absolute value; $p_i^{\texttt{truth}}$ and $p_i^{\texttt{est}}$ are the true model coefficient and the estimated coefficient, corresponding to the coefficient p_i, respectively. The main point to define such criteria is the fact that in some PDEs the scale of coefficients for different terms are not in the same range (such as Allen-Cahn equation with $\frac{\partial u}{\partial t} = 0.0001u_{xx} - 5u^3 + 5u$), therefore it is logical to quantify the errors for each element of the parameter vector separately.

In this article we examine the parameter estimations for three PDEs: (i) the Allen-Cahn equation, (ii) the Burgers' equation, and (iii) the Korteweg-de Vries (KdV) equation. We recall that in our setting we do not consider the boundary conditions of PDEs, since our goal is not solving them, rather we desire to estimate their parameters. For each example, the simulation is divided into two parts. In the first part, we employ Q-DEIM on the given snapshot matrix for a PDE to identify the

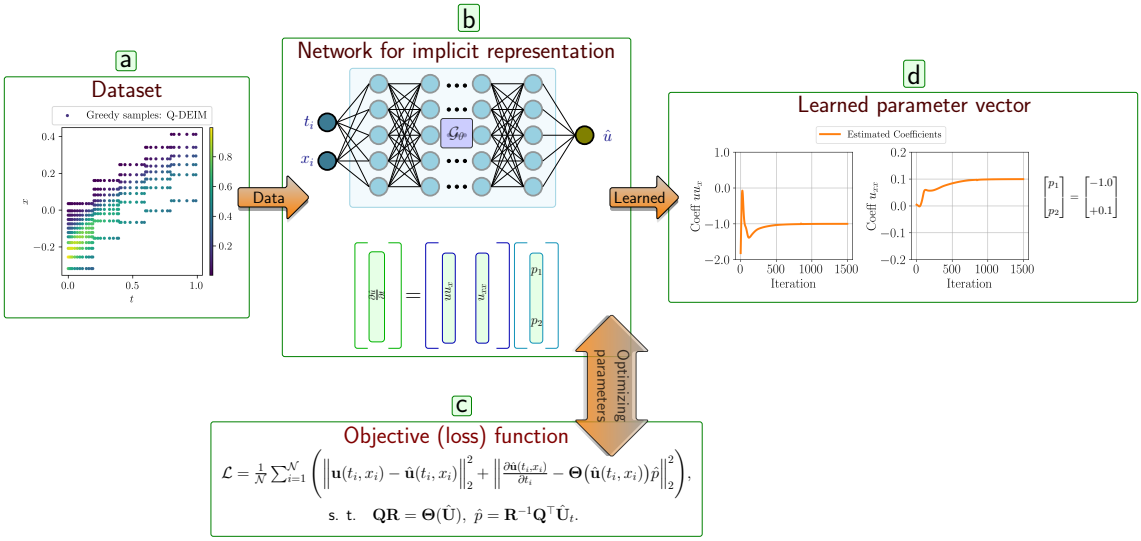

Figure 1: A schematic diagram of GS-PINN for PDE parameter estimation. (a) sampling PDE dataset with Q-DEIM algorithm, (b) feeding the pairs (t_i, x_i) time and space resulted from Q-DEIM into DNN, (c) using the output of the DNN as the function approximator, construct matrix $\mathcal{U}$, (d) estimating the parameters of the DNN with stochastic gradient descent and parameter vector p by considering the loss function $\mathcal{L}$.

most informative samples. This step requires to have a trade off between the number of divisions in time domain $\mathtt{t_{div}}$ and precision value $\epsilon_{\mathtt{thr}}$, since we can manipulate the cardinality of our sample set. Then we apply Algorithm 2 to estimate the parameters of the PDE. In the second part, we do a comparison between Q-DEIM sampling and random sampling approach based on the same sample size. To identify the minimum and maximum number of samples for the comparison, in case of the Q-DEIM algorithm, the time domain division is considered to have integer values $\mathtt{t_{div}} = 1, \ldots, 4$ and we choose the precision value $\epsilon_{\mathtt{thr}}$ to be equally spaced logarithmic values, spanning the range from a minimum value to maximum value, e.g., 20 equally spaced logarithmic values, spanning the range from 10^{-10} to 10^{-2}. This approach is commonly used in scientific and computational applications, such as when specifying tolerance levels for numerical calculations or for testing a range of values in a logarithmic scale. Having the collection of samples and for each pair $(\mathtt{t_{div}},\ \epsilon_{\mathtt{thr}})$ we apply our GS-PINN Algorithm 2 for PDE parameter estimation. By doing so it reveals the effect of time domain division $\mathtt{t_{div}}$, the precision value $\epsilon_{\mathtt{thr}}$ and the sample size on the GS-PINN performance for parameter estimation. To choose the sample size for the simulation of random sampling approach we make use of sample range computed via Q-DEIM algorithm for pairs $(\mathtt{t_{div}},\ \epsilon_{\mathtt{thr}})$ as explained earlier for each PDE. In the sense that we select the minimum and the maximum value among all sample sizes that we identified by Q-DEIM algorithm for pairs $(\mathtt{t_{div}},\ \epsilon_{\mathtt{thr}})$ and we start from the minimum sample size and we reach to the maximum value with 10 linearly spaced steps. For each sample size we pick samples from PDE snapshot matrix randomly and we employ our PINN setup. We plan to conduct simulations 5 times for each sample size without specifying a random seed, resulting in a total of 55 experiments. For the sake of simplicity the PINN architecture which make use of random sampling is named Random Sampling PINN (RS-PINN). The details of each simulation example will be provided accordingly.

Data generation. We acquired our dataset associated to each PDE from the repository provided by the works reported in [37,46]. Moreover, we perform a data-processing step before feeding them to the Q-DEIM and DNN for both time t and the space x domains. In particular, we map the time domain t into the interval $[0,\ 1]$ and the space domain x into the interval $[-1,\ 1]$. For each PDE we also mention the original range of the space-time domain dataset.

Architecture. We utilize multi-layer perceptron networks with periodic activation functions, particularly embracing the `SIREN` architecture as introduced by [47]. This approach allows us to obtain an implicit representation from measurement data, and we will customize the number of hidden layers and neurons for each specific example. Our `DNN` architecture for all the examples have 3 hidden layers, each having 128 neurons.

Hardware. In our neural network training and parameter estimation efforts to uncover governing equations, we employed an NVIDIA®RTX A4000 GPU with 16 GB RAM. For CPU-intensive tasks such as data generation, we harnessed the power of a 12th Gen Intel® Core™i5-12600K processor equipped with 32 GB RAM.

Training set-up. We use the Adam optimizer with $\texttt{learning\_rate} = 10^{-5}$ to update the parameter vector p that is trained alongside the `DNN` parameters [48]. In particular we employ `CyclicLR`, which is a learning rate scheduling technique commonly use in deep learning with the `PyTorch` framework [49]. It allows for dynamic adjustments of the learning rate during training to potentially improve the training process. The key parameters include $\texttt{base\_lr} = 0.1 \times \texttt{learning\_rate}$ and $\texttt{max\_lr} = 10 \times \texttt{learning\_rate}$, which define the lower and upper bounds of the learning rate respectively, `cycle_momentum` to control cycling of momentum, and `mode` to specify the learning rate cycling strategy. In our setting, it employs the `exp_range` mode, which means the learning rate oscillates exponentially between the specified lower and upper bounds over a set number of iterations as determined by $\texttt{step\_size\_up} = 1000$. This technique can be effective in training deep neural networks by promoting faster convergence and potentially helping to escape local minima during optimization. The positive constant parameters $\mu_{1,2}$ are considered to have equal value, i.e. $\mu_{1,2} = 1$. Number of time domain division ($\texttt{t}_{\texttt{div}}$) to apply `Q-DEIM` for each `PDE` dataset, precision value ($\epsilon_{\texttt{thr}}$), and maximum number of iterations (`max-iter`) will be mentioned separately for each example.

5.1. Allen-Cahn equation

The Allen-Cahn equation lies at the heart of modelling the transformation of a physical quantity, often called the order parameter, as a material undergoes a transition from one phase to another. This equation elegantly encapsulates phenomena such as solidification, crystal growth and the emergence of domain patterns in magnetic materials. It belongs to the broader class of reaction-diffusion equations, which relate diffusion processes to a double-well potential energy function. Consequently, the equation encapsulates the inherent drive of the system to minimise its free energy, leading to the formation of domains with distinctive properties. These domains are characterised by smooth variations in the order parameter, with sharp transitions occurring at domain walls. Their versatility extends to a wide range of applications, including the study of grain boundaries in materials science, the modelling of phase separation in binary fluids, and the elucidation of the intricacies of pattern formation in neural networks in neuroscience. The paramount importance of understanding phase transitions and the intricate development of complex patterns in complex patterns in physical systems underlines the invaluable role played by the Allen-Cahn equation [50]. I confirmed your modifications up to here

Let us extract the Allen-Cahn equation by using Ginzburg-Landau free energy [51]

$$\mathcal{F} = \int_w \frac{\gamma_1}{2} \left|\nabla u\right|^2 + \frac{\gamma_2}{4}(u^2 - 1)\, dx, \tag{15}$$

where γ_1 and γ_2 are parameters. By computing L^2 gradient flow, we obtain the Allen-Cahn equation

$$\begin{aligned} &\frac{\partial u}{\partial t} + \gamma_1\ \Delta u + \gamma_2\ (u - u^3) = 0, \\ &\frac{\partial u}{\partial t} = -\gamma_1\ u_{xx} - \gamma_2\ (u - u^3). \end{aligned} \tag{16}$$

We employed `GS-PINN` for `PDE` parameter estimator on a dataset corresponding to Allen-Cahn equation with periodic boundary conditions in one dimension with nominal parameters $\gamma_1 = 0.0001$,

and $\gamma_2 = 5$. In this example the original dataset is taken from the work reported in [51]. The space domain x and time domain t were already mapped into the interval $x \in [-1,\ 1]$ and $t \in [0,\ 1]$ and were discretized into $n = 512$ locations and $m = 201$ points, respectively. Therefore the snapshot matrix $\mathcal{U} \in \mathbb{R}^{n \times m}$ has 102912 elements which represents the curse of dimensionality of training the PINN architecture for the PDE parameter estimation. The snapshot matrix of the original dataset is shown in the Figure 2 (left) which represents the process of phase separation in time. To apply Q-DEIM algorithm we set the time domain division $\mathtt{t_{div}} = 2$, precision value $\epsilon_{\mathtt{thr}} = 10^{-8}$. In the Figure 2 (right) we see the most informative samples that are selected with Q-DEIM algorithm. In total, 394 samples are selected to be utilized in PINN parameter estimation with maximum number of iterations max-iter $= 1500$. The trajectory of the coefficients estimated by GS-PINN are depicted in Figure 3. The parameter vector is $p = [4.95, -4.95, -0.0003]^\top$ which based on (16) $\gamma_1 = -p_3$ and $\gamma_2 = -p_1 = -p_2$. We observe that with having 0.383% of the entire dataset (i.e. $\frac{394}{102912} \approx 0.00383 \times 100 = 0.383\%$) we earned acceptable precision. We notice that GS-PINN can not estimate the parameter γ_1 properly since its value is considerably less than γ_2 in the nominal equation.

In the second scenario, i.e. comparing GS-PINN with RS-PINN, we choose 20 equally spaced logarithmic values for precision $\epsilon_{\mathtt{thr}}$, spanning the range from 10^{-13} to 10^{-4} corresponding to each time division $\mathtt{t_{div}} = 1,\ 2,\ 3,\ 4$. The minimum number of samples corresponds to the pair $(\mathtt{t_{div}},\ \epsilon_{\mathtt{thr}}) = (3,\ 10^{-4})$ with 86 samples, while the maximum number of samples corresponds to $(\mathtt{t_{div}},\ \epsilon_{\mathtt{thr}}) = (1,\ 10^{-13})$ with 1444 samples. The performance of GS-PINN for each pair $(\mathtt{t_{div}},\ \epsilon_{\mathtt{thr}})$ is shown in the figure Figure 4. The first notable problem that we recognize is that the GS-PINN does not have a good estimation about coefficient corresponding to the term u_{xx}, however GS-PINN is able to recover the coefficients corresponding to the terms u as well as u^3. As we can see $\mathtt{t_{div}} = 1$ has the worst performance with both highest relative error and highest sample range. In the sense that for the fixed relative error $\mathtt{t_{div}} = 1$ almost requires more samples compare to the $\mathtt{t_{div}} = 2,\ 3,\ 4$. This result reveals the necessity of finding the suitable value for $\mathtt{t_{div}}$ to have a less relative error in PDE parameter estimation with GS-PINN and therefore better performance. Moreover, we see that for the case of $\mathtt{t_{div}} = 3$ and $\mathtt{t_{div}} = 4$ the sample range and error values are almost the same and it is hard to distinguish which one performs better. This shows that with a fixed precision value $\epsilon_{\mathtt{thr}}$ increasing the time division $\mathtt{t_{div}}$ has no sensible effect on the quality of the estimation for Allen-Cahn PDE equation. It is worth to highlight that the minimum relative error corresponds to the pair $(\mathtt{t_{div}}, \epsilon_{\mathtt{thr}}) = (3,\ 1.833 \times 10^{-9})$ with 457 samples.

Having the sample range computed by GS-PINN we are ready to evaluate the performance of RS-PINN to recover the PDE coefficients. The result of this comparison between GS-PINN and RS-PINN is shown in the Figure 5. For the sake of simplicity in the plots related to the performance of RS-PINN, we take the average value of the relative errors for a given fixed number of samples associated to 5 experiments. Since the range of samples is related to the time division $\mathtt{t_{div}}$, for GS-PINN we employ K-means clustering with $k = 20$, see Appendix A and [52], to find the nearest cluster centroid corresponding to each pair sample size and relative error. This significantly simplifies the evaluation of GS-PINN and RS-PINN under different configurations. These curves are shown in the Figure 5 for both GS-PINN (dashed line) and RS-PINN (solid line). We see that RS-PINN can not estimate the coefficient corresponding to the term u_{xx}, however it has lower relative error with respect to GS-PINN. This is due to the fact that the DNN is not capable to reach a small precision value in the range 0.0001 which coefficient corresponding to the term u_{xx} is. Regarding the estimated values for the terms u and u^3, we see that GS-PINN outperforms RS-PINN in all the settings except in one case.

5.2. Burgers' equation

Burgers' equation captures the dynamics of a quantity, typically denoted as the velocity or density, within a medium characterized by convection and diffusion processes. It represents a crucial intermediary between the simpler linear convection-diffusion equation and the more complex nonlinear conservation equations. The equation is characterized by its capacity to describe a multitude of phenomena, including the formation of shocks, rarefaction waves, and the emergence of complex wave patterns. Within the realm of fluid dynamics, Burgers' equation finds application in modeling traffic flow, the dynamics of shock waves in compressible fluids, and the behavior of

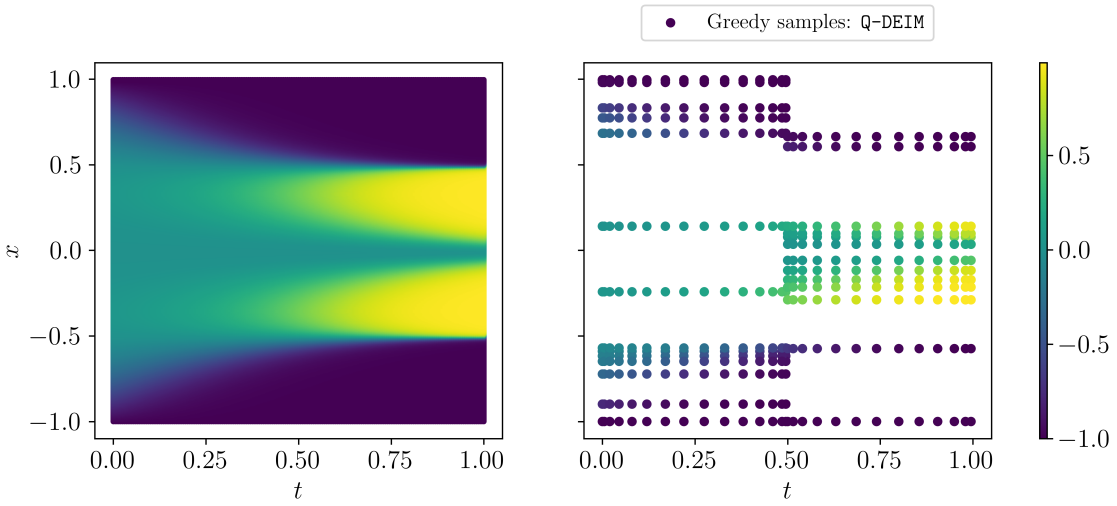

Figure 2: (left) Entire dataset; (right) Greedy samples by `Q-DEIM` algorithm for Allen-Cahn equation (Section 5.1)

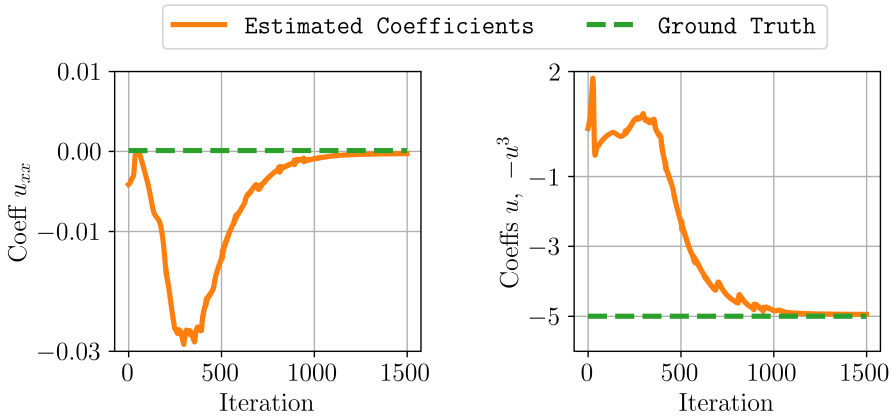

Figure 3: Estimated coefficients by `GS-PINN` for Allen-Cahn equation (Section 5.1)

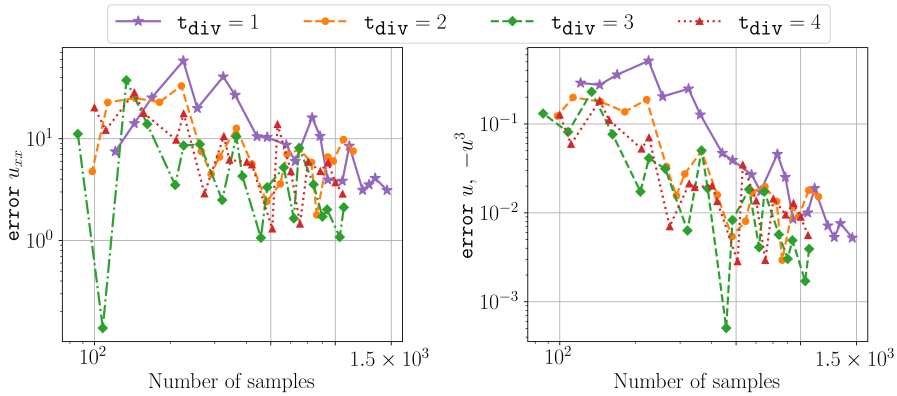

Figure 4: Performance of `GS-PINN` to estimate coefficients corresponding to Allen-Cahn `PDE` equation (Section 5.1) with different time divisions t_{div} and precision values ϵ_{thr}

dispersive waves in shallow water. Furthermore, it has significance in other fields such as plasma physics, acoustics, and the study of nonlinear optical systems [53]. The one-dimensional form of

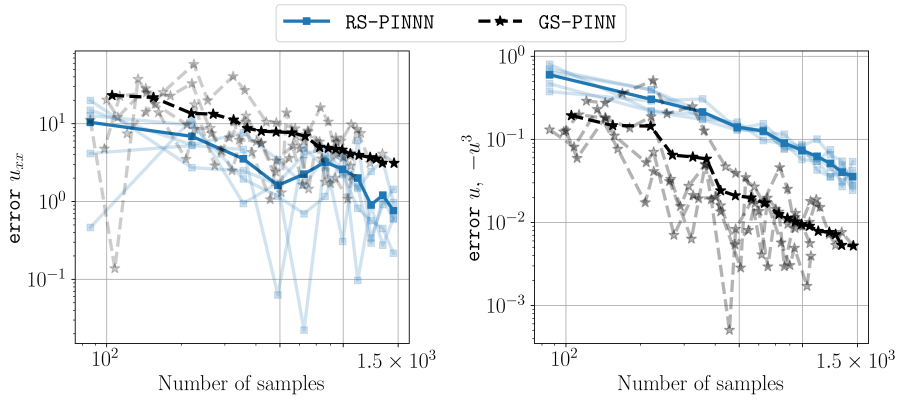

Figure 5: Comparing GS-PINN with RS-PINN to estimate coefficients corresponding to Allen-Cahn PDE equation (Section 5.1) with different time divisions $\mathtt{t_{div}}$ and precision values $\epsilon_{\mathtt{thr}}$

the Burgers' Equation is typically written as:

$$\frac{\partial u}{\partial t} + \lambda\, uu_x + \nu\, u_{xx} = 0. \tag{17}$$

In order to study the Burgers' equation, as described in [46], it is discretized over the domain $x \in [-8, 8]$ into $n = 256$ points and in a time interval $t \in [0, 10]$ with $m = 101$ time steps.

Hence, the snapshot matrix $\mathcal{U}$, with dimensions $n \times m$, comprises a total of $25,856$ elements, highlighting the challenge posed by the curse of dimensionality when training the PINN architecture for PDE parameter estimation. The left side of Figure 6 displays the snapshot matrix of the original dataset, illustrating the Burgers' equation dynamic. To employ the Q-DEIM algorithm, we specify a time domain division of $\mathtt{t_{div}} = 5$ and a precision threshold value of $\epsilon_{\mathtt{thr}} = 10^{-6}$.

In Figure 6 on the right-hand side, we observe the selection of the most informative samples accomplished using the Q-DEIM algorithm. A total of 359 samples have been chosen for utilization in GS-PINN parameter estimation, with a maximum iteration set at max-iter = 1500. The evolution of the coefficients estimated by the GS-PINN method is illustrated in Figure 7. The computed parameter vector p after 1500 iteration is $p^\top = [-1.000,\ 0.0996]$ where based on (17) we have $\lambda = -p_1$ and $\nu = -p_2$. We see that with approximately 1.39% of the entire dataset we can recover Burgers' PDE parameters.

In Figure 8 the relative error corresponding to different pairs $(\mathtt{t_{div}}, \epsilon_{\mathtt{thr}})$ is shown. We select 20 equally spaced logarithmic values for precision $\epsilon_{\mathtt{thr}}$, spanning the range from 10^{-10} to 10^{-2} corresponding to each time division $\mathtt{t_{div}} = 1,\ 2,\ 3,\ 4$. The minimum number of samples corresponds to the case of $\mathtt{t_{div}} = 3$ with associated sample size 50. We see that $\mathtt{t_{div}} = 1$ has the worst performance in the interval with less number of samples, however it improves with more number of samples and in some cases outperforms the others. Moreover, in the interval corresponding to higher sample size manipulating $\mathtt{t_{div}}$ has no sensible effect on the performance of GS-PINN ; this is more evident for the relative error corresponding to the term uu_x in Figure 8. The lowest relative error associated to the term u_{xx} corresponds to the pair $(\mathtt{t_{div}},\ \epsilon_{\mathtt{thr}}) = (2,\ 2 \times 10^{-4})$, while for the term uu_x lowest relative error corresponds to the pair $(\mathtt{t_{div}},\ \epsilon_{\mathtt{thr}}) = (4,\ 1.13 \times 10^{-05})$.

With the sample range calculated using GS-PINN, we are now prepared to assess how well RS-PINN performs in the recovery of PDE coefficients. The outcome of this comparison between GS-PINN and RS-PINN can be seen in Figure 9. To simplify the plots related to the performance of RS-PINN, we take the average value of the relative errors across five experiments for a fixed number of samples.

Like the previous example, as the sample range is linked to the time division parameter, denoted as $\mathtt{t_{div}}$, we employ K-means clustering with $k = 20$, within the context of GS-PINN. This clustering technique is utilized to determine the nearest cluster centroid corresponding to each pair of sample size and relative error. This approach greatly simplifies the assessment of GS-PINN and RS-PINN performance across various configurations. The resulting curves, illustrated in Figure 9, represent GS-PINN with dashed lines and RS-PINN with solid lines. It is obvious that GS-PINN out-

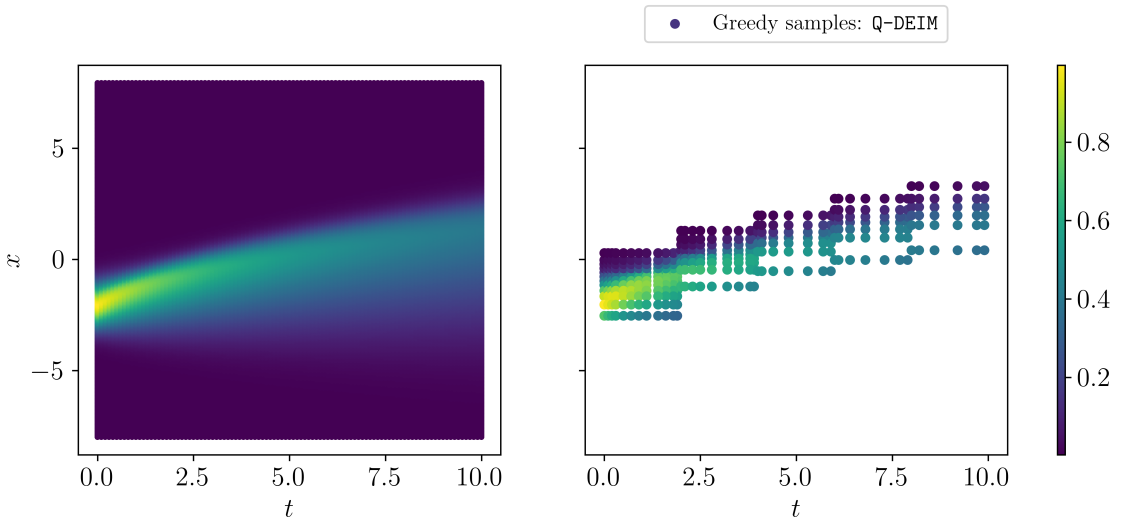

Figure 6: (left) Entire dataset ; (right) Greedy samples by `Q-DEIM` algorithm for Burger's equation (Section 5.2)

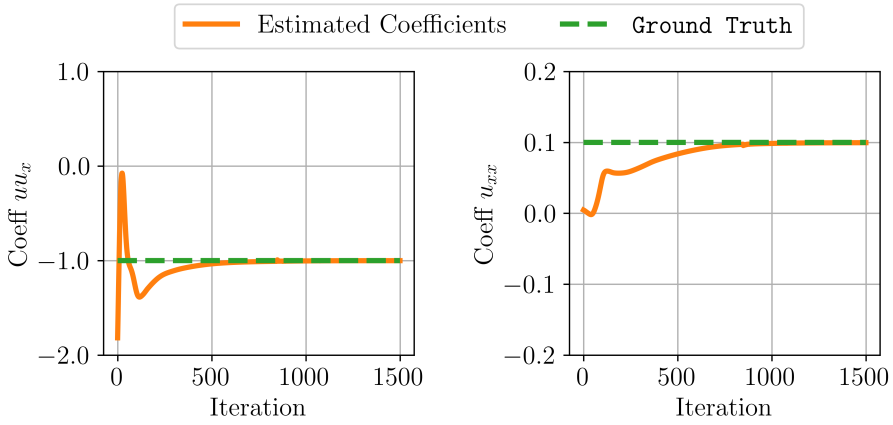

Figure 7: Estimated coefficients by `GS-PINN` for Burger's equation (Section 5.2)

performs `RS-PINN` almost in all the settings except for some cases that has `RS-PINN` has better or comparable performance which corresponds to the higher sample sizes specifically associated to the term uu_x.

5.3. Korteweg-de Vries (KdV) equation

The Korteweg-de Vries (KdV) equation is a fundamental partial differential equation in the field of mathematical physics and nonlinear waves. The KdV equation is particularly significant because it provides a mathematical description of certain types of nonlinear waves in various physical systems, including shallow water waves, plasma physics, and optics. The KdV equation can be written as a partial differential equation in the form:

$$\frac{\partial u}{\partial t} + c\ uu_x + \alpha\ u_{xxx} = 0, \tag{18}$$

where c and α are constant variables. It is known for its ability to model waves that maintain their shape while traveling through a dispersive medium, such as water with varying depths, and it can describe both solitons (solitary wave solutions) and other types of nonlinear wave phenomena. The discretization for the KdV equation, as described in [46], is performed at 512 space points

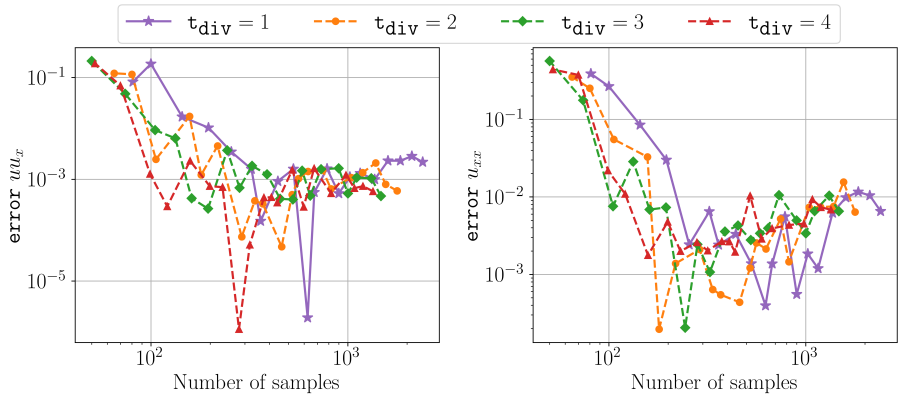

Figure 8: Performance of `GS-PINN` to estimate coefficients corresponding to Burger's `PDE` equation (Section 5.2) with different time divisions $\mathtt{t_{div}}$ and precision values $\epsilon_{\mathtt{thr}}$

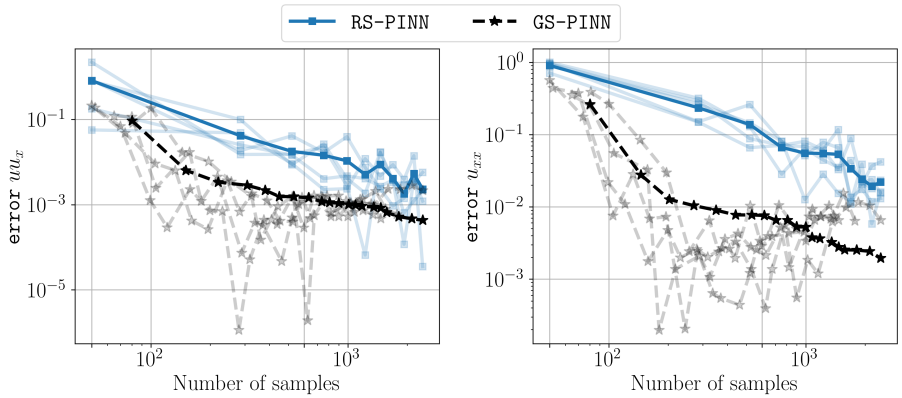

Figure 9: Comparing `GS-PINN` with `RS-PINN` to estimate coefficients corresponding to Burger's `PDE` equation (Section 5.2) with different time divisions $\mathtt{t_{div}}$ and precision values $\epsilon_{\mathtt{thr}}$

($n = 512$) and 201 ($m = 201$) temporal points within the original spatial domain of $x \in [-30,\ 30]$ and a time range of $t \in [0,\ 20]$, respectively. Having a snapshot matrix $\mathcal{U} \in \mathbb{R}^{512\times 201}$ demonstrates the curse of dimensionality in the training of the `PINN` for the `PDE` parameter estimation, therefore we employ `Q-DEIM` algorithm to extract the most informative samples. To do so, we set the time domain division $\mathtt{t_{div}} = 2$ and the precision value $\epsilon_{\mathtt{thr}} = 10^{-3}$ whereby we earn 288 samples which is approximately 0.28% of our entire dataset. The entire dataset (left) and selected samples (right) by `Q-DEIM` are shown in Figure 10. Number of iteration for training is set at `max-iter` = 1000 and we feed these samples into our `GS-PINN` architecture to recover the parameters of the underlying `PDE`. The convergence of the coefficients to their true values corresponding to the terms uu_x and u_{xxx} is depicted in Figure 11. The computed parameter vector $p^\top = [-5.971,\ -0.973]$ which based on (18) we have $c = -p_1$ and $\alpha = -p_2$. We can see that with having 0.28% of the entire dataset we reach to relative errors, $\mathtt{error}_1 = 0.00478$ and $\mathtt{error}_1 = 0.0267$ corresponding to the terms uu_x and u_{xxx}, respectively. This verifies the effectiveness of choosing most valuable and informative samples in the training of `PINN`.

Figure 12 presents the relative error associated with various pairs of $(\mathtt{t_{div}}, \epsilon_{\mathtt{thr}})$. To achieve this, we have chosen 20 equally spaced logarithmic values for precision $\epsilon_{\mathtt{thr}}$, covering the range from 10^{-10} to 10^{-2}, and linked each of these values to time domain divisions $\mathtt{t_{div}} = 1,\ 2,\ 3,\ 4$. One common observation by comparing curves corresponding to the pairs $(\mathtt{t_{div}}, \epsilon_{\mathtt{thr}})$ is that for the fixed $\mathtt{t_{div}}$ decreasing $\epsilon_{\mathtt{thr}}$ has no significant impact on the number of samples that are chosen by `Q-DEIM` and as well as corresponding relative errors. Moreover, we see that with approximately 300 samples almost all $\mathtt{t_{div}}$s have acceptable performance. The curve corresponding to $\mathtt{t_{div}} = 3$ with approximately 2500 samples gives the minimum relative error associated to the terms uu_x

and u_{xxx} compared to the others. Interesting observation is that time domain division $\mathtt{t_{div}}$ limit the maximum number of samples chosen by `Q-DEIM` at lower values of $\epsilon_{\mathtt{thr}}$.

Having the sample range through `GS-PINN`, we are ready to evaluate the performance of `RS-PINN` in recovering the `PDE` coefficients. The results of this comparative analysis between `GS-PINN` and `RS-PINN` are depicted in Figure 13. To streamline the presentation of `RS-PINN`'s performance in the plots, we calculate the average relative error across five experiments for a fixed number of samples. Like the previous examples we utilize `K-means` clustering with a parameter of $k = 20$ to identify the closest cluster centroid for each combination of sample size and relative error. As we can see for the lower value of sample range both `GS-PINN` and `RS-PINN` do not have good performance, however `RS-PINN` requires almost 4000 samples to reach almost the same relative errors compared to `GS-PINN` in the sample range $200 - 400$ which demonstrates the significant impact of greedy sampling on `PDE`parameter estimation. In the higher sampling range both `RS-PINN` and `GS-PINN` have almost the same performance.

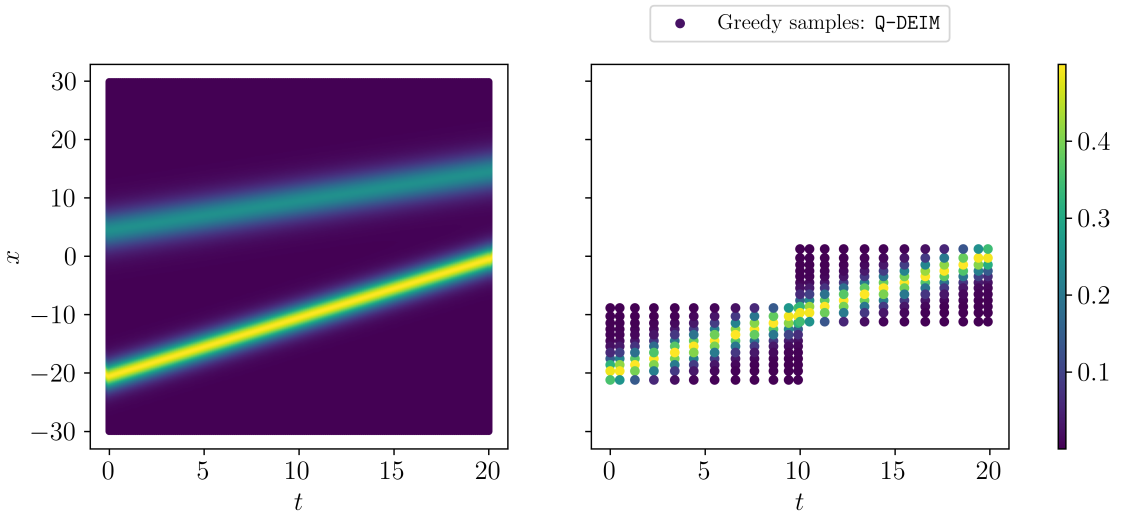

Figure 10: (left) Entire dataset ; (right) Greedy samples by `Q-DEIM` algorithm for KdV equation (Section 5.3)

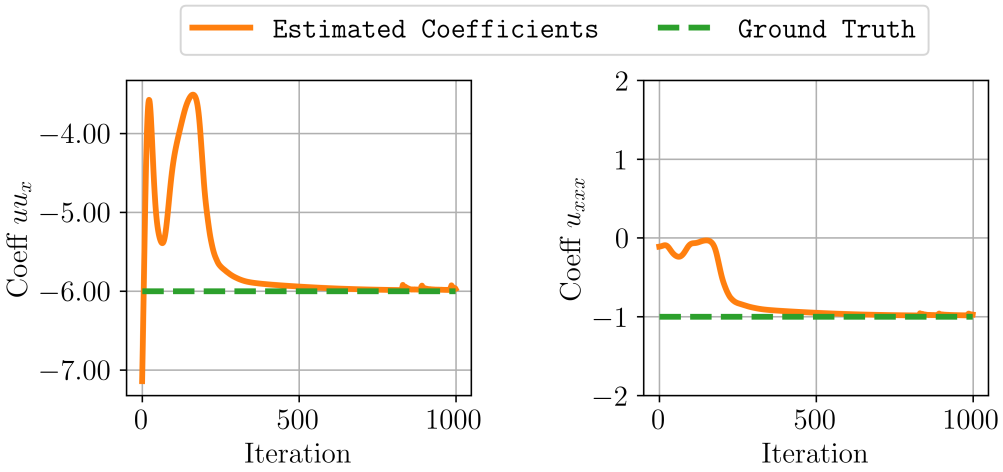

Figure 11: Estimated coefficients by `GS-PINN` for KdV equation (Section 5.3)

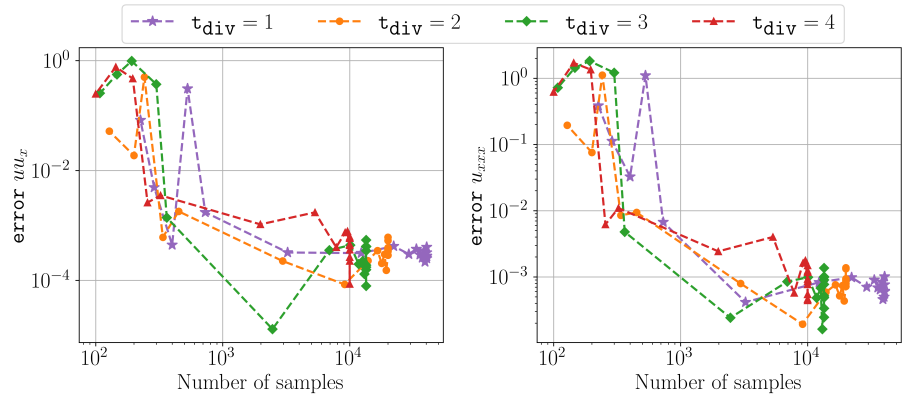

Figure 12: Performance of GS-PINN to estimate coefficients corresponding to KdV PDE equation (Section 5.3) with different time divisions t_{div} and precision values ϵ_{thr}

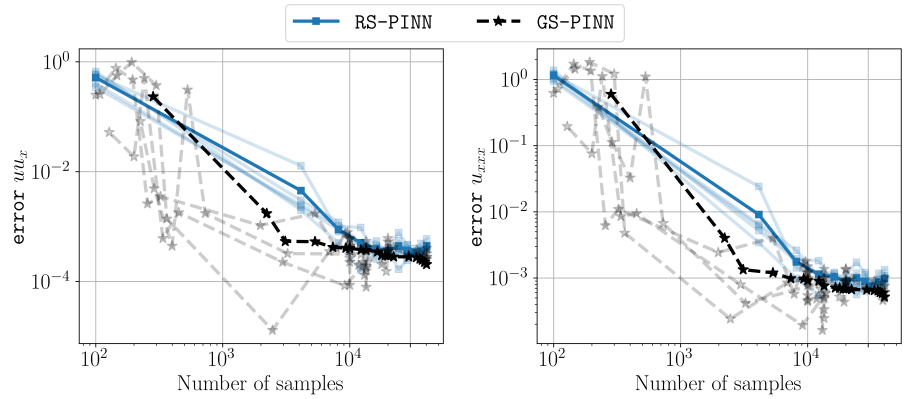

Figure 13: Comparing GS-PINN with RS-PINN to estimate coefficients corresponding to KdV PDE equation (Section 5.3) with different time divisions t_{div} and precision values ϵ_{thr}

6. Conclusion

In this paper we employed a greedy sampling approach, namely Q-DEIM, on the dataset resulted from descretization of the PDEs to do parameter estimation using PINN. Q-DEIM selects the most informative samples from a given snapshot matrix corresponding to a PDE. Through various simulation examples we have shown that greedy sampling approach based on Q-DEIM algorithm significantly outperforms common random sampling method for the PDE parameter estimation both in required training time for the PINN architecture as well as relative estimation error. Our investigation revealed that by using time domain division of the snapshot matrix of a PDE and Q-DEIM algorithm we can capture local dynamics, hence acquire more valuable samples which improves the quality of parameter estimation via PINN. Leveraging Q-DEIM and PINN we have estimated the corresponding parameters of three well known PDEs, (i) Allen-Cahn equation, (ii) Burgers' equation, and (iii) Korteweg-de Vries equation with 0.383%, 1.39%, and 0.28% of the dataset, respectively.

References

[1] F. De Barros, W. Mills, and R. Cotta, "Integral transform solution of a two-dimensional model for contaminant dispersion in rivers and channels with spatially variable coefficients," *Environmental Modelling & Software*, vol. 21, no. 5, pp. 699–709, 2006.

[2] M. Zeneli, A. Nikolopoulos, S. Karellas, and N. Nikolopoulos, "Numerical methods for solid-

liquid phase-change problems," in *Ultra-high Temperature Thermal Energy Storage, Transfer and Conversion*. Elsevier, 2021, pp. 165–199.

[3] B. Bai, H. Ci, H. Lei, and Y. Cui, "A local integral-generalized finite difference method with mesh-meshless duality and its application," *Engineering Analysis with Boundary Elements*, vol. 139, pp. 14–31, 2022.

[4] J. Blechschmidt and O. G. Ernst, "Three ways to solve partial differential equations with neural networks – a review," *GAMM-Mitteilungen*, vol. 44, no. 2, p. e202100006, 2021.

[5] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[6] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[7] Y. Chen, L. Lu, G. E. Karniadakis, and L. D. Negro, "Physics-informed neural networks for inverse problems in nano-optics and metamaterials," *Opt. Express*, vol. 28, pp. 11 618–11 633, 2020.

[8] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020.

[9] G. Pang, L. Lu, and G. E. Karniadakis, "fPINNs: Fractional physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 41, no. 4, pp. A2603–A2626, 2019.

[10] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis, "Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems," *Journal of Computational Physics*, vol. 397, p. 108850, 2019.

[11] L. Yang, D. Zhang, and G. E. Karniadakis, "Physics-informed generative adversarial networks for stochastic differential equations," *SIAM Journal on Scientific Computing*, vol. 42, no. 1, pp. A292–A317, 2020.

[12] M. A. Nabian and H. Meidani, "A deep learning solution approach for high-dimensional random differential equations," *Probabilistic Engineering Mechanics*, vol. 57, pp. 14–25, 2019.

[13] D. Zhang, L. Guo, and G. E. Karniadakis, "Learning in modal space: Solving time-dependent stochastic pdes using physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 42, no. 2, pp. A639–A665, 2020.

[14] E. A. Antonelo, E. Camponogara, L. O. Seman, E. R. de Souza, J. P. Jordanou, and J. F. Hubner, "Physics-informed neural nets for control of dynamical systems," *arXiv preprint arXiv:2104.02556*, 2021.

[15] S. Mowlavi and S. Nabi, "Optimal control of PDEs using physics-informed neural networks," *Journal of Computational Physics*, vol. 473, p. 111731, 2023.

[16] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerische Mathematik*, vol. 2, pp. 84–90, 1960.

[17] W.-S. L. Tien-Tsin Wong and P.-A. Heng, "Sampling with hammersley and halton points," *Journal of Graphics Tools*, vol. 2, no. 2, pp. 9–24, 1997.

[18] I. M. Sobol, "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967.

[19] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 403, p. 115671, 2023.

[20] S. Chaturantabut and D. C. Sorensen, "Nonlinear model reduction via discrete empirical interpolation," *SIAM Journal on Scientific Computing*, vol. 32, no. 5, pp. 2737–2764, 2010.

[21] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera, "An empirical interpolation method: application to efficient reduced-basis discretization of partial differential equations," *Comptes Rendus Mathematique*, vol. 339, no. 9, pp. 667–672, 2004.

[22] Z. Drmac and S. Gugercin, "A new selection operator for the discrete empirical interpolation method—improved a priori error bound and extensions," *SIAM Journal on Scientific Computing*, vol. 38, no. 2, pp. A631–A648, 2016.

[23] E. Haber, *Computational Methods in Geophysical Electromagnetics*. SIAM, 2014.

[24] S. W. Fung and L. Ruthotto, "A multiscale method for model order reduction in PDE parameter estimation," *Journal of Computational and Applied Mathematics*, vol. 350, pp. 19–34, 2019.

[25] A. M. Tartakovsky, C. O. Marrero, P. Perdikaris, G. D. Tartakovsky, and D. Barajas-Solano, "Physics-informed deep neural networks for learning parameters and constitutive relationships in subsurface flow problems," *Water Resources Research*, vol. 56, no. 5, p. e2019WR026731, 2020.

[26] Q. Ye, "Accurate inverses for computing eigenvalues of extremely ill-conditioned matrices and differential operators," *Mathematics of Computation*, vol. 87, no. 309, pp. 237–259, 2018.

[27] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009, vol. 2.

[28] R. Tibshirani, "Regression shrinkage and selection via the Lasso," *J. Roy. Statist. Soc.: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[29] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky, "An interior-point method for large-scale l_1 -regularized least squares," *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 606–617, 2007.

[30] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, p. 1, 2010.

[31] G.-J. Both, G. Tod, and R. Kusters, "Model discovery in the sparse sampling regime," *arXiv preprint arXiv:2105.00400*, 2021.

[32] Z. Zhang, X. Yang, and G. Lin, "POD-based constrained sensor placement and field reconstruction from noisy wind measurements: A perturbation study," *Mathematics*, vol. 4, no. 2, p. 26, 2016.

[33] S. Joshi and S. Boyd, "Sensor selection via convex optimization," *IEEE Transactions on Signal Processing*, vol. 57, no. 2, pp. 451–462, 2008.

[34] J. Ranieri, A. Chebira, and M. Vetterli, "Near-optimal sensor placement for linear inverse problems," *IEEE Transactions on signal processing*, vol. 62, no. 5, pp. 1135–1146, 2014.

[35] S. Lau, R. Eichardt, L. Di Rienzo, and J. Haueisen, "Tabu search optimization of magnetic sensor systems for magnetocardiography," *IEEE Transactions on Magnetics*, vol. 44, no. 6, pp. 1442–1445, 2008.

[36] Z. Wang, H.-X. Li, and C. Chen, "Reinforcement learning-based optimal sensor placement for spatiotemporal modeling," *IEEE Transactions on Cybernetics*, vol. 50, no. 6, pp. 2861–2871, 2019.

[37] G.-J. Both, S. Choudhury, P. Sens, and R. Kusters, "Deepmod: Deep learning for model discovery in noisy data," *Journal of Computational Physics*, vol. 428, p. 109985, 2021.

[38] K. Kunisch and S. Volkwein, "Galerkin proper orthogonal decomposition methods for a general equation in fluid dynamics," *SIAM J. Numer. Anal.*, vol. 40, no. 2, pp. 492–515, 2002.

[39] K. Manohar, B. W. Brunton, J. N. Kutz, and S. L. Brunton, "Data-driven sparse sensor placement for reconstruction: Demonstrating the benefits of exploiting known patterns," *IEEE Control Systems Magazine*, vol. 38, no. 3, pp. 63–86, 2018.

[40] M. Gavish and D. L. Donoho, "The optimal hard threshold for singular values is $\frac{4}{\sqrt{3}}$," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 5040–5053, 2014.

[41] S. Manavi, E. Fattahi, and T. Becker, "A trial solution for imposing boundary conditions of partial differential equations in physics-informed neural networks," *Engineering Applications of Artificial Intelligence*, vol. 127, p. 107236, 2024.

[42] K. Eshkofti and S. M. Hosseini, "A gradient-enhanced physics-informed neural network (gPINN) scheme for the coupled non-fickian/non-fourierian diffusion-thermoelasticity analysis: A novel gpinn structure," *Engineering Applications of Artificial Intelligence*, vol. 126, p. 106908, 2023.

[43] N. Wandel, M. Weinmann, M. Neidlin, and R. Klein, "Spline-pinn: Approaching pdes without data using fast, physics-informed hermite-spline cnns," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8529–8538.

[44] A. Forootani, P. Goyal, and P. Benner, "A robust SINDy approach by combining neural networks and an integral form," *arXiv preprint arXiv:2309.07193*, 2023.

[45] P. Goyal and P. Benner, "Neural ordinary differential equations with irregular and noisy data," *Roy. Soc. Open Sci.*, vol. 10, no. 7, p. 221475, 2023.

[46] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, no. 4, p. e1602614, 2017.

[47] V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[48] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[50] X. Feng and A. Prohl, "Numerical analysis of the Allen-Cahn equation and approximation for mean curvature flows," *Numerische Mathematik*, vol. 94, pp. 33–65, 2003.

[51] C. L. Wight and J. Zhao, "Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks," *Communications in Computational Physics*, vol. 29, no. 3, pp. 930–954, 2021.

[52] D. Arthur and S. Vassilvitskii, "`K-means++`: the advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1035.

[53] T. Öziş, E. Aksan, and A. Özdeş, "A finite element approach for solution of Burger's equation," *Applied Mathematics and Computation*, vol. 139, no. 2-3, pp. 417–428, 2003.

A. Appendix

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into k distinct, non-overlapping subsets (clusters). The goal of the algorithm is to group similar data points together and assign them to clusters, with the number of clusters, k, specified by the user.

The step-by-step explanation of how the k-means algorithm works: (i) Initialization: Randomly select k data points from the dataset as the initial centroids. The centroids are the points that will represent the center of each cluster. (ii) Assignment: assign each data point to the cluster whose centroid is closest to it. This is typically done using a distance metric, such as Euclidean distance. (iii) Update Centroids: recalculate the centroids of the clusters by taking the mean of all the data points assigned to each cluster. (iv) Repetition: repeat steps (ii) and (iii) until convergence is reached. Convergence occurs when the centroids no longer change significantly between iterations or when a certain number of iterations is reached. (v) Output: The algorithm produces k clusters, and each data point is assigned to one of these clusters.

The final result of the k-means clustering algorithm is a set of k cluster centroids and a labeling of each data point to its respective cluster. K-means clustering algorithm has been already implemented in Machine Learning software packages [1].

To apply the K-means clustering algorithm on the results of the GS-PINN regarding pairs $(\mathtt{t_{div}}, \epsilon_{\mathtt{thr}})$ we construct a matrix consist of 80 rows and 2 columns, i.e. $\mathtt{t_{div}} = 1,\ 2,\ 3,\ 4$ where each have corresponding 20 relative error as mentioned in the draft. The first column is the number of samples for each pair $(\mathtt{t_{div}}, \epsilon_{\mathtt{thr}})$ and the second column corresponds to the associated relative error. Employing K-means clustering algorithm with $k = 20$ will result 20 centroids each is a pair corresponds to the number of samples and relative error value. The clustering is performed with 100 different initializations to enhance robustness.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html