An Extensible Benchmark Suite for Learning to Simulate Physical Systems

Karl Otness, Arvi Gjoka, Joan Bruna, Daniele Panozzo, Benjamin Peherstorfer, Teseo Schneider & Denis Zorin

NYU

ICLR 2021 Workshop
Deep Learning for Simulation (simDL)

Introduction

Time integration of models of physical systems is a core task of scientific computing. Recently, there has been a surge of interest in data-driven methods that learn from data a model of the physical system and then integrate it in time to make predictions. This work introduces benchmarks for evaluating data-driven methods on a variety of physical systems and proposes evaluation scenarios. The proposed benchmarks comprise three representative physical systems (spring, spring mesh, wave) and a collection of classical time integrators as baselines. For demonstration purposes, we apply several data-driven methods to the benchmarks and report accuracy and computational efficiency.

Contributions

1. A set of simple, yet representative, physical models with a range of training and evaluation boundary conditions, coefficients, and parameters with reference high-accuracy solutions which are used to evaluate data-driven methods
2. Reference implementations of traditional time integration algorithms, which are used as baselines for evaluation
3. Implementations of widely used data-driven methods, including physics-agnostic multi-layer perceptrons (MLPs), efficient kernel machines, and geometric deep learning models based on graph neural networks

Our benchmark suite is also modular, permitting extensions with limited code changes.

Systems

Consider a time-dependent PDE of the form $\partial_t u = \mathcal{L}(u)$, where u is the solution function and $\mathcal{L}$ is a potentially nonlinear operator that includes spatial derivatives of u. Discretizing in space one obtains a dynamical system with an N-dimensional state $x(t) \in \mathbb{R}^N$ for times $t \in [0, T]$ and initial conditions $x_0 \in \mathbb{R}^N$. For second-order systems we consider their formulation as a first-order system via position q and velocity p, $\dot{x}(t) = [\dot{q}(t), \dot{p}(t)] = f(x(t))$.

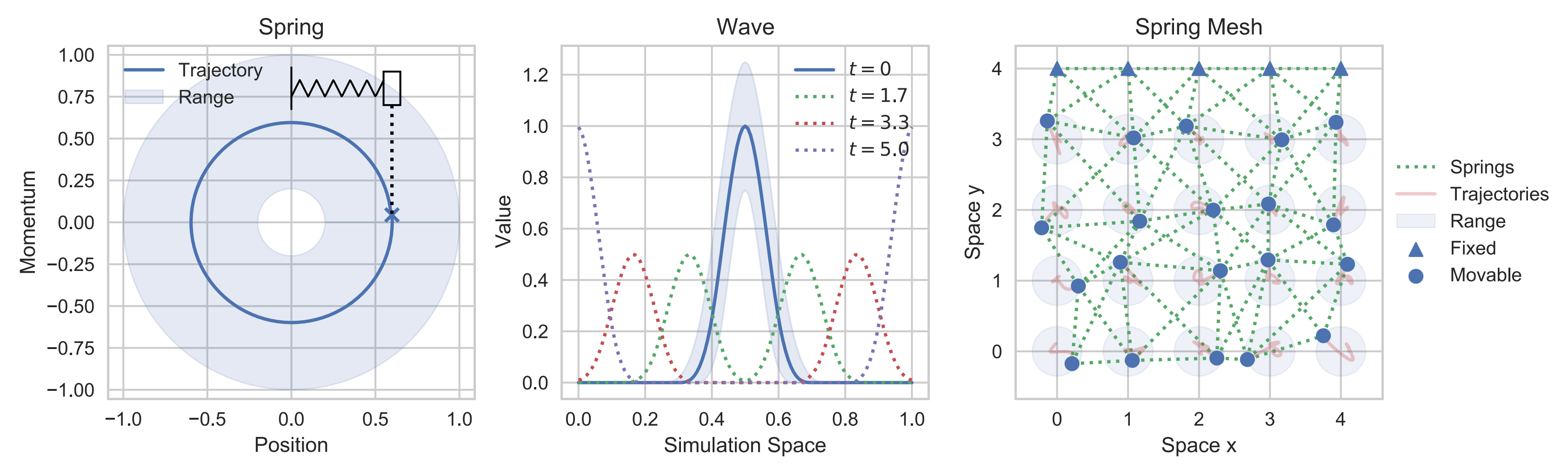
We test our numerical integrators and learning methods on three sample systems:
1. Spring: a single spring in one dimension
2. Spring mesh: a two-dimensional grid of masses linked by springs
3. Wave: a wave equation with 125 spatial grid points

For each of these systems we define a distribution over initial conditions for $[q, p]$. We sample from these distributions to select initial configurations of these systems, then integrate with a numerical integrator to produce high-accuracy, reference solutions. Sample initial conditions, as well as the distributions for training samples, are illustrated at the top of the middle column.

When evaluating these systems with standard numerical integrators, we apply:
1. Forward Euler
2. Leapfrog
3. Runge-Kutta 4
4. Backward Euler

System	# Train Trajectories	# Eval Trajectories	Time Step Size	# Steps
Spring	10, 500, 1000	30	0.00781, ÷128	805
Wave	10, 25, 50	6	0.00049, ÷8	10205
Spring Mesh	25, 50, 100	15	0.00781, ÷128	805

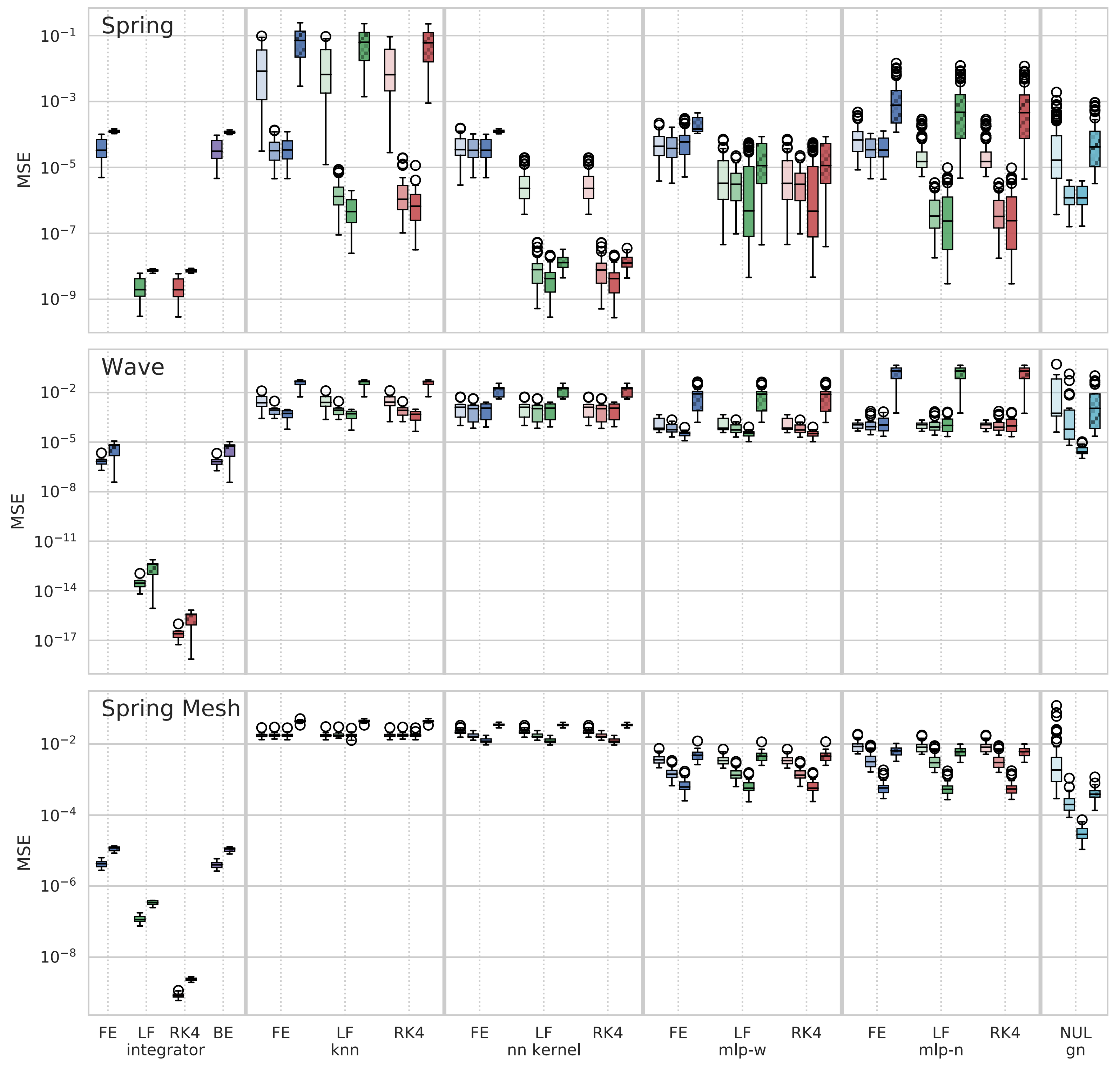
Parameters used to generate data sets for training and evaluation. Evaluation sets and training sets of three sizes are generated using the specified number of trajectories, each of which is integrated with the time step sizes and number of steps listed.



$$[\dot{q}(t), \dot{p}(t)] = [p(t), -q(t)]$$

$$\begin{bmatrix}\dot{q}(t)\\ \dot{p}(t)\end{bmatrix} = \begin{bmatrix}0 & I\\ c^2 D_{xx} & 0\end{bmatrix}\begin{bmatrix}q(t)\\ p(t)\end{bmatrix}$$

Representative initial conditions for the three systems. Shaded blue regions denote the sampling range for initial states.



Error distribution for all systems. MSE is averaged across all evaluation snapshots for three independently trained neural networks on training sets of several sizes, as well as an evaluation on samples taken outside the distribution of training data.

Base Integrator		mlp-w		mlp-n		nn-kernel		gn	knn	
		FE	RK4	FE	RK4	FE	RK4	NUL	FE	RK4
Spring	FE	19.9 / 1	19.9 / 1	23.2 / 1	23.2 / 1	19.2 / 1	19.2 / 1	376.2 / 1	106.8 / 1	106.8 / 1
	LF	29.9 / 32	29.9 / 32	34.7 / 32	34.7 / 32	28.9 / 32	28.9 / 32	506.7 / 4	160.2 / 32	160.2 / 32
	RK4	19.9 / 1	19.9 / 1	23.2 / 1	23.2 / 1	19.2 / 1	19.2 / 1	541.0 / 32	106.8 / 1	106.8 / 1
	BE	29.8 / 1	29.8 / 1	34.6 / 1	34.6 / 1	28.8 / 1	28.8 / 1	562.1 / 1	159.6 / 1	159.6 / 1
Wave	FE	69.8 / 4	69.8 / 4	50.7 / 8	50.7 / 8	104.7 / 4	104.7 / 4	873.6 / 2	318.8 / 8	318.8 / 8
	LF	103.2 / 128	103.2 / 128	39.5 / 128	39.5 / 128	154.7 / 128	154.7 / 128	2049.4 / 64	248.0 / 128	248.0 / 128
	RK4	123.3 / 128	123.3 / 128	47.1 / 128	47.1 / 128	184.9 / 128	184.9 / 128	1652.7 / 128	296.2 / 128	296.2 / 128
	BE	29.7 / 16	29.7 / 16	58.9 / 64	58.9 / 64	230.9 / 64	230.9 / 64	183.6 / 4	370.0 / 64	370.0 / 64
Spring Mesh	FE	3.5 / 8	3.5 / 8	2.8 / 8	2.8 / 8	4.0 / 16	4.0 / 16	34.2 / 8	102.7 / 16	102.7 / 16
	LF	3.5 / 8	3.5 / 8	2.8 / 8	2.8 / 8	4.0 / 16	4.0 / 16	34.2 / 8	102.7 / 16	102.7 / 16
	RK4	3.5 / 8	3.5 / 8	2.8 / 8	2.8 / 8	4.0 / 16	4.0 / 16	34.2 / 8	102.7 / 16	102.7 / 16
	BE	3.5 / 8	3.5 / 8	2.8 / 8	2.8 / 8	4.0 / 16	4.0 / 16	34.2 / 8	102.7 / 16	102.7 / 16

Computational slowdowns for equivalent error for each learned method. For each base integrator, we match the error rates for the learned method by increasing time step sizes. Each cell reports the slowdown factor and (after the slash) the time step scaling necessary.

Numerical Experiments

Given M initial conditions $x_0^{(1)}, \ldots, x_0^{(M)}$ and the corresponding M trajectories $X^{(i)} = [x_0^{(i)}, \ldots, x_K^{(i)}]$ obtained with a time integration scheme from the dynamical systems, we consider the problem of learning and approximation $\tilde{f}$ of the right-hand side function f. This gives an approximate $\dot{\tilde{x}}$ that is then numerically integrated to produce a trajectory $\tilde{X}$ for an initial condition x_0. The aim is that $\tilde{X}$ approximates well the true trajectory X obtained with f for the same initial conditions.

We evaluate the learned models on their ability to predict derivatives producing good approximate trajectories from randomly sampled initial conditions. During evaluation, we use initial conditions drawn independently from those used to produce training data, both from the same distribution as the training samples, as well as from a distribution with support outside the training range.

We consider a variety of common machine learning methods:
1. A k-nearest neighbors (KNN) regressor which memorizes input-output pairs from the training set
2. A neural network kernel
3. Two simple MLP architectures
4. A graph neural network derived from Pfaff et al. (2020)

This last network predicts accelerations for each system and integrates these to produce predictions for the velocity. The graph structure is selected to match a static mesh chosen for each of our systems.

We consider the performance of each of these methods on our three benchmark systems. For each system, we select a time step size at which numerical integration succeeds with acceptable error. We measure the accuracy of trajectories $\tilde{X}$ computed from the approximated right-hand side functions when combined with several integration schemes, as well as the computational overheads for each scheme. The plot and table in the center of the middle column illustrate the results of our measurements.

Our results show that the learning methods successfully approximate these systems even without access to the underlying model. Potential improvements could address accuracy, data requirements, and computational overhead. In most cases, the kernel method—though one of the simplest models—performs well. The graph network also demonstrates impressive stability, even producing good results outside the distribution of training samples. This may be due to the encoding of the true mesh structure directly into the network architecture, demonstrating the potential of such techniques.

Conclusion

A standardized set of problems is valuable to provide consistent accuracy and performance measurements as more learning methods for data-driven time integration emerge. The focus of our benchmarks is on simplicity and the setting where training samples are available but access to the underlying model is not. In the future, we hope to extend this benchmark suite to include additional systems with different physical behavior and to cover a wider range of data-driven tasks in scientific computing.