ENHANCING ACCURACY: NUMERICAL ERROR REDUCTION AND UNCERTAINTY QUANTIFICATION WITH DEEP LEARNING

Final Semester Project Report
Submitted in partial fulfillment of the
Requirements for the Degree of
Bachelor of Technology in Engineering and Computational Mechanics

by
Mohit Garg & Jai Shankar

Under the Guidance of
Prof. Rajdip Nayek

Department of Applied Mechanics
Indian Institute of Technology Delhi
New Delhi-110016

November 29, 2023

Contents

1 Introduction and Research Objectives 1

1.1 Motivation . . . 1

1.2 Addressing Information Gaps with Neural Networks . . . 1

1.3 Extending to Nonlinear Dynamics . . . 2

1.4 Methodology . . . 2

2 Literature Review 3

2.1 Problem Formulation . . . 4

2.1.1 Two Discretizations of the PDE . . . 4

2.1.2 Hybrid Solver with Neural Networks . . . 4

2.1.3 Error Quantification . . . 5

2.1.4 Learning Objective . . . 5

2.2 Definitions . . . 5

2.3 Conclusion . . . 6

3 Starting from Ordinary Differential Equations 7

3.1 Introduction . . . 7

3.2 ODE and Numerical Methods . . . 7

3.3 Euler's Method . . . 8

3.4 RK4 Method . . . 8

3.5 Results and Analysis . . . 8

3.5.1 Neural Network Integration . . . 9

3.5.2 Cumulative Error Comparison: Euler vs. RK4 for Testing Data . . . 9

3.5.3 Evaluation Metrics for Different k Values . . . 10

4 Spring-Mass Dynamic System 11

4.1 Physical Models . . . 11

4.1.1 1-DOF Linear System . . . 11

4.1.2 General Form for N-DOF Linear System . . . 11

4.1.3 Vectorization of the Dynamic Equation . . . 12

4.1.4 Closed-Form Solution for Vector Equation . . . 12

4.1.5 Non-Linear Spring-Mass System . . . 14

4.1.6 Excitation Signal u . . . 14

4.2 Machine Learning Models for Linear Simulator . . . 15

4.2.1 SOL_{linear} - Solver in Loop Model with Linear Simulator . . . 15

4.2.2 $SOL^{\epsilon}_{\text{linear}}$ - Solver in Loop Epsilon Model with Linear Simulator . . . 15

4.2.3 Target, Models, and Simulator for 1-DOF Dynamic System 16
4.2.4 Target, Models, and Simulator for 5-DOF Dynamic System 17
4.3 Machine Learning Models for Non-Linear Simulator 19
4.3.1 $SOL_{\text{nonlinear}}$ -Solver in Loop Model with Non-Linear Simulator . . 19
4.3.2 $SOL^{\epsilon}_{\text{nonlinear}}$ - Solver in Loop Epsilon Model with Non-Linear Simulator 19
4.3.3 Target, Models and Simulator for 1 dof dynamic system 19
4.3.4 Target, Models and Simulator for 5 dof dynamic system 21
4.4 Results and Plots 22
4.4.1 Linear Simulator for 1-DOF 22
4.4.2 Linear Simulator for 5-DOF 23
4.4.3 Nonlinear Simulator for 1-DOF 24
4.4.4 Nonlinear Simulator for 5-DOF 25
4.5 Conclusion 29

5 Future Work **30**
5.1 Stochastic Differential Equations (SDEs) 30
5.2 M-Step Analysis 30
5.3 Hyperparameter Tuning 30
5.4 Exploration of Alternative Approaches 31

Bibliography **32**

6 Appendix **33**
6.1 Matrices for 5-DOF Non-Linear System 33
6.2 Non-Linear Terms for 5-DOF System 34
6.3 State-Space Equation for Non-Linear System 34
6.3.1 Physical Model 35
6.3.2 Partial Differential Equations 35
6.3.3 Numerical Approximation of Partial Differential Equations 35
6.4 Differentiable Physics 36
6.4.1 Differentiable Physics (DP): A Literature Review 36
6.5 Integrating Neural Networks (NN) with Differentiable Physics (DP) . . . 37
6.5.1 Introduction 37
6.5.2 Unrolling for Enhanced Integration 38
6.5.3 Benefits of DP in Training 40

List of Figures

2.1 Transformed solutions of the reference sequence computed on R (blue) differ from solutions computed on the source manifold S (orange). A correction function C (green) updates the state after each iteration to more closely match the projected reference trajectory on S. [1] 3

3.1 Denoting the Euler and RK4 solution for 3 different values of K 9
3.2 Denoting the Euler and RK4 solution for 3 different values of K 10
3.3 Denoting the Euler and RK4 solution for 3 different values of K 10

4.1 Excitation vs Time 15
4.2 Testing 1-DOF - Linear Simulator 22
4.3 Combined Evaluation Metrics for Linear 1-DOF System 23
4.4 Testing 5-DOF - Linear Simulator 24
4.5 Evaluation Metrics for Linear 5-DOF System - Displacement 25
4.6 Evaluation Metrics for Linear 5-DOF System - Velocity 26
4.7 Testing 1-DOF - Nonlinear Simulator 26
4.8 Combined Evaluation Metrics for Nonlinear 1-DOF System 27
4.9 Testing 5-DOF - Nonlinear Simulator 27
4.10 Evaluation Metrics for Nonlinear 5-DOF System - Displacement 28
4.11 Evaluation Metrics for Nonlinear 5-DOF System - Velocity 28

6.1 A network produces an input to a PDE solver, which provides a gradient for training during the backpropagation step. 38
6.2 A PDE solver produces an output which is processed by an NN 38
6.3 Time stepping with interleaved DP and NN operations for k solver iterations. The dashed gray arrows indicate optional intermediate evaluations of loss terms (similar to the solid gray arrow for the last k step), and intermediate outputs of the NN are indicated with a tilde. 39

1 Introduction and Research Objectives

The accurate resolution of partial differential equations (PDEs) stands as a cornerstone in various scientific and engineering disciplines, impacting applications ranging from climate modeling to fluid dynamics. Our research is motivated by the groundbreaking work of Kiwon Um et al [1], who presented a Solver-in-the-Loop approach, demonstrating the effectiveness of learning from differentiable physics to interact with iterative PDE solvers. In their work, they achieved high fidelity by employing finer grids and showcased remarkable results.

Building upon the insights derived from Kiwon Um et al.'s research, and the 34th Conference on Neural Information Processing Systems (NeurIPS 2020) [1], our investigation endeavors to delve deeper into the factors influencing solution accuracy. Inspired by their success in utilizing finer grids for high-fidelity data generation, we aim to explore further dimensions of the problem to enrich our understanding and extend the direction set by their pioneering work.

1.1 Motivation

Our investigation initiates by questioning the influence of numerical methods on solution fidelity. Inspired by the concept of differentiable physics and the related solver in loop concept [2], we focus on the choice of numerical solvers. Specifically, we aim to discern the impact of using higher-fidelity models (instead of fine grid as in original text), such as the Runge-Kutta (RK4) method compared to the traditional Euler method . This exploration seeks to quantify the advantages of employing more sophisticated numerical solvers in enhancing the accuracy of solution outcomes.

1.2 Addressing Information Gaps with Neural Networks

Acknowledging the potential limitations of physical models in capturing the complexities of unknown equations, we pivot to the question of whether neural networks (NN) can serve as compensatory tools. To address this, we leverage data from nonlinear dynamic equations and employ a hybrid approach. Our methodology involves combining a linear model with neural networks for systems with one and five degrees of freedom (1-DOF and 5-DOF) [3]. By integrating traditional mathematical models with machine learning techniques, we aim to explore the extent to which neural networks can compensate for information gaps within dynamic systems [4].

1.3 Extending to Nonlinear Dynamics

Recognizing the inherent challenges posed by nonlinear systems, our investigation extends beyond linear models. We posit that the adaptability and performance of neural networks may be better evaluated in the context of dynamic systems governed by nonlinear equations [5]. This expansion aims to unravel the synergy between traditional mathematical models and machine learning techniques in not only compensating for information gaps but also addressing the complexities introduced by nonlinear dynamics.

1.4 Methodology

Our comprehensive methodology involves a meticulous comparative analysis encompassing numerical solvers, linear and non-linear models, and neural networks. We systematically evaluate the influence of numerical methods on solution fidelity and investigate the compensatory capabilities of neural networks within dynamic equations. The study is designed to provide nuanced insights into the individual and combined contributions of numerical and machine learning approaches.

2 Literature Review

The foundation of our research lies in the Solver-in-the-Loop approach proposed by Kiwon Um et al. [1]. They demonstrated the efficacy of learning from differentiable physics to interact with iterative PDE solvers, achieving high fidelity by employing finer grids. Our investigation stems from this work, aiming to build upon their insights and extend the understanding of factors influencing solution accuracy.

In the original work by Kiwon Um et al., they considered two discrete versions of the same PDE, denoted as P_r and P_s, representing a more accurate discretization and an approximate version, respectively. States, represented as phase space points, evolve over time in each solution manifold. The discretization in time involves reference sequences in the solution manifold R and source sequences in the solution manifold S. A mapping operator T transforms a phase space point from one solution manifold to another. The differences between solutions in the two manifolds are quantified using an L^2-norm, and their learning goal is to derive a correction operator $C(s)$ using a deep neural network, ensuring a lower error for corrected solutions compared to unmodified ones. This correction is recursively applied over time to achieve a more accurate solution [1].

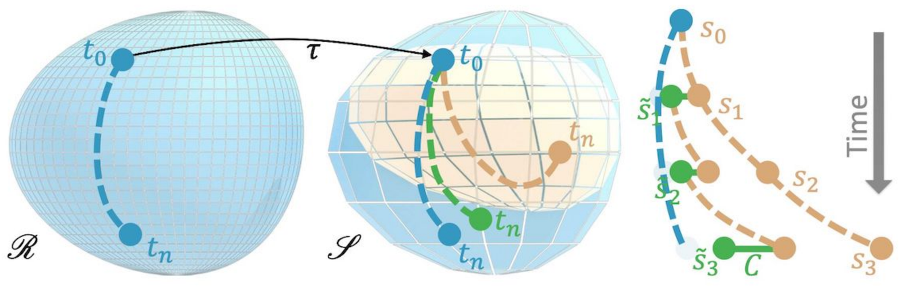


Figure 2.1: Transformed solutions of the reference sequence computed on R (blue) differ from solutions computed on the source manifold S (orange). A correction function C (green) updates the state after each iteration to more closely match the projected reference trajectory on S. [1]

In the original paper, r and s are considered as states at a certain instance in time, representing phase space points. Evolutions over time are given by a trajectory in each solution manifold. In the discrete setting, a solution over time consists of a reference sequence $\{r_t, r_{t+\Delta t}, \ldots, r_{t+k\Delta t}\}$ in the solution manifold R, and correspondingly, a more coarsely approximated source sequence $\{s_t, s_{t+\Delta t}, \ldots, s_{t+k\Delta t}\}$ exists in the solution manifold S [1].

In our adaptation of the methodology, we distinguish our approach by considering s not as coarsely approximated but as approximated through a different physical model, introducing lower fidelity. For instance, we explore different numerical solvers such as Euler and Runge-Kutta (RK4), representing distinct physical models with varying levels of fidelity. This modification allows us to investigate the impact of the choice of numerical solvers on solution accuracy, extending the study beyond a mere coarser approximation.

Additionally, when addressing nonlinear dynamic systems, our approach involves approximating solutions through different physical models, acknowledging the inherent challenges posed by nonlinear equations. The adaptability and performance of neural networks are evaluated in this context, aiming to compensate for information gaps and address complexities introduced by nonlinear dynamics.

2.1 Problem Formulation

To address the challenge of reducing errors in numerical solvers for partial differential equations (PDEs), we introduce a problem formulation that leverages the concept of a physically differentiable solver. Traditional supervised learning methods often fall short when dealing with iterative solvers due to the accumulation of minor inference errors over time. These errors can lead to significant shifts in the data distribution, negatively impacting the solver's stability and performance.

2.1.1 Two Discretizations of the PDE

We consider two different discretizations of the same PDE $\mathcal{P}^*$:

- The *reference* version, denoted as $\mathcal{P}_r$, is assumed to be accurate and produces solutions $\mathbf{r} \in R$, where R represents the manifold of solutions of $\mathcal{P}_r$.
- The *source* version, denoted as $\mathcal{P}_s$, provides a less accurate approximation of the same PDE, with solutions $\mathbf{s} \in S$.

2.1.2 Hybrid Solver with Neural Networks

After training, we aim to create a *hybrid* solver that combines $\mathcal{P}_s$ with a trained neural network (NN) to obtain improved solutions, closer to those produced by $\mathcal{P}_r$. We assume that $\mathcal{P}$ advances a solution by a time step Δt, represented as

$$\mathcal{P}_s^n(\mathcal{T}\mathbf{r}_t) = \mathcal{P}_s(\mathcal{P}_s(\cdots \mathcal{P}_s(\mathcal{T}\mathbf{r}_t)\cdots)).$$

, where $\mathcal{T}$ is a mapping operator that transfers a reference solution to the source manifold.

2.1.3 Error Quantification

To quantify the deviation between the source state $\mathbf{s}_{t+n}$ and the corresponding reference state $\mathbf{r}_{t+n}$, we utilize an L^2-norm-based error function:

$$e(\mathbf{s}_t, \mathcal{T}\mathbf{r}_t) = \|\mathbf{s}_t - \mathcal{T}\mathbf{r}_t\|_2.$$

2.1.4 Learning Objective

The correction function $\mathcal{C}(\mathbf{s}|\theta)$ is represented as a deep neural network with weights θ and receives the state $\mathbf{s}$ to infer an additive correction field with the same dimension. To distinguish the original states $\mathbf{s}$ from the corrected ones, we'll denote the latter with an added tilde $\tilde{\mathbf{s}}$. The overall learning goal now becomes

$$\arg\min_\theta \left((\mathcal{P}_s\mathcal{C})^n(\mathcal{T}\mathbf{r}_t) - \mathcal{T}\mathbf{r}_{t+n} \right)^2$$

The correction depends on the modified states, i.e., it is a function of $\mathcal{C}(\tilde{\mathbf{s}}|\theta)$, states that don't exist beforehand actually evolve over time when training.

2.2 Definitions

$\mathbf{s}$: State in the source manifold.

$\mathbf{r}$: State in the reference manifold.

$\mathcal{T}$: Mapping operator transferring a reference solution to the source manifold.

$\mathcal{P}_r$: Reference version of the PDE with accurate discretization.

$\mathcal{P}_s$: Source version of the PDE with less accurate discretization.

$\mathcal{C}(s|\theta)$: Correction function represented as a deep neural network.

$\tilde{\mathbf{s}}$: Corrected state.

These definitions provide clarity to the key components in the problem formulation and learning objectives.

2.3 Conclusion

In conclusion, the literature review has established the foundation for our research by exploring the Solver-in-the-Loop approach and its application in solving partial differential equations. Building upon this foundation, we have formulated our problem, introducing a physically differentiable solver and addressing the challenges posed by iterative solvers. The problem formulation includes two discretizations of the PDE, a hybrid solver with neural networks, error quantification, and the learning objective. Our adaptations to the original methodology aim to explore the influence of numerical solvers and extend the study to nonlinear dynamics, providing a comprehensive understanding of the interplay between numerical and machine learning approaches in enhancing solution accuracy.

The next chapters will delve into the methodology, experimental setup, results, and discussions, further advancing our understanding and contributing to the evolving field of numerical solvers and machine learning applications in scientific computing.

3 Starting from Ordinary Differential Equations

Consider a first-order ordinary differential equation (ODE) given as:

$$\frac{dy}{dx} = f(k, x, y)$$

where y is the dependent variable, x is the independent variable, and k is a parameter. The function $f(k, x, y)$ represents a family of functions that depend on the parameter k, as well as the variables x and y.

The objective is to numerically solve this ODE using the Euler method, a relatively simple numerical integration technique. However, to enhance the accuracy of the Euler method, a neural network (NN) will be introduced to provide corrections to the predictions in an iterative manner.

3.1 Introduction

In this study, we compare the results obtained by two numerical methods, Euler's method and the fourth-order Runge-Kutta (RK4) method, for solving the ordinary differential equation (ODE) given by $\frac{df}{dx} = x - y$. We applied both methods to obtain two distinct data series and calculated the loss between them.

3.2 ODE and Numerical Methods

The ODE under consideration is $\frac{df}{dx} = x - y$. To solve this equation numerically, we employed two different methods: Euler's method and the RK4 method.

3.3 Euler's Method

Euler's method is a simple and computationally inexpensive numerical method for solving ODEs. The update formula is given by:

$$f_{n+1} = f_n + h \cdot (x_n - y_n), \tag{3.1}$$

where h is the step size, and (x_n, y_n) represents the current values of the variables.

3.4 RK4 Method

The RK4 method is a higher-order numerical method that provides more accurate results compared to Euler's method. The update formula for RK4 is given by:

$$k_1 = h \cdot (x_n - y_n), \tag{3.2}$$

$$k_2 = h \cdot (x_n + \frac{h}{2} - (y_n + \frac{k_1}{2})), \tag{3.3}$$

$$k_3 = h \cdot (x_n + \frac{h}{2} - (y_n + \frac{k_2}{2})), \tag{3.4}$$

$$k_4 = h \cdot (x_n + h - (y_n + k_3)), \tag{3.5}$$

$$f_{n+1} = f_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{3.6}$$

3.5 Results and Analysis

We applied both Euler's method and RK4 method to solve the family of ODE, obtaining two distinct data series. The loss between these series was calculated to assess the difference in accuracy between the two methods. The ODE we are solving is:

$$\frac{dy}{dx} = k \cdot (x - y) \tag{3.7}$$

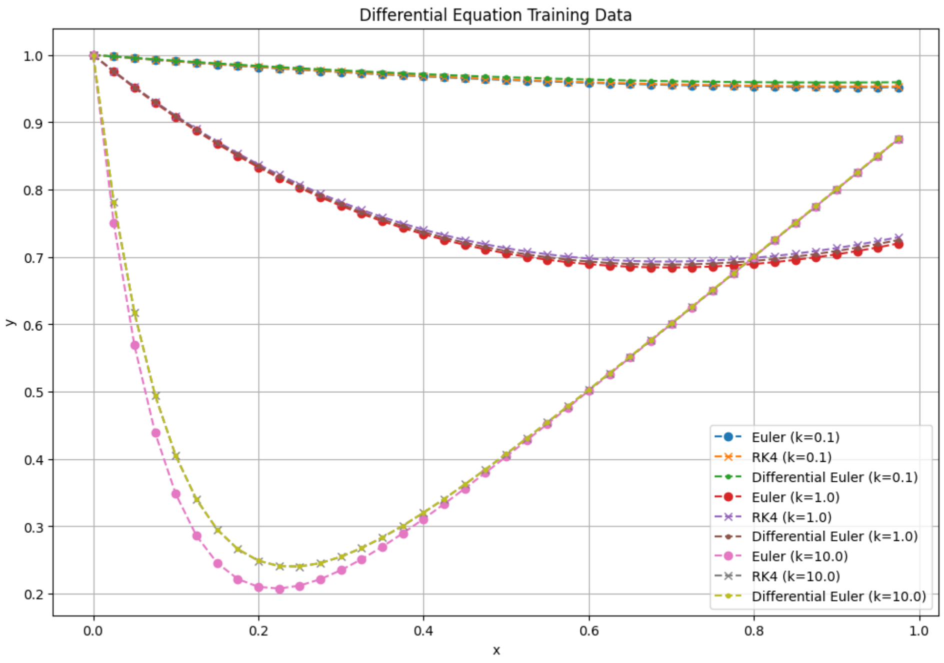


Figure 3.1: Denoting the Euler and RK4 solution for 3 different values of K

3.5.1 Neural Network Integration

For enhanced accuracy, a neural network (NN) is introduced. The NN architecture, denoted as `NeuralNet(3, [8, 16, 8], 1, nn.Sigmoid())`, is trained using the Adam optimizer with a learning rate of 0.1, maximum epochs set to 2000, and a mini-batch size of 40.

3.5.2 Cumulative Error Comparison: Euler vs. RK4 for Testing Data

We evaluate the cumulative errors for both Euler and RK4 methods concerning testing data. The cumulative error is calculated up to the final data point index, offering insights into the overall performance.

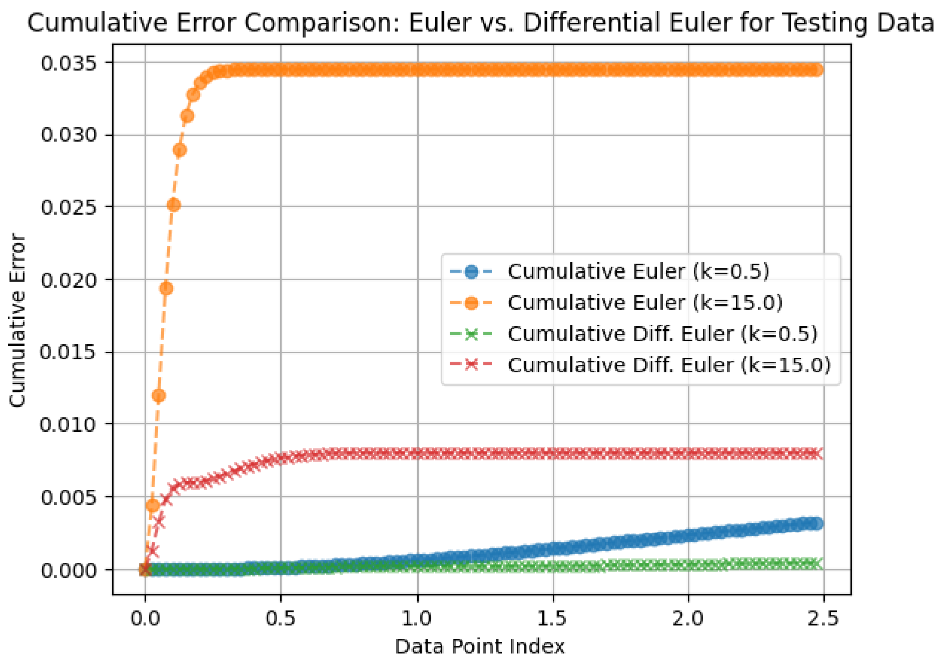


Figure 3.2: Denoting the Euler and RK4 solution for 3 different values of K

3.5.3 Evaluation Metrics for Different k Values

For a comprehensive assessment, we employ evaluation metrics such as R2 score, RMSE, and L2 loss for different k values. These metrics provide a quantitative measure of the models' accuracy and reliability.

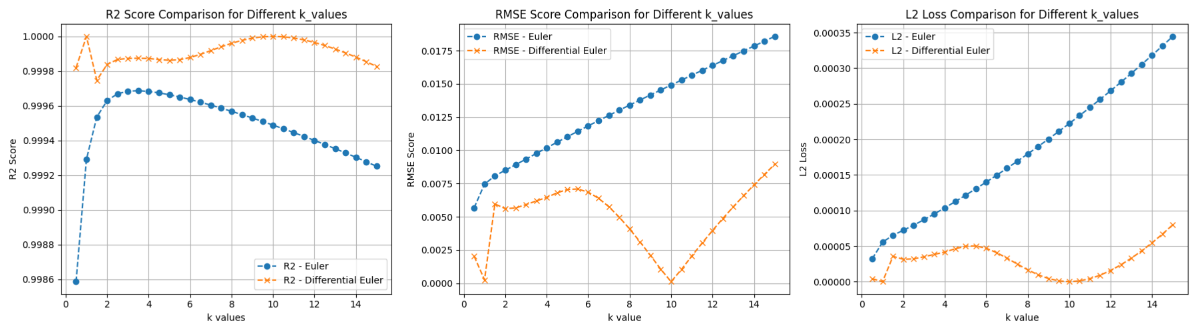


Figure 3.3: Denoting the Euler and RK4 solution for 3 different values of K

The chapter concludes with a visual representation of the results, providing an insightful overview of the comparative performance of Euler's method, RK4 method, and the integrated neural network.

4 Spring-Mass Dynamic System

4.1 Physical Models

4.1.1 1-DOF Linear System

The 1-degree-of-freedom (1-DOF) linear system comprises a mass connected by a linear spring and damper. The spring force is directly proportional to the displacement of the mass, and the damper resistance is directly proportional to the velocity of the mass.

Linear Spring Equation:

$$F_{\text{spring}} = k \cdot x \tag{4.1}$$

Linear Damper Equation:

$$F_{\text{damper}} = c \cdot \dot{x} \tag{4.2}$$

Where:

c is the damping coefficient of the linear damper,
$\dot{x}$ is the velocity of the mass,
k is the stiffness constant of the linear spring,
x is the displacement of the mass.

The 1-DOF system is governed by second-order ordinary differential equations derived from Newton's second law of motion. The equation of motion is expressed as:

$$m \cdot \ddot{x}(t) + c \cdot \dot{x}(t) + k \cdot x(t) = u(t) \tag{4.3}$$

4.1.2 General Form for N-DOF Linear System

The general form for the N-DOF linear system is:

$$M\ddot{x} + C\dot{x} + Kx = U \tag{4.4}$$

Where:

M is the mass matrix,
C is the damping matrix,
K is the stiffness matrix,
x is the displacement vector,
$\dot{x}$ is the velocity vector,
U is the external force vector.

4.1.3 Vectorization of the Dynamic Equation

To represent the above dynamic equation in a state-space form, define the state vector $\mathbf{y}$ for each degree of freedom:

$$\mathbf{y}_i = [x_i, \dot{x}_i]$$

Now, express the system of second-order differential equations as a set of first-order differential equations:

$$\dot{\mathbf{y}}_i = [\dot{x}_i, \ddot{x}_i] = [\dot{x}_i, M_i^{-1}(U_i - C_i\dot{x}_i - K_i x_i)]$$

Define the state vector $\mathbf{y}$ for the entire system as:

$$\mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]$$

where n is the number of degrees of freedom.

4.1.4 Closed-Form Solution for Vector Equation

TFor a dynamical system of the form $M\ddot{x} + C\dot{x} + Kx = U$, the state vector can be idealized as $y = [x, \dot{x}]^T$. Now, the equation for the dynamical system can be rearranged as:

$$\dot{y} = A_c y + B_c u,$$

where

$$A_c = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix}$$

and

$$B_c = \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix}.$$

Here:

- $\dot{y}$ represents the derivative of the state vector y with respect to time.
- A_c is the state matrix that characterizes the inherent dynamics of the system.
- B_c is the input matrix relating the input excitation u to the system dynamics.

To obtain a closed-form solution, consider the homogeneous solution, particular solution, and their combination:

Homogeneous Solution:

$$\mathbf{y}_h(t) = e^{\mathbf{A}_c t}\mathbf{y}_0$$

where $\mathbf{y}_0$ is the initial state vector.

Particular Solution:

$$\mathbf{y}_p(t) = (\mathbf{A}_c - \mathbf{I})^{-1}\mathbf{y}_0 + (\mathbf{A}_c - \mathbf{I})^{-1}\int_0^t e^{-\mathbf{A}_c s}\mathbf{B}_c\mathbf{U}(s)\,ds$$

The inverse Laplace transform of $\mathbf{Y}(s)$ gives the particular solution in the time domain:

$$\mathbf{y}_p(t) = e^{\mathbf{A}_c t}\int_0^t e^{-\mathbf{A}_c s}\mathbf{B}_c\mathbf{U}(s)\,ds$$

Complete Solution:

The complete solution is the sum of the homogeneous and particular solutions:

$$\mathbf{y}(t) = \mathbf{y}_h(t) + \mathbf{y}_p(t)$$

$$\mathbf{y}(t) = e^{\mathbf{A}_c t}\mathbf{y}_0 + e^{\mathbf{A}_c t}\int_0^t e^{-\mathbf{A}_c s}\mathbf{B}_c\mathbf{U}(s)\,ds$$

This expression provides the closed-form solution for the linear vector dynamic system in state-space representation. The matrices $\mathbf{A}_c$ and $\mathbf{B}_c$, and the input excitation $\mathbf{U}$ play crucial roles in determining the dynamic behavior of the system. The matrix exponential $e^{\mathbf{A}_c t}$ is calculated using numerical methods or pre-existing solutions tailored to the specific system properties.

Additionally, for the discrete-time system, the evolution of the state vector is given by:

$$\mathbf{y}_{t+1} = \mathbf{A}_d \mathbf{y}_t + \mathbf{B}_d \mathbf{u}_t$$

where $\mathbf{A}_d = \exp(\mathbf{A}_c \Delta t)$ and $\mathbf{B}_d = (\mathbf{A}_d - \mathbf{I})\mathbf{A}_c^{-1}\mathbf{B}_c$.

4.1.5 Non-Linear Spring-Mass System

In our investigation, we introduced non-linearities in the 1-DOF system by considering cubic stiffness non-linearity ($k_3 x^3$) and quadratic damping linearity ($c_2 x^2$). This choice reflects real-world scenarios where systems often exhibit complex, non-linear behavior. The equation of motion for our non-linear system is given by:

$$m \cdot \ddot{x}(t) + c \cdot \dot{x}(t) + k \cdot x(t) = u(t) - \eta(x(t))$$

Here, m is the mass, c is the damping coefficient, k is the stiffness constant, $x(t)$ is the displacement, $\dot{x}(t)$ is the velocity, $u(t)$ is the input excitation, and $\eta(x(t))$ captures the non-linear effects.

The state-space equation for the non-linear system can be expressed as:

$$\dot{y}(t) = A \cdot y(t) + B \cdot u(t) + C$$

Here, $\dot{y}(t)$ represents the derivative of the state vector $x(t)$ with respect to time, A is the state matrix characterizing the inherent dynamics, B is the input matrix relating the input excitation to the system dynamics, and C is a term incorporating non-linear effects.

To harness the solution for the state-space equation, we employed the Runge-Kutta 4th order (RK-4) numerical integration method. RK-4 is a robust technique that iteratively calculates the state vector's evolution over time, providing a numerical approximation of the system's response to the non-linear dynamics. This approach allows us to obtain a solution for the state vector that captures the intricate behavior introduced by the non-linearities in the system.

4.1.6 Excitation Signal u

In our simulations, we employ a filtered Gaussian White Noise as the input excitation, which is achieved using a lowpass Butterworth filter with a cutoff frequency of 20 Hz.

We conduct training on three distinct datasets where σ_u (standard deviation of the excitation) is set to 1, 3, and 5. To assess the robustness and generalization of our results, we perform testing across various values of σ_u within the specified range.

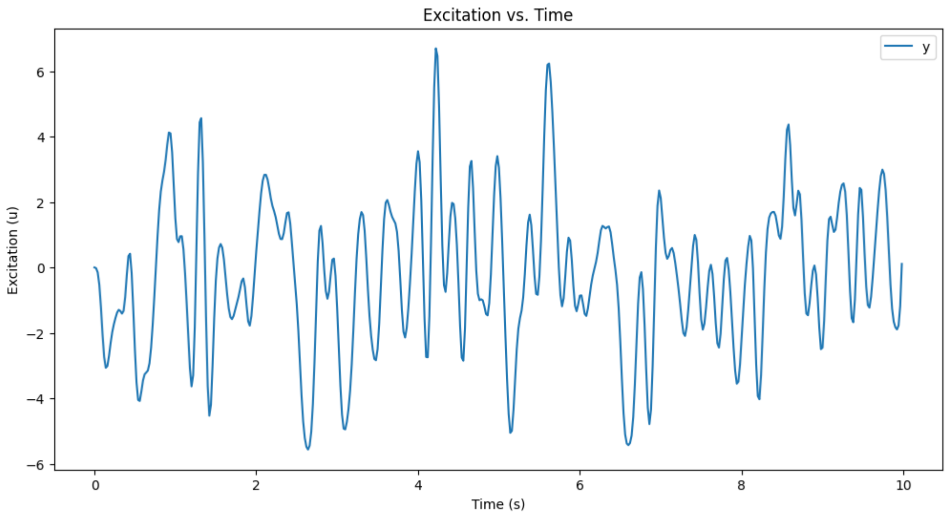


Figure 4.1: Excitation vs Time

4.2 Machine Learning Models for Linear Simulator

4.2.1 $SOL_{\textsf{linear}}$ - Solver in Loop Model with Linear Simulator

Update Process:

1. **System State Evolution:** Simulate the next state using the physical model: $y_{t+1} \rightarrow \text{simulator}(y_t, u_t)$.
2. **Correction Step:** Update the simulated state with the correction from Neural Network: $y_{t+1} \rightarrow y_{t+1} + \text{NN}_{linear}(u_t, y_{t+1})$.

y_t represents the initial system state, and after each update step, y_{t+1} represents the simulated system state with the nonlinear model correction. Here, u_t represents the excitation at the given time t.

4.2.2 $SOL^{\epsilon}_{\textsf{linear}}$ - Solver in Loop Epsilon Model with Linear Simulator

Update Process:

1. **System State Evolution:** Simulate the next state using the physical model: $y_{t+1} \rightarrow \text{simulator}(y_t, u_t)$.
2. **Correction Step:** Compute the correction term using the neural network in epsilon model: $\epsilon \rightarrow \text{NN}^{\epsilon}_{linear}(u_t, y_t)$.

3. **Update Simulated State:** Update the simulated state incorporating the correction: $y_{t+1} \to y_t + \text{simulator}(\epsilon)$.

y_t represents the initial system state, and after each update step, y_{t+1} represents the simulated system state with the nonlinear epsilon model correction. Here, u_t represents the excitation at the given time t.

4.2.3 Target, Models, and Simulator for 1-DOF Dynamic System

System Parameters:

- $m = 1.0$ - mass
- $c = 0.2$ - damping coefficient
- $k = 100.0$ - spring constant

Time:

- $t_0 = 0$ - initial time
- $t_f = 10$ - final time

Frequency:

- $fs = 50$ - frequency

Input Excitation:

- u is filtered Gaussian White Noise using a lowpass Butterworth filter with a cutoff frequency of 20 Hz.

Target Solution: Non-Zero Coefficients for Cubic Terms

Consider a scenario with non-zero coefficients for cubic terms:

$$[k3] = [10000], \quad [c2] = [0]$$

The system equation becomes:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) + C_{\text{cubic stiffness}}$$

Here, $C_{\text{cubic stiffness}}$ incorporates the effects of cubic and quadratic terms based on the specified $[k3]$.

SOL_{linear} - **Solver in Loop Model with Linear Simulator**

Neural Network Architecture:

- Feedforward structure with five hidden layers (16, 32, 64, 32, 16).
- Input layer size: 3 (concatenation of input u_t and current state y_t).
- Output layer size: 2, representing the system's states.
- Activation Function: Rectified Linear Unit (ReLU) applied between hidden layers.

$SOL^{\epsilon}_{\text{linear}}$ - **Solver in Loop Epsilon Model with Linear Simulator**

Neural Network Architecture:

- Feedforward structure with five hidden layers (16, 32, 64, 32, 16).
- Input layer size: 3 (concatenation of input u_t and current state y_t).
- Output layer size: 1, representing the system's degrees of freedom.
- Activation Function: Rectified Linear Unit (ReLU) applied between hidden layers.

Training Process:

- Initial 50,000 iterations with lr (learning rate) of 0.0005 and $mstep$ (time steps per iteration) set to 1.
- Additional 50,000 iterations with an increased $mstep$ of 3.

4.2.4 Target, Models, and Simulator for 5-DOF Dynamic System

System Parameters for 5-DOF Dynamic System:

- $ndof = 5$ - Number of degrees of freedom
- $kvec = [95, 98, 100, 104, 97]$ - Vector of spring constants for each degree of freedom
- $cvec = [0.18, 0.22, 0.24, 0.18, 0.19]$ - Vector of damping coefficients for each degree of freedom
- $mvec = [1, 1, 1, 0.8, 0.8]$ - Vector of masses for each degree of freedom

Time:

- $t_0 = 0$ - Initial time
- $t_f = 10$ - Final time

Frequency:

- $fs = 50$ - Frequency

Target Solution: Non-Zero Coefficients for Cubic Terms

Consider a scenario with non-zero coefficients for cubic terms:

$$[k3] = [10000, 0, 0, 5000, 0], \quad [c2] = [0, 0, 0, 0, 0]$$

The system equation becomes:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) + C_{\text{cubic stiffness}}$$

Here, $C_{\text{cubic stiffness}}$ incorporates the effects of cubic and quadratic terms based on the specified $[k3]$.

$SOL_{\textsf{linear}}$ - Solver in Loop Model with Linear Simulator

Neural Network Architecture:

- Feedforward structure with five hidden layers (16, 32, 64, 32, 16).
- Input layer size: 11 (concatenation of input u_t and current state y_t).
- Output layer size: 10, representing the system's states.
- Activation Function: Rectified Linear Unit (ReLU) applied between hidden layers.

$SOL^{\epsilon}_{\textsf{linear}}$ - Solver in Loop Epsilon Model with Linear Simulator

Neural Network Architecture:

- Feedforward structure with five hidden layers (16, 32, 64, 32, 16).
- Input layer size: 11 (concatenation of input u_t and current state y_t).
- Output layer size: 5, representing the system's degrees of freedom.
- Activation Function: Rectified Linear Unit (ReLU) applied between hidden layers.

Training Process:

- Initial 100,000 iterations with lr (learning rate) of 0.0005 and $mstep$ (time steps per iteration) set to 1.

4.3 Machine Learning Models for Non-Linear Simulator

4.3.1 $SOL_{\textsf{nonlinear}}$ -Solver in Loop Model with Non-Linear Simulator

Update Process:

1. **System State Evolution:** Simulate the next state using the physical model: $y_{t+1} \to \text{simulator}(y_t, u_t)$.
2. **Correction Step:** Update the simulated state with the correction from neural network: $y_{t+1} \to y_{t+1} + \text{NN}_{nonlinear}(u_t, y_{t+1})$.

y_t represents the initial system state, and after each update step, y_{t+1} represents the simulated system state with the nonlinear model correction. Here, u_t represents the excitation at the given time t.

4.3.2 $SOL^{\epsilon}_{\textsf{nonlinear}}$ - Solver in Loop Epsilon Model with Non-Linear Simulator

Update Process:

1. **Input Correction:** Update the excitation signal accounting for non-linearity term calculated by neural network: $u_t \to u_t + \text{NN}^{\epsilon}_{nonlinear}(u_t, y_t)$.
2. **System State Evolution:** Simulate the next state using the physical model: $y_{t+1} \to \text{simulator}(y_t, u_t)$.

y_t represents the initial system state, and after each update step, y_{t+1} represents the simulated system state with the nonlinear epsilon model correction. Here, u_t represents the excitation at the given time t.

4.3.3 Target, Models and Simulator for 1 dof dynamic system

Target Solution:- Non-Zero Coefficients for Cubic Terms

Consider a scenario with non-zero coefficients for cubic terms:

$$[k3] = [10000], \quad [c2] = [0]$$

The system equation becomes:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) + C_{\text{cubic stiffness}}$$

Here, $C_{\text{cubic stiffness}}$ incorporates the effects of cubic and terms based on the specified $[k3]$.

Simulator:- Non-Zero Coefficients for Quadratic Damping Terms

Consider another scenario with non-zero coefficients for linear damping terms:

$$[k3] = [0], \quad [c2] = [1]$$

The system equation becomes:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) + C_{\text{quadratic damping}}$$

Here, $C_{\text{quadratic damping}}$ incorporates quadratic damping terms based on the specified $[c2]$.

$SOL_{\textsf{nonlinear}}$ - Solver in Loop Model with Non-Linear Simulator

Neural Network Architecture:

- Feedforward structure with five hidden layers (16, 32, 64, 32, 16).
- Input layer size: 3 (concatenation of input u_t and current state y_t).
- Output layer size: 2, representing the system's states.
- Activation Function: Rectified Linear Unit (ReLU) applied between hidden layers.

$SOL^{\epsilon}_{\textsf{nonlinear}}$ - Solver in Loop Epsilon Model with Non-Linear Simulator

Neural Network Architecture:

- Feedforward structure with five hidden layers (16, 32, 64, 32, 16).
- Input layer size: 3 (concatenation of input u_t and current state y_t).
- Output layer size: 1, representing the system's degrees of freedom.
- Activation Function: Rectified Linear Unit (ReLU) applied between hidden layers.

Training Process:

- Initial 50,000 iterations with lr (learning rate) of 0.0005 and $mstep$ (time steps per iteration) set to 1.
- Additional 50,000 iterations with an increased $mstep$ of 3.

4.3.4 Target, Models and Simulator for 5 dof dynamic system

Target Solution:- Non-Zero Coefficients for Cubic Terms

Consider a scenario with non-zero coefficients for cubic terms:

$$[k3] = [10000, 0, 0, 5000, 0], \quad [c2] = [0, 0, 0, 0, 0]$$

The system equation becomes:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) + C_{\text{cubic stiffness}}$$

Here, $C_{\text{cubic stiffness}}$ incorporates the effects of cubic and quadratic terms based on the specified $[k3]$.

Simulator:- Non-Zero Coefficients for Quadratic Damping Terms

Consider another scenario with non-zero coefficients for linear damping terms:

$$[k3] = [0, 0, 0, 0, 0], \quad [c2] = [1, 0, 0.8, 0, 0]$$

The system equation becomes:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) + C_{\text{quadratic damping}}$$

Here, $C_{\text{quadratic damping}}$ incorporates quadratic damping terms based on the specified $[c2]$.

$SOL_{\textbf{nonlinear}}$ -Solver in Loop Model with Non-Linear Simulator

Neural Network Architecture:

- Feedforward structure with five hidden layers (16, 32, 64, 32, 16).
- Input layer size: 11 (concatenation of input u_t and current state y_t).
- Output layer size: 10, representing the system's states.
- Activation Function: Rectified Linear Unit (ReLU) applied between hidden layers.

$SOL^{\epsilon}_{\text{nonlinear}}$ **- Solver in Loop Epsilon Model with Non-Linear Simulator**

Neural Network Architecture:

- Feedforward structure with five hidden layers (16, 32, 64, 32, 16).
- Input layer size: 11 (concatenation of input u_t and current state y_t).
- Output layer size: 5, representing the system's degrees of freedom.
- Activation Function: Rectified Linear Unit (ReLU) applied between hidden layers.

Training Process:

- Initial 100,000 iterations with lr (learning rate) of 0.0005 and $mstep$ (time steps per iteration) set to 1.

4.4 Results and Plots

We will be using R^2, RMSE, and L^2 as evaluation parameters in the following subsections.

4.4.1 Linear Simulator for 1-DOF

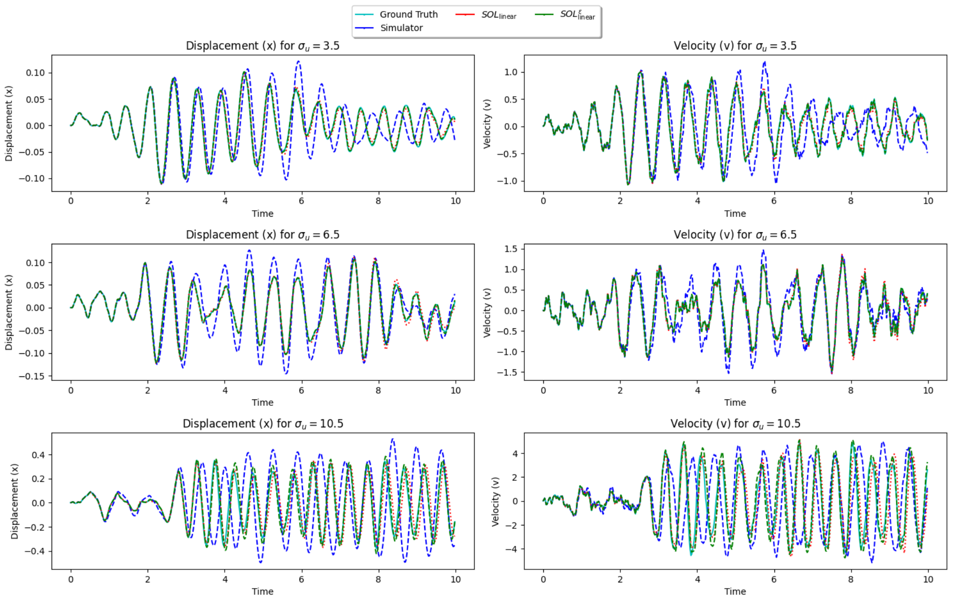


Figure 4.2: Testing 1-DOF - Linear Simulator

Evaluation Metrics vs σ_u

Figure 4.3 shows the combined evaluation metrics for the Linear 1-DOF system.

In the case of the Linear 1-DOF system, the SOL_{linear} model demonstrates strong performance until σ_u reaches 6. Beyond this point, deviations become noticeable in the evaluation plots. This behavior suggests that SOL_{linear} might struggle to generalize well for higher excitation levels. On the other hand, the $SOL^{\epsilon}_{\text{linear}}$ model, incorporating an epsilon formulation, consistently performs better across a broader range of σ_u values. The epsilon formulation appears to enhance the model's ability to capture and adapt to variations in the input, resulting in improved accuracy.

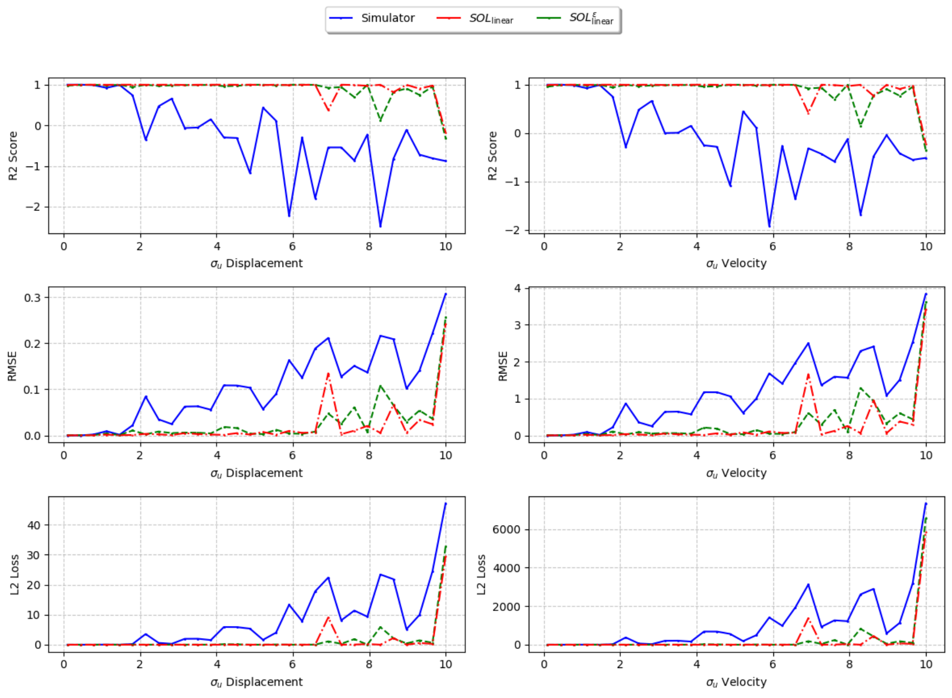


Figure 4.3: Combined Evaluation Metrics for Linear 1-DOF System

4.4.2 Linear Simulator for 5-DOF

Evaluation Metrics vs σ_u - Displacement

Figure 4.5 presents the evaluation metrics for the Linear 5-DOF system in terms of displacement.

Evaluation Metrics vs σ_u - Velocity

Figure 4.6 illustrates the evaluation metrics for the Linear 5-DOF system concerning velocity.

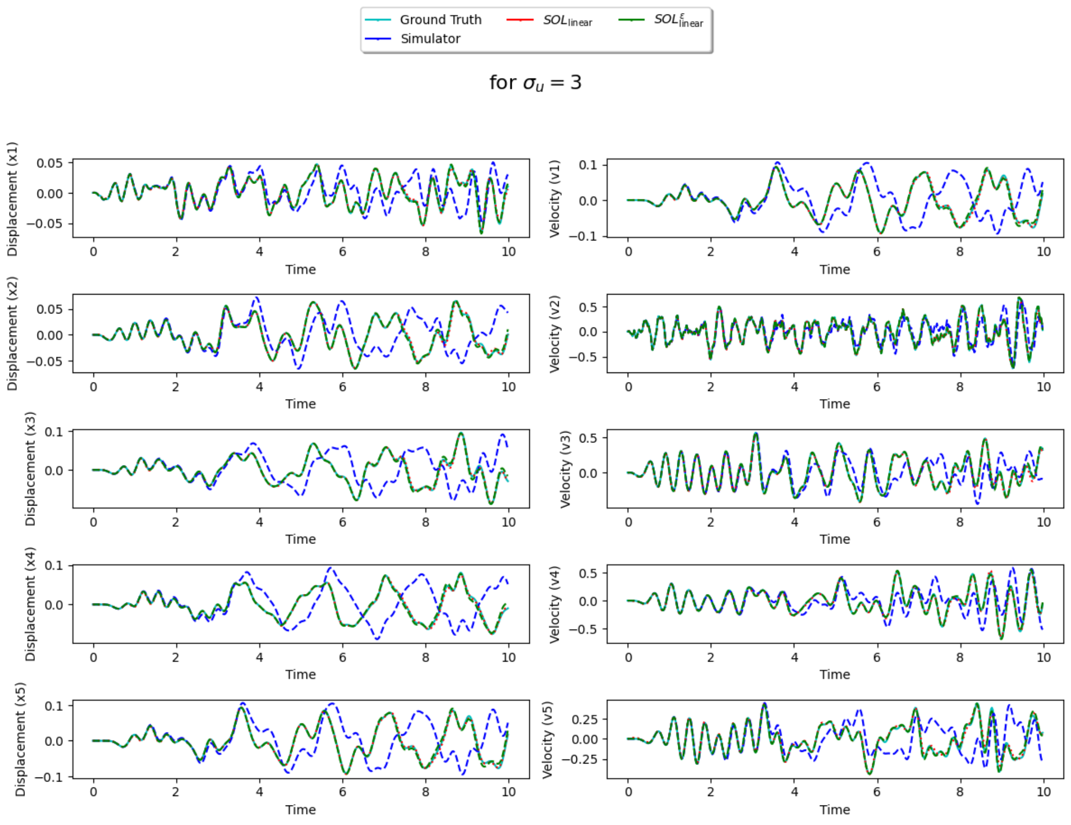


Figure 4.4: Testing 5-DOF - Linear Simulator

For the Linear 5-DOF system, deviations in SOL_{linear} are observed when σ_u equals 5, indicating potential limitations in the model's ability to handle higher excitation levels. Interestingly, while $SOL^{\epsilon}_{\text{linear}}$ fits the training data significantly better (at the $1e-7$ order compared to $1e-4$ for SOL_{linear}), it exhibits lower performance. This discrepancy is attributed to overfitting, where the model becomes too tailored to the training data and struggles to generalize well to new inputs.

4.4.3 Nonlinear Simulator for 1-DOF

Evaluation Metrics vs σ_u

Figure 4.8 displays the combined evaluation metrics for the Nonlinear 1-DOF system.

In the Nonlinear 1-DOF system, $SOL^{\epsilon}_{\text{nonlinear}}$ performs exceptionally well, maintaining accuracy up to $\sigma_u = 8$. In contrast, the normal $SOL_{\text{nonlinear}}$ model performs satisfactorily until $\sigma_u = 6$. The epsilon formulation proves effective in handling higher values of σ_u, showcasing its adaptability and robustness in capturing nonlinear dynamics.

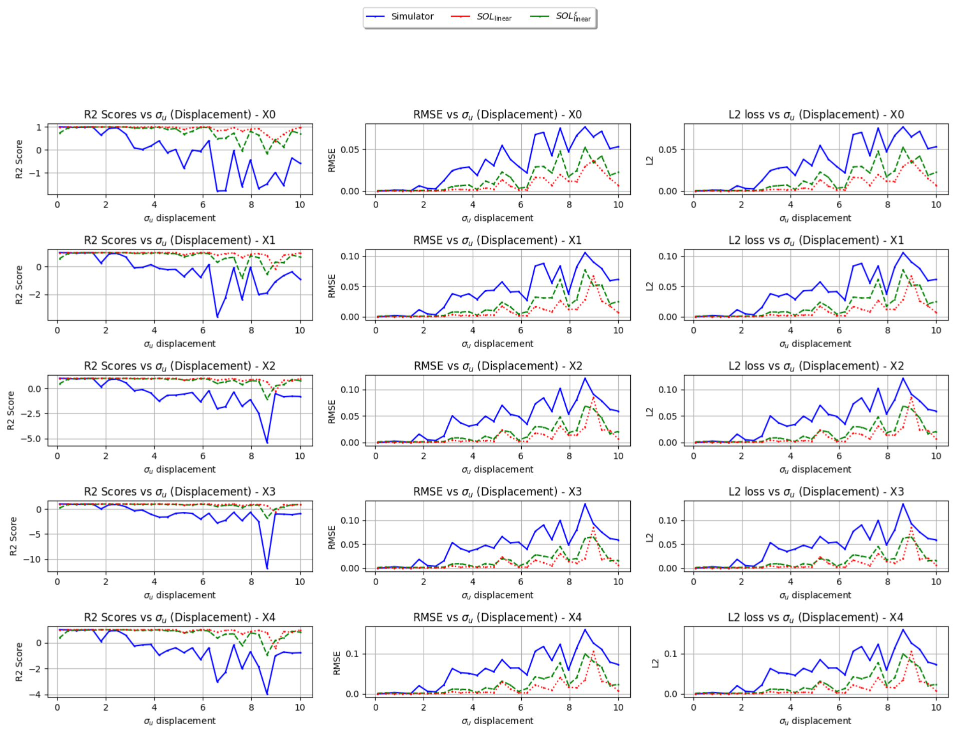


Figure 4.5: Evaluation Metrics for Linear 5-DOF System - Displacement

4.4.4 Nonlinear Simulator for 5-DOF

Evaluation Metrics vs σ_u - Displacement

Figure 4.10 exhibits the evaluation metrics for the Nonlinear 5-DOF system in terms of displacement.

Evaluation Metrics vs σ_u - Velocity

Figure 4.11 showcases the evaluation metrics for the Nonlinear 5-DOF system concerning velocity.

In the Nonlinear 5-DOF system, $SOL^{\epsilon}_{\text{nonlinear}}$ outperforms its counterpart, especially for σ_u values beyond 3. It demonstrates stability until $\sigma_u = 3$, after which deviations become apparent. On the other hand, the normal $SOL_{\text{nonlinear}}$ model faces challenges, yielding poor results in the σ_u range of (3, 5). This emphasizes the enhanced performance of the epsilon formulation in capturing the complexities of the nonlinear system.

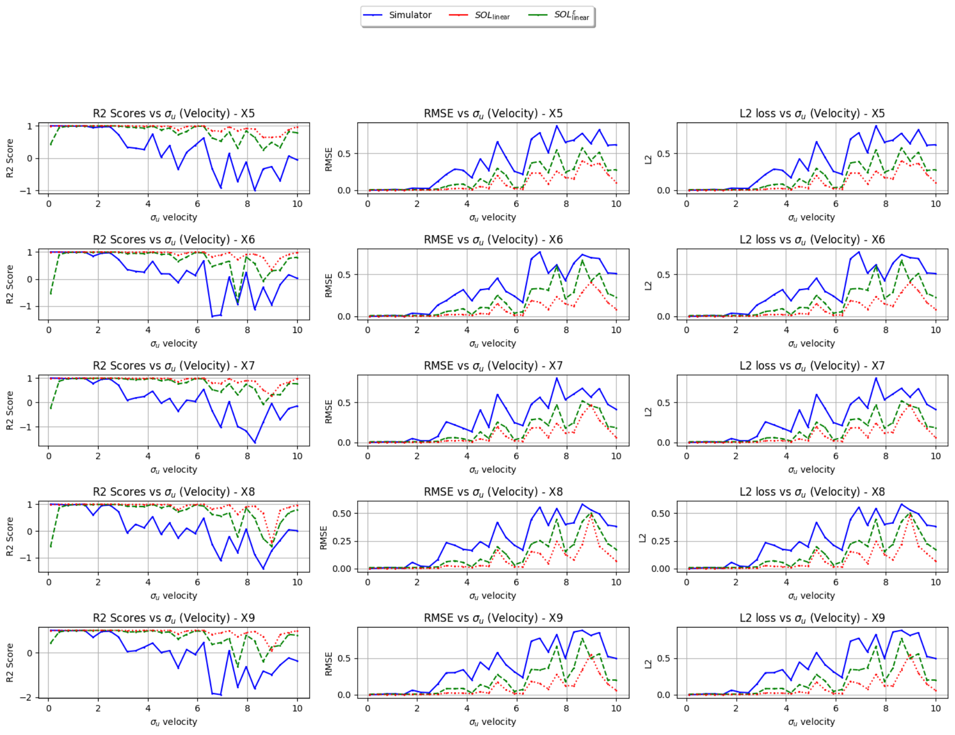


Figure 4.6: Evaluation Metrics for Linear 5-DOF System - Velocity

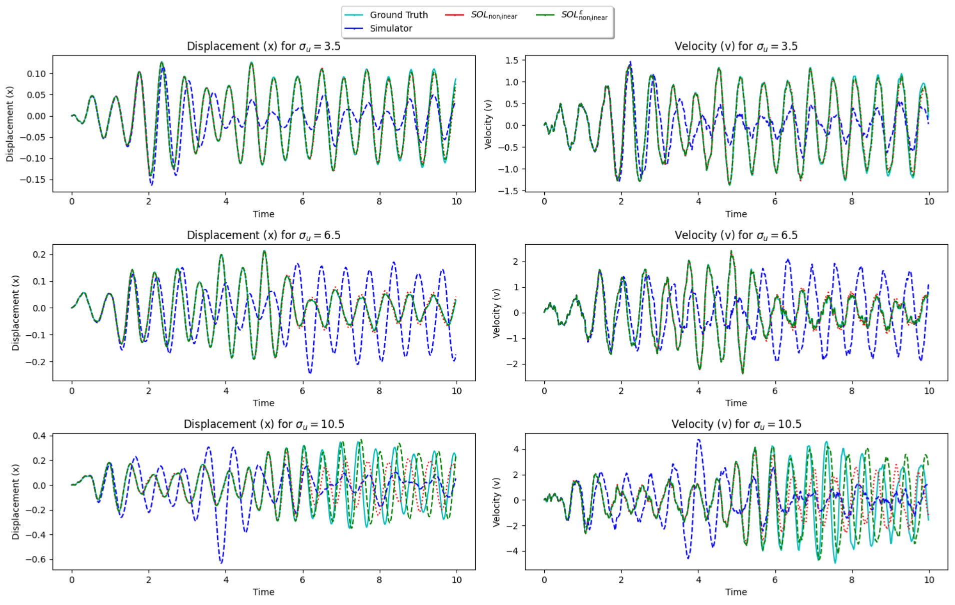


Figure 4.7: Testing 1-DOF - Nonlinear Simulator

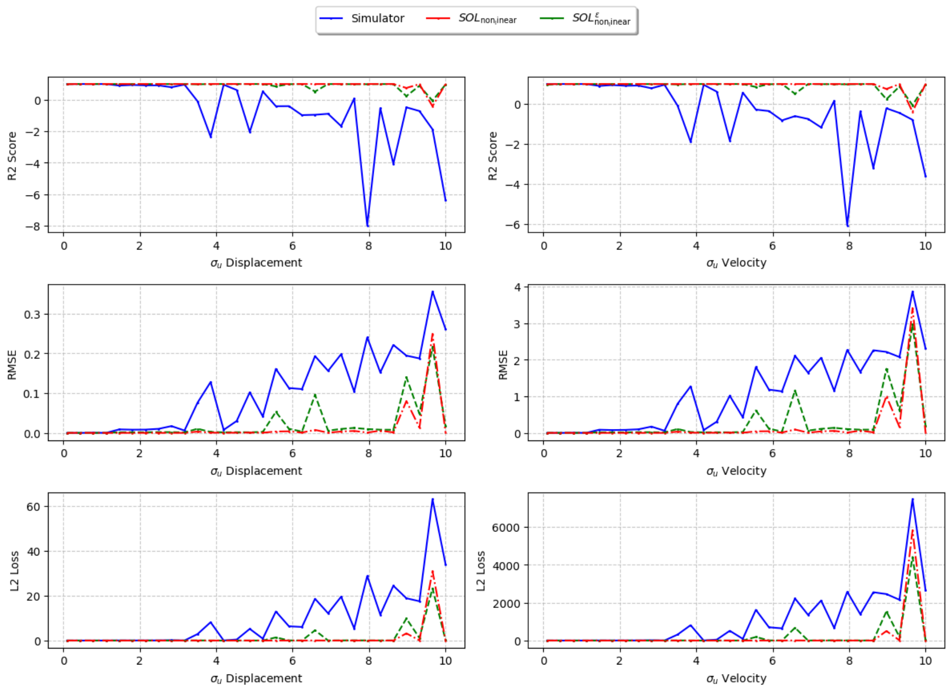


Figure 4.8: Combined Evaluation Metrics for Nonlinear 1-DOF System

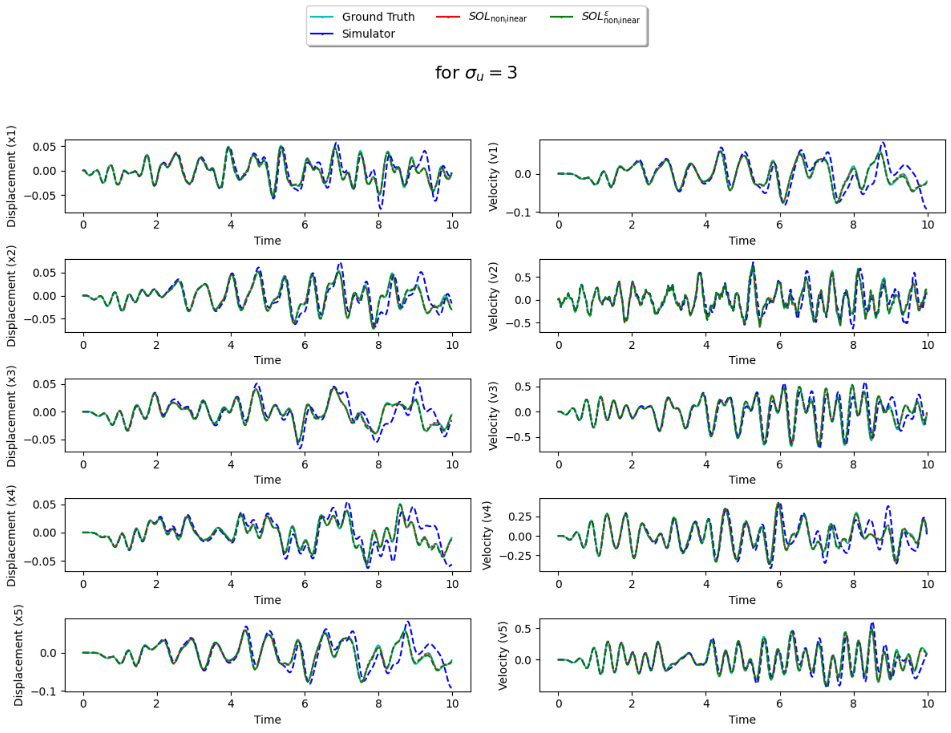


Figure 4.9: Testing 5-DOF - Nonlinear Simulator

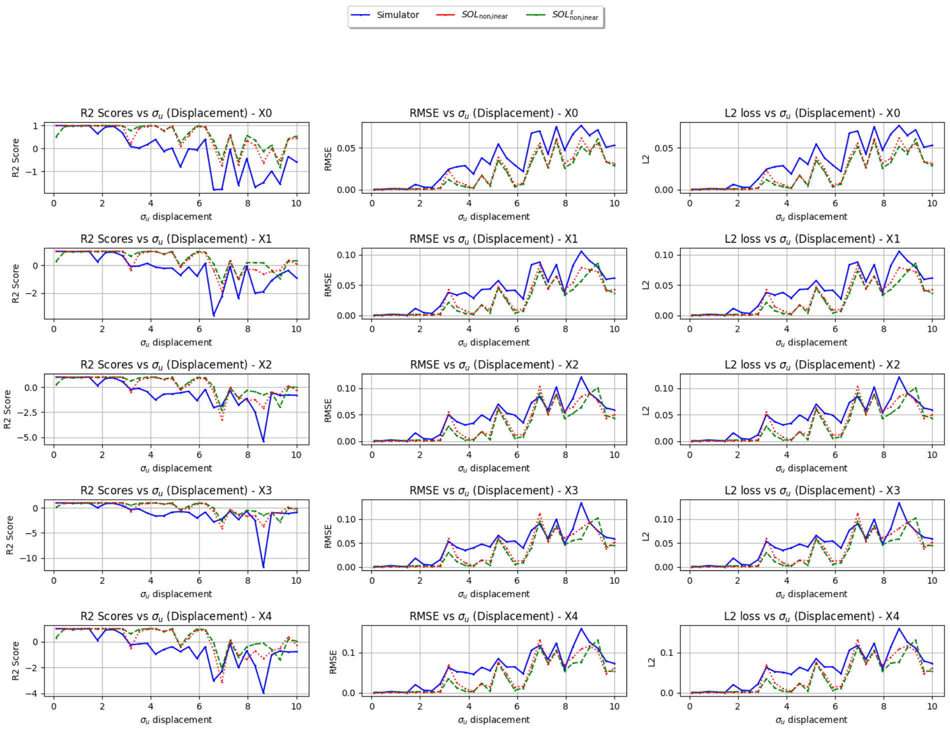


Figure 4.10: Evaluation Metrics for Nonlinear 5-DOF System - Displacement

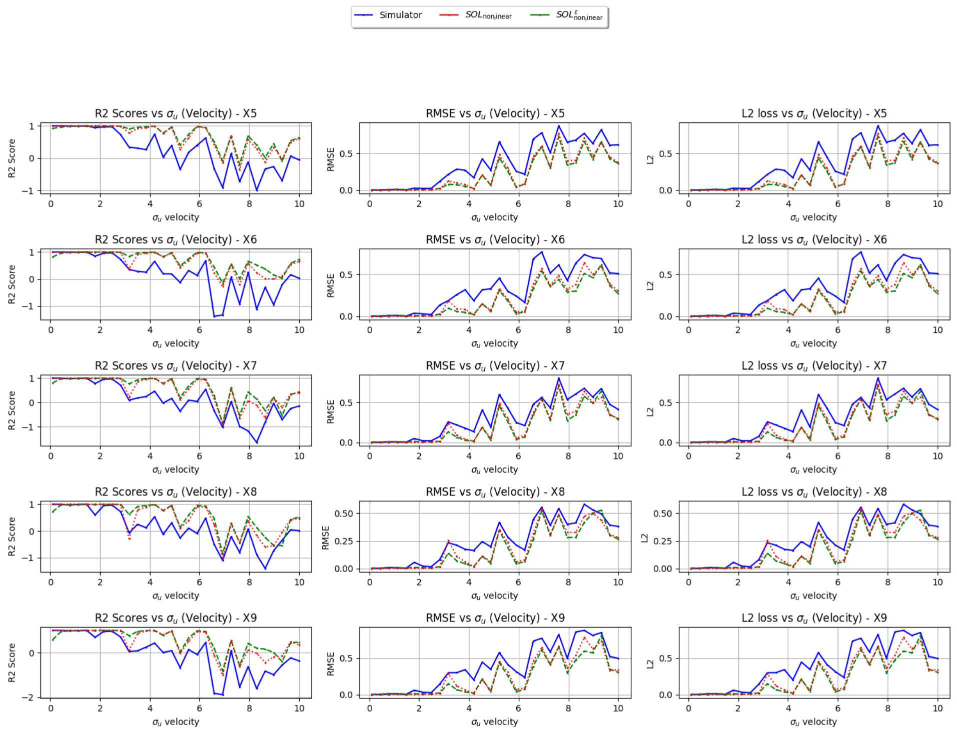


Figure 4.11: Evaluation Metrics for Nonlinear 5-DOF System - Velocity

4.5 Conclusion

In conclusion, the epsilon formulations ($SOL^{\epsilon}_{\text{linear}}$ and $SOL^{\epsilon}_{\text{nonlinear}}$) exhibit improved performance, particularly in scenarios involving higher excitation (σ_u) or nonlinear dynamics. The epsilon formulation introduces a flexibility that allows the model to adapt to a broader range of input conditions, leading to enhanced accuracy and robustness. However, it is crucial to strike a balance, as overly complex models may risk overfitting and compromise generalization.

5 Future Work

The exploration of solving differential equations using neural networks provides a solid foundation for future investigations. Several avenues can be pursued to extend and enhance the current study:

5.1 Stochastic Differential Equations (SDEs)

While the current study focuses on ordinary differential equations (ODEs), the extension to stochastic differential equations (SDEs) presents an intriguing direction for future research. Incorporating stochastic elements into the equations introduces a new layer of complexity, mirroring real-world scenarios where systems are influenced by random processes. Investigating the application of neural networks to solve SDEs could yield valuable insights into modeling dynamic systems under uncertain conditions.

5.2 M-Step Analysis

The current study acknowledges a limitation in analyzing the impact of the mstep parameter due to computational constraints. The analysis is restricted to mstep - 2 in the linear model and mstep - 1 in the non-differentiable model for the 5-DOF system. Future work should aim to overcome these constraints, allowing for a comprehensive examination of the mstep parameter's influence on model performance. Understanding the optimal mstep value could provide valuable insights into the dynamics of the system and improve the efficiency of the neural network-based solution.

5.3 Hyperparameter Tuning

The study recognizes the importance of hyperparameter tuning in optimizing the performance of neural network models. Future work should delve deeper into systematically tuning hyperparameters, such as learning rates, batch sizes, and network architectures, to uncover the most effective configurations for solving differential equations. A rigorous hyperparameter tuning process can lead to more robust and accurate models.

5.4 Exploration of Alternative Approaches

As the field of machine learning continues to evolve, exploring alternative approaches and architectures for solving differential equations remains a promising avenue. Investigating advanced neural network architectures, regularization techniques, or novel optimization algorithms may uncover more effective strategies for tackling the challenges posed by differential equations.

In conclusion, the current study sets the stage for future research endeavors in the domain of solving differential equations using neural networks. The proposed directions aim to expand the scope, enhance analytical capabilities, and explore innovative approaches, contributing to the ongoing advancement of this interdisciplinary field.

Bibliography

[1] Kiwon Um, Robert Brand, Yun Fei, Philipp Holl, Nils Thuerey. *Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers.* (2021).

[2] Nils Thuerey, Philipp Holl, Maximilian Mueller, Patrick Schnell, Felix Trost, Kiwon Um. *Physics-based Deep Learning.* `https://physicsbaseddeeplearning.org`. (2021).

[3] Subramanian, A., Mahadevan, S. (2023). Probabilistic physics-informed machine learning for dynamic systems. *Reliability Engineering and System Safety*, 230, 108899. `https://doi.org/10.1016/j.ress.2022.108899`.

[4] Philipp Holl, Vladlen Koltun, Nils Thuerey. *Learning to control PDEs with differentiable physics.* `https://ge.in.tum.de/publications/2020-iclr-holl/`. (2019).

[5] Moss, J. D., Opolka, F. L., Dumitrascu, B., & Lió, P. (2021). Approximate Latent Force Model Inference. ArXiv. `https://arxiv.org/abs/2109.11851`.

6 Appendix

6.1 Matrices for 5-DOF Non-Linear System

The values of the matrices for the 5-degree-of-freedom (5-DOF) non-linear system are as follows:

$$A = \begin{bmatrix} 0 & I_5 \\ -M^{-1}K & -M^{-1}C \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix}$$

$$M = \begin{bmatrix} m_1 & 0 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 & 0 \\ 0 & 0 & m_3 & 0 & 0 \\ 0 & 0 & 0 & m_4 & 0 \\ 0 & 0 & 0 & 0 & m_5 \end{bmatrix}$$

$$C = \begin{bmatrix} c_1 + c_2 & -c_2 & 0 & 0 & 0 \\ -c_2 & c_2 + c_3 & -c_3 & 0 & 0 \\ 0 & -c_3 & c_3 + c_4 & -c_4 & 0 \\ 0 & 0 & -c_4 & c_4 + c_5 & -c_5 \\ 0 & 0 & 0 & -c_5 & c_5 \end{bmatrix}$$

$$K = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & 0 & 0 \\ -k_2 & k_2 + k_3 & -k_3 & 0 & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 & 0 \\ 0 & 0 & -k_4 & k_4 + k_5 & -k_5 \\ 0 & 0 & 0 & -k_5 & k_5 \end{bmatrix}$$

6.2 Non-Linear Terms for 5-DOF System

For the non-linear terms in the 5-DOF non-linear system, we consider cubic stiffness non-linearity (k_3x^3) and quadratic damping linearity (c_2x^2). These terms are incorporated into the state-space equation as follows:

$$A_{k3} \text{ is generated from } k3_{\text{vec}}$$

$$A_{c2} \text{ is generated from } c2_{\text{vec}}$$

The non-linearity term C is given by:

$$C = \begin{bmatrix} 0 \\ \eta \end{bmatrix}$$

where η represents the function capturing the system's non-linear behavior.

6.3 State-Space Equation for Non-Linear System

The state-space equation for the non-linear spring-mass system is given by:

$$\dot{y}(t) = A \cdot y(t) + B \cdot u(t) + C$$

where:

- $\dot{y}(t)$ represents the derivative of the state vector $x(t)$ with respect to time.
- A is the state matrix characterizing the inherent dynamics.
- B is the input matrix relating the input excitation $u(t)$ to the system dynamics.
- C is a term incorporating non-linear effects.

6.3.1 Physical Model

A physical model $\mathcal{P}$ refers to a mathematical representation that describes the behavior of physical systems or phenomena using fundamental laws and principles of physics. These models are typically expressed as partial differential equations (PDEs) or other mathematical equations that capture the relationships between various physical quantities, such as forces, velocities, temperatures, or concentrations, as they evolve and space.

These models are designed and formulated in a way that allows them to be differentiable, meaning that gradients can be computed concerning their inputs. This property is crucial for seamlessly combining physical simulations with neural networks, as it enables the flow of gradient information during training and optimization processes.

6.3.2 Partial Differential Equations

Partial Differential Equations (PDEs) provide the most fundamental description of evolving systems, spanning from quantum mechanics to turbulent flows. PDEs are mathematical equations that establish relationships among the partial derivatives of an unknown function. In the context of a physical system denoted as $u(x,t)$, the governing PDE can be expressed as:

$$\frac{\partial u}{\partial t} = P(u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, \ldots, F(t)) \tag{6.1}$$

Here, the function P models the physical behavior of the system, and $F(t)$ represents an optional external force factor.

6.3.3 Numerical Approximation of Partial Differential Equations

Analytic solutions, which are closed-form expressions, are only applicable to a small subset of Partial Differential Equations (PDEs). To address this limitation, numerical methods are employed to discretize PDEs, transforming continuous equations into systems with a finite number of unknowns. This discretization involves introducing a time step size (Δt) for the temporal dimension. For spatial dimensions, two common approaches are used: the Eulerian perspective, which assigns values to grid cells, and the Lagrangian perspective, which tracks particles.

In a discrete setting, numerical methods provide approximations of a smooth function denoted as u, inevitably introducing errors into the computations. These errors can be quantified by measuring how much they deviate from the exact analytical solution. Specifically, in the context of discrete simulations involving Partial Differential Equations (PDEs), these errors are often expressed as functions related to truncation, commonly represented as $O(\Delta t^k)$, where higher-order methods with larger values of k are generally favored but tend to pose practical implementation challenges.

For practical solution schemes, there are no readily available closed-form expressions to precisely describe these truncation errors. Furthermore, it's crucial to note that these errors tend to exhibit exponential growth as solutions are integrated over time.

6.4 Differentiable Physics

6.4.1 Differentiable Physics (DP): A Literature Review

Introduction

Differentiable Physics (DP) represents a significant advancement in the fusion of deep learning methods with physical simulations. This review explores the key concepts and methodologies within DP, highlighting its relevance in various applications. DP extends beyond traditional physics-informed loss functions and empowers neural networks (NNs) to seamlessly interact with physical solvers, enabling improved learning feedback and generalization. This review covers the fundamental principles of DP, its applications, and the underlying mathematical foundations.

Fundamental Principles of DP

DP begins with a continuous representation of a physical model, denoted as $\mathcal{P}^*(\mathbf{x}, \nu)$, describing a physical quantity of interest represented as $\mathbf{u}(\mathbf{x}, t)$. Here, $\mathbf{u}$ maps from the space $R^d \times R^+$ to R^d, with model parameters ν. The central goal is to understand the temporal evolution of this system. To achieve this, the model is discretized to obtain a formulation, $\mathcal{P}(\mathbf{x}, \nu)$, which rearranges the calculation of the future state of the system after a time step of Δt. This is achieved through a sequence of operations denoted as $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$, applied successively to the initial state $\mathbf{u}(t)$, considering model parameters ν. The core idea is expressed as:

$$\mathbf{u}(t + \Delta t) = \mathcal{P}_m \circ \cdots \circ \mathcal{P}_2 \circ \mathcal{P}_1(\mathbf{u}(t), \nu)$$

The composition of functions is represented by $\circ$, akin to $f(g(x))$. To simplify, it is assumed that $\mathbf{u}$ is solely a function of space, remaining constant over time. This reformulated process is denoted as $\mathcal{P}$, which maps a state at time t to a new state at an evolved time $t + \Delta t$:

$$d(t + \Delta t) = \mathcal{P}(d(t), \mathbf{u}, t + \Delta t)$$

Applications of DP

One of the primary applications of DP lies in finding velocity fields $\mathbf{u}$. These fields transform an initial scalar density state d^0 at time t^0 into a state that evolves through $\mathcal{P}$ to a later time t^e, achieving a specific shape or configuration d^{target}. This objective is expressed as an L^2 loss between the two states:

$$L = |d(t^e) - d^{\text{target}}|^2$$

The minimization problem seeks to find the optimal $\mathbf{u}$ that minimizes this loss:

$$\arg\ \min_{\mathbf{u}} |\mathcal{P}(d^0, \mathbf{u}, t^e) - d^{\text{target}}|^2$$

This optimization task can be tackled using gradient descent (GD), where the gradient is determined through the differentiable physics approach.

Mathematical Foundations of DP

DP relies on the concept of differentiable operators. These operators are built upon existing numerical solvers and involve a continuous formulation of physical effects. The fundamental requirement for DP integration is that each operator provides gradients with respect to its inputs. Jacobian matrices play a crucial role in computing derivatives for vector-valued functions.

Implicit gradient calculations are employed for more complex problems, such as Poisson's equation. In such cases, iterative solvers approximate the matrix-vector products necessary for backpropagation. By carefully defining gradient operations, it is possible to efficiently compute gradients without explicitly storing intermediate states, avoiding memory overhead and computational inefficiency.

6.5 Integrating Neural Networks (NN) with Differentiable Physics (DP)

6.5.1 Introduction

The integration of neural network (NN) layers with differentiable physics (DP) operators represents a powerful approach to solving complex problems in the realm of physics-based simulations and modeling. This section delves into various strategies for seamlessly combining NNs and DP operators, with a special emphasis on the "unrolling" technique. Through unrolling, NNs gain the capability to influence the state of dynamic systems, paving the way for innovative solutions to intricate physical challenges.

0.3

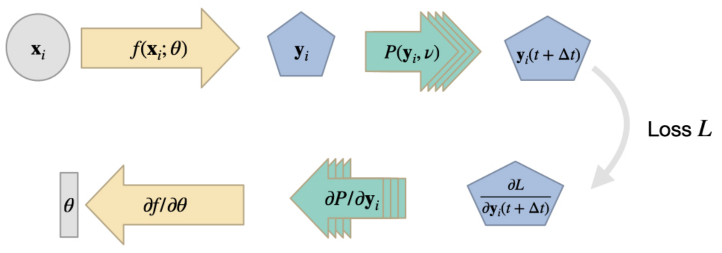

$\mathbf{x}_i$
$f(\mathbf{x}_i;\theta)$
$\mathbf{y}_i$
$P(\mathbf{y}_i,\nu)$
$\mathbf{y}_i(t+\Delta t)$
Loss L
$\frac{\partial L}{\partial \mathbf{y}_i(t+\Delta t)}$
$\partial P/\partial \mathbf{y}_i$
$\partial f/\partial \theta$
θ

Figure 6.1: A network produces an input to a PDE solver, which provides a gradient for training during the backpropagation step.

0.3

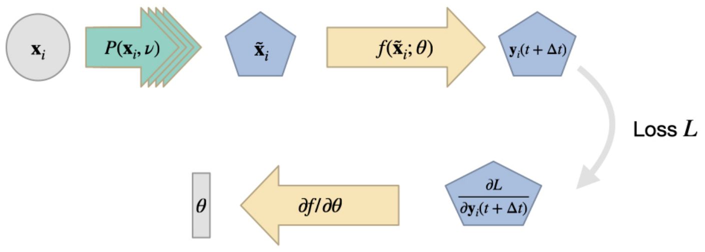

$\mathbf{x}_i$
$P(\mathbf{x}_i,\nu)$
$\tilde{\mathbf{x}}_i$
$f(\tilde{\mathbf{x}}_i;\theta)$
$\mathbf{y}_i(t+\Delta t)$
Loss L
$\frac{\partial L}{\partial \mathbf{y}_i(t+\Delta t)}$
$\partial f/\partial \theta$
θ

Figure 6.2: A PDE solver produces an output which is processed by an NN

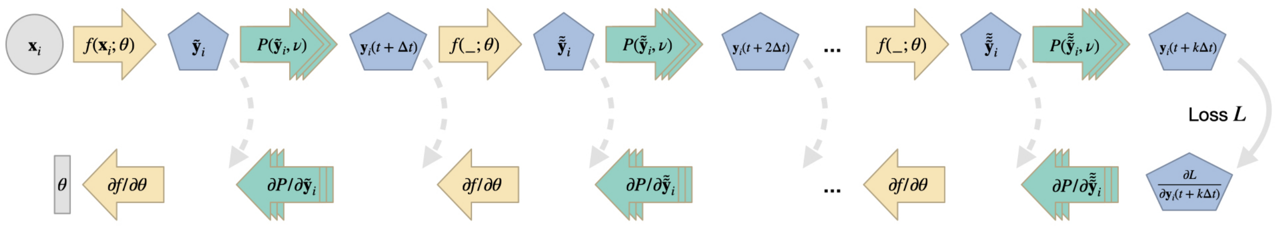
6.5.2 Unrolling for Enhanced Integration

Solver in the Loop

In the visual representation Figure 2.3, we observe the NN, denoted as f, generating an entire state for a physical system, which is then input into the DP operator $\mathcal{P}$. To elucidate this process further, let's consider a state at time $t + j\Delta t$, where the NN produces an intermediate state, $\tilde{x}(t + j\Delta t) = f(x(t + j\Delta t); \theta)$. Subsequently, the solver utilizes this intermediate state to compute the next state:

$$x(t + (j + 1)\Delta t) = \mathcal{P}(\tilde{x}(t + j\Delta t))$$

While this approach is valid and applicable in various scenarios, it may not always be the optimal choice. Particularly, when the primary role of the NN is to provide a correction to



$\mathbf{x}_i$
$f(\mathbf{x}_i;\theta)$
$\tilde{\mathbf{y}}_i$
$P(\tilde{\mathbf{y}}_i,\nu)$
$\mathbf{y}_i(t+\Delta t)$
$f(\_;\theta)$
$\tilde{\tilde{\mathbf{y}}}_i$
$P(\tilde{\tilde{\mathbf{y}}}_i,\nu)$
$\mathbf{y}_i(t+2\Delta t)$
...
$f(\_;\theta)$
$\tilde{\tilde{\tilde{\mathbf{y}}}}_i$
$P(\tilde{\tilde{\tilde{\mathbf{y}}}}_i,\nu)$
$\mathbf{y}_i(t+k\Delta t)$
Loss L
θ
$\partial f/\partial\theta$
$\partial P/\partial\tilde{\mathbf{y}}_i$
$\partial f/\partial\theta$
$\partial P/\partial\tilde{\tilde{\mathbf{y}}}_i$
...
$\partial f/\partial\theta$
$\partial P/\partial\tilde{\tilde{\tilde{\mathbf{y}}}}_i$
$\frac{\partial L}{\partial \mathbf{y}_i(t+k\Delta t)}$

Figure 6.3: Time stepping with interleaved DP and NN operations for k solver iterations. The dashed gray arrows indicate optional intermediate evaluations of loss terms (similar to the solid gray arrow for the last k step), and intermediate outputs of the NN are indicated with a tilde.

the current state, there arises an opportunity to optimize the allocation of NN resources. This optimization involves reusing portions of the existing state that are already correct. To facilitate this, we introduce an operator denoted as $\circ$, which combines both x and $\tilde{x}$:

$$x(t + (j+1)\Delta t) = \mathcal{P}(x(t + j\Delta t) \circ \tilde{x}(t + j\Delta t))$$

In this context, $\circ$ can represent an addition, although other operations such as multiplication or integration schemes are also viable options. In the case of addition, the update to the next state is calculated as $\mathcal{P}(x + \tilde{x})$. Consequently, the NN's task is streamlined, focusing solely on modifying components of x that do not already align with the desired learning objective.

Formalizing Integration

To provide a more rigorous and formal description of the integration process, we delve into the mathematical representation of how the NN influences the system. Specifically, we employ Jacobians to express the update step for x. Let's break down the components of this representation:

- Mini-batches are denoted by an index i.
- The loss function is represented as L.
- The total number of unrolled steps is defined as k.
- The state x of batch i at time step j is expressed as $x_i^{(j)}$.
- The gradient of the network weights is computed as follows:

$$\frac{\partial L}{\partial \theta} = \sum_i \sum_{j=0}^{k-1} \frac{\partial L}{\partial x_i^{(j+1)}} \cdot \left(\prod_{l=0}^{j} \frac{\partial x_i^{(l+1)}}{\partial x_i^{(l)}} \right) \cdot \frac{\partial x_i^{(0)}}{\partial \theta}$$

At first glance, this expression may appear complex, but it adheres to a structured pattern:

- The first summation accumulates contributions from all entries within a mini-batch.
- The outer summation extends over all time steps from 0 to $k-1$.
- Within each time step, we trace dependencies backward from the final state $x_i^{(j+1)}$ to the initial state $x_i^{(0)}$.
- At each step in this chain, we calculate a Jacobian with respect to time, which inherently depends on the corrections introduced by the NN, represented by $\tilde{x}(t + j\Delta t)$ (not explicitly detailed here).

The final step focuses on determining changes in terms of the network output and its weights at the j-th time step. Aggregating these contributions from different time steps yields a final update that plays a pivotal role in the optimizer during the training process.

6.5.3 Benefits of DP in Training

Training with a differentiable physics simulator, as described here, offers a plethora of advantages, especially when compared to alternative approaches. To illustrate the significance of this integration, let's consider a comparison with an "unrolled supervised" training approach, particularly in complex cases like turbulent mixing. Incorporating differentiability significantly enhances stability and accuracy, as evidenced by turbulence metrics and visualizations.

In conclusion, the integration of NNs with DP operators unlocks a world of possibilities for addressing challenging problems in physics and simulation. By leveraging techniques such as unrolling and thoughtful composition of NN and solver components, we can optimize the learning process and achieve superior results in complex simulations and physical modeling scenarios. The synergy between NNs and DP operators not only enhances the efficiency of training but also expands the horizons of what can be achieved in the realm of differentiable physics.