Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 403 (2023) 115671
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks

Chenxi Wu[a,1], Min Zhu[a,1], Qinyang Tan[b], Yadhu Kartha[c], Lu Lu[a,*]

[a] Department of Chemical and Biomolecular Engineering, University of Pennsylvania, Philadelphia, PA 19104, USA
[b] Department of Mathematics, University of Southern California, Los Angeles, CA 90089, USA
[c] College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA

Received 20 July 2022; received in revised form 21 September 2022; accepted 24 September 2022
Available online 22 October 2022

Abstract

Physics-informed neural networks (PINNs) have shown to be effective tools for solving both forward and inverse problems of partial differential equations (PDEs). PINNs embed the PDEs into the loss of the neural network using automatic differentiation, and this PDE loss is evaluated at a set of scattered spatio-temporal points (called residual points). The location and distribution of these residual points are highly important to the performance of PINNs. However, in the existing studies on PINNs, only a few simple residual point sampling methods have mainly been used. Here, we present a comprehensive study of two categories of sampling for PINNs: non-adaptive uniform sampling and adaptive nonuniform sampling. We consider six uniform sampling methods, including (1) equispaced uniform grid, (2) uniformly random sampling, (3) Latin hypercube sampling, (4) Halton sequence, (5) Hammersley sequence, and (6) Sobol sequence. We also consider a resampling strategy for uniform sampling. To improve the sampling efficiency and the accuracy of PINNs, we propose two new residual-based adaptive sampling methods: residual-based adaptive distribution (RAD) and residual-based adaptive refinement with distribution (RAR-D), which dynamically improve the distribution of residual points based on the PDE residuals during training. Hence, we have considered a total of 10 different sampling methods, including six non-adaptive uniform sampling, uniform sampling with resampling, two proposed adaptive sampling, and an existing adaptive sampling. We extensively tested the performance of these sampling methods for four forward problems and two inverse problems in many setups. Our numerical results presented in this study are summarized from more than 6000 simulations of PINNs. We show that the proposed adaptive sampling methods of RAD and RAR-D significantly improve the accuracy of PINNs with fewer residual points for both forward and inverse problems. The results obtained in this study can also be used as a practical guideline in choosing sampling methods.
© 2022 Elsevier B.V. All rights reserved.

Keywords: Partial differential equations; Physics-informed neural networks; Residual point distribution; Non-adaptive uniform sampling; Uniform sampling with resampling; Residual-based adaptive sampling

1. Introduction

Physics-informed neural networks (PINNs) [1] have emerged in recent years and quickly became a powerful tool for solving both forward and inverse problems of partial differential equations (PDEs) via deep neural

* Corresponding author.
E-mail address: lulu1@seas.upenn.edu (L. Lu).
[1] These authors contributed equally to this work.

https://doi.org/10.1016/j.cma.2022.115671
0045-7825/© 2022 Elsevier B.V. All rights reserved.

networks (DNNs) [2–4]. PINNs embed the PDEs into the loss of the neural network using automatic differentiation. Compared with traditional numerical PDE solvers, such as the finite difference method (FDM) and the finite element method (FEM), PINNs are meshfree and therefore highly flexible. Moreover, PINNs can easily incorporate both physics-based constraints and data measurements into the loss function. PINNs have been applied to tackle diverse problems in computational science and engineering, such as inverse problems in nano-optics, metamaterials [5], and fluid dynamics [2], parameter estimation in systems biology [6,7], and problems of inverse design and topology optimization [8]. In addition to standard PDEs, PINNs have also been extended to solve other types of PDEs, including integro-differential equations [3], fractional PDEs [9], and stochastic PDEs [10].

Despite past success, addressing a wide range of PDE problems with increasing levels of complexity can be theoretically and practically challenging. Thus, many aspects of PINNs still require further improvements to achieve more accurate prediction, higher computational efficiency, and training robustness [4]. A series of extensions to the vanilla PINN have been proposed to boost the performance of PINNs from various aspects. For example, better loss functions have been discovered via meta-learning [11], and gradient-enhanced PINNs (gPINNs) have been developed to embed the gradient information of the PDE residual into the loss [12]. In PINNs, the total loss is a weighted summation of multiple loss terms corresponding to the PDE and initial/boundary conditions. Different methods have been developed to automatically tune these weights and balance the losses [13–15]. Moreover, different weights for each loss term could be set at every training point [8,16–18]. For problems in a large domain, the decomposition of the spatio-temporal domain accelerates the training of PINNs and improves their accuracy [19–21]. For time-dependent problems, it is usually helpful to first train PINNs within a short time domain and then gradually expand the time intervals of training until the entire time domain is covered [22–26]. In addition to these general methods, other problem-specific techniques have also been developed, e.g., enforcing Dirichlet or periodic boundary conditions exactly by constructing special neural network architectures [8,27,28].

PINNs are mainly optimized against the PDE loss, which guarantees that the trained network is consistent with the PDE to be solved. PDE loss is evaluated at a set of scattered residual points. Intuitively, the effect of residual points on PINNs is similar to the effect of mesh points on FEM. Thus, the location and distribution of these residual points should be highly important to the performance of PINNs. However, in previous studies on PINNs, two simple residual point sampling methods (i.e., an equispaced uniform grid and uniformly random sampling) have mainly been used, and the importance of residual point sampling has largely been overlooked.

1.1. Related work and our contributions

Different residual point sampling methods can be classified into two categories: uniform sampling and nonuniform sampling. Uniform sampling can be obtained in multiple ways. For example, we could use the nodes of an equispaced uniform grid as the residual points or randomly sample the points according to a continuous uniform distribution in the computational domain. Although these two sampling methods are simple and widely used, alternative sampling methods may be applied. The Latin hypercube sampling (LHS) [29,30] was used in Ref. [1], and the Sobol sequence [31] was first used for PINNs in Ref. [9]. The Sobol sequence is one type of quasi-random low-discrepancy sequence among other sequences, such as the Halton sequence [32], and the Hammersley sequence [33]. Low-discrepancy sequences usually perform better than uniformly distributed random numbers in many applications such as numerical integration; hence, a comprehensive comparison of these methods for PINNs is required. However, very few comparisons [34,35] have been performed. In this study, we

- extensively compared the performance of different uniform sampling methods, including (1) equispaced uniform grid, (2) uniformly random sampling, (3) LHS, (4) Sobol sequence, (5) Halton sequence, and (6) Hammersley sequence.

In supervised learning, the dataset is fixed during training, but in PINNs, we can select residual points at any location. Hence, instead of using the same residual points during training, in each optimization iteration, we could select a new set of residual points, as first emphasized in Ref. [3]. While this strategy has been used in some works, it has not yet been systematically tested. Thus, in this study, we

- tested the performance of such a resampling strategy and investigated the effect of the number of residual points and the resampling period for the first time.

Uniform sampling works well for some simple PDEs, but it may not be efficient for those that are more complicated. To improve the accuracy, we could manually select the residual points in a nonuniform way, as was done in Ref. [36] for high-speed flows, but this approach is highly problem-dependent and usually tedious and time-consuming. In this study, we focus on automatic and adaptive nonuniform sampling. Motivated by the adaptive mesh refinement in FEM, Lu et al. [3] proposed the first adaptive nonuniform sampling for PINNs in 2019, the residual-based adaptive refinement (RAR) method, which adds new residual points in the locations with large PDE residuals. In 2021, another sampling strategy [37] was developed, where all the residual points were resampled according to a probability density function (PDF) proportional to the PDE residual. In this study, motivated by these two ideas, we proposed two new sampling strategies:

- residual-based adaptive distribution (RAD), where the PDF for sampling is a nonlinear function of the PDE residual;
- residual-based adaptive refinement with distribution (RAR-D), which is a hybrid method of RAR and RAD, i.e., the new residual points are added according to a PDF.

During the preparation of this paper, a few new studies appeared [38–45] that also proposed modified versions of RAR or PDF-based resampling. Most of these methods are special cases of the proposed RAD and RAR-D, and our methods can achieve better performance. We include a detailed comparison of these strategies in Section 2.4, after introducing several notations and our new proposed methods.

In this study, we have considered a total of 10 different sampling methods, including seven non-adaptive sampling methods (six different uniform samplings and one uniform sampling with resampling) and three adaptive sampling approaches (RAR, RAD, and RAR-D).

- We compared the performance of these sampling methods for four forward problems of PDEs and investigated the effect of the number of residual points.
- We also compared their performance for two inverse problems that have not yet been considered in the literature.
- We performed more than 6000 simulations of PINNs to obtain all the results shown in this study.

1.2. Organization

This paper is organized as follows. In Section 2, after providing a brief overview of PINNs and different non-adaptive sampling strategies, two new adaptive nonuniform sampling strategies (RAD and RAR-D) are proposed. In Section 3, we compare the performance of 10 different methods for six different PDE problems, including four forward problems and two inverse problems. Section 4 summarizes the findings and concludes the paper.

2. Methods

This section briefly reviews physics-informed neural networks (PINNs) in solving forward and inverse partial differential equations (PDEs). Then different types of uniformly sampling are introduced. Next, two nonuniform residual-based adaptive sampling methods are proposed to enhance the accuracy and training efficiency of PINNs. Finally, a comparison of related methods is presented.

2.1. PINNs in solving forward and inverse PDEs

We consider the PDE parameterized by $\boldsymbol{\lambda}$ defined on a domain $\Omega \subset \mathbb{R}^d$,

$$f(\mathbf{x}; u(\mathbf{x})) = f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} = (x_1, \ldots, x_d) \in \Omega,$$

with boundary conditions on $\partial\Omega$

$$\mathcal{B}(u, \mathbf{x}) = 0,$$

and $u(\mathbf{x})$ denotes the solution at $\mathbf{x}$. In PINNs, the initial condition is treated as the Dirichlet boundary condition.

A forward problem is aimed to obtain the solution u across the entire domain, where the model parameters $\boldsymbol{\lambda}$ are known. In practice, the model parameters $\boldsymbol{\lambda}$ might be unknown, but some observations from the solution u are

available, which lead to an inverse problem. An inverse problem is aimed to discover parameters $\boldsymbol{\lambda}$ that best describe the observed data from the solution.

PINNs are capable of addressing both forward and inverse problems. To solve a forward problem, the solution u is represented with a neural network $\hat{u}(\mathbf{x};\boldsymbol{\theta})$. The network parameters $\boldsymbol{\theta}$ are trained to approximate the solution u, such that the loss function is minimized [1,3]:

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{T}) = w_f\mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f) + w_b\mathcal{L}_b(\boldsymbol{\theta};\mathcal{T}_b),$$

where

$$\mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|}\sum_{\mathbf{x}\in\mathcal{T}_f}\left| f(\mathbf{x};\frac{\partial\hat{u}}{\partial x_1},\ldots,\frac{\partial\hat{u}}{\partial x_d};\frac{\partial^2\hat{u}}{\partial x_1\partial x_1},\ldots,\frac{\partial^2\hat{u}}{\partial x_1\partial x_d};\ldots;\boldsymbol{\lambda})\right|^2, \tag{1}$$

$$\mathcal{L}_b(\boldsymbol{\theta};\mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|}\sum_{\mathbf{x}\in\mathcal{T}_b}\left|\mathcal{B}(\hat{u},\mathbf{x})\right|^2,$$

and w_f and w_b are the weights. Two sets of points are samples both inside the domain ($\mathcal{T}_f$) and on the boundaries ($\mathcal{T}_b$). Here, $\mathcal{T}_f$ and $\mathcal{T}_b$ are referred to as the sets of "residual points", and $\mathcal{T} = \mathcal{T}_f \cup \mathcal{T}_b$.

To solve the inverse problem, an additional loss term corresponding to the misfit of the observed data at the locations $\mathcal{T}_i$, defined as

$$\mathcal{L}_i(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|}\sum_{\mathbf{x}\in\mathcal{T}_i}|\hat{u}(\mathbf{x}) - u(\mathbf{x})|^2,$$

is added to the loss function. The loss function is then defined as

$$\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}) = w_f\mathcal{L}_f(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}_f) + w_b\mathcal{L}_b(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}_b) + w_i\mathcal{L}_i(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}_i),$$

with an additional weight w_i. Then the network parameters $\boldsymbol{\theta}$ are trained simultaneously with $\boldsymbol{\lambda}$.

For certain PDE problems, it is possible to enforce boundary conditions directly by constructing a special network architecture [8,12,27,28], which eliminates the loss term of boundary conditions. In this study, the boundary conditions are enforced exactly and automatically. Hence, for a forward problem, the loss function is

$$\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}) = \mathcal{L}_f(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}_f).$$

For an inverse problem, the loss function is

$$\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}) = w_f\mathcal{L}_f(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}_f) + w_i\mathcal{L}_i(\boldsymbol{\theta},\boldsymbol{\lambda};\mathcal{T}_i),$$

where we choose $w_f = w_i = 1$ for the diffusion–reaction equation in Section 3.6, and $w_f = 1$, $w_i = 1000$ for the Korteweg–de Vries equation in Section 3.7.

2.2. Uniformly-distributed non-adaptive sampling

The training of PINNs requires a set of residual points ($\mathcal{T}_f$). The sampling strategy of $\mathcal{T}_f$ plays a vital role in promoting the accuracy and computational efficiency of PINNs. Here, we discuss several sampling approaches.

2.2.1. Fixed residual points

In most studies of PINNs, we specify the residual points at the beginning of training and never change them during the training process. Two simple sampling methods (equispaced uniform grids and uniformly random sampling) have been commonly used. Other sampling methods, such as the Latin hypercube sampling (LHS) [29,30] and the Sobol sequence [31], have also been used in some studies [1,9,34]. The Sobol sequence is one type of quasi-random low-discrepancy sequence. Low-discrepancy sequences are commonly used as a replacement for uniformly distributed random numbers and usually perform better in many applications such as numerical integration. This study also considers other low-discrepancy sequences, including the Halton sequence [32] and the Hammersley sequence [33].

We list the six uniform sampling methods as follows, and the examples of 400 points generated in $[0,1]^2$ using different methods are shown in Fig. 1.

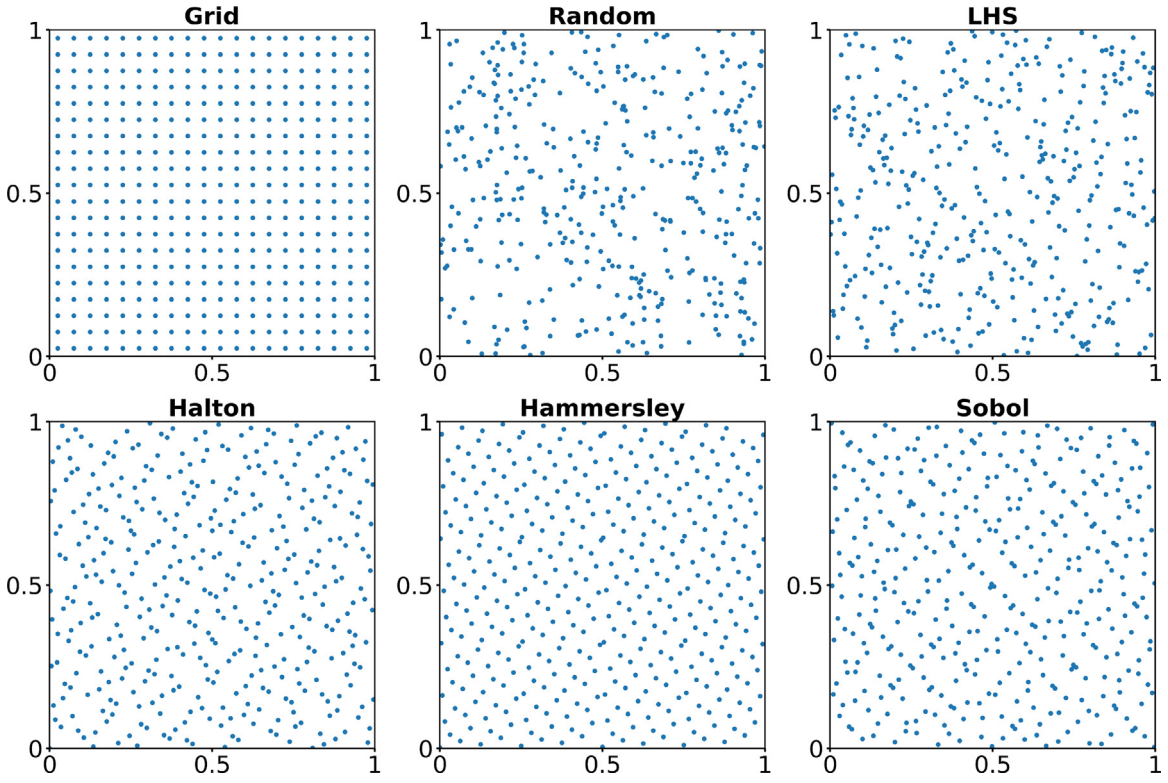

Fig. 1. Examples of 400 points generated in $[0, 1]^2$ using different uniform sampling methods in Section 2.2.1.

1. **Equispaced uniform grid (Grid)**: The residual points are chosen as the nodes of an equispaced uniform grid of the computational domain.
2. **Uniformly random sampling (Random)**: The residual points are randomly sampled according to a continuous uniform distribution over the domain. In practice, this is usually done using pseudo-random number generators such as the PCG-64 algorithm [46].
3. **Latin hypercube sampling (LHS)** [29,30]: The LHS is a stratified Monte Carlo sampling method that generates random samples that occur within intervals on the basis of equal probability and with normal distribution for each range.
4. Quasi-random low-discrepancy sequences:
   (a) **Halton sequence (Halton)** [32]: The Halton samples are generated according to the reversing or flipping the base conversion of numbers using primes.
   (b) **Hammersley sequence (Hammersley)** [33]: The Hammersley sequence is the same as the Halton sequence, except in the first dimension where points are located equidistant from each other.
   (c) **Sobol sequence (Sobol)** [31]: The Sobol sequence is a base-2 digital sequence that fills in a highly uniform manner.

2.2.2. Uniform points with resampling

In PINNs, a point at any location can be used to evaluate the PDE loss. Instead of using the fixed residual points during training, we could also select a new set of residual points in every certain optimization iteration [3]. The specific method to sample the points each time can be chosen from those methods discussed in Section 2.2.1. We can even use different sampling methods at different times, so many possible implementations make it impossible to be completely covered in this study.

In this study, we only consider Random sampling with resampling (**Random-R**). The Random-R method is the same as the Random method, except that the residual points are resampled for every N iteration. The *resampling*

period N is also an important hyperparameter for accuracy, as we demonstrate in our empirical experiments in Section 3.

2.3. Nonuniform adaptive sampling

Although the uniform sampling strategies were predominantly employed, recent studies on the nonuniform adaptive sampling strategies [3,37] have demonstrated promising improvement in the distribution of residual points during the training processes and achieved better accuracy.

2.3.1. Residual-based adaptive refinement with greed (RAR-G)

The first adaptive sampling method for PINNs is the residual-based adaptive refinement method (RAR) proposed in Ref. [3]. RAR aims to improve the distribution of residual points during the training process by sampling more points in the locations where the PDE residual is large. Specifically, after every certain iteration, RAR adds new points in the locations with large PDE residuals (Algorithm 1). RAR only focuses on the points with large residual, and thus it is a greedy algorithm. To better distinguish from the other sampling methods, the RAR method is referred to as RAR-G in this study.

Algorithm 1: RAR-G [3].

1 Sample the initial residual points $\mathcal{T}$ using one of the methods in Section 2.2.1;
2 Train the PINN for a certain number of iterations;
3 **repeat**
4 Sample a set of dense points $\mathcal{S}_0$ using one of the methods in Section 2.2.1;
5 Compute the PDE residuals for the points in $\mathcal{S}_0$;
6 $\mathcal{S} \leftarrow m$ points with the largest residuals in $\mathcal{S}_0$;
7 $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{S}$;
8 Train the PINN for a certain number of iterations;
9 **until** *the total number of iterations or the total number of residual points reaches the limit*;

2.3.2. Residual-based adaptive distribution (RAD)

RAR-G significantly improves the performance of PINNs when solving certain PDEs of solutions with steep gradients [3,12]. Nevertheless, RAR-G focuses mainly on the location where the PDE residual is largest and disregards the locations of smaller residuals. Another sampling strategy was developed later in Ref. [37], where all the residual points are resampled according to a probability density function (PDF) $p(\mathbf{x})$ proportional to the PDE residual. Specifically, for any point $\mathbf{x}$, we first compute the PDE residual $\varepsilon(\mathbf{x}) = \left|f(\mathbf{x}; \hat{u}(\mathbf{x}))\right|$, and then compute a probability as

$$p(\mathbf{x}) \propto \varepsilon(\mathbf{x}), \qquad \text{i.e.,} \qquad p(\mathbf{x}) = \frac{\varepsilon(\mathbf{x})}{A},$$

where $A = \int_\Omega \varepsilon(\mathbf{x}) dx$ is a normalizing constant. Then all the residual points are sampled according to $p(\mathbf{x})$.

This approach works for certain PDEs, but as we show in our numerical examples, it does not work well in some cases. Following this idea, we propose an improved version called the residual-based adaptive distribution (RAD) method (Algorithm 2), where we use a new PDF defined as

$$p(\mathbf{x}) \propto \frac{\varepsilon^k(\mathbf{x})}{\mathbb{E}[\varepsilon^k(\mathbf{x})]} + c. \tag{2}$$

Here, $k \geq 0$ and $c \geq 0$ are two hyperparameters. $\mathbb{E}[\varepsilon^k(\mathbf{x})]$ can be approximated by a numerical integration such as Monte Carlo integration. We note that the Random-R method in Section 2.2.2 is a special case of RAD by choosing $k = 0$ or $c \to \infty$.

In RAD (Algorithm 2 line 4), we need to sample a set of points according to $p(\mathbf{x})$, which can be done in a few ways. When $\mathbf{x}$ is low-dimensional, we can sample the points approximately in the following brute-force way:

1. Sample a set of dense points $\mathcal{S}_0$ using one of the methods in Section 2.2.1;
2. Compute $p(\mathbf{x})$ for the points in $\mathcal{S}_0$;

Algorithm 2: RAD.

Sample the initial residual points $\mathcal{T}$ using one of the methods in Section 2.2.1;
Train the PINN for a certain number of iterations;
repeat
  $\mathcal{T} \leftarrow$ A new set of points randomly sampled according to the PDF of Eq. (2);
  Train the PINN for a certain number of iterations;
until *the total number of iterations reaches the limit*;

3. Define a probability mass function $\tilde{p}(\mathbf{x}) = \frac{p(\mathbf{x})}{A}$ with the normalizing constant $A = \sum_{\mathbf{x}\in\mathcal{S}_0} p(\mathbf{x})$;
4. Sample a subset of points from $\mathcal{S}_0$ according to $\tilde{p}(\mathbf{x})$.

This method is simple, easy to implement, and sufficient for many PDE problems. For more complicated cases, we can use other methods such as inverse transform sampling, Markov chain Monte Carlo (MCMC) methods, and generative adversarial networks (GANs) [47].

The two hyperparameters k and c in Eq. (2) control the profile of $p(\mathbf{x})$ and, thus, the distribution of sampled points. We illustrate the effect of k and c using a simple 2D example,

$$\varepsilon(x, y) = 2^{4a} x^a (1-x)^a y^a (1-y)^a, \tag{3}$$

with $a = 10$ in Fig. 2. When $k = 0$, it becomes a uniform distribution. As the value of k increases, more residual points with large PDE residuals are sampled. As the value of c increases, the residual points exhibit an inclination to be uniformly distributed. Compared with RAR, RAD provides more freedom to balance the points in the locations with large and small residuals by tuning k and c. The optimal values of k and c are problem-dependent, and based on our numerical results, the combination of $k = 1$ and $c = 1$ is usually a good default choice.

2.3.3. Residual-based adaptive refinement with distribution (RAR-D)

We also propose a hybrid method of RAR-G and RAD, namely, residual-based adaptive refinement with distribution (RAR-D) (Algorithm 3). Similar to RAR-G, RAR-D repeatedly adds new points to the training dataset; similar to RAD, the new points are sampled based on the PDF in Eq. (2). We note that when $k \to \infty$, only points with the largest PDE residual are added, which recovers RAR-G. The optimal values of k and c are problem-dependent, and based on our numerical results, the combination of $k = 2$ and $c = 0$ is usually a good default choice.

Algorithm 3: RAR-D.

Sample the initial residual points $\mathcal{T}$ using one of the methods in Section 2.2.1;
Train the PINN for a certain number of iterations;
repeat
  $\mathcal{S} \leftarrow m$ points randomly sampled according to the PDF of Eq. (2);
  $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{S}$;
  Train the PINN for a certain number of iterations;
until *the total number of iterations or the total number of residual points reaches the limit*;

2.4. Comparison with related work

As discussed in Section 2.3, our proposed RAD and RAR-D are improved versions of the methods in Refs. [3,37]. Here, we summarize the similarities between their methods and ours.

- Lu et al. [3] (in July 2019) proposed RAR (renamed to RAR-G here), which is a special case of RAR-D by choosing a large value of k.
- The method proposed by Nabian et al. [37] (in April 2021) is a special case of RAD by choosing $k = 1$ and $c = 0$.

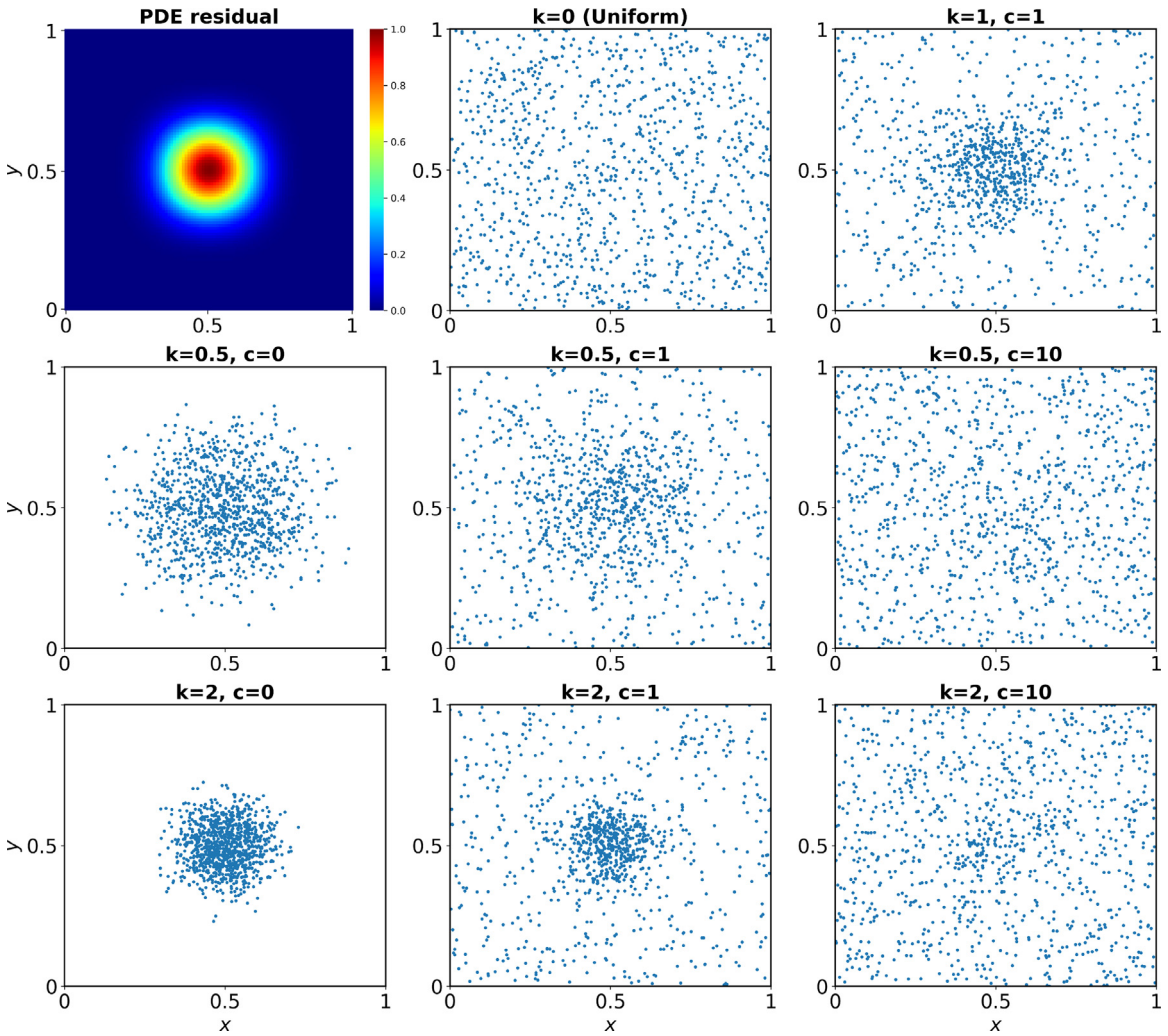

Fig. 2. Examples of 1000 residual points sampled by RAD with different values of k and c for the PDE residual $\varepsilon(x, y)$ in Eq. (3).

During the preparation of this paper, a few new papers appeared [38–45] that also proposed similar methods. Here, we summarize the similarities and differences between these studies.

- The method proposed by Gao et al. [40] (in December 2021) is a special case of RAD by choosing $c = 0$.
- Tang et al. [41] (in December 2021) proposed two methods. One is a special case of RAD by choosing $k = 2$ and $c = 0$, and the other is a special case of RAR-D by choosing $k = 2$ and $c = 0$.
- Zeng et al. [43] (in April 2022) proposed a subdomain version of RAR-G. The entire domain is divided into many subdomains, and then new points are added to the several subdomains with large average PDE residuals.
- Similar to RAR-G, Peng et al. [42] (in May 2022) proposed to add more points with large PDE residual, but they used the node generation technology proposed in Ref. [48]. We note that this method only works for a two-dimensional space.
- Zapf et al. [38] (in May 2022) proposed a modified version of RAR-G, where some points with small PDE residual are removed while adding points with large PDE residual. They show that compared with RAR, this reduces the computational cost, but the accuracy keeps similar.
- Hanna et al. [44] (in May 2022) proposed a similar method as RAR-D, but they chose $p(\mathbf{x}) \propto \max\{\log(\varepsilon(\mathbf{x})/\varepsilon_0), 0\}$, where ε_0 is a small tolerance.

Table 1
The hyperparameters used for each numerical experiment.

Problems	Depth	Width	Optimizer
Section 3.2 Diffusion equation	4	32	Adam
Section 3.3 Burgers' equation	4	64	Adam + L-BFGS
Section 3.4 Allen–Cahn equation	4	64	Adam + L-BFGS
Section 3.5 Wave equation	6	100	Adam + L-BFGS
Section 3.6 Diffusion–reaction equation (inverse)	4	20	Adam
Section 3.7 Korteweg–de Vries equation (inverse)	4	100	Adam

The learning rate of Adam optimizer is chosen as 0.001.

- Similar to the work of Zapf et al. Daw et al. [39] (in July 2022) also proposed to remove the points with small PDE residual, but instead of adding new points with large PDE residual, they added new uniformly random sampled points.
- Subramanian et al. [45] (in July 2022) proposed a cosine annealing strategy that balances the fraction of points sampled uniformly and non-uniformly based on either the PDE residual or its gradient.

Thus all these methods are special cases of our proposed RAD and RAR-D (or with minor modification). However, in our study, two tunable variables k and c are introduced. As we show in our results, the values of k and c could be crucial since they significantly influence the residual points distribution. By choosing proper values of k and c, our methods would outperform the other methods.

We also note that the point-wise weighting [8,16–18] can be viewed as a special case of adaptive sampling, described as follows. When the residual points are randomly sampled from a uniform distribution $\mathcal{U}(\Omega)$, and the number of residual points is large, the PDE loss in Eq. (1) can be approximated by $\mathbb{E}_{\mathcal{U}}[\varepsilon^2(\mathbf{x})]$. If we consider a point-wise weighting function $w(\mathbf{x})$, then the loss becomes $\mathbb{E}_{\mathcal{U}}[w(\mathbf{x})\varepsilon^2(\mathbf{x})]$, while for RAD the loss is $\mathbb{E}_p[\varepsilon^2(\mathbf{x})]$. If we choose $w(\mathbf{x})$ (divided by a normalizing constant) as the PDF $p(\mathbf{x})$, then the two losses are equal.

3. Results

We apply PINNs with all ten sampling methods in Section 2 to solve six forward and inverse PDE problems. In all examples, the hyperbolic tangent (tanh) is selected as the activation function. Table 1 summarizes the network width, depth, and optimizers used for each example. More details of the hyperparameters and training procedure can be found in each section of the specific problem.

For both forward and inverse problems, to evaluate the accuracy of the solution $\hat{u}$, the L^2 relative error is used:

$$\frac{\|\hat{u} - u\|_2}{\|u\|_2}.$$

For inverse problems, to evaluate the accuracy of the predicted coefficients $\hat{\lambda}$, the relative error is also computed:

$$\frac{|\hat{\lambda} - \lambda|}{|\lambda|}.$$

As the result of PINN has randomness due to the random sampling, network initialization, and optimization, thus, for each case, we run the same experiment at least ten times and then compute the geometric mean and standard deviation of the errors. The code in this study is implemented by using the library DeepXDE [3] and is publicly available from the GitHub repository https://github.com/lu-group/pinn-sampling.

3.1. Summary

Here, we first present a summary of the accuracy of all the methods for the forward and inverse problems listed in Tables 2 and Table 3, respectively. A relatively small number of residual points is chosen to show the difference among different methods. In the specific section of each problem (Sections 3.2–3.7), we discuss all the detailed analyses, including the convergence of error during the training process, the convergence of error with respect to the number of residual points, and the effects of different hyperparameters (e.g., the period of resampling in

Table 2
L^2 relative error of the PINN solution for the forward problems.

	Diffusion	Burgers'	Allen–Cahn	Wave
No. of residual points	30	2000	1000	2000
Grid	0.66 ± 0.06%	13.7 ± 2.37%	93.4 ± 6.98%	81.3 ± 13.7%
Random	0.74 ± 0.17%	13.3 ± 8.35%	22.2 ± 16.9%	68.4 ± 20.1%
LHS	0.48 ± 0.24%	13.5 ± 9.05%	26.6 ± 15.8%	75.9 ± 33.1%
Halton	0.24 ± 0.17%	4.51 ± 3.93%	0.29 ± 0.14%	60.2 ± 10.0%
Hammersley	0.17 ± 0.07%	3.02 ± 2.98%	**0.14 ± 0.14**%	58.9 ± 8.52%
Sobol	0.19 ± 0.07%	3.38 ± 3.21%	0.35 ± 0.24%	57.5 ± 14.7%
Random-R	**0.12 ± 0.06**%	1.69 ± 1.67%	0.55 ± 0.34%	**0.72 ± 0.90**%
RAR-G [3]	0.20 ± 0.07%	**0.12 ± 0.04**%	0.53 ± 0.19%	0.81 ± 0.11%
RAD	**0.11 ± 0.07**%	**0.02 ± 0.00**%	**0.08 ± 0.06**%	**0.09 ± 0.04**%
RAR-D	**0.14 ± 0.11**%	**0.03 ± 0.01**%	**0.09 ± 0.03**%	**0.29 ± 0.04**%

Bold font indicates the smallest three errors for each problem. Underlined text indicates the smallest error for each problem.

Table 3
L^2 relative error of the PINN solution and relative error of the inferred parameters for the inverse problems.

	Diffusion–reaction		Korteweg–de Vries		
	$u(x)$	$k(x)$	$u(x,t)$	λ_1	λ_2
No. of residual points	15		600		
Grid	0.36 ± 0.12%	8.58 ± 2.14%	24.4 ± 11.1%	53.7 ± 30.7%	42.0 ± 22.3%
Random	0.35 ± 0.17%	5.77 ± 2.05%	8.86 ± 2.80%	16.4 ± 7.33%	16.8 ± 7.40%
LHS	0.36 ± 0.14%	7.00 ± 2.62%	10.9 ± 2.60%	22.0 ± 6.68%	22.6 ± 6.36%
Halton	0.23 ± 0.08%	6.16 ± 1.08%	8.76 ± 3.33%	16.7 ± 6.16%	17.2 ± 6.20%
Hammersley	0.28 ± 0.08%	6.37 ± 0.91%	4.49 ± 3.56%	5.24 ± 7.08%	5.71 ± 7.32%
Sobol	**0.21 ± 0.06**%	**3.09 ± 0.75**%	8.59 ± 3.67%	15.8 ± 6.15%	15.6 ± 5.79%
Random-R	**0.19 ± 0.09**%	**3.43 ± 1.80**%	**0.97 ± 0.15**%	**0.41 ± 0.30**%	**1.14 ± 0.31**%
RAR-G [3]	1.12 ± 0.11%	15.9 ± 1.53%	8.83 ± 1.98%	15.4 ± 9.29%	14.5 ± 9.25%
RAD	**0.17 ± 0.09**%	**2.76 ± 1.32**%	**0.77 ± 0.11**%	**0.31 ± 0.19**%	**0.86 ± 0.25**%
RAR-D	0.76 ± 0.24%	10.3 ± 3.28%	**2.36 ± 0.98**%	**3.49 ± 2.21**%	**3.18 ± 2.02**%

Bold font indicates the smallest three errors for each problem. Underlined text indicates the smallest error for each problem.

Random-R, the values of k and c in RAD and RAR-D, and the number of new points added each time in RAR-D). We note that Random-R is a special case of RAD by choosing $k = 0$ or $c \to \infty$, and RAR-G is a special case of RAR-D by choosing $k \to \infty$.

Our main findings from the results are as follows.

- The proposed RAD method has always performed the best among the 10 sampling methods when solving all forward and inverse problems.
- For PDEs with complicated solutions, such as the Burgers' and multi-scale wave equation, the proposed RAD and RAR-D methods are predominately effective and yield errors that are magnitudes lower.
- For PDEs with smooth solutions, such as the diffusion equation and diffusion–reaction equation, some uniform sampling methods, such as the Hammersley and Random-R, also produce sufficiently low errors.
- Compared with other uniform sampling methods, Random-R usually demonstrates better performance.
- Among the six uniform sampling methods with fixed residual points, the low-discrepancy sequences (Halton, Hammersley, and Sobol) generally perform better than Random and LHS, and both are better than Grid.

3.2. Diffusion equation

We first consider the following one-dimensional diffusion equation:

$$\begin{aligned}
&\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + e^{-t}\left(-\sin(\pi x) + \pi^2 \sin(\pi x)\right), \quad x \in [-1, 1], t \in [0, 1],\\
&u(x, 0) = \sin(\pi x),\\
&u(-1, t) = u(1, t) = 0,
\end{aligned}$$

where u is the concentration of the diffusing material. The exact solution is $u(x, t) = \sin(\pi x)e^{-t}$.

We first compare the performance of the six uniform sampling methods with fixed residual points (Fig. 3A). The number of residual points is ranged from 10 to 80 with an increment of 10 points each time. For each number of residual points, the maximum iteration is set to be 15 000 with Adam as the optimizer. When the number of points is large (e.g., more than 70), all these methods have similar performance. However, when the number of residual points is small such as 50, the Hammersley and Sobol sequences perform better than others, and the equispaced uniform grid and random sampling have the largest errors (about one order of magnitude larger than Hammersley and Sobol).

We then test the Random-R method using 30 residual points (Fig. 3B). The accuracy of Random-R has a strong dependence on the period of resampling, and the optimal period of resampling in this problem is around 200. Compared with Random without resampling, the Random-R method always leads to lower L^2 relative errors regardless of the period of resampling. The error can be lower by one order of magnitude by choosing a proper resampling period. Among all the non-adaptive methods, Random-R performs the best.

Next, we test the performance of the nonuniform adaptive sampling methods. In Algorithms 2 and 3, the neural network is first trained using 10 000 steps of Adam. In the RAD method, we use 30 residual points and resample every 1000 iterations. The errors of RAD with different values of k and c are shown in Figs. 3C and D. We note that Random-R is a special case of RAD with either $c \to \infty$ or $k = 0$. Here, RAD with large values of c or small values of k leads to better accuracy, i.e., the points are almost uniformly distributed. For the RAR-D method (Figs. 3E and F), one residual point is added after every 1000 iterations starting from 10 points. When using $k = 2$ and $c = 0$ (the two red lines in Figs. 3E red F, RAR-D perform the best.

When using 30 residual points, the errors of all the methods are listed in Table 2. In this diffusion equation, all the methods achieve a good accuracy ($< 1\%$). Compared with Random-R (0.12%), RAD and RAR-D (0.11%) are not significantly better. The reason could be that the solution of this diffusion equation is very smooth, so uniformly distributed points are good enough. In our following examples, we show that RAD and RAR-D work significantly better and achieve an error of orders of magnitude smaller than the non-adaptive methods.

3.3. Burgers' equation

The Burgers' equation is considered defined as:

$$\begin{aligned}
&\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], t \in [0, 1],\\
&u(x, 0) = -\sin(\pi x),\\
&u(-1, t) = u(1, t) = 0,
\end{aligned}$$

where u is the flow velocity and ν is the viscosity of the fluid. In this study, ν is set at $0.01/\pi$. Different from the diffusion equation with a smooth solution, the solution of the Burgers' equation has a sharp front when $x = 0$ and t is close to 1.

We first test the uniform sampling methods by using the number of residual points ranging from 1000 to 10 000 (Fig. 4A). The maximum iteration is 15 000 steps, with Adam as the optimizer, followed by 15 000 steps of L-BFGS. Fig. 4A shows that the Hammersley method converges the fastest and reaches the lowest L^2 relative error among all the uniform sampling methods, while the Halton and Sobol sequences also perform adequately.

Fig. 4B shows the L^2 relative error as a function of the period of resampling using the Random-R method with 2000 residual points. Similar to the diffusion equation, the Random-R method always outperforms the Random

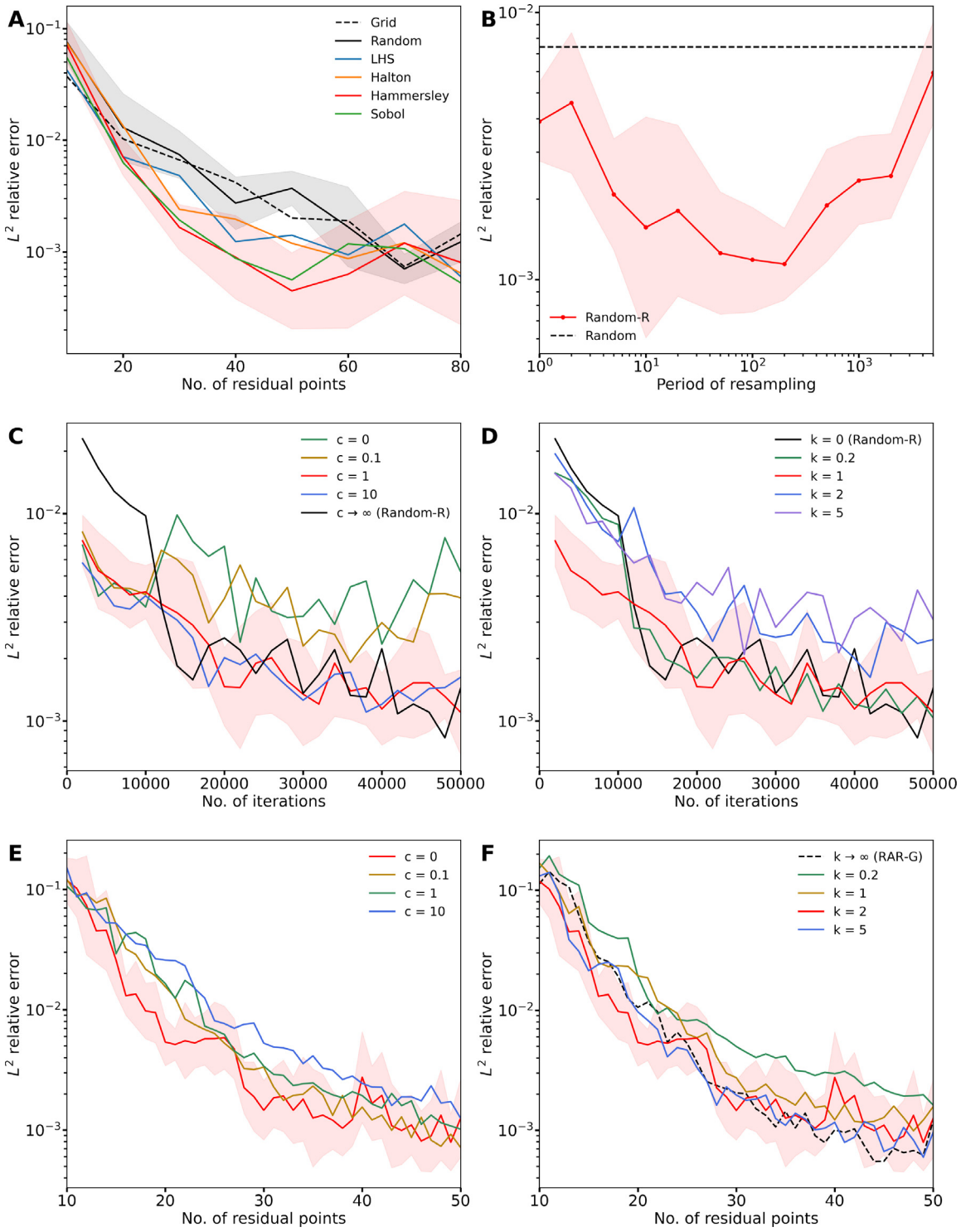

Fig. 3. L^2 relative errors of different sampling methods for the diffusion equation in Section 3.2. (A) Six uniform sampling with fixed residual points. (B) Random-R with different periods of resampling when using 30 residual points. (C and D) The training trajectory of RAD with different values of k and c when using 30 residual points. (C) $k = 1$. (D) $c = 1$. (E and F) RAR-D with different values of k and c. Each time one new point is added. (E) $k = 2$. (F) $c = 0$. The curves and shaded regions represent the geometric mean and one standard deviation of 10 runs. For clarity, only some standard deviations are plotted.

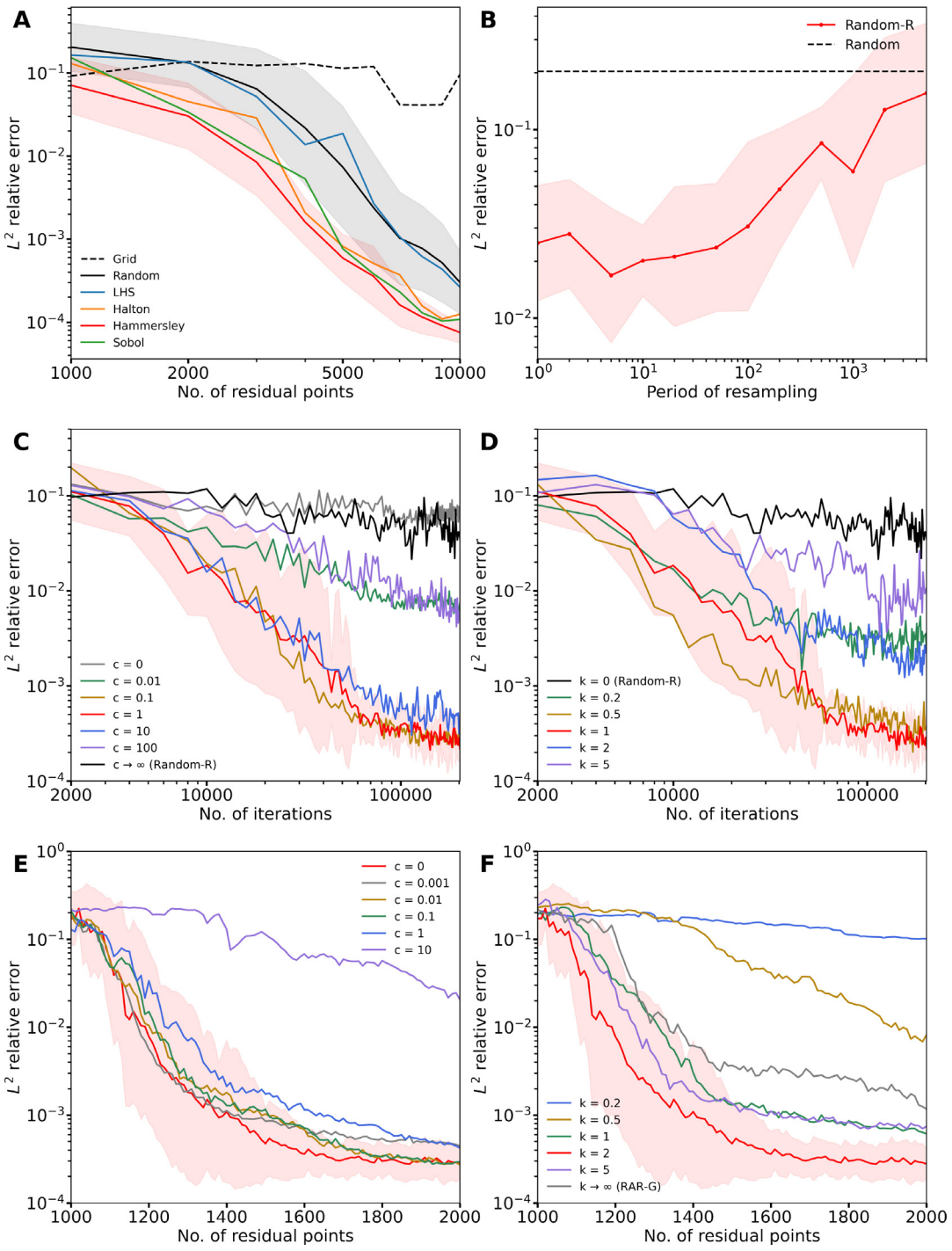

Fig. 4. L^2 relative errors of different sampling methods for the Burgers' equation in Section 3.3. (A) Six uniform sampling with fixed residual points. (B) Random-R with different periods of resampling when using 2000 residual points. (C and D) The training trajectory of RAD with different values of k and c when using 2000 residual points. (C) $k = 1$. (D) $c = 1$. (E and F) RAR-D with different values of k and c. Each time 10 new points are added. (E) $k = 2$. (F) $c = 0$. The curves and shaded regions represent the geometric mean and one standard deviation of 10 runs. For clarity, only some standard deviations are plotted.

method. However, the performance of Random-R is not sensitive to the period of resampling if the period is smaller than 100. Choosing a period of resampling too large can negatively affect its performance.

When applying the nonuniform adaptive methods, the neural network is first trained using 15 000 steps of Adam and then 1000 steps of L-BFGS. In the RAD method, we use 2000 residual points, which are resampled every 2000 iterations (1000 iterations using Adam followed by 1000 iterations using L-BFGS). As indicated by Fig. 4C, the RAD method possesses significantly greater advantages over the Random-R method (a special case of RAD by choosing $k = 0$ or $c \to \infty$), whose L^2 relative errors barely decrease during the training processes. This fact reflects that both extreme cases show worse performance. In contrast, for $k = 1$ and $c = 1$ (the red lines in Figs. 4C and D), the L^2 relative error declines rapidly and quickly reaches $\sim 2 \times 10^{-4}$. The RAD method is also effective when choosing a set of k and c in a moderate range.

For the RAR-D method, 1000 residual points are selected in the pre-trained process, and 10 residual points are added every 2000 iterations (1000 iterations using Adam and 1000 iterations using L-BFGS as optimizer) until the total number of residual points reaches 2000. Shown by Figs. 4E and F, the optimal values for k and c are found to be 2 and 0, respectively.

Since the solution of Burgers' equation has a very steep region, when using 2000 residual points, both RAD and RAR-D have competitive advantages over the uniform sampling methods in terms of accuracy and efficiency. For the following three forward PDE problems (Allen–Cahn equation in Section 3.4, wave equation in Section 3.5, and diffusion–reaction equation in Section 3.6), unless otherwise stated, the maximum iterations, the use of optimizer, and the training processes remain the same as the Burgers' equation.

Table 2 summarizes the L^2 relative error for all methods when we fix the number of residual points at 2000. All uniform sampling methods fail to capture the solution well. The L^2 relative errors given by the Halton, Hammersley, and Sobol methods ($\sim 4\%$) are around one-fourth of that given by the Grid, Random, and LHS methods ($> 13\%$). Even though the Random-R performs the best among all uniform methods ($1.69 \pm 1.67\%$), the proposed RAD and RAR-D methods can achieve an L^2 relative error two orders of magnitude lower than that (0.02%).

3.4. Allen–Cahn equation

Next, we consider the Allen–Cahn equation in the following form:

$$\begin{aligned}
&\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + 5(u - u^3), \quad x \in [-1, 1], t \in [0, 1],\\
&u(x, 0) = x^2 \cos(\pi x),\\
&u(-1, t) = u(1, t) = -1,
\end{aligned}$$

where the diffusion coefficient $D = 0.001$. Fig. 5 outlines the L^2 relative errors of different sampling methods for the Allen–Cahn equation.

Similar patterns are found for the nonadaptive uniform sampling as in the previous examples. The Hammersley method has the best accuracy (Fig. 5A). As the number of residual points becomes significantly large, the difference between these uniform sampling methods becomes negligible. Except for the equispaced uniform grid method, other uniform sampling methods converge to L^2 relative errors of 10^{-3}, about the same magnitude as the number of residual points reaching 10^4. Fig. 5B shows that when using 1000 residual points for Random-R, lower L^2 relative errors can be obtained if we select a period of resampling less than 500.

We next test the performance of RAD for different values of k and c when using a different number of residual points. In Figs. 5C and D, we resampled 500 residual points every 2000 iterations, while in Figs. 5E and F, we used 1000 residual points instead. For both cases, the combination of $k = 1$ and $c = 1$ (the red lines in Figs. 5C–F) gives good accuracy. When fewer residual points (e.g., 500) are used, the RAD methods boost the performance of PINNs.

Similarly, we also test RAR-D in Figs. 5G–J. In Figs. 5G and H, we pre-train the neural network with 500 residual points and add 10 residual points after every 2000 iterations until the total number of residual points reaches 1000. In Figs. 5I and J, we pre-train the neural network using 1000 residual points and heading to 2000 residual points in the same fashion. We recognize that 2 and 0 are the best k and c values for the RAR-D method for both scenarios, which outperform the RAR-G method.

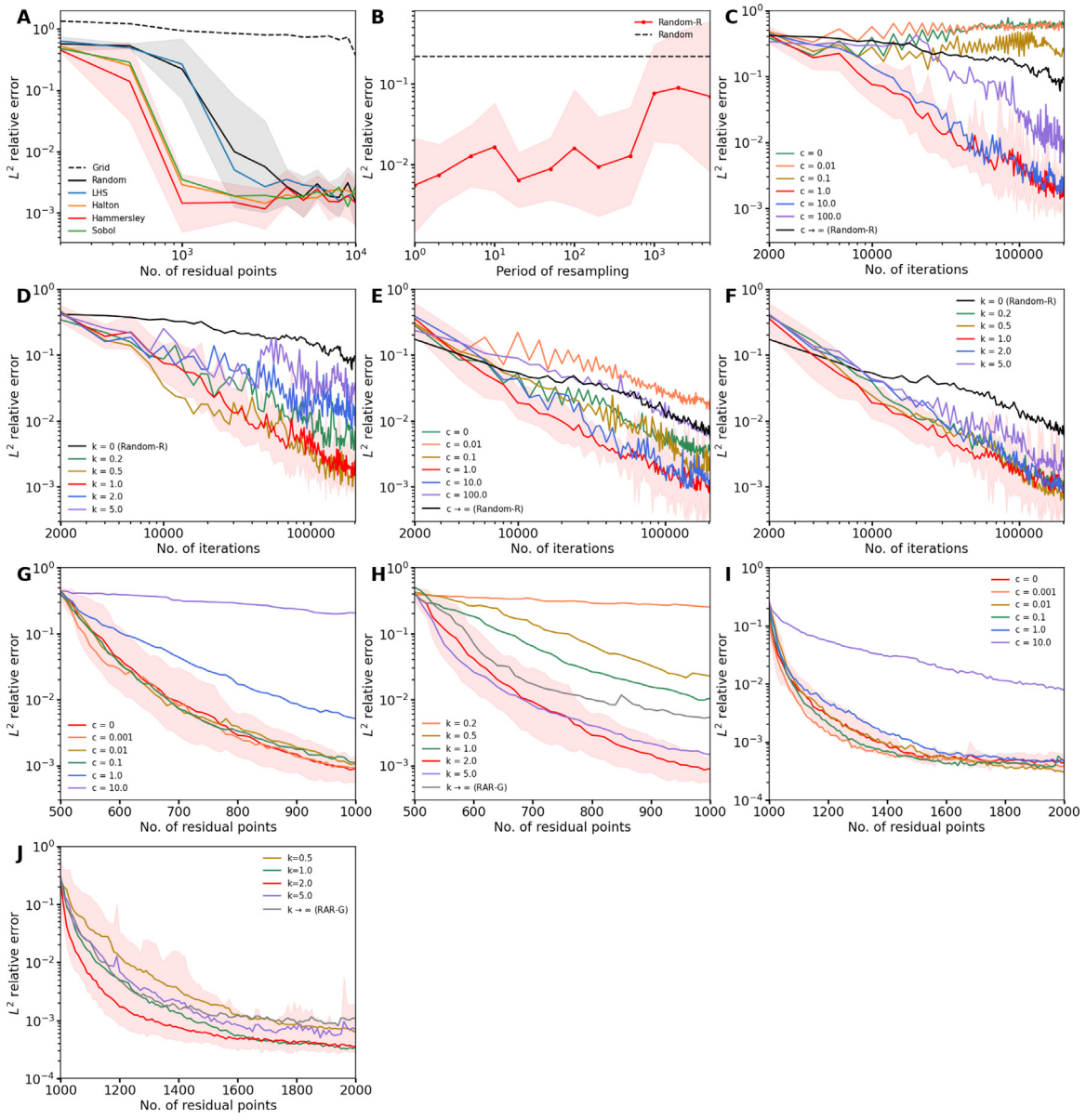

Fig. 5. L^2 relative errors of different sampling methods for the Allen–Cahn equation in Section 3.4. (A) Six uniform sampling with fixed residual points. (B) Random-R with different periods of resampling when using 1000 residual points. (C–F) The training trajectory of RAD with different values of k and c. (C and D) 500 residual points are used. (C) $k = 1$. (D) $c = 1$. (E and F) 1000 residual points are used. (E) $k = 1$. (F) $c = 1$. (G–J) RAR-D with different values of k and c. (G and H) The number of residual points is increased from 500 to 1000. Each time 10 new points are added. (G) $k = 2$. (H) $c = 0$. (I and J) The number of residual points is increased from 1000 to 2000. Each time 10 new points are added. (I) $k = 2$. (J) $c = 0$. The curves and shaded regions represent the geometric mean and one standard deviation of 10 runs. For clarity, only some standard deviations are plotted.

As proven in this example, when applying the RAD and the RAR-D methods, the optimal values of k and c remain stable even though we choose a different number of residual points. In addition, we find that the optimal k and c for the Burgers' and Allen–Cahn equations are the same for both the RAD and the RAR-D methods. Thus, we could choose $(k = 1, c = 1)$ for the RAD methods and $(k = 2, c = 0)$ for the RAR-D methods by default when first applied these methods to a new PDE problem.

To make a comparison across all sampling methods, Table 2 shows the L^2 relative error for the Allen–Cahn equation when we fix the number of residual points at 1000. The Grid, Random, and LHS methods are prone to substantial errors, which are all larger than 20%. Nevertheless, the other four uniform methods (Halton, Hammersley, Sobol, and Random-R) have greater performance and can achieve L^2 relative errors of less than 1%. Remarkably, the RAD and RAR-D methods we proposed can further bring down the L^2 relative error below 0.1%.

3.5. Wave equation

In this example, the following one-dimensional wave equation is considered:

$$\begin{aligned}
&\frac{\partial^2 u}{\partial t^2} - 4\frac{\partial^2 u}{\partial x^2} = 0, \quad x \in [0, 1], t \in [0, 1],\\
&u(0, t) = u(1, t) = 0, \quad t \in [0, 1],\\
&u(x, 0) = \sin(\pi x) + \frac{1}{2}\sin(4\pi x), \quad x \in [0, 1],\\
&\frac{\partial u}{\partial t}(x, 0) = 0, \quad x \in [0, 1],
\end{aligned}$$

where the exact solution is given as:

$$u(x, t) = \sin(\pi x)\cos(2\pi t) + \frac{1}{2}\sin(4\pi x)\cos(8\pi t).$$

The solution has a multi-scale behavior in both spatial and temporal directions.

When we test the six uniform sampling methods, the number of residual points are ranged from 1000 to 6000, with an increment of 1000 each time. The Hammersley method achieves the lowest L^2 relative error with the fastest rate (Fig. 6A). When the number of residual points approaches 6000, the Random, Halton, and Hammersley methods can all obtain an L^2 relative error $\sim 10^{-3}$.

To determine the effectiveness of Random-R when using different numbers of residual points, we test the following three scenarios: small (1000 points), medium (4000 points), and large (10 000) sets of residual points (Figs. 6B, C, and D). In the medium case (Fig. 6C), the Random-R attains L^2 relative errors that are magnitudes lower than the Random method. However, in the small and large cases (Figs. 6B and D), the Random-R methods show no advantage over the Random method regardless of the period of resampling. This is because when the number of residual points is small, both the Random and Random-R methods fail to provide accurate predictions. On the other hand, if the number of residual points is large, the predictions by the Random method are already highly accurate, so the Random-R is unable to further improve the accuracy.

Since the optimal sets of k and c for both RAD and RAR-D methods are found to be the same for the Burgers' and the Allen–Cahn equations, in this numerical experiment, we only apply the default settings (i.e., RAD: $k = 1$ and $c = 1$; RAR-D: $k = 2$ and $c = 0$) to investigate the effect of other factors, including the number of residual points for the RAD method and the number of points added to the RAR-D method.

In Fig. 6E, we compare the performance of three nonuniform adaptive sampling methods under the same number of residual points from 1000 to 10 000. We first train the network using 15 000 iterations of Adam and 1000 iterations of L-BFGS, and then after each resampling in RAD or adding new points in RAR-D/RAR-G, we train the network with 1000 iterations of L-BFGS. For the RAR-G and the RAR-D methods, we first train the network with 50% of the final number of the residual points and add 10 residual points each time until reaching the total number of residual points. As we can see from Fig. 6E, the RAD achieves much better results when the number of residual points is small. As the number of residual points increases, the RAR-D method acts more effectively and eventually reaches comparable accuracy to the RAD method. Since the RAD method is more computationally costly than the RAR-D methods with the same number of residual points, we suggest applying the RAD method when the number of residual points is small and the RAR-D method when the number of residual points is large.

We next investigate the RAD method with a different number of residual points (i.e., 1000, 2000, 5000, and 10 000). Fig. 6F illustrates that if we increase the number of residual points, lower L^2 relative error can be achieved but with diminishing marginal effect. We train the network for more than 500 000 iterations to see if the L^2 relative error can further decrease. However, the L^2 relative errors converge and remain relatively stable after 100 000 iterations.

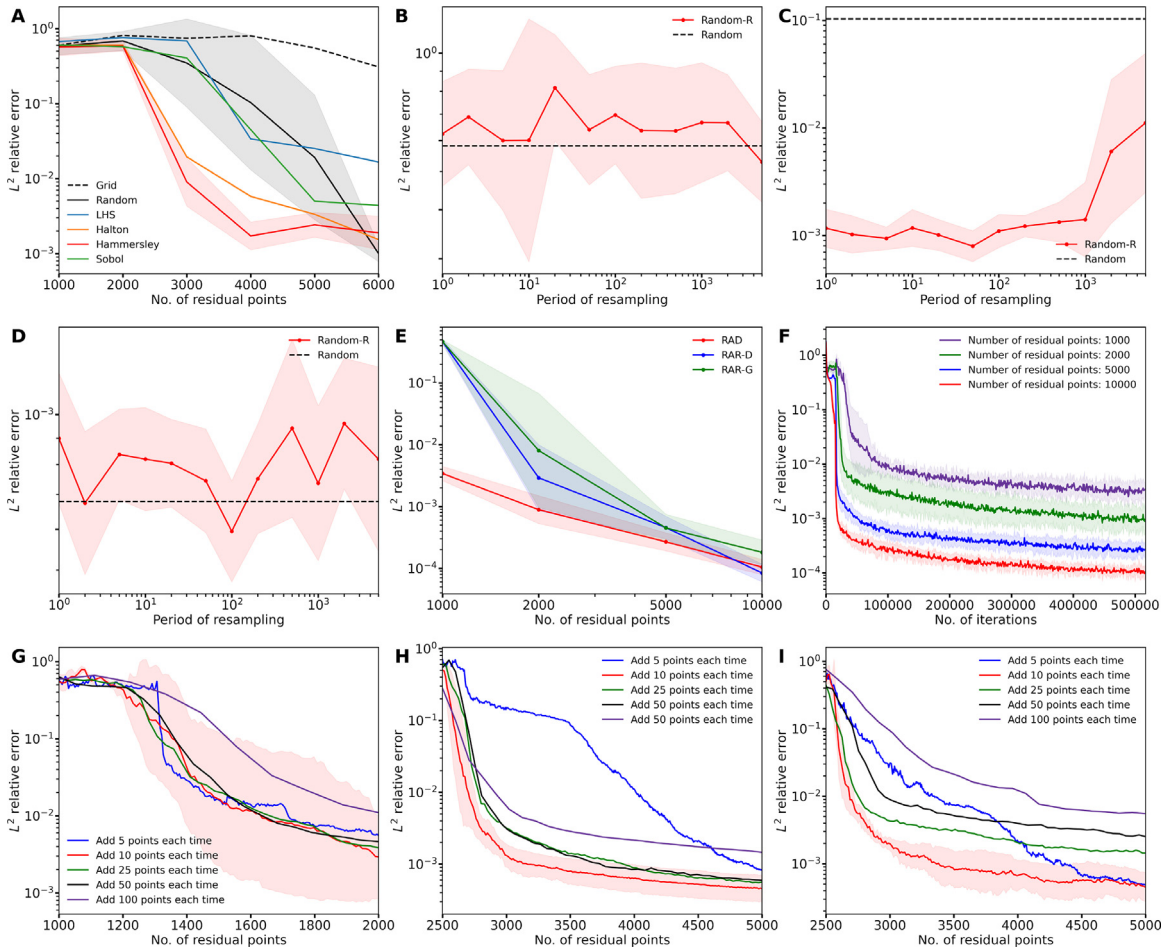

Fig. 6. L^2 relative errors of different sampling methods for the wave equation in Section 3.5. (A) Six uniform sampling with fixed residual points. (B, C, and D) Random-R with different periods of resampling when using (B) 1000 residual points, (C) 4000 residual points, and (D) 10 000 residual points. (E) Comparison among RAD ($k = 1$ and $c = 1$), RAR-D ($k = 2$ and $c = 0$), and RAR-G for different numbers of residual points. (F) The training trajectory of RAD ($k = 1$ and $c = 1$) uses different numbers of residual points. (G and H) Convergence of RAR-D ($k = 2$ and $c = 0$) when adding a different number of new points each time. (G) New points are added starting from 1000 residual points. (H) New points are added starting from 2500 residual points. (I) Convergence of RAR-G when adding a different number of new points each time. New points are added starting from 2500 residual points. The curves and shaded regions represent the geometric mean and one standard deviation of 10 runs. For clarity, only some standard deviations are plotted.

One important factor to consider in the RAR-D and the RAR-G methods is how new points are added. We can either add a small number of residual points each time and prolong the training process or add a large number of residual points each time and shorten the process. In Fig. 6G, we first train the network with 1000 residual points and then add new residual points at different rates until the total number of residual points reaches 2000. After adding new residual points each time, we train the network using 1000 steps of L-BFGS. Likewise, in Fig. 6H, we first train the network with 2500 residual points and add new points at different rates until the total number of residual points reaches 5000. In both cases (Figs. 6G and H) that use the RAR-D methods, we find that the best strategy is to add 10 points each time. However, shown by two red-shaded regions in Figs. 6G and H, the results are more stable when we use a larger number of residual points. Fig. 6I is set up the same way as Fig. 6H but tests the RAR-G method. The best strategy for the RAR-G is identical to that of the RAR-D.

Table 2 outlines the L^2 relative error for the wave equation using all methods when the number of residual points equals 2000. All uniform methods with fixed residual points perform poorly (error $> 50\%$) and fail to approximate the truth values. Random-R, as a special case of the proposed RAD, gives L^2 relative errors of around 1%. The

RAR-D method significantly enhances the prediction accuracy resulting in L^2 relative errors under 0.3%. In addition, the RAD with the default setting of k and c converges to L^2 relative errors under 0.1%.

3.6. Diffusion–reaction equation

The first inverse problem we consider is the diffusion–reaction system as follows:

$$\lambda \frac{d^2u}{dx^2} - k(x)u = f, \quad x \in [0, 1],$$

where $f = \sin(2\pi x)$ is the source term. $\lambda = 0.01$ is the diffusion coefficient, and u is the solute concentration. In this problem, we aim to infer the space-dependent reaction rate $k(x)$ with given measurements on the solution u. The exact unknown reaction rate is

$$k(x) = 0.1 + e^{-0.5\frac{(x-0.5)^2}{0.15^2}}.$$

We aim to learn the unknown function $k(x)$ and solve for $u(x)$ by using eight observations of u, which are uniformly distributed on the domain $x \in [0, 1]$, including two points on both sides of the boundaries. The L^2 relative errors for both the solution u (Figs. 7A, C, and E) and the unknown function k (Figs. 7B, D, and F) are computed. The maximum number of iterations is 50 000 steps of Adam. Figs. 7A and B summarize the performance of all uniform sampling methods. We note that in 1D, the Hammersley and Halton sequences are identical and outperform other uniform methods. We fix the residual points at 15 and compare the Random method with the Random-R method. The L^2 relative errors (Figs. 7C and D) given by the Random-R remain steady, disregarding the changes in the period of resampling, and are approximately the same as that produced by the Random method. This is because the diffusion–reaction system is fairly simple and can be easily handled by uniform sampling methods without resampling.

Next, we compare the Random, RAD, RAR-G, and RAR-D methods with default settings (i.e., RAD: $k = 1$ and $c = 1$; RAR-D: $k = 2$ and $c = 0$) using a different number of residual points. For the random and RAD methods, the maximum number of iterations is 50 000 steps of Adam. For the RAR-G/RAR-D, we first train the neural network with 50% of the total number of residual points for 10 000 steps of Adam; then, we add one point each time and train for 1000 steps of Adam until we meet the total number of residual points. As shown by Figs. 7E and F, the RAD method surpasses other methods and is able to produce low L^2 relative error even when the number of residual points is very small. However, RAR-G and RAR-D are even worse than the Random sampling.

To sum up, we fix the number of residual points at 15 and present the L^2 relative error for both the solution and unknown function in Table 3. The RAD yields the minimum L^2 relative error (0.17% for $u(x)$; 2.76% for $k(x)$). However, due to the simplicity of this PDE problem, some uniform sampling methods, especially the Sobol and Random-R, have comparable performance to the RAD. Generally speaking, we recognize that the uniform sampling methods are adequate when solving this inverse PDE with smooth solutions. Still, the RAD method can further enhance the performance of PINNs, especially when the number of residual points is small.

3.7. Korteweg–de Vries equation

The second inverse problem we solve is the Korteweg–de Vries (KdV) equation:

$$\frac{\partial u}{\partial t} + \lambda_1 u \frac{\partial u}{\partial x} + \lambda_2 \frac{\partial^3 u}{\partial x^3} = 0, \qquad x \in [-1, 1], \quad t \in [0, 1],$$

where λ_1 and λ_2 are two unknown parameters. The exact values for λ_1 and λ_2 are 1 and 0.0025, respectively. The initial condition is $u(x, t = 0) = \cos(\pi x)$, and periodic boundary conditions are used. To infer λ_1 and λ_2, we assume that we have the observations of two solution snapshots $u(x, t = 0.2)$ and $u(x, t = 0.8)$ at 64 uniformly distributed points at each time.

In Fig. 8, the first column (Figs. 8A, D, and G) shows the L^2 relative error of the solution u, while the second column (Figs. 8B, E, and H) and the third column (Figs. 8C, F, and I) illustrate the relative errors for λ_1 and λ_2, respectively. The maximum iteration is 100 000 steps of Adam. Hammersley achieves better accuracy than the other uniform sampling methods. The Sobol and Halton methods behave comparably as these two curves (the yellow and

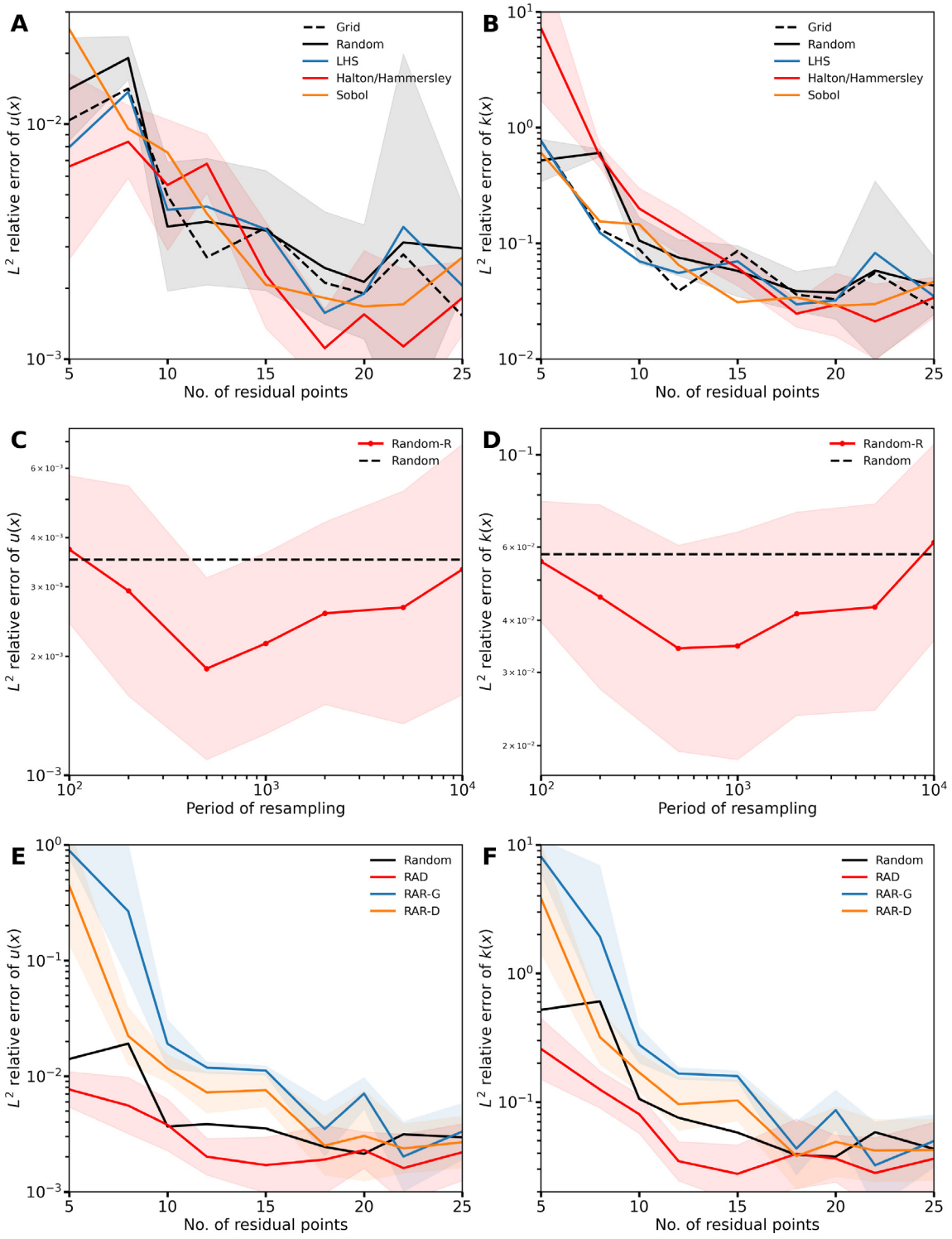

Fig. 7. L^2 relative errors of different sampling methods for u and k in the diffusion–reaction equation in Section 3.6. (A and B) Six uniform sampling with fixed residual points. (C and D) Random-R with different periods of resampling when using 15 residual points. (E and F) Comparison among Random, RAD ($k = 1$ and $c = 1$), RAR-G, and RAR-D ($k = 2$ and $c = 0$) for different numbers of residual points. The curves and shaded regions represent the geometric mean and one standard deviation of 10 runs. For clarity, only some standard deviations are plotted.

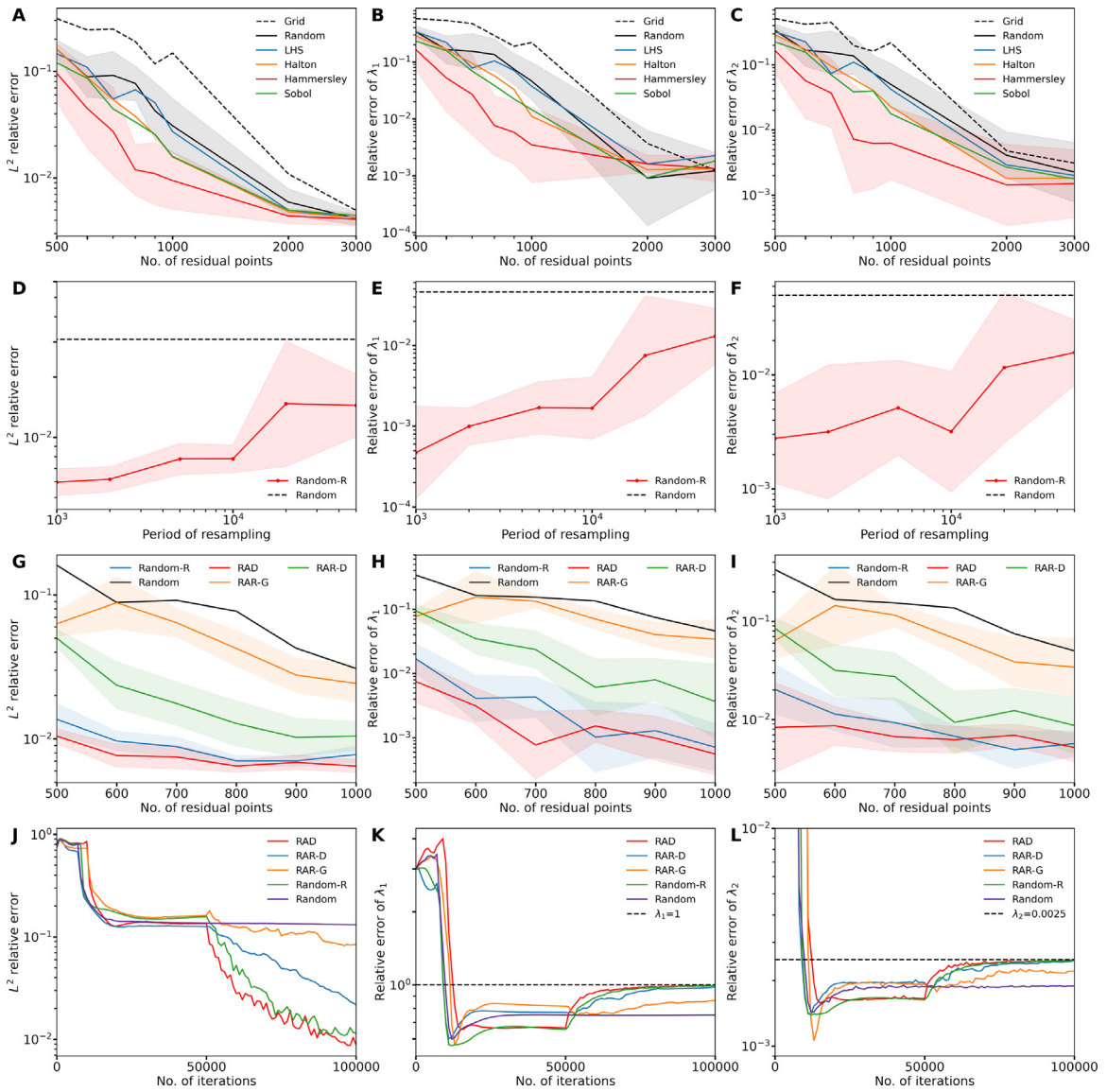

Fig. 8. L^2 relative errors of u and relative errors of λ_1 and λ_2 using different sampling methods for the Korteweg–de Vries equation in Section 3.7. (A, B, and C) Six uniform sampling with fixed residual points. (D, E, and F) Random-R with different periods of resampling when using 1000 residual points. (G, H, and I) Comparison among Random, Random-R, RAD ($k = 1$ and $c = 1$), RAR-G, and RAR-D ($k = 2$ and $c = 0$) for different number of residual points. (J, K, and L) Examples of the training trajectories using Random, Random-R, RAD ($k = 1$ and $c = 1$), RAR-G, and RAR-D ($k = 2$ and $c = 0$) with 600 residual points. The curves and shaded regions represent the geometric mean and one standard deviation of 10 runs. For clarity, only some standard deviations are plotted. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

green curves in Figs. 8A, B, and C) are almost overlapping. Shown in Figs. 8D, E, and F, the Random-R method yields higher accuracy than the Random method by about one order of magnitude in all cases when using 1000 residual points. A smaller period of resampling leads to smaller errors.

Figs. 8G, H, and I compare the Random-R, Random, RAD, RAR-G, and RAR-D methods using the same number of residual points and the total number of iterations. For the Random and the Random-R methods, we train the network for 100 000 steps of Adams. For the RAD methods, we first train the network using 50 000 steps of Adams; then, we resample the residual points and train for 1000 steps of Adams 50 times. In order to fix the total number

of iterations for the RAR-G/RAR-D methods to 100 000, we accordingly adjust the number of new residual points added each time. For example, if the final number of residual points is 500, we first train the network using 250 residual points (i.e., 50% of the total number of residual points) with 50 000 steps of Adams; and we consequently add 5 points and train for 1000 steps of Adams each time. If the final number of residual points is 1000, we first train the network using 500 residual points with 50 000 steps of Adams; and then we add 10 points and train for 1000 steps of Adams each time. As demonstrated by Figs. 8G, H, and I, the RAD method is the best, while the Random-R method is also reasonably accurate. We show one example of the training process (Figs. 8J, K, and L) when the number of residual points is 600 to illustrate the convergence of the solution, λ_1, and λ_2 during training. The resampling strategies, especially the RAD method, achieve the greatest success among all sampling methods.

Table 3 demonstrates the L^2 relative errors for the solution $u(x,t)$ and the relative error of two unknown parameters, λ_1 and λ_2, for all methods when the number of residual points is set at 600. The lowest L^2 relative errors for uniform sampling with fixed points are given by Hammersley ($\sim 5\%$). The Random-R is the second-best method and provides L^2 relative errors of around 1%. With the smallest errors ($< 1\%$) and standard deviations, the RAD method has compelling advantages over all other methods in terms of accuracy and robustness. It is noteworthy that the RAR-D method provides adequate accuracy ($\sim 3\%$) and is less expensive than the Random-R and RAD methods when the number of residual points is the same. Therefore, the RAR-D is also a valuable approach to consider.

Compared with the non-adaptive uniform sampling, in addition to the network training, Random-R and the adaptive sampling methods have the computational overhead to sample the residual points every certain iterations. Here, we test the computational cost for 600 residual points. The training time for 2000 iterations is about 20.28 s on a NVIDIA GeForce RTX 3080 Ti GPU. Random-R takes $\sim 10^{-4}$ seconds to sample once on an AMD Threadripper 3960X CPU, and the sampling time for non-uniform methods (RAD, RAR-G, and RAR-D) is about 0.73 s. In this case, the computational overhead is about 3%, and it can be further reduced if we use a longer resampling period.

4. Conclusions

In this paper, we present a comprehensive study of two categories of sampling for physics-informed neural networks (PINNs), including non-adaptive uniform sampling and adaptive nonuniform sampling. For the non-adaptive uniform sampling, we have considered six methods: (1) equispaced uniform grid (Grid), (2) uniformly random sampling (Random), (3) Latin hypercube sampling (LHS), (4) Halton sequence (Halton), (5) Hammersley sequence (Hammersley), and (6) Sobol sequence (Sobol). We have also considered a resampling strategy for uniform sampling (Random-R). For the adaptive nonuniform sampling, motivated by the residual-based adaptive refinement with greed (RAR-G) [3], we proposed two new residual-based adaptive sampling methods: residual-based adaptive distribution (RAD) and residual-based adaptive refinement with distribution (RAR-D).

We extensively investigated the performance of these ten sampling methods in solving four forward and two inverse problems of partial differential equations (PDEs) with many setups, such as a different number of residual points. Our results show that the proposed RAD and RAR-D significantly improve the accuracy of PINNs by orders of magnitude, especially when the number of residual points is small. RAD and RAR-D also have great advantages for the PDEs with complicated solutions, e.g., the solution of the Burgers' equation with steep gradients and the solution of the wave equation with a multi-scale behavior. A summary of the comparison of these methods can be found in Section 3.1.

Based on our empirical results, we summarize the following suggestions as a practical guideline in choosing sampling methods for PINNs.

- RAD with $k = 1$ and $c = 1$ can be chosen as the default sampling method when solving a new PDE. The hyperparameters k and c can be tuned to balance the points in the locations with large and small PDE residuals.
- RAR-D can achieve comparable accuracy to RAD, but RAR-D is more computationally efficient as it gradually increases the number of residual points. Hence, RAR-D ($k = 2$ and $c = 0$ by default) is preferable for the case with limited computational resources.
- Random-R can be used in situation where adaptive sampling is not allowed, e.g., it is difficult to sample residual points according to a probability density function. The period of resampling should not be chosen as too small or too large.

- A low-discrepancy sequence (e.g., Hammersley) should be considered rather than Grid, Random, or LHS, when we have to use a fixed set of residual points, such as in PINNs with the augmented Lagrangian method (hPINNs) [8].

In this study, we sample residual points in RAD and RAR-D by using a brute-force approach, which is simple, easy to implement, and sufficient for many PDEs. However, for high-dimensional problems, it is computationally expensive, and we need to use other methods, such as generative adversarial networks (GANs) [47], as was done in Ref. [41]. Moreover, the probability of sampling a point $\mathbf{x}$ is only considered as $p(\mathbf{x}) \propto \frac{\varepsilon^k(\mathbf{x})}{\mathbb{E}[\varepsilon^k(\mathbf{x})]} + c$. While this probability works very well in this study, it is possible that there exists another better choice. We can learn a new probability density function by meta-learning, as was done for loss functions of PINNs in Ref. [11].

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgment

This work was supported by the U.S. Department of Energy [DE-SC0022953].

References

[1] 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.
[2] Maziar Raissi, Alireza Yazdani, George Em Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026–1030.
[3] Lu Lu, Xuhui Meng, Zhiping Mao, George Em Karniadakis, DeepXDE: A deep learning library for solving differential equations, SIAM Rev. 63 (1) (2021) 208–228.
[4] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, Liu Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.
[5] Yuyao Chen, Lu Lu, George Em Karniadakis, Luca Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (8) (2020) 11618.
[6] Alireza Yazdani, Lu Lu, Maziar Raissi, George Em Karniadakis, Systems biology informed deep learning for inferring parameters and hidden dynamics, PLoS Comput. Biol. 16 (11) (2020).
[7] Mitchell Daneker, Zhen Zhang, George Em Karniadakis, Lu Lu, Systems biology: Identifiability analysis and parameter identification via systems-biology informed neural networks, 2022, arXiv preprint arXiv:2202.01723.
[8] Lu Lu, Raphaël Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, Steven G. Johnson, Physics-informed neural networks with hard constraints for inverse design, SIAM J. Sci. Comput. 43 (6) (2021).
[9] Guofei Pang, Lu Lu, George Em Karniadakis, fPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019).
[10] Dongkun Zhang, Lu Lu, Ling Guo, George Em Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, J. Comput. Phys. 397 (2019) 108850.
[11] Apostolos F. Psaros, Kenji Kawaguchi, George Em Karniadakis, Meta-learning PINN loss functions, J. Comput. Phys. 458 (2022) 111121.
[12] Jeremy Yu, Lu Lu, Xuhui Meng, George Em Karniadakis, Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114823.
[13] Sifan Wang, Yujun Teng, Paris Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (5) (2021) A3055–A3081.
[14] Sifan Wang, Xinling Yu, Paris Perdikaris, When and why PINNs fail to train: A neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768.
[15] Zixue Xiang, Wei Peng, Xu Liu, Wen Yao, Self-adaptive loss balanced physics-informed neural networks, Neurocomputing (2022).
[16] Levi McClenny, Ulisses Braga-Neto, Self-adaptive physics-informed neural networks using a soft attention mechanism, 2020, arXiv preprint arXiv:2009.04544.
[17] Yiqi Gu, Haizhao Yang, Chao Zhou, SelectNet: Self-paced learning for high-dimensional partial differential equations, J. Comput. Phys. 441 (2021) 110444.

[18] Wensheng Li, Chao Zhang, Chuncheng Wang, Hanting Guan, Dacheng Tao, Revisiting PINNs: Generative adversarial physics-informed neural networks and point-weighting method, 2022, arXiv preprint arXiv:2205.08754.
[19] Xuhui Meng, Zhen Li, Dongkun Zhang, George Em Karniadakis, PPINN: Parareal physics-informed neural network for time-dependent pdes, Comput. Methods Appl. Mech. Engrg. 370 (2020) 113250.
[20] Khemraj Shukla, Ameya D. Jagtap, George Em Karniadakis, Parallel physics-informed neural networks via domain decomposition, J. Comput. Phys. 447 (2021) 110683.
[21] Ameya D. Jagtap, George Em Karniadakis, Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (5) (2020) 2002–2041.
[22] Colby L. Wight, Jia Zhao, Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks, 2020, arXiv preprint arXiv:2007.04542.
[23] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, Michael W Mahoney, Characterizing possible failure modes in physics-informed neural networks, Adv. Neural Inf. Process. Syst. 34 (2021) 26548–26560.
[24] Revanth Mattey, Susanta Ghosh, A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations, Comput. Methods Appl. Mech. Engrg. 390 (2022) 114474.
[25] Katsiaryna Haitsiukevich, Alexander Ilin, Improved training of physics-informed neural networks with model ensembles, 2022, arXiv preprint arXiv:2204.05108.
[26] Sifan Wang, Shyam Sankaran, Paris Perdikaris, Respecting causality is all you need for training physics-informed neural networks, 2022, arXiv preprint arXiv:2203.07404.
[27] Pola Lydia Lagari, Lefteri H Tsoukalas, Salar Safarkhani, Isaac E Lagaris, Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions, Int. J. Artif. Intell. Tools 29 (05) (2020) 2050009.
[28] Suchuan Dong, Naxian Ni, A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks, J. Comput. Phys. 435 (2021) 110242.
[29] Michael D. McKay, Richard J. Beckman, William J. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 42 (1) (2000) 55–61.
[30] Michael Stein, Large sample properties of simulations using Latin hypercube sampling, Technometrics 29 (2) (1987) 143–151.
[31] Il'ya Meerovich Sobol', On the distribution of points in a cube and the approximate evaluation of integrals, Zhurnal Vychislitel'Noi Matematiki I Matematicheskoi Fiziki 7 (4) (1967) 784–802.
[32] John H. Halton, On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals, Numer. Math. 2 (1) (1960) 84–90.
[33] J.M. Hammersley, D.C. Handscomb, Monte-Carlo Methods, Mathuen, Lon, 1964.
[34] Hongwei Guo, Xiaoying Zhuang, Xiaoyu Meng, Timon Rabczuk, Analysis of three dimensional potential problems in non-homogeneous media with deep learning based collocation method, 2020, arXiv preprint arXiv:2010.12060.
[35] Sourav Das, Solomon Tesfamariam, State-of-the-art review of design of experiments for physics-informed deep learning, 2022, arXiv preprint arXiv:2202.06416.
[36] Zhiping Mao, Ameya D. Jagtap, George Em Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Engrg. 360 (2020) 112789.
[37] Mohammad Amin Nabian, Rini Jasmine Gladstone, Hadi Meidani, Efficient training of physics-informed neural networks via importance sampling, Comput.-Aided Civ. Infrastruct. Eng. (2021).
[38] Bastian Zapf, Johannes Haubner, Miroslav Kuchta, Geir Ringstad, Per Kristian Eide, Kent-Andre Mardal, Investigating molecular transport in the human brain from MRI with physics-informed neural networks, 2022, arXiv preprint arXiv:2205.02592.
[39] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, Anuj Karpatne, Rethinking the importance of sampling in physics-informed neural networks, 2022, arXiv preprint arXiv:2207.02338.
[40] Wenhan Gao, Chunmei Wang, Active learning based sampling for high-dimensional nonlinear partial differential equations, 2021, arXiv preprint arXiv:2112.13988.
[41] Kejun Tang, Xiaoliang Wan, Chao Yang, DAS: A deep adaptive sampling method for solving partial differential equations, 2021, arXiv preprint arXiv:2112.14038.
[42] Wei Peng, Weien Zhou, Xiaoya Zhang, Wen Yao, Zheliang Liu, RANG: a residual-based adaptive node generation method for physics-informed neural networks, 2022, arXiv preprint arXiv:2205.01051.
[43] Shaojie Zeng, Zong Zhang, Qingsong Zou, Adaptive deep neural networks methods for high-dimensional partial differential equations, J. Comput. Phys. 463 (2022) 111232.
[44] John M Hanna, Jose V Aguado, Sebastien Comas-Cardona, Ramzi Askri, Domenico Borzacchiello, Residual-based adaptivity for two-phase flow simulation in porous media using physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 396 (2022) 115100.
[45] Shashank Subramanian, Robert M. Kirby, Michael W. Mahoney, Amir Gholami, Adaptive self-supervision algorithms for physics-informed neural networks, 2022, arXiv preprint arXiv:2207.04084.
[46] Melissa E. O'Neill, PCG: A family of simple fast space-efficient statistically good algorithms for random number generation, ACM Trans. Math. Software (2014).
[47] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, Generative adversarial nets, Adv. Neural Inf. Process. Syst. 27 (2014).
[48] Bengt Fornberg, Natasha Flyer, Fast generation of 2-D node distributions for mesh-free PDE discretizations, Comput. Math. Appl. 69 (7) (2015) 531–544.