Numerical Algorithms (2024) 96:1163–1187
https://doi.org/10.1007/s11075-024-01826-8

ORIGINAL PAPER

Stabilization of parareal algorithms for long-time computation of a class of highly oscillatory Hamiltonian flows using data

Rui Fang[1] · Richard Tsai[2]

Received: 1 September 2023 / Accepted: 21 March 2024 / Published online: 3 April 2024

Abstract

Applying parallel-in-time algorithms to multiscale Hamiltonian systems to obtain stable long-time simulations is very challenging. In this paper, we present novel data-driven methods aimed at improving the standard parareal algorithm developed by Lions et al. in 2001, for multiscale Hamiltonian systems. The first method involves constructing a correction operator to improve a given inaccurate coarse solver through solving a Procrustes problem using data collected online along parareal trajectories. The second method involves constructing an efficient, high-fidelity solver by a neural network trained with offline generated data. For the second method, we address the issues of effective data generation and proper loss function design based on the Hamiltonian function. We show proof-of-concept by applying the proposed methods to a Fermi-Pasta-Ulam (FPU) problem. The numerical results demonstrate that the Procrustes parareal method is able to produce solutions that are more stable in energy compared to the standard parareal. The neural network solver can achieve comparable or better runtime performance compared to numerical solvers of similar accuracy. When combined with the standard parareal algorithm, the improved neural network solutions are slightly more stable in energy than the improved numerical coarse solutions.

Keywords Hamiltonian systems · Parareal algorithms · Machine learning · Neural networks

✉ Rui Fang
rfang@utexas.edu

Richard Tsai
ytsai@math.utexas.edu

1 Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX 78712, USA

2 Department of Mathematics and Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX 78712, USA

1 Introduction

Hamiltonian systems are ubiquitous in astronomy, molecular dynamics, classical mechanics, and theoretical physics. We concentrate on the separable Hamiltonian case

$$H(\mathbf{p}, \mathbf{q}) = \frac{1}{2}\mathbf{p}^T M^{-1}\mathbf{p} + U(\mathbf{q}), \quad \mathbf{p}, \mathbf{q} \in \mathbb{R}^d \tag{1}$$

where $\mathbf{p}$ and $\mathbf{q}$ are respectively the generalized momentum and position in a d-dimensional space, M a diagonal matrix denoting the masses, and U a smooth scalar function depending on the position $\mathbf{q}$. Physically, the Hamiltonian is interpreted as the total energy of a system, consisting of the kinetic energy $K(\mathbf{p}) := \frac{1}{2}\mathbf{p}^T M^{-1}\mathbf{p}$ and the potential energy $U(\mathbf{q})$. The dynamics of the system is given by Hamilton's equations

$$\dot{\mathbf{q}} = H_{\mathbf{p}} = M^{-1}\mathbf{p}, \quad \dot{\mathbf{p}} = -H_{\mathbf{q}} = -\nabla U(\mathbf{q}). \tag{2}$$

Geometric integrators (methods that preserve geometric properties of the exact flow) such as the velocity Verlet method are frequently used to simulate Hamiltonian flows [1]. It is proved that the preservation of geometric structures may greatly improve long-time numerical integration compared with general-purpose methods.

Even with the improved long-time stability and accuracy of geometric integrators, the computational complexity remains high for many physical applications. In particular, for systems with multiple time scales, accurate long-time integration is very difficult because the small stepsize required for stable integration of the fast motions lead to a large number of time steps. Computational multiscale algorithms aim at reducing computational complexity by exploiting the underlying multiscale structures. For example, for systems with sufficiently wide separation of scales and certain homogeneity and ergodic properties, the heterogeneous multiscale methods (HMM) can compute the effective systems with significantly reduced complexity [2].

Recently, due to the increased availability of parallel processors in modern supercomputers, there has been rising interest in developing time-domain parallelization algorithms to reduce the wall-clock computation time for time-dependent problems. As the algorithm of choice, the work in this paper will rely on the parareal method, a parallel-in-time algorithm introduced by Lions et al. [3]. To set up, the time domain $[0, T]$ is divided into N subintervals of length $\Delta t = T/N$. The parareal method involves two numerical solvers that advance a solution by Δt: an efficient but low-fidelity coarse solver, denoted by $C_{\Delta t}$, and an accurate but expensive fine solver, denoted by $F_{\Delta t}$. The coarse solver solutions are iteratively corrected by fine solver solutions computed on smaller time intervals in parallel. Formally, letting $\mathbf{u}_n^{(k)}$ denote the solution computed at iteration k and time $t_n = n\Delta t$, the parareal iterations are given by

$$\begin{aligned} \mathbf{u}_{n+1}^{(k+1)} &= C_{\Delta t}\mathbf{u}_n^{(k+1)} + \left(F_{\Delta t}\mathbf{u}_n^{(k)} - C_{\Delta t}\mathbf{u}_n^{(k)}\right), \\ \mathbf{u}_0^{(k)} &= \mathbf{u}_0, \quad \mathbf{u}_{n+1}^{(0)} = C_{\Delta t}\mathbf{u}_n^{(0)}, \quad k = 0, 1, 2, ...; n = 0, 1, ..., N-1. \end{aligned} \tag{3}$$

Ideally, with a sufficiently accurate coarse solver, the iterations will quickly converge to the sequential solution computed by the fine solver $F^n_{\Delta t}\mathbf{u}_0$. However, a closer analysis reveals that the convergence relies on the stability of the parareal iterations, and the standard parareal is only stable for dissipative problems [4, 5]. For oscillatory and hyperbolic problems (such as Hamiltonian systems), the standard parareal scheme is known to perform badly because the convergence restricts the length of integration time [6]. To be more specific, let $\mathbf{e}^{(k)}_n := \mathbf{u}^{(k)}_n - F^n_{\Delta t}\mathbf{u}_0$. The bound for the amplification $\mathbf{e}^{(k+1)}_n/\mathbf{e}^{(k)}_n$ depends on the sum $\sum_{j=0}^{n-k-2} \|C_{\Delta t}\|^j$. For dissipative problems, $\|C_{\Delta t}\| < 1$, and so the sum can be bounded by some constant independent of n. In contrast, for purely hyperbolic problems, $\|C_{\Delta t}\|$ is close to 1 and the sum will be proportional to n, which causes the iterations to become unstable.

Over the past years, many efforts have been devoted to developing stable parareal schemes for oscillatory and hyperbolic problems. For example, Dai et al. [7] proposed a symmetric variant of the parareal scheme and coupled it with projections to the constant energy manifold. Another class of methods was developed based on the idea of making correction to parareal solutions using data. We will review more of these works in Section 2.

In this paper, we will focus on multiscale Hamiltonian systems where $H = H_\epsilon$. Here, ϵ is a small parameter indicating the small time or length scale. Our aim is to improve the computational efficiency for long-time simulations of multiscale Hamiltonian systems by leveraging recent advancement in machine learning (ML) and parallel-in-time algorithms. Specifically, we would like to develop data-driven methods to stabilize the standard parareal scheme. Our idea is to use an improved solver $\Phi^{\theta,k}_{\Delta t}$ in place of $C_{\Delta t}$ in (4). The superscript θ in $\Phi^{\theta,k}_{\Delta t}$ denotes the unknown parameters defining the solver. The superscript k indicates this solver may depend on the iteration. We propose two approaches to construct $\Phi^{\theta,k}_{\Delta t}$:

1. Enhance an existing coarse solver $C_{\Delta t}$ through a correction operator that aligns the "phase" in the coarse solutions $C_{\Delta t}\mathbf{u}^{(k)}_n$ and fine solutions $F_{\Delta t}\mathbf{u}^{(k)}_n$ for each iteration. In other words, $\Phi^{\theta,k}_{\Delta t} := \Psi^{(k)}_{\Delta t} \circ C_{\Delta t} \approx F_{\Delta t}$, where $\Psi^{(k)}_{\Delta t}$ is the correction operator constructed from data collected *online* during parareal iterations. Because $\Psi^{(k)}_{\Delta t}$ is obtained by solving an orthogonal Procrustes problem, this approach is named the Procrustes parareal method. The Procrustes parareal iteration is given by
$$\mathbf{u}^{(k+1)}_{n+1} = \Psi^{(k)}_{\Delta t} \circ C_{\Delta t}\mathbf{u}^{(k+1)}_n + \left(F_{\Delta t}\mathbf{u}^{(k)}_n - \Psi^{(k)}_{\Delta t} \circ C_{\Delta t}\mathbf{u}^{(k)}_n\right). \tag{4}$$

2. Approximate the fine solver (namely the solution map with a fixed stepsize Δt) using a neural network (NN), i.e., $\Phi^{\theta,k}_{\Delta t} := \Phi^{\mathrm{NN}}_{\Delta t} \approx F_{\Delta t}$ where $\Phi^{\mathrm{NN}}_{\Delta t}$ stands for a NN solver. Unlike in the Procruste parareal approach, the NN solver is constructed using *offline* training data. For this approach, we will address the issue of suitable training data generation and loss function design. The parareal iteration with an NN solver is as follows:
$$\mathbf{u}^{(k+1)}_{n+1} = \Phi^{\mathrm{NN}}_{\Delta t}\mathbf{u}^{(k+1)}_n + \left(F_{\Delta t}\mathbf{u}^{(k)}_n - \Phi^{\mathrm{NN}}_{\Delta t}\mathbf{u}^{(k)}_n\right). \tag{5}$$

The paper is organized as follows. Section 2 presents the Procrustes parareal approach. Section 3 presents the neural network approach for approximating the solution map. Section 4 presents a case study for the Fermi-Pasta-Ulam (FPU) problem. We then conclude our findings in Section 5.

2 Enhancing the coarse solvers by data

In order to address the instability issue in the standard parareal iterations, several methods involving a correction to bridge the gap between fine and coarse solutions were proposed in the past. For example, Farhat and Chandesris [8] used a Newton-type iteration to reduce the jumps between the fine and coarse solutions. In [5], Ariel et al. proposed the θ-parareal scheme, which uses an interpolation-based linear operator to enhance the coarse solver for oscillatory systems. In [9], Nguyen and Tsai focused on the second-order wave equations and developed a correction operator based on minimizing the wave energy residual of the fine and coarse solutions. The resulting correction successfully stabilizes the parareal iterations by aligning the "phase" in the wave fields computed by the fine and coarse solver respectively. Later, the same authors proposed in [10] a deep learning approach to enhance the coarse solver to reduce the "phase" errors in wave propagation.

The success for the wave equations in [9] directly inspires the development of a similar approach for a class of Hamiltonian systems where the notion of "phase" can be suitably defined.

In this section, we first provide our definition of "phase" for Hamiltonian systems, and then introduce the procedure to obtain a correction operator $\Psi_{\Delta t}^{(k)}$ from data computed in previous iterations

$$\{C_{\Delta t}\mathbf{u}_n^{(k')}, F_{\Delta t}\mathbf{u}_n^{(k')}\} \quad n = 0, 1, \cdots, N-1;\; k' = 0, 1, \cdots, k-1.$$

2.1 A practical notion of "phase" for Hamiltonian systems

For integrable Hamiltonian systems such as harmonic oscillators or Kepler systems, the "phase" can be naturally defined as the angle variables from the action-angle coordinates. For non-integrable systems, such as the FPU problem, where action-angle coordinates are not available, we need alternative definitions for "phase."

Because phase is an angle-like object, for a separable Hamiltonian (1), it is natural to consider a transform function Λ, which maps the energy level set to a hypersphere, whose radius satisfies

$$\|\Lambda\left([\mathbf{p},\mathbf{q}]\right)\|_2^2 = H\left(\mathbf{p},\mathbf{q}\right) + \text{constant}. \tag{6}$$

This way, we can define the "phase" difference between $[\mathbf{p}_1, \mathbf{q}_1]$ and $[\mathbf{p}_2, \mathbf{q}_2]$ as the angle between the transformed vectors $\Lambda\left([\mathbf{p}_1, \mathbf{q}_1]\right)$ and $\Lambda\left([\mathbf{p}_2, \mathbf{q}_2]\right)$. We call Λ the energy transform because the l_2 norm of the transformed vector is related to the energy.

The specific form of Λ and the pseudo-inverse $\Lambda^\dagger$ depends on the Hamiltonian. Noticeably, not all Hamiltonian functions allow a valid definition of Λ. For example, for a 1D harmonic oscillator whose Hamiltonian is $H(p,q) = \frac{1}{2}p^2 + \frac{1}{2}q^2$,

$$\Lambda\left(\begin{bmatrix} p \\ q \end{bmatrix}\right) := \begin{bmatrix} p/\sqrt{2} \\ q/\sqrt{2} \end{bmatrix}, \quad \Lambda^\dagger\left(\begin{bmatrix} \tilde{p} \\ \tilde{q} \end{bmatrix}\right) := \begin{bmatrix} \sqrt{2}\tilde{p} \\ \sqrt{2}\tilde{q} \end{bmatrix}. \tag{7}$$

For a 2D Kepler problem where $H(\mathbf{p},\mathbf{q}) = \frac{1}{2}\mathbf{p}^T\mathbf{p} - \frac{1}{\|\mathbf{q}\|}$, Λ does not exist because of the negative potential term.

In this paper, we will work with the FPU problem and will show the corresponding definition of Λ in Section 4.

2.2 The Procrustes parareal

In the following, we use $\mathbf{u}$ to represent the concatenated vector $[\mathbf{p},\mathbf{q}]$. Assuming Λ and $\Lambda^\dagger$ are both known, we define the correction operator as

$$\Psi_{\Delta t}^{(k)} := \Lambda^\dagger \circ \Omega_{\Delta t}^{(k)} \circ \Lambda, \tag{8}$$

where $\Omega_{\Delta t}^{(k)}$ is an orthogonal transformation to be determined. We see the advantage of defining $\Lambda(\mathbf{u})$—any orthogonal transformation on $\Lambda(\mathbf{u})$ preserves $H(\mathbf{u})$. Hence, the correction operator preserves the energy.

The Procrustes parareal method is given as follows: (function composition symbols are left out for brevity)

$$\begin{aligned} \mathbf{u}_{n+1}^{(k+1)} &= \Lambda^\dagger\Omega_{\Delta t}^{(k)}\Lambda C_{\Delta t}\mathbf{u}_n^{(k+1)} + \left(F_{\Delta t}\mathbf{u}_n^{(k)} - \Lambda^\dagger\Omega_{\Delta t}^{(k)}\Lambda C_{\Delta t}\mathbf{u}_n^{(k)}\right), \\ \mathbf{u}_0^{(k)} &= \mathbf{u}_0, \quad \mathbf{u}_{n+1}^{(0)} = C_{\Delta t}\mathbf{u}_n^{(0)}, \quad k = 0, 1, 2, ...; n = 0, 1, ..., N-1. \end{aligned} \tag{9}$$

Here, $\Omega_{\Delta t}^{(k)}$ is obtained by solving the orthogonal Procrustes problem

$$\begin{aligned} \Omega_{\Delta t}^{(k)} &:= \arg\min_{\Omega} \sum_{n=0}^{N-1} \|f_n - \Omega g_n\|_2^2 \quad \text{s.t.} \quad \Omega\Omega^T = \Omega^T\Omega = I, \\ f_n &:= \Lambda F_{\Delta t}\mathbf{u}_n^{(k)}, \\ g_n &:= \Lambda C_{\Delta t}\mathbf{u}_n^{(k)}. \end{aligned} \tag{10}$$

The geometric interpretation is to minimize the sum of the phase errors between fine and coarse solutions computed along the trajectory from the last iteration. Hence, $\Omega_{\Delta t}^{(k)}$ is referred to as the phase corrector.

We follow the standard way to solve the orthogonal Procrustes problem, which uses the singular value decomposition (SVD) of the correlation matrix $M := FG^T$. Here,

$F := [f_0 \; f_1 \; \cdots \; f_{N-1}]$, $G := [g_0 \; g_1 \; \cdots \; g_{N-1}]$. Let $U\Sigma V^T$ be the SVD of M. If M has full rank, then the minimizer is uniquely $\Omega_* = UV^T$. We refer readers to [11] for more details of the Procrustes problem.

The pseudo-code for the Procrustes parareal method is provided in Algorithm 1.

Algorithm 1 Procrustes parareal method

```
Require: initial state $\mathbf{u}_0$, number of time intervals $N$, number of iterations $K$, coarse solver $C_{\Delta t}$, fine solver $F_{\Delta t}$, map from phase space variables to energy components $\Lambda$, map from energy components to phase space variables $\Lambda^\dagger$
Ensure: solutions $\mathbf{u}_n^{(k)}$ for $k = 0, 1, ...; n = 0, 1, ..., N$
  Compute initial solution:
  $\mathbf{u}_0^{(0)} = \mathbf{u}_0$
  for $n = 0 : N - 1$ do
      $\mathbf{u}_{n+1}^{(0)} = C_{\Delta t}\mathbf{u}_n^{(0)}$
  end for

  $k = 0$
  while $k \leq K$ do
      Compute coarse and fine solutions in parallel:
      for $n = 0 : N - 1$ do
          $f_n = F_{\Delta t}\mathbf{u}_n^{(k)}$
          $g_n = C_{\Delta t}\mathbf{u}_n^{(k)}$
      end for

      Construct data matrices and solve the Procrustes problem:
      $F = [\Lambda(f_0) \; \Lambda(f_1) \; \cdots \; \Lambda(f_{N-1})]$
      $G = [\Lambda(g_0) \; \Lambda(g_1) \; \cdots \; \Lambda(g_{N-1})]$
      $M = FG^T$
      $[U, \Sigma, V] = \mathrm{SVD}(M)$
      $\Omega = UV^T$

      Compute solution of current iteration sequentially:
      $\mathbf{u}_0^{(k+1)} = \mathbf{u}_0$
      for $n = 0 : N - 1$ do
          $\mathbf{u}_{n+1}^{(k+1)} = \Lambda^\dagger\Omega\Lambda C_{\Delta t}\mathbf{u}_n^{(k+1)} + \left(F_{\Delta t}\mathbf{u}_n^{(k)} - \Lambda^\dagger\Omega\Lambda C_{\Delta t}\mathbf{u}_n^{(k)}\right)$
      end for
      $k = k + 1$
  end while
```

3 Neural network approximation of the solution map

In this section, we present our second data-driven approach, which is to use a neural network (NN) to approximate the solution map $F_{\Delta t}$. To construct the NN solver $\Phi_{\Delta t}^{\mathrm{NN}}$, the main task is to solve an optimization problem

$$\Phi_{\Delta t}^{\mathrm{NN}} = \underset{\Phi_{\Delta t}^{\mathrm{NN}} \in X}{\arg\min} \frac{1}{|\mathcal{D}_0|} \sum_{\mathbf{u}_0 \in \mathcal{D}_0} l\left(\mathbf{u}_0, \Phi_{\Delta t}^{\mathrm{NN}}, F_{\Delta t}\right), \tag{11}$$

where X is a function space determined by the network architecture, $\mathcal{D}_0$ a set of input data points, and $l\left(\mathbf{u}_0, \Phi_{\Delta t}^{\mathrm{NN}}, F_{\Delta t}\right)$ the misfit term for each data point $\mathbf{u}_0$ given the reference solution map $F_{\Delta t}$ and the approximated map $\Phi_{\Delta t}^{\mathrm{NN}}$. We describe our setup for each of these components as follows.

3.1 Choice of neural network architecture

For simplicity, we choose fully connected residual networks (ResNets). Compared to a regular multilayer perceptron, the residual network adds skip connections between pairs of hidden layers.

Let L denote the number of hidden layers and n the number of nodes per hidden layer. The layer outputs are defined as follows:

$$\begin{aligned}
\text{input layer:}\quad & y^{(0)} := x \in \mathbb{R}^{2d}, \\
\text{1st hidden layer:}\quad & y^{(1)} := \sigma(W^{(1)} y^{(0)} + b^{(1)}) \in \mathbb{R}^{n}, \\
l\text{-th hidden layer:}\quad & y^{(l)} := y^{(l-1)} + \frac{1}{L}\sigma(W^{(l)} y^{(l-1)} + b^{(l)}) \in \mathbb{R}^{n}, \quad l = 2, ..., L \\
\text{output layer:}\quad & y^{(L+1)} := W^{(L+1)} y^{(L)} + b^{(L+1)} \in \mathbb{R}^{2d}.
\end{aligned} \tag{12}$$

Here, $W^{(l)}$ and $b^{(l)}$, $l = 1, ..., L+1$ are weights and biases to be determined through the training procedure. We use the Exponential Linear Unit (ELU) [12] for the nonlinear activation function σ. Note that we adopt a scaling factor $1/L$ for the hidden layers with skip connections. This technique was proposed in [13] to make the network performance more robust against hyperparameter change.

3.2 Design of misfit term

The function to be approximated is a solution map that maps a phase space state to another state. This allows us to use a sequence of successive time steps to construct the misfit term. Suppose that for an input $\mathbf{u}_0$ and a sequence length S, we generate $\{\mathbf{u}_i\}_{1\leq i\leq S}$, $\mathbf{u}_i = (F_{\Delta t})^i\, \mathbf{u}_0$ as the target sequence and $\{\tilde{\mathbf{u}}_i\}_{1\leq i\leq S}$, $\tilde{\mathbf{u}}_i = \left(\Phi_{\Delta t}^{\mathrm{NN}}\right)^i \mathbf{u}_0$ as the approximated sequence. The misfit is then computed between the approximated sequence and the target sequence

$$l\left(\mathbf{u}_0, \Phi_{\Delta t}^{\mathrm{NN}}, F_{\Delta t}\right) = \frac{1}{S}\sum_{i=1}^{S} \operatorname{diff}(\mathbf{u}_i, \tilde{\mathbf{u}}_i). \tag{13}$$

We remark that because the network is applied recursively for obtaining $\tilde{\mathbf{u}}_i$, this multi-step loss essentially makes training the network like training a recurrent neural network.

There are several ways to measure the difference between $(\mathbf{u}_i, \tilde{\mathbf{u}}_i)$. One common approach is to use the Euclidean metric of $\mathbb{R}^{2d}$. This is known as the mean squared error

$$\mathrm{MSE}\,(\mathbf{u}_i, \tilde{\mathbf{u}}_i) = \|\mathbf{u}_i - \tilde{\mathbf{u}}_i\|_2^2\,. \tag{14}$$

The Euclidean metric puts equal weights on $\mathbf{p}$ and $\mathbf{q}$ components of $\mathbf{u}$. While minimizing the mean squared error aligns with the goal of reducing the trajectory error, it often leads to imbalanced energy error because the Hamiltonian function does not always weight $\mathbf{p}$ and $\mathbf{q}$ similarly. Therefore, to naturally balance the components based on the Hamiltonian, we adopt the energy transform Λ as defined in Section 2.1. We define the energy balanced error

$$\mathrm{EBE}\,(\mathbf{u}_i, \tilde{\mathbf{u}}_i) = \|\Lambda\mathbf{u}_i - \Lambda\tilde{\mathbf{u}}_i\|_2^2\,. \tag{15}$$

To put together, suppose we use the energy balanced error, the misfit term for an initial state $\mathbf{u}_0$ and a sequence length S is given by

$$l\left(\mathbf{u}_0, \Phi_{\Delta t}^{\mathrm{NN}}, F_{\Delta t}\right) = \frac{1}{S}\sum_{i=1}^{S}\left\|\Lambda\left((F_{\Delta t})^i\,\mathbf{u}_0\right) - \Lambda\left(\left(\Phi_{\Delta t}^{\mathrm{NN}}\right)^i\mathbf{u}_0\right)\right\|_2^2. \tag{16}$$

3.3 Generation of input data set

The next problem is to generate a proper set of initial conditions $\mathbf{u}_0$ for training. Unlike in the Procrustes parareal approach, where the data are collected online during parareal iterations, here, we have to generate training data offline to construct an effective NN solver.

In order to understand "what is a reasonable distribution to sample in the phase space," we regard the misfit term in the optimization problem (11) as the mean error over a continuous distribution of $\mathbf{u}$ in the phase space:

$$\frac{1}{|\mathcal{D}_0|}\sum_{\mathbf{u}\in\mathcal{D}_0} l\left(\mathbf{u}, \Phi_{\Delta t}^{\mathrm{NN}}, F_{\Delta t}\right) \approx \int_{\mathbb{R}^{2d}} l\left(\mathbf{u}, \Phi_{\Delta t}^{\mathrm{NN}}, F_{\Delta t}\right) d\mu(\mathbf{u}). \tag{17}$$

It is thus natural to consider using a relevant invariant measure of the Hamiltonian flow for μ.

Suppose we are interested in simulations of the Hamiltonian flow with a fixed total energy. Then, we should consider sampling an invariant measure on an energy level set

$$\mathcal{M}_{H_0} := \left\{(\mathbf{p}, \mathbf{q}) \in \mathbb{R}^{2d} \mid H(\mathbf{p}, \mathbf{q}) = H_0\right\}. \tag{18}$$

However, the numerical approximations may not preserve the total energy. To start, the accurate fine solver used to approximate the true solution map is a symplectic integrator for which exact energy preservation is not possible. In addition, there is no guarantee for energy preservation by the general NN solver considered in this paper.

Notice that by Liouville's theorem, the Hamiltonian flow preserves phase space volume. Then, we can construct an invariant measure in $\mathbb{R}^{2d}$ that concentrates on the chosen energy level H_0 as follows, using the coarea formula:

$$\begin{aligned}&\int_{\mathbb{R}^{2d}} l\left(\mathbf{u}, \Phi_{\Delta t}^{\mathrm{NN}}, F_{\Delta t}\right) \exp^{-(H(\mathbf{u})-H_0)^2/2\sigma^2} d\mathbf{u} \\ &= \int_{\mathbb{R}} \exp^{-(E-H_0)^2/2\sigma^2} \int_{\{H(\mathbf{u})=E\}} l\left(\mathbf{u}, \Phi_{\Delta t}^{\mathrm{NN}}, F_{\Delta t}\right) \frac{dS}{|\nabla H(\mathbf{u})|} dE.\end{aligned} \tag{19}$$

In other words, one can separately sample the invariant densities on the energy level sets and the Gaussian density in the normal directions on the energy level sets.

Motivated by this observation, we propose a novel sampling algorithm called HMC-H_0. The name comes from its resemblance to the Hamiltonian Monte-Carlo (HMC) algorithm [14]. Starting with $\mathbf{q} = \mathbf{q}_0$, we generate a chain of points by repeating the following two steps:

1. Momentum refreshment: randomly sample $\mathbf{p}$ from the hypersphere defined by

$$\left\{\mathbf{p} \in \mathbb{R}^d \mid \mathbf{p}^T M^{-1} \mathbf{p} = 2\left(H_0 - U(\mathbf{q})\right)\right\} \tag{20}$$

2. Time integration: $(\mathbf{p}, \mathbf{q}) \leftarrow F_{\delta t}(\mathbf{p}, \mathbf{q})$

Note that our approach is different from the original HMC algorithm in the momentum refreshment step. In the original HMC, $\mathbf{p}$ is randomly sampled from a Gaussian distribution independent of current $\mathbf{q}$, whereas in our approach, the distribution depends on $\mathbf{q}$ and a fixed H_0.

We shall compare the HMC-H_0 algorithm to the following naive approach, combining random sampling in the momentum space and generating trajectories in the phase space using the flow. We first sample a set of momenta, then using the sampled momenta and $\mathbf{q}_0$, we generate an ensemble of trajectories by flowing the points for a duration of time. The points along the trajectories are collected. This is a naïve attempt to sample the Liouville measures on the energy level sets, assuming ergodicity of the flows. We call this algorithm TrajEnsemble-H_0.

Full descriptions of the algorithms are given in Algorithms 2 and 3. For both algorithms, we can leverage parallel computation to obtain a large number of data samples.

4 Case study: the Fermi-Pasta-Ulam problem

We consider the Fermi-Pasta-Ulam (FPU) problem as a model problem to demonstrate properties of our proposed methods. All code and data accompanying the experiments are publicly available at https://github.com/tsai-lab-ut/multiscale-hamiltonian.

First studied in 1955, the FPU problem [15] describes a simple yet important model for nonlinear physics, which exhibits unexpected dynamical behaviors after

Algorithm 2 HMC-H_0

```
Require: target energy $H_0$, initial position $\mathbf{q}_0 \in \mathbb{R}^d$, standard deviation $\sigma$, mass matrix $M$, number of
  chains $N_{\text{chains}}$, number of transitions per chain $N_{\text{trans}}$, integration time $\delta t$, reference solution map $F_{\delta t}$
Ensure: a set of phase space points $\{(\mathbf{p}^{i,j}, \mathbf{q}^{i,j})\}_{1 \le i \le N_{\text{chains}}, 1 \le j \le N_{\text{trans}}}$
  for $i = 1 : N_{\text{chains}}$ do
    $\mathbf{q}^{i,0} = \mathbf{q}_0$
    for $j = 1 : N_{\text{trans}}$ do
      Step 1: Momentum refreshment
      $K' = 0$
      while $K' \le 0$ do
        sample $K' \sim \mathcal{N}(H_0 - U(\mathbf{q}^{i,j-1}), \sigma^2)$
      end while
      sample $\tilde{\mathbf{p}}' \sim \mathcal{U}(\mathbb{S}^{d-1})$
      $\mathbf{p}' = \sqrt{2K'} M^{1/2} \tilde{\mathbf{p}}'$

      Step 2: Time integration
      $\mathbf{p}^{i,j}, \mathbf{q}^{i,j} = F_{\delta t}(\mathbf{p}', \mathbf{q}^{i,j-1})$
    end for
  end for
```

Algorithm 3 TrajEnsemble-H_0

```
Require: target energy $H_0$, initial position $\mathbf{q}_0 \in \mathbb{R}^d$, standard deviation $\sigma$, mass matrix $M$, number of
  energy level sets $N_{\text{levelsets}}$, number of trajectories per level set $N_{\text{traj}}$, length of trajectory $L$, stepsize of
  trajectory $\delta t$, reference solution map $F_{\delta t}$
Ensure: a set of phase space points $\{(\mathbf{p}^{i,j,k}, \mathbf{q}^{i,j,k})\}_{1 \le i \le N_{\text{levelsets}}, 1 \le j \le N_{\text{traj}}, 1 \le k \le L}$
  for $i = 1 : N_{\text{levelsets}}$ do
    $K^i = 0$
    while $K^i \le 0$ do
      sample $K^i \sim \mathcal{N}(H_0 - U(\mathbf{q}_0), \sigma^2)$
    end while
    for $j = 1 : N_{\text{traj}}$ do
      sample $\tilde{\mathbf{p}}^{i,j} \sim \mathcal{U}(\mathbb{S}^{d-1})$
      $\mathbf{p}^{i,j} = \sqrt{2K^i} M^{1/2} \tilde{\mathbf{p}}^{i,j}$
      for $k = 1 : L$ do
        $\mathbf{p}^{i,j,k}, \mathbf{q}^{i,j,k} = F_{\delta t}^k(\mathbf{p}^{i,j}, \mathbf{q}_0)$
      end for
    end for
  end for
```

long enough integration time. The model involves a chain of particles connected by springs that obey Hooke's law but with a weak nonlinear perturbation. Here, we adopt a version of the problem presented in [1]. Suppose there are $2m$ mass points connected by alternating soft nonlinear springs and stiff linear springs. The variables $q_1, \cdots, q_{2m}$ ($q_0 = q_{2m+1} = 0$) represent the displacements of the mass points from equilibrium, and $p_i = dq_i/dt$ represent velocities. The Hamiltonian of this system is given by

$$H(\mathbf{p}, \mathbf{q}) = \frac{1}{2} \sum_{i=1}^{m} \left(p_{2i-1}^2 + p_{2i}^2 \right) + \frac{\omega^2}{4} \sum_{i=1}^{m} (q_{2i} - q_{2i-1})^2 + \sum_{i=0}^{m} (q_{2i+1} - q_{2i})^4 , \tag{21}$$

where $\omega \gg 1$ is the frequency of the stiff linear springs.

The dynamics of such a system has different behaviors on several different time scales. On the smallest time scale $\mathcal{O}(\omega^{-1})$, the linear springs show almost-harmonic oscillations with period close to π/ω. On the time scale $\mathcal{O}(\omega^0)$, the motion of the nonlinear springs becomes apparent. On the time scale $\mathcal{O}(\omega)$, there is slow energy exchange among the stiff springs. An illustration of motion on different time scales can be found in Section XIII.2 in [1].

In our experiments, we aim to obtain stable simulation of the system on a time scale of $\mathcal{O}(\omega)$ using solvers with Δt on the time scale of $\mathcal{O}(\omega^0)$, i.e., $\Delta t = 1.0$. We will use $m = 3$ (hence the degree of freedom $d = 6$) and $\omega = 300$. This is a challenging regime because the separation of characteristic time scales is large, i.e., from $1/300$ to 300. We will run the algorithms from the initial condition

$$\mathbf{p}_{\text{init}} = \begin{bmatrix} 0 \ \sqrt{2} \ 0 \ 0 \ 0 \ 0 \end{bmatrix}^T, \quad \mathbf{q}_{\text{init}} = \begin{bmatrix} \frac{1-\omega^{-1}}{\sqrt{2}} \ \frac{1+\omega^{-1}}{\sqrt{2}} \ 0 \ 0 \ 0 \ 0 \end{bmatrix}^T. \tag{22}$$

The corresponding energy is

$$H(\mathbf{p}_{\text{init}}, \mathbf{q}_{\text{init}}) = 2 + 3\omega^{-2} + \frac{1}{2}\omega^{-4}. \tag{23}$$

Given a reference solution $\mathbf{u}^{\text{ref}} = \left(\mathbf{p}^{\text{ref}}, \mathbf{q}^{\text{ref}}\right)$ and a computed solution $\mathbf{u} = (\mathbf{p}, \mathbf{q})$, we shall report the trajectory error

$$\text{traj err} = \left\| \mathbf{u} - \mathbf{u}^{\text{ref}} \right\| = \sqrt{\left\|\mathbf{p} - \mathbf{p}^{\text{ref}}\right\|^2 + \left\|\mathbf{q} - \mathbf{q}^{\text{ref}}\right\|^2}, \tag{24}$$

and the energy error

$$\text{energy err} = \frac{\left|H(\mathbf{p}, \mathbf{q}) - H(\mathbf{p}^{\text{ref}}, \mathbf{q}^{\text{ref}})\right|}{\left|H\left(\mathbf{p}^{\text{ref}}, \mathbf{q}^{\text{ref}}\right)\right|}. \tag{25}$$

4.1 Definition of the energy transform

We first present our definition of the energy transform Λ, since both the Procrustes parareal and the NN solver will rely on this function. Based on the Hamiltonian (21), we define

$$\Lambda_1 : \mathbf{p} \in \mathbb{R}^{2m} \mapsto \frac{\mathbf{p}}{\sqrt{2}} \in \mathbb{R}^{2m}, \tag{26}$$

$$\Lambda_2 : \mathbf{q} \in \mathbb{R}^{2m} \mapsto \begin{bmatrix} \mathbf{dq}_{\text{stiff}} \\ \mathbf{dq}_{\text{soft}} \end{bmatrix} \in \mathbb{R}^{2m+1}, \tag{27}$$

where

$$\mathbf{dq}_{\text{stiff}} := \begin{bmatrix} \frac{\omega}{2}(q_2 - q_1) \\ \vdots \\ \frac{\omega}{2}(q_{2i} - q_{2i-1}) \\ \vdots \\ \frac{\omega}{2}(q_{2m} - q_{2m-1}) \end{bmatrix} \in \mathbb{R}^m, \quad \mathbf{dq}_{\text{soft}} := \begin{bmatrix} (q_1 - q_0)^2 \\ \vdots \\ (q_{2i+1} - q_{2i})^2 \\ \vdots \\ (q_{2m+1} - q_{2m})^2 \end{bmatrix} \in \mathbb{R}^{m+1}. \tag{28}$$

Then, Λ can be written as

$$\Lambda : \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} \in \mathbb{R}^{4m} \mapsto \begin{bmatrix} \Lambda_1(\mathbf{p}) \\ \Lambda_2(\mathbf{q}) \end{bmatrix} \in \mathbb{R}^{4m+1}. \tag{29}$$

One can check the l_2 norm squared of Λ ($[\mathbf{p}, \mathbf{q}]$) recovers (21).

To define the pseudo-inverse $\Lambda^\dagger$, the main task is to recover $\mathbf{q}$ from a given vector $\Lambda_2(\mathbf{q})$. This can be done by solving a nonlinear least squares problem: given $\tilde{\mathbf{dq}}_{\text{stiff}}, \tilde{\mathbf{dq}}_{\text{soft}}$, find

$$\mathbf{q}_* = \arg\min_{\mathbf{q}} \left\| \Lambda_2(\mathbf{q}) - \begin{bmatrix} \tilde{\mathbf{dq}}_{\text{stiff}} \\ \tilde{\mathbf{dq}}_{\text{soft}} \end{bmatrix} \right\|_2^2. \tag{30}$$

We use an adaptive nonlinear least squares algorithm, called NL2SOL [16], to solve the problem. In the Procrustes parareal setup, $\Lambda^\dagger$ is only evaluated when applying the correction operator $\Psi_{\Delta t}^{(k)} := \Lambda^\dagger \Omega_{\Delta t}^{(k)} \Lambda$ to some solution $\mathbf{u}$. Since the correction by the unitary matrix $\Omega_{\Delta t}^{(k)}$ is expected to be small, we use the $\mathbf{q}$ component of $\mathbf{u}$ as an initial guess in the least squares algorithm.

4.2 The Procrustes parareal method

In this section, we discuss our choice of numerical integrators and present results for the Procrustes parareal method.

Choice of numerical solvers

For Hamiltonian systems, we use symplectic integrators for the fine and coarse solvers. The stepsize h of the integrator should be order $\mathcal{O}(\omega^{-1})$ or smaller for accurate integration. For the coarse solver $C_{\Delta t}$, we use a 4th-order symplectic algorithm developed by Calvo and Sanz-Serna [17]. We consider two stepsizes $h = 2^{-9}$ and $h = 2^{-8}$ for comparison. For the fine solver $F_{\Delta t}$, we use an 8th-order symplectic algorithm developed by Kahan and Li [18] with a stepsize $h = 2^{-18}$. The numerical integrators are denoted by $\Phi_{\Delta t}^{\text{CSS4},h=2^{-9}}$, $\Phi_{\Delta t}^{\text{CSS4},h=2^{-8}}$ and $\Phi_{\Delta t}^{\text{KL8},h=2^{-18}}$.

Because we aim to simulate for a long time interval $\mathcal{O}(\omega)$, generating the reference trajectory requires $\omega/h \approx 8 \times 10^7$ fine steps. Hence, we perform fine solver computations in octuple precision to reduce the accumulated rounding errors. The computations

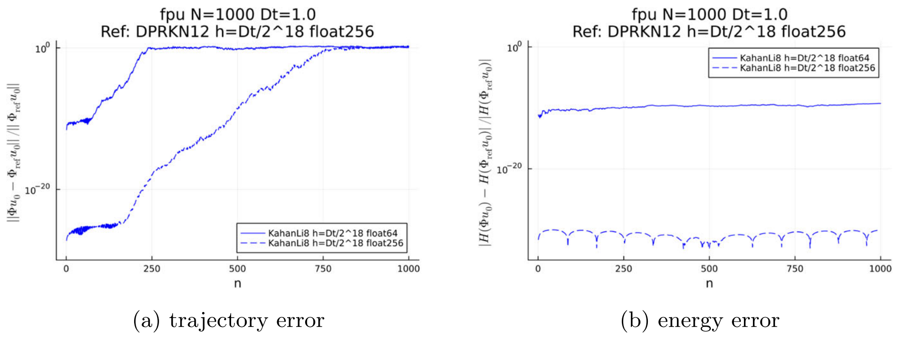


Fig. 1 Errors in trajectories generated by applying the fine solver $F_{\Delta t} = \Phi_{\Delta t}^{\text{KL8},h=2^{-18}}$, implemented in double precision and octuple precision, sequentially for $N = 1000$ steps of $\Delta t = 1.0$. Float64 refers to double precision and float256 refers to octuple precision. The numerical integrator used for generating the reference trajectory here is $\Phi_{\Delta t}^{\text{DPRKN12},h=2^{-18}}$, implemented in octuple precision

of the coarse solver are done in double precision. We perform all numerical computations in Julia. We use the MultiFloats library to obtain octuple precision numbers.

To demonstrate the significance of rounding errors and to access the quality of the fine solutions, in Fig. 1, we plot the global errors of the fine solutions computed up to $T = 1000$ in double precision versus in octuple precision. Here, we use a method of

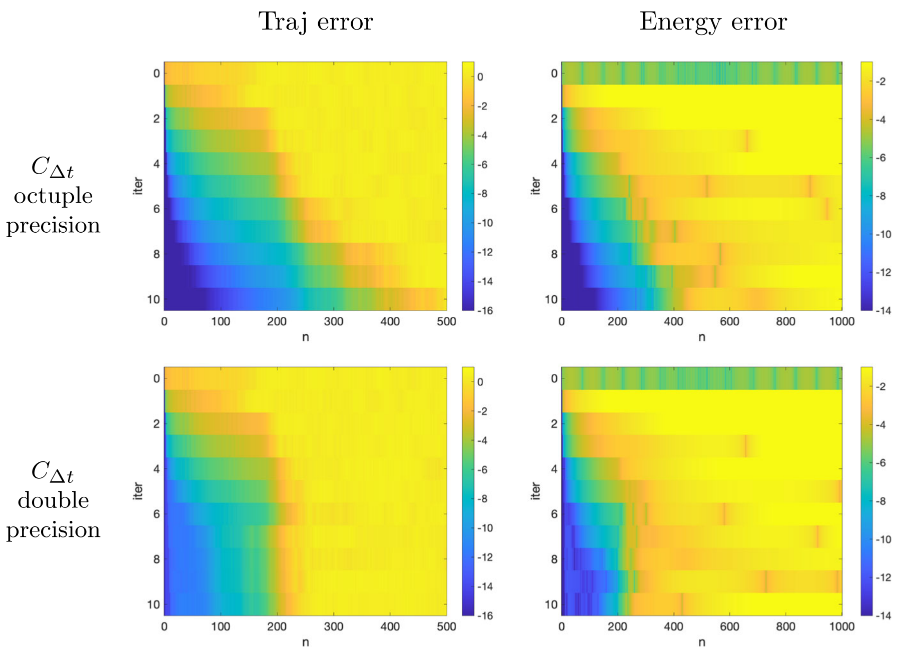


Fig. 2 Log (base 10) errors in plain parareal solutions computed with $C_{\Delta t} = \Phi_{\Delta t}^{\text{CSS4},h=2^{-9}}$ implemented in double precision and octuple precision, and $F_{\Delta t} = \Phi_{\Delta t}^{\text{KL8},h=2^{-18}}$ implemented in octuple precision only ($\Delta t = 1.0$, $T = 1000$)

even higher order, the 12th-order explicit Runge–Kutta-Nyström method, with stepsize $h = 2^{-18}$ implemented in octuple precision to serve as the reference solution map. We can see the trajectory errors grow over time while the energy errors are stable (expected since it is a symplectic integrator) for both precisions. The trajectory errors grow linearly in time at first, and then grow exponentially. Based on the trajectory errors, we conclude the fine solutions computed in octuple precision lose digits much later than fine solutions computed in double precision. Even with octuple precision, the fine solutions are not reliable after $n = 500$. In the rest of the paper, unless otherwise mentioned, we compare the trajectory errors against the reference only up to $n = 500$. For energy errors, we may compare for $n > 500$ since the reference energy is almost a constant.

We also observe an effect of floating point precision on parareal iterations. Figure 2 shows the errors in parareal solutions computed with a coarse solver implemented in double precision versus octuple precision. The fine solver is fixed using octuple precision. We found the error plots differ significantly after $n = 200$ steps. Using double precision for the coarse solver prevents the parareal solutions from improving after $n = 200$. Unfortunately, we have to use double precision for the coarse solver for the rest of comparisons, given the facts that (1) the library for the least squares algorithm involved in inverting Λ only supports double precision, and (2) the NN solver is double precision.

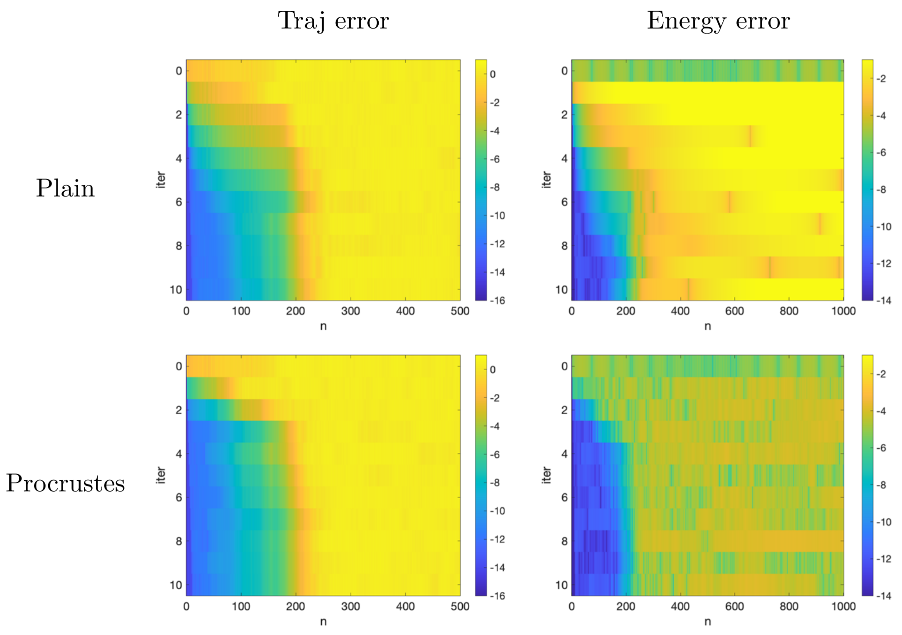


Fig. 3 Log (base 10) errors in plain parareal solutions and Procrustes parareal solutions ($\Delta t = 1.0$, $T = 1000$, $C_{\Delta t} = \Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$, $F_{\Delta t} = \Phi_{\Delta t}^{\mathrm{KL8},h=2^{-18}}$)

Plain versus Procrustes

Using the introduced numerical solvers, we run the plain parareal method and the Procrustes parareal method for $N = 1000$ steps and $k = 10$ iterations, from the same initial condition.

Figure 3 shows errors in the computed trajectories for $C_{\Delta t} = \Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$. Overall, both methods improve the initial coarse solution, but the improvement becomes small after several iterations. Comparing the trajectory error plots, we observe the Procrustes parareal solutions improve faster than the plain parareal solutions over iterations. Comparing the energy error plots, we see the Procrustes parareal solutions not only improve faster, but they are also more stable in energy than the plain parareal solutions (in particular, see how the energy errors grow from iteration 0 to iteration 1 in plain parareal versus in Procrustes parareal). The stability issue can be seen more clearly in Fig. 4, where we plot the energy of the three stiff springs and their total energy from the computed solutions. As shown in the reference energy profile in Fig. 5, during the simulated time range, there is energy exchange among the stiff springs while the total energy of the stiff springs remains almost a constant. For the plain parareal solution at iteration 3, the total energy blows up by the end of the simulation. On the contrary, the total energy for the Procrustes parareal solution is almost conserved.

The same can be observed for a less accurate coarse solver $C_{\Delta t} = \Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-8}}$ (see Figs. 11 and 12). Compared with using $C_{\Delta t} = \Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$, the improvement of Procrustes parareal over plain parareal is more substantial.

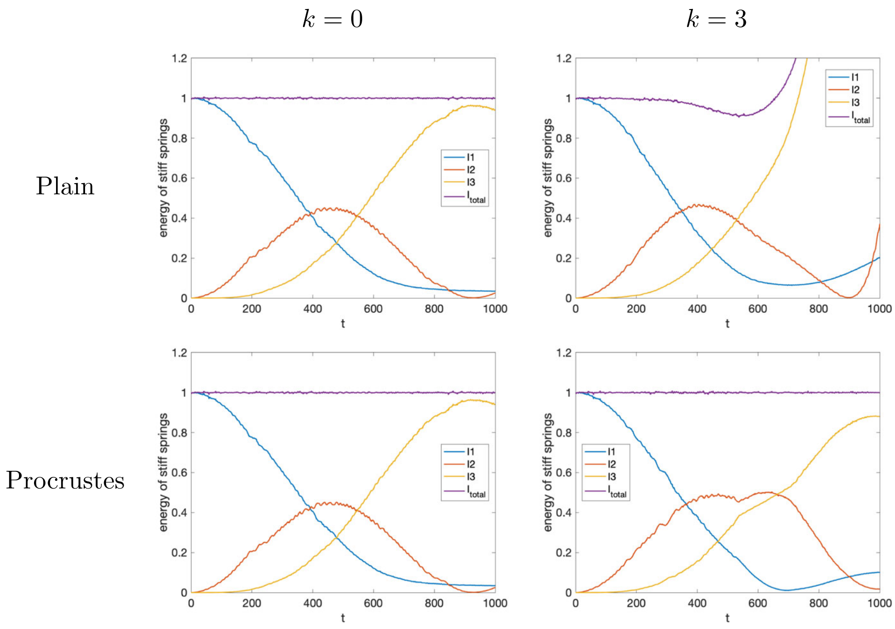


Fig. 4 Energy profiles of the stiff springs computed from plain parareal solutions and Procrustes parareal solutions at iteration 0 and iteration 3 ($\Delta t = 1.0$, $T = 1000$, $C_{\Delta t} = \Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$, $F_{\Delta t} = \Phi_{\Delta t}^{\mathrm{KL8},h=2^{-18}}$)

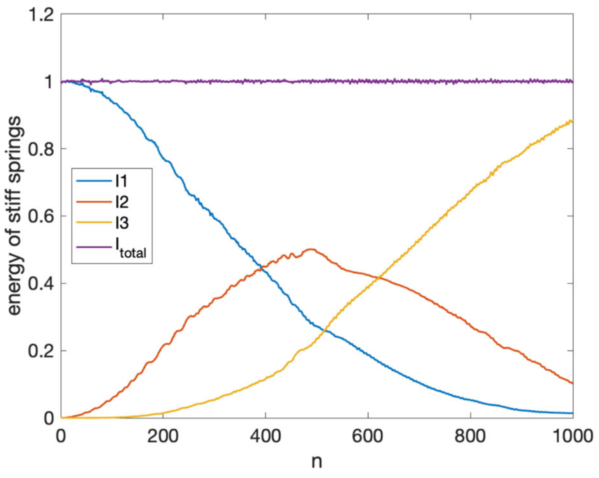


Fig. 5 Reference energy profile of the stiff springs ($\Delta t = 1.0$, $T = 1000$)

Lastly, we compare runtime of the parareal methods. We used 40 computing cores to perform parallel computation. For $C_{\Delta t} = \Phi_{\Delta t}^{\text{CSS4},h=2^{-9}}$ and 10 iterations, the total runtime of the plain parareal method is 3.2×10^3 s, and the total runtime of the Procrustes parareal method is 3.6×10^3 s. As a baseline, the runtime of the sequential fine computation on $[0, T]$ is 1.2×10^5 s.

4.3 NN solution map

In this section, we present the setups for learning the solution map $\Phi_{\Delta t}^{\text{NN}}$ and study how its quality is affected by different options of training data, loss function, and network architecture. To evaluate performance of a learned $\Phi_{\Delta t}^{\text{NN}}$, we generate a 1000-step trajectory from the initial condition (22) by sequential applications of $\Phi_{\Delta t}^{\text{NN}}$.

Using the proposed data generation algorithms, we generated two sets of input data, denoted by $\mathcal{D}_0^{\text{HMC-}H_0}$ and $\mathcal{D}_0^{\text{TrajEnsemble-}H_0}$ where $H_0 = H(\mathbf{p}_{\text{init}}, \mathbf{q}_{\text{init}})$. The parameters for data generation algorithms are given in Table 1. Each dataset has 200k examples of inputs $\mathbf{u}_0$. We emphasize that $(\mathbf{p}_{\text{init}}, \mathbf{q}_{\text{init}})$ is not included in either dataset even though $\mathbf{q}_{\text{init}}$ was used in the parameters. This is because both algorithms randomly sample $\mathbf{p}$ given $\mathbf{q}_{\text{init}}$.

Table 1 Descriptions of input data sets

Dataset	Parameters
$\mathcal{D}_0^{\text{TrajEnsemble-}H_0}$	$\mathbf{q}_0 = \mathbf{q}_{\text{init}}$, $\sigma = 0.1$, $N_{\text{levelsets}} = 400$, $N_{\text{traj}} = 10$, $L = 50$, $\delta t = 0.1$, $F_{\delta t} = \Phi_{\delta t}^{\text{CSS4},h=5^{-6}}$
$\mathcal{D}_0^{\text{HMC-}H_0}$	$\mathbf{q}_0 = \mathbf{q}_{\text{init}}$, $\sigma = 0.1$, $N_{\text{chains}} = 100$, $N_{\text{trans}} = 2000$, $\delta t = 0.4$, $F_{\delta t} = \Phi_{\delta t}^{\text{CSS4},h=5^{-6}}$

Given a $\mathcal{D}_0$, we generate the full training dataset $\mathcal{D}$ by propagating each $\mathbf{u}_0$ for 5 steps using the fine solver $F_{\Delta t}$, and then collecting the input and target sequence pairs:

$$\mathcal{D} = \left\{ \left(\mathbf{u}_0, \{(F_{\Delta t})^i \, \mathbf{u}_0\}_{i=1,\cdots,5} \right) : \mathbf{u}_0 \in \mathcal{D}_0 \right\}. \tag{31}$$

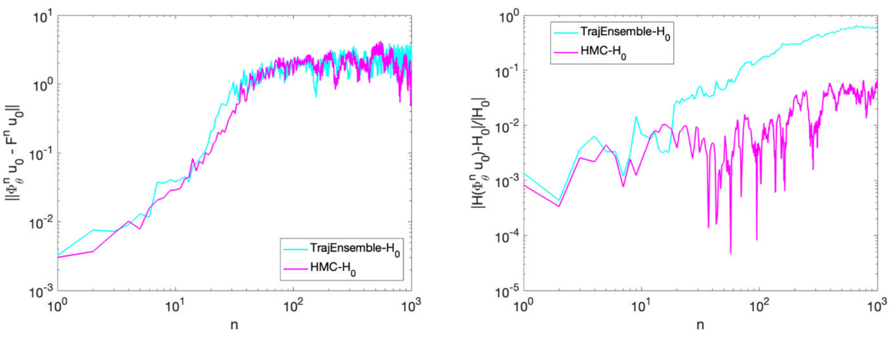
This leads to two training datasets $\mathcal{D}^{\text{TrajEnsemble-}H_0}$ and $\mathcal{D}^{\text{HMC-}H_0}$.

Let ResNet(L, n) denote a network with L hidden layers and n nodes per hidden layer. We considered several ResNet architectures, including a shallow network ResNet(4, 1000) and a deep network ResNet(75, 200). The two networks have a similar number of trainable parameters, which is around 3×10^6. Performances of the two networks are similar. Hence, we will just report results for ResNet(4, 1000).

The neural networks are implemented using PyTorch. We trained the networks using the mini-batch Adam algorithm [19] with weight decay [20]. To accelerate the training procedure, we used a one-cycle learning rate scheduler [21] that anneals the learning rate from an initial value to some maximum value and then to some minimum value within a fixed number of epochs. We used 10,000 epochs to train a ResNet(4, 1000) and 5000 epochs to train a ResNet(75, 200).

Effects of training data

We trained the shallow network ResNet(4, 1000) with different datasets and with the one-step ($S = 1$) MSE loss. Figure 6 shows the network trained with $\mathcal{D}^{\text{HMC-}H_0}$ is better than the network trained with $\mathcal{D}^{\text{TrajEnsemble-}H_0}$, especially in terms of the energy stability. We attribute this to the fact that the set of inputs $\mathcal{D}_0^{\text{HMC-}H_0}$ better represent the target distribution. As shown in Fig. 7, the minimum distance from each point along the reference trajectory to the set $\mathcal{D}_0^{\text{HMC-}H_0}$ is on average a lot smaller than the minimum distance to the set $\mathcal{D}_0^{\text{TrajEnsemble-}H_0}$. What is more, the minimum distance to $\mathcal{D}_0^{\text{TrajEnsemble-}H_0}$ increases along the trajectory, while the minimum distance to $\mathcal{D}_0^{\text{HMC-}H_0}$ stays stable over time.



(a) trajectory error (b) energy error

Fig. 6 Errors in trajectories generated by NN solvers learned with different data

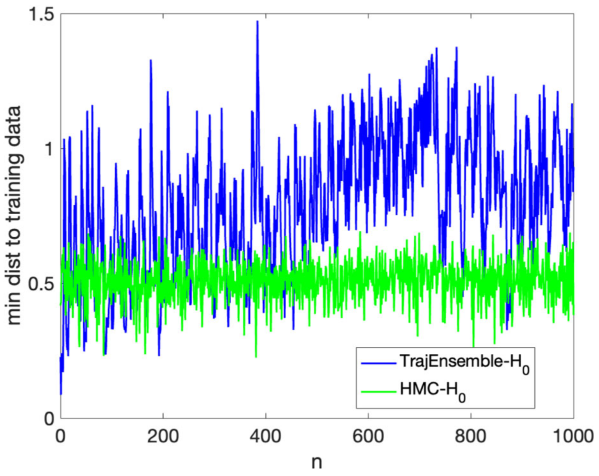


Fig. 7 Minimum distance from each point on the reference trajectory to different training data in the phase space

Effects of sequence length in loss function

We trained the shallow network ResNet(4, 1000) using $\mathcal{D}^{\text{HMC-}H_0}$ and multi-step MSE loss for different sequence length S. Figure 8 shows that longer sequence length yields slightly better accuracy for the first ten steps. After ten steps, there is no significant difference between results of different sequence lengths. In Fig. 9, we repeated the comparison for a different initial condition $(\sqrt{2}\mathbf{p}_{\text{init}}, \mathbf{q}_{\text{init}})$. Note that the corresponding energy level is higher than H_0 for generating the training data. In other words, we are testing the generalization ability of the NN solvers for out-of-distribution examples. Based on the results, the NN solvers are able to achieve accuracy on par with the accuracy for in-distribution examples for at least the first few steps. Moreover, we found longer training sequences result in significantly better generalization ability.

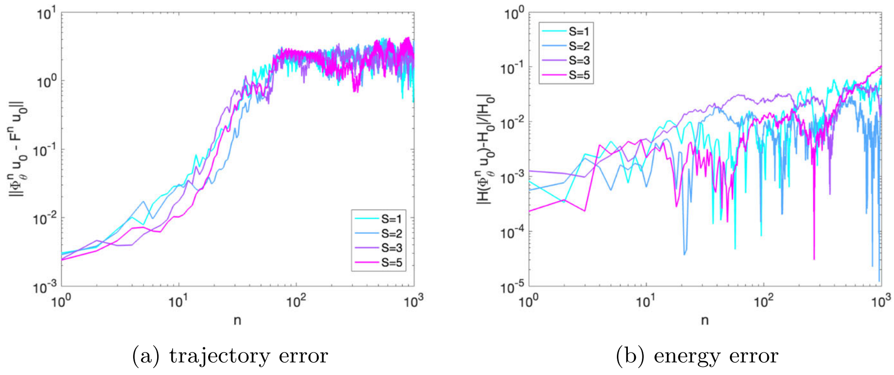
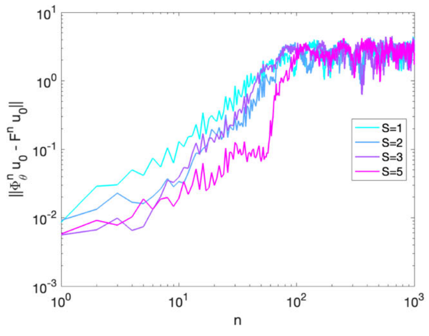
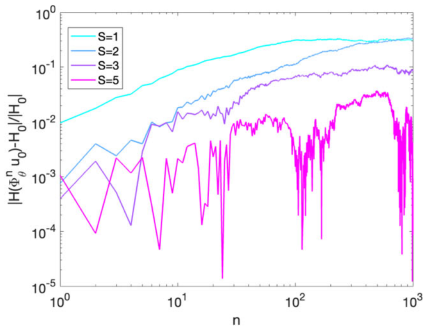


Fig. 8 Errors in trajectories generated by NN solvers learned with different sequence length S (initial condition = $(\mathbf{p}_{\text{init}}, \mathbf{q}_{\text{init}})$)



(a) trajectory error



(b) energy error

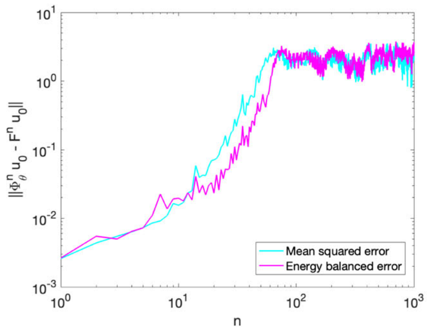
Fig. 9 Errors in trajectories generated by NN solvers learned with different sequence length S (initial condition = $(\sqrt{2}\mathbf{p}_{\text{init}}, \mathbf{q}_{\text{init}})$)

Effects of loss function metric

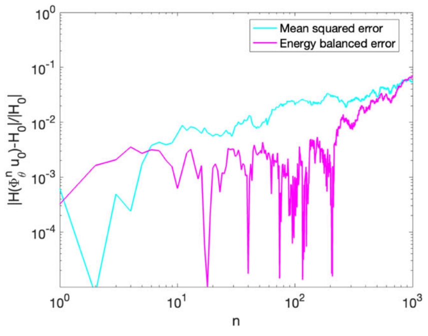
We trained the shallow network ResNet(4, 1000) using $\mathcal{D}^{\text{HMC-}H_0}$ and different loss metrics with sequence length $S = 5$. As displayed in Fig. 10, compared to using MSE, using EBE leads to smaller trajectory error and energy error for over 100 steps. In particular, the energy error is not only smaller but also more stable over a long time period.

Comparison with numerical solvers

We present in Table 2 the one-step accuracy and runtime performance of various solvers. The fine solver $F_{\Delta t}$ is $\Phi_{\Delta t}^{\text{KL8},h=2^{-18}}$ implemented in octuple precision. The NN solver $\Phi_{\Delta t}^{\text{NN}}$ is ResNet(4, 1000), trained using $\mathcal{D}^{\text{HMC-}H_0}$ and the multi-step ($S = 5$) EBE loss. For comparison, we include several numerical solvers: $\Phi_{\Delta t}^{\text{CSS4},h=2^{-9}}$ and $\Phi_{\Delta t}^{\text{VV},h=2^{-14}}$, whose one-step trajectory error is comparable to that of $\Phi_{\Delta t}^{\text{NN}}$, as well as



(a) trajectory error



(b) energy error

Fig. 10 Errors in trajectories generated by NN solvers learned with different metrics

Table 2 Accuracy and runtime performance comparison for various solvers

	Trajectory error	Energy error	Runtime
$F_{\Delta t}$	0	0	12.3 s
$\Phi_{\Delta t}^{\mathrm{NN}}$	0.00264	3.2×10^{-4}	0.272 ms*
$\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$	0.00262	4.3×10^{-6}	0.246 ms
$\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-8}}$	0.0435	1.7×10^{-4}	0.150 ms
$\Phi_{\Delta t}^{\mathrm{VV},h=2^{-14}}$	0.00419	3.4×10^{-7}	4.231 ms
$\Phi_{\Delta t}^{\mathrm{VV},h=2^{-11}}$	0.231	6.5×10^{-4}	0.578 ms

*Runtime for the NN solver implemented in Python. The rest of the solvers are written and optimized in Julia

$\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-8}}$ and $\Phi_{\Delta t}^{\mathrm{VV},h=2^{-11}}$, whose one-step energy error is comparable to that of $\Phi_{\Delta t}^{\mathrm{NN}}$. Here, VV stands for the 2nd-order velocity Verlet scheme.

It can be seen that, with the same level of trajectory error, the numerical solvers achieve lower energy error than $\Phi_{\Delta t}^{\mathrm{NN}}$. However, in terms of runtime, $\Phi_{\Delta t}^{\mathrm{NN}}$ is as good as $\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$ and is about 17 times faster than $\Phi_{\Delta t}^{\mathrm{VV},h=2^{-14}}$. We emphasize that the runtime measurements took place in different environments: the NN solver is implemented in Python, and numerical solvers are implemented in Julia. We have fully optimized the Julia code for runtime and memory efficiency. We expect to further optimize the NN implementation for better runtime performance in the future.

4.4 NN solution map in parareal iterations

In this section, we present results of using $\Phi_{\Delta t}^{\mathrm{NN}}$ as the coarse solver in parareal methods.

We first study the plain parareal method. Figure 11 compares the plain parareal solutions computed by different coarse solvers, including $\Phi_{\Delta t}^{\mathrm{NN}}$, $\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$, and $\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-8}}$. Clearly, $\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-8}}$ performs the worst, as expected because it is the least accurate among the three solvers. Based on the trajectory errors, we see using $\Phi_{\Delta t}^{\mathrm{NN}}$ as the coarse solver provides slower accuracy improvement over iterations compared to using $\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$. Comparing the energy errors, we observe that when using $\Phi_{\Delta t}^{\mathrm{NN}}$, the stability in energy is not destroyed as much as in using $\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$ as the coarse solver (see also the energy profiles at iteration 3 in Fig. 13).

We will now compare different coarse solvers used in the Procrustes parareal method. In Section 4.2, we found the Procrustes parareal method improves accuracy by stabilizing the energy of the solutions. As shown in Figs. 12 and 14, the best improvement is again yielded by $\Phi_{\Delta t}^{\mathrm{CSS4},h=2^{-9}}$. There is no significant gain from combining Procrustes parareal with the NN solution map. In fact, the improvement over the Procrustes parareal iterations even deteriorates compared to the improvement over the plain parareal iterations.

We speculate that $\Phi_{\Delta t}^{\mathrm{NN}}$ does not perform well in parareal iterations because it was not trained using suitable data. As described in Section 3.3, we sampled training

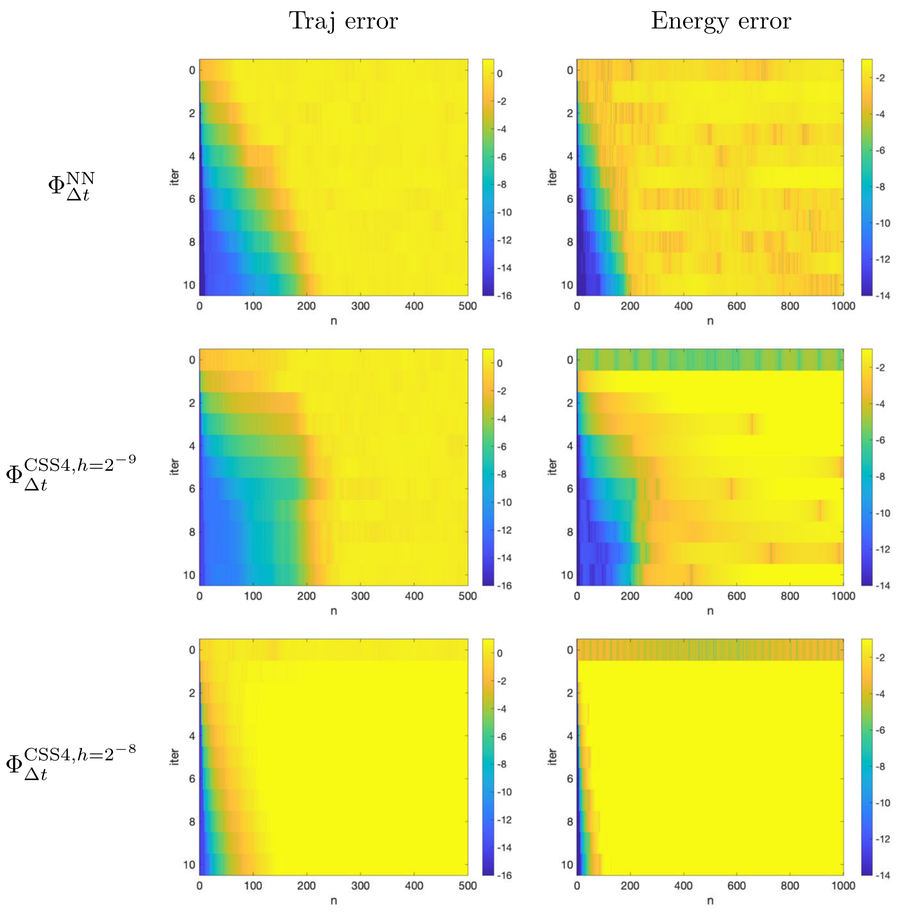


Fig. 11 Log (base 10) errors in plain parareal solutions by various coarse solvers ($\Delta t = 1.0$, $T = 1000$)

data points from the Liouville density, mainly because it is a natural distribution for learning a solution map to be used in a sequential algorithm. In parareal schemes, since the coarse solver is applied differently than in a sequential algorithm, we would need a different data distribution. A similar issue has been investigated in [10] for approximating the correction operator using the NN approach for wave equations. There, the authors demonstrated the importance of using training data closer to the ones encountered in the simulations.

5 Conclusion

In this paper, we presented two data-driven approaches for stabilization of the standard parareal algorithm for long-time computation of highly oscillatory Hamiltonian systems. The Procrustes parareal approach uses solutions computed along the parareal iterations to construct a correction operator to align the “phase” of the fine and coarse

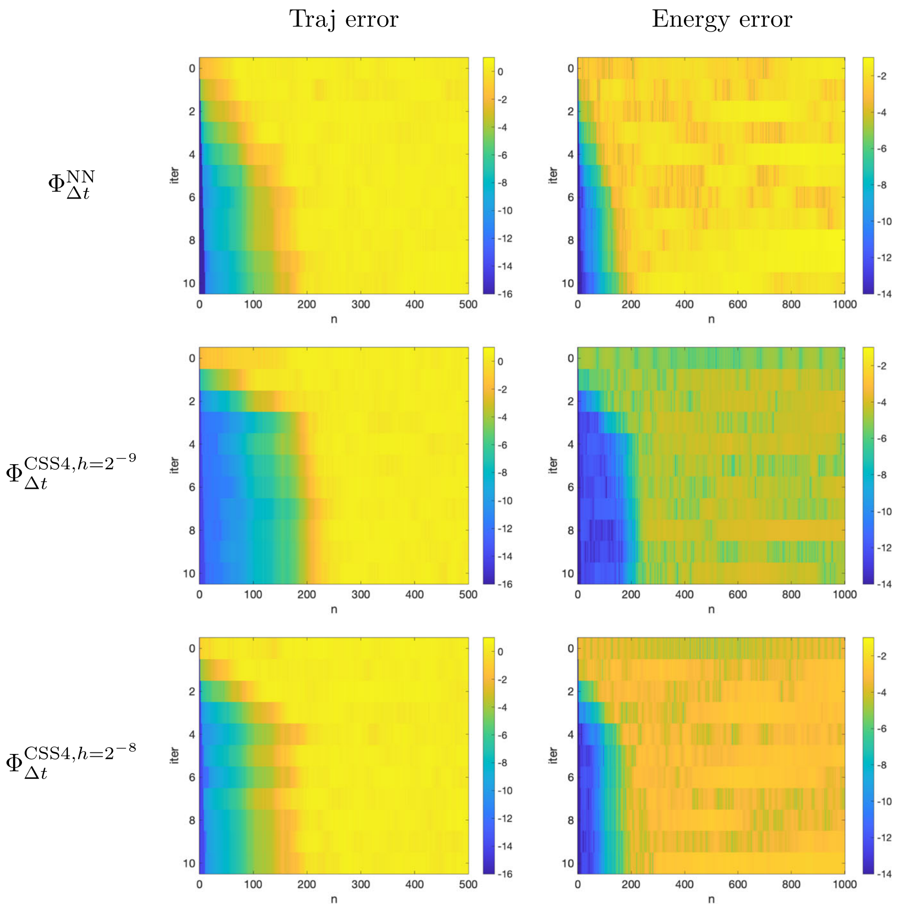


Fig. 12 Log (base 10) errors in Procrustes parareal solutions by various coarse solvers ($\Delta t = 1.0, T = 1000$)

solvers. Numerical results for the FPU problem demonstrated that the constructed correction can successfully stabilize the parareal iterations, which helps improve the accuracy of the computed solutions. The second approach we proposed is to use a neural network (NN) to approximate the reference solution map that advances the given state forward in time by a large fixed time step. We developed a sampling algorithm called HMC-H_0 to sample phase space points from the neighborhood of an energy level set. We also designed a loss function which considers the energy-balanced errors between approximated trajectories and reference trajectories. The resulting NN solver for the FPU problem is able to achieve comparable or better runtime performance compared to numerical solvers of similar accuracy. When combined with parareal iterations, solutions computed by the NN solver are not as accurate as solutions computed by a comparable numerical solver, although the NN energy errors are slightly smaller. We think that this may be improved if we train the network using data suitably sampled for the discrete trajectories computed by the parareal schemes.

The FPU problem is too small to reveal the potential benefit of using NNs. It is small enough that optimized high-order symplectic integrators are extremely efficient and accurate. For more complicated problems, where the phase space is large and lower-order accurate methods are the only feasible choice, we think that the investigated NN approach may become viable.

Appendix: A Energy profiles of parareal solutions by various coarse solvers

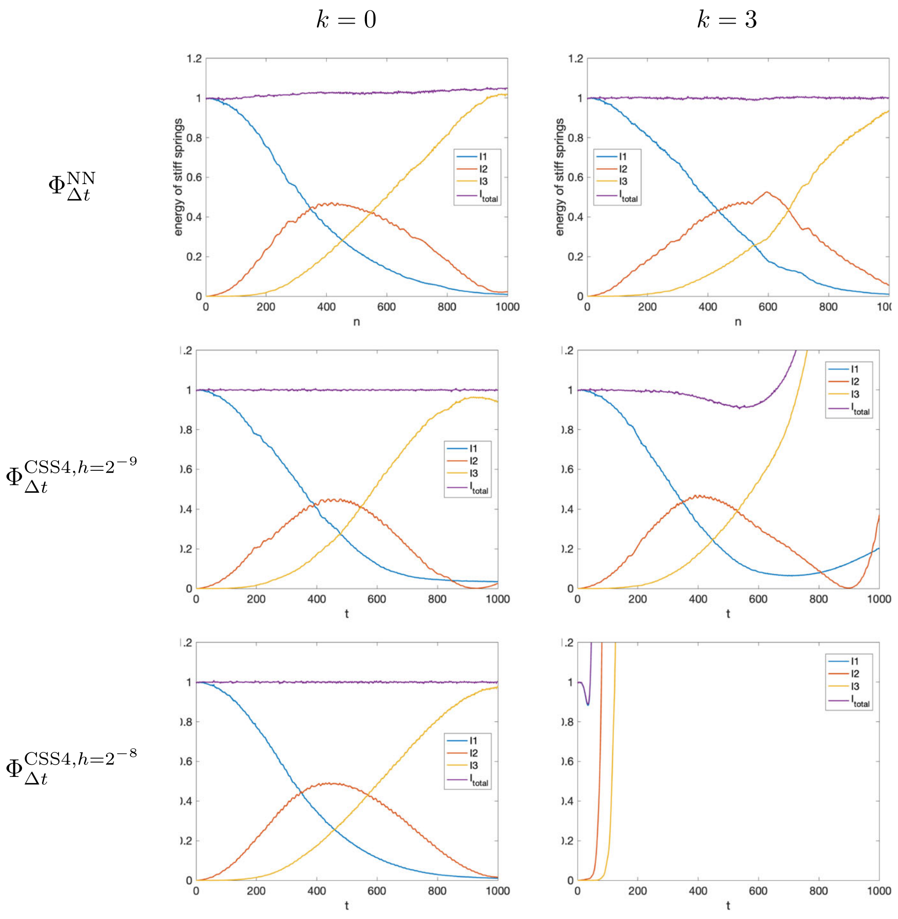


Fig. 13 Energy profiles of the stiff springs computed from plain parareal solutions by various coarse solvers ($\Delta t = 1.0$, $T = 1000$)

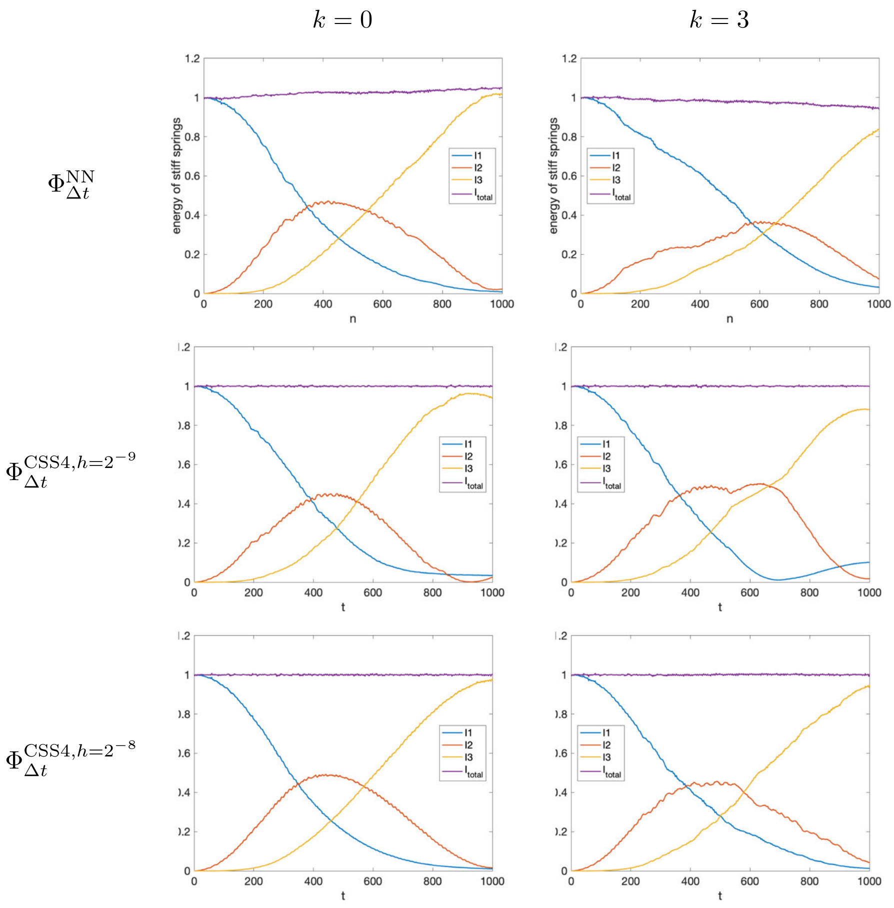


Fig. 14 Energy profiles of the stiff springs computed from Procrustes parareal solutions by various coarse solvers ($\Delta t = 1.0$, $T = 1000$)

Acknowledgements The authors thank the Texas Advanced Computing Center (TACC) for providing computing resources.

Author contribution R.F.: conceptualization, methodology, software, visualization, writing. R.T.: conceptualization, funding acquisition, methodology, project administration, supervision, writing.

Funding The authors are partially supported by the National Science Foundation grant DMS-2208504.

Data availability The datasets generated and analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Ethical approval Not applicable

Conflict of interest The authors declare no competing interests.

References

1. Hairer, E., Lubich, C., Wanner, G.: Geometric numerical integration, 2nd edn. Springer Series in Computational Mathematics, vol. 31, p. 644. Springer, Berlin (2006)
2. Engquist, B., Tsai, Y.-H.: Heterogeneous multiscale methods for stiff ordinary differential equations. Math. Comput. **74**(252), 1707–1742 (2005)
3. Lions, J., Maday, Y., Turinici, G.: A “parareal” in time discretization of PDE’s. comptes rendus de l’acadmie des sciences-series i-mathematics **332**, 661–668 (2001)
4. Gander, M.J., Vandewalle, S.: Analysis of the parareal time-parallel time-integration method. SIAM J. Sci. Comput. **29**(2), 556–578 (2007)
5. Ariel, G., Kim, S.J., Tsai, R.: Parareal multiscale methods for highly oscillatory dynamical systems. SIAM J. Sci. Comput. **38**(6), 3540–3564 (2016)
6. Gander, M.J., Hairer, E.: Analysis for parareal algorithms applied to Hamiltonian differential equations. J. Comput. Appl. Math. **259**, 2–13 (2014)
7. Dai, X., Le Bris, C., Legoll, F., Maday, Y.: Symmetric parareal algorithms for Hamiltonian systems. ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique **47**(3), 717–742 (2013)
8. Farhat, C., Chandesris, M.: Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications. Int. J. Numer. Meth. Eng. **58**(9), 1397–1434 (2003)
9. Nguyen, H., Tsai, R.: A stable parareal-like method for the second order wave equation. J. Comput. Phys. **405**, 109156 (2020)
10. Nguyen, H., Tsai, R.: Numerical wave propagation aided by deep learning. J. Comput. Phys. **475**, 111828 (2023)
11. Gower, J.C., Dijksterhuis, G.B.: Procrustes problems, vol. 30. Oxford University Press, Oxford (2004)
12. Clevert, D.-A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (ELUs). arXiv:1511.07289 (2015)
13. E, W.: Machine learning and computational mathematics. arXiv:2009.14596 (2020)
14. Duane, S., Kennedy, A.D., Pendleton, B.J., Roweth, D.: Hybrid Monte Carlo. Phys. Lett. B **195**(2), 216–222 (1987). https://doi.org/10.1016/0370-2693(87)91197-X
15. Fermi, E., Pasta, P., Ulam, S., Tsingou, M.: Studies of the nonlinear problems. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (1955)
16. Dennis, J.E., Gay, D.M., Welsch, R.E.: Algorithm 573: NL2SOL—an adaptive nonlinear least-squares algorithm [E4]. ACM Transactions on Mathematical Software (TOMS) **7**(3), 369–383 (1981). https://doi.org/10.1145/355958.355966
17. Calvo, M.P., Sanz-Serna, J.M.: The development of variable-step symplectic integrators, with application to the two-body problem. SIAM J. Sci. Comput. **14**(4), 936–952 (1993)
18. Kahan, W., Li, R.-C.: Composition constants for raising the orders of unconventional schemes for ordinary differential equations. Math. Comput. **66**(219), 1089–1099 (1997)
19. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv:1412.6980 (2014)
20. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv:1711.05101 (2017)
21. Smith, L.N., Topin, N.: Super-convergence: very fast training of neural networks using large learning rates. In: Artificial Intelligence and Machine Learning for Multi-domain Operations Applications, vol. 11006, pp. 369–386. SPIE (2019)