Towards Strict Conservation of Quadratic Conservation Laws in Operator Learning

Moritz Elias Hauschulz

Words: 7498

University of Oxford

A thesis submitted for the degree of

MSc in Mathematical Sciences

Trinity 2025

Abstract

Machine learning methods have become increasingly capable of modeling dynamical systems governed by ordinary and partial differential equations. However, ensuring that model outputs exhibit realistic physical behaviour beyond the training set remains a challenge. This paper is dedicated to developing architectures to strictly enforce quadratic conservation laws which are known to be conserved by the underlying system on which the models are trained. To motivate the approach, we propose two schemes for enforcing energy conservation in Deep Operator Networks (DeepONets) and show their effectiveness on the example of the harmonic oscillator. We then transfer these approaches to the spectral domain, where besides general theory we consider the special case of the wave equation. This leads us to propose a new class of models which we term DeepONet Spectral Operators (DSOs). We show that approximate energy conservation can be achieved with little computational overhead, but also provide an argument for why exact energy preservation through the proposed methods suffers from errors due to non-linear feedback. We further present a fully spectral alternative, which is related to the Fourier Neural Operator (FNO). We show empirically that this Augmented Fourier Neural Operator (AFNO) can conserve energy exactly through appropriate normalisation, besides outperforming the DSO by an order of magnitude.

Acknowledgements

I am deeply grateful for the support of Georg Maierhofer and Nicolas Boullé for proposing and leading this project in an accessible, structured and open-minded way. They proposed the initial QR post-processing approach for DeepONets learning the harmonic oscillator, and suggested probing similar techniques in the spectral domain. I thank Kathryn Gillow for attending my presentation and providing useful feedback. Lastly, I would like to acknowledge the many inspiring discussions with the other students on the project.

Contents

List of Figures

Notation

Selected Acronyms

DeepONet	Deep Operator Network
DSO	DeepONet Spectral Operator
FNO	Fourier Neural Operator
AFNO	Augmented Fourier Neural Operator
AD	auto differentiation
ODE	ordinary differential equation
PDE	partial differential equation
(I)FT	(inverse) Fourier transform
(I)DFT	(inverse) discrete Fourier transform
(I)FDFT	(inverse) fast discrete Fourier transform
LHS	left hand side (of an equation)
RHS	right hand side (of an equation)
w.r.t.	with respect to
s.t.	such that

Selected Symbols

$\boldsymbol{x}$	vector
$\boldsymbol{x}_i$	vector, indexed by i
x_i	i-th component of $\boldsymbol{x}$
$\boldsymbol{A}$	matrix
$\boldsymbol{A}_{i,\cdot}$	i-th row of A
$\boldsymbol{A}_{\cdot,j}$	j-th column of A
$\boldsymbol{A}^\dagger$	Hermitian conjugate of $\boldsymbol{A}$
$\mathbb{R}$	real numbers
$\mathbb{C}$	complex numbers
$\mathbb{Z}$	integers
$\overline{x}$	complex conjugate
$C(\Omega)$	real-valued continuous functions on Ω
$C^\infty(\Omega)$	real-valued smooth functions on Ω
$C^\infty(\Omega, \mathbb{C})$	complex-valued smooth functions on Ω
$L^2(\Omega)$	real-valued square integrable functions on Ω
$L^2(\Omega, \mathbb{C})$	complex-valued square integrable functions on Ω
$u^{(\alpha)}(t)$	α-th derivative u
$\frac{\mathrm{d}x}{\mathrm{d}t}$	derivative of x w.r.t. t
$\partial_i^\alpha u(\boldsymbol{x})$	α-th partial derivative w.r.t. i-th component of $\boldsymbol{x}$
∂^α	derivative with multiindex $\alpha = (\alpha_1, \ldots, \alpha_n)$, i.e. $\partial_1^{\alpha_1}\partial_2^{\alpha_2}\cdots\partial_n^{\alpha_n}$
$\circ$	element-wise function evaluation or multiplication
$[n]$	set $\{0, \ldots, n\}$
$\hat{f} = \mathcal{F}(f)$	FT of f
$f = \mathcal{F}^{-1}(\hat{f})$	IFT of $\hat{f}$
$\hat{\boldsymbol{x}} = \mathcal{F}(\boldsymbol{x})$	DFT of $\boldsymbol{x}$
$\boldsymbol{x} = \mathcal{F}^{-1}(\hat{\boldsymbol{x}})$	IDFT of $\hat{\boldsymbol{x}}$
f_θ	function f parameterised by θ
Ψ_θ	operator Ψ parameterised by θ
$k \mod n$	k modulo n

Chapter 1

Introduction

As machine learning disrupts the scientific domain, applications to modeling the behaviour of dynamical systems have attracted increasing interest in fields from engineering to the Earth sciences. This has led to the emergence of operator learning as a method to model mappings between infinite dimensional spaces of functions. In this paper, we are interested in the solution operators of ordinary and partial differential equations, which typically map a function on the boundary of a domain to the solution function on its interior. Various approaches have been proposed to this end, among which the most popular are the Deep Operator Network (DeepONet) [1] and the Fourier Neural Operator (FNO) [2]. Despite these approaches and their adaptations having shown strong results in theory and on practical applications (cf. [3][4]), they still lag behind traditional numerical methods when it comes to the preservation structure. While structure preservation can be interpreted in many ways, we choose to focus on the conservation of integral quantities, such as energy, that are to be conserved by the underlying dynamics. Our main contributions are as follows:

1. We motivate two novel approaches for energy conservation in DeepONets on the example of harmonic oscillators.

2. We propose the DeepONet Spectral Operator (DSO), which extends the methods to a broad class of dynamics and conservation laws by using the DeepONet to model time-dependent Fourier coefficients.

3. We hypothesise that spectral approaches in space *and* time are most suitable for exact energy conservation due to the ability to compute gradients without differentiating the architecture. We show that this can be achieved by a framework inspired by the FNO which we term the Augmented FNO (AFNO).

4. We show empirically, on the wave equation, that the conservation of integral quantities in practice leads to better test set performance and lower generalisation errors besides predictions that are closer to the truth in a distributional sense.

Chapter 2

Background

The background section is split into two parts. First we provide a primer on Fourier analysis, which will be referenced throughout the paper. The second part contains an introduction to operator learning, focusing on the DeepONet and FNO architectures.

2.1 Fourier Analysis

2.1.1 Continuous

In this discussion, we focus on functions on $f : \mathbb{R}^n \to \mathbb{C}$, but all results generalise to vector valued functions. The results are established, but a useful reference is [5].

Definition 2.1.1. *For a function $f : \mathbb{R}^n \to \mathbb{C}$, the **Fourier transform** is defined as*

$$\hat{f}(\boldsymbol{\xi}) = \int_{\mathbb{R}^n} f(\boldsymbol{x})\, e^{-i2\pi \boldsymbol{x}\cdot\boldsymbol{\xi}}\, d\boldsymbol{x}$$

Its inverse is given by

$$f(\boldsymbol{x}) = \int_{\mathbb{R}^n} \hat{f}(\boldsymbol{\xi})\, e^{i2\pi \boldsymbol{x}\cdot\boldsymbol{\xi}}\, d\boldsymbol{\xi}$$

The operators mapping $f \mapsto \hat{f}$ and $\hat{f} \mapsto f$ are denoted by $\mathcal{F}$ and $\mathcal{F}^{-1}$ respectively.

Recall further that a periodic function on $\Omega = [0, L_1] \times ... \times [0, L_n] \subset \mathbb{R}^n$ can be expanded in a Fourier series:

Definition 2.1.2. *Any **periodic** function $f : \Omega \to \mathbb{C}$ can be expressed as a **Fourier series***

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \hat{f}[\boldsymbol{k}] e^{i2\pi \sum_{j=1}^n \frac{k_j}{L_j} x_j} \tag{2.1}$$

where

$$\hat{f}[\boldsymbol{k}] = \frac{1}{\prod_{j=1}^n L_j} \int_\Omega f(\boldsymbol{x}) e^{-i2\pi \sum_{j=1}^n \frac{k_j}{L_j} x_j}\, d\boldsymbol{x} \quad \text{for } \boldsymbol{k} \text{ in } \mathbb{Z}^n \tag{2.2}$$

are known as Fourier coefficients, or Fourier modes.

It can be shown that for such periodic functions the Fourier transform simplifies to a sum of products of Dirac delta functions and Fourier coefficients as follows:

$$\hat{f}(\boldsymbol{\xi}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \hat{f}[\boldsymbol{k}] \prod_{j=1}^{n} \delta(\xi_j - \frac{k_j}{L_j}) \tag{2.3}$$

Before moving to the discrete time setting, we finally state two important theorems. The first one, Parceval's theorem, yields an expression for inner products in $L^2(\Omega, \mathbb{C})$ in terms of Fourier coefficients. The second one, known as the convolution theorem, demonstrates how convolution in physical space corresponds to multiplication in the spectral domain.

Theorem 2.1.1 (Parceval's Theorem)**.** *Let $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ be two complex-valued, square-integrable, periodic functions on Ω. Write their Fourier series as*

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \hat{f}[\boldsymbol{k}] e^{i2\pi \sum_{j=1}^{n} \frac{k_j}{L_j} x_j}$$

and

$$g(\boldsymbol{x}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \hat{g}[\boldsymbol{k}] e^{i2\pi \sum_{j=1}^{n} \frac{k_j}{L_j} x_j}$$

Then

$$\sum_{\boldsymbol{k}\in\mathbb{Z}^n} \hat{f}[\boldsymbol{k}]\overline{\hat{g}[\boldsymbol{k}]} = \frac{1}{\prod_{j=1}^{n} L_j} \int_{\Omega} f(\boldsymbol{x})\overline{g(\boldsymbol{x})} d\boldsymbol{x}$$

Theorem 2.1.2 (Convolution Theorem)**.** *Given two functions $f, g \in L^1(\mathbb{R}^n)$ with Fourier transforms $\mathcal{F}(f) = \hat{f}$ and $\mathcal{F}(g) = \hat{g}$, their convolution, denoted by $f * g$, satisfies $(f * g)(\boldsymbol{x}) = \mathcal{F}^{-1}(\hat{f} \cdot \hat{g})(\boldsymbol{x})$.*

2.1.2 Discrete

Suppose now that we have infinitely many samples of a function f at points in $\mathbb{R}^n$ spaced with distance T_j apart from each other along the j-th component, denoted $\{f(\boldsymbol{m} \circ \boldsymbol{T})\}_{\boldsymbol{m}\in\mathbb{Z}^n}$ with $\boldsymbol{m}$ indexing the samples. Here, $\boldsymbol{T}$ refers to the vector of T_js. Then we may define a discrete analogue to Definition 2.1.1 by approximating the integral with a summation. We adapt our presentation from [6]. Note that the first definition is not exactly equivalent to the discrete Fourier transform (DFT), which will be explained later.

Definition 2.1.3. *Given an infinite number of samples of a function s, the **discretised Fourier transform** is defined as*

$$\hat{f}_{\boldsymbol{T}}(\boldsymbol{\xi}) = \sum_{\boldsymbol{m}\in\mathbb{Z}^n} \underbrace{\prod_{j=1}^{n} T_j f(\boldsymbol{m} \circ \boldsymbol{T})}_{f[\boldsymbol{m}]} e^{-i2\pi \sum_{j=1}^{n} T_j m_j \cdot \xi_j}$$

Noting that this is a Fourier series over frequencies it follows that this is periodic with period $\frac{1}{T_i}$ in the j-th dimension, and we may invert the DFT by (2.2):

$$f[\boldsymbol{m}] = \prod_{j=1}^{N} T_j \int_{[0,1/T_1]\times\cdots\times[0,1/T_n]^n} \hat{f}_{\boldsymbol{T}}(\boldsymbol{\xi}) e^{i2\pi \sum_{j=1}^{n} T_j m_j \xi_j} d\boldsymbol{\xi}$$

Now we assume periodicity of the data after N_j samples in dimension j, and define the index set $\mathcal{I} = \prod_{j=1}^{n} \{0, \ldots, N_j - 1\}$. Then, as in (2.3), our DFT can be shown to simplify to a weighted Dirac comb expressible as

$$\begin{aligned}
\hat{f}_{\boldsymbol{T}}(\boldsymbol{\xi}) &= \frac{1}{\prod_{i=1}^{n} N_i T_i} \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \sum_{\boldsymbol{m}\in\mathcal{I}} f[\boldsymbol{m}] e^{-i2\pi \sum_{j=1}^{n} T_j m_j \cdot \xi_j} \prod_{j=1}^{n} \delta(\xi_j - \frac{k_j}{N_j T_j}) \\
&= \frac{1}{\prod_{i=1}^{n} N_i T_i} \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \hat{f}[\boldsymbol{k}] \prod_{j=1}^{n} \delta(\xi_j - \frac{k_j}{N_j T_j})
\end{aligned}$$

where we defined a new quantity, which is commonly known as the DFT, given by

$$\hat{f}[\boldsymbol{k}] = \sum_{\boldsymbol{m}\in\mathcal{I}} f[\boldsymbol{m}] e^{-i2\pi \sum_{j=1}^{n} \frac{k_j}{N_j} m_j} \quad \text{for} \quad \boldsymbol{k} \in \mathbb{Z}^n.$$

The inverse also simplifies to a sum, known as the inverse DFT (IDFT)

$$f[\boldsymbol{m}] = \frac{1}{\prod_{j=1}^{n} N_j} \sum_{\boldsymbol{k}\in\mathcal{I}} \hat{f}[\boldsymbol{k}] \cdot e^{i2\pi \sum_{j=1}^{N} \frac{k_j}{N_j} \cdot m_j} \quad \text{for} \quad \boldsymbol{m} \in \mathbb{Z}^n.$$

We have thus obtained a tractable way of transforming between function evaluations on the grid $\mathcal{I}$ and Fourier modes whose inverse transform evaluates to the observed data on the grid points, under the assumption that the measured function f is periodic. A useful property is that $f[\boldsymbol{m}]$ are real valued if and only if $\hat{f}[\boldsymbol{k}] = \overline{\hat{f}[-\boldsymbol{k} \mod \boldsymbol{N}]}$, where $\boldsymbol{N}$ is the stacked vector of N_js.

2.1.2.1 Fast Fourier Transforms

The DFT is not only tractable, but can be efficiently computed for a data tensor $\boldsymbol{f} = (f(\boldsymbol{m} \circ \boldsymbol{T}))$ with a parallelised algorithm, yielding a tensor of Fourier coefficients $\hat{\boldsymbol{f}}$ of the same dimensionality. Computing the discrete Fourier transform of $\boldsymbol{f}$ naïvely has complexity $\mathcal{O}(n^2)$. Several algorithms exist that speed up the computation to $\mathcal{O}(n \log n)$. This class of algorithms is known as fast discrete Fourier transforms FDFT. The most widely used such algorithm is the Cooley-Tukey algorithm [7].[1]

[1] which, coincidentally, turns 60 this month

2.2 Operator Learning

Contrary to classical machine learning, which amounts to the approximation of functions between finite dimensional spaces, operator learning aims to learn mappings between infinite dimensional spaces *of* functions. That is, we aim to learn an operator $\Psi : \mathcal{V} \to \mathcal{U}$ where $\mathcal{V}$ and $\mathcal{U}$ are function spaces on some domains $\Omega_{\mathcal{V}}$ and $\Omega_{\mathcal{U}}$. For an operator learned with a neural network architecture with parameters θ, we typically write Ψ_θ. Deploying a neural network architecture typically restricts the space in which solutions can be represented, so that $\Psi_\theta : \mathcal{V} \to \tilde{U} \subset \mathcal{U}$. For instance, on a compact domain $\Omega \subset \mathbb{R}^n$ the output from a neural network with smooth activation functions is necessarily a subset of $C^\infty(\Omega)$. We may restrict ourselves further, for example by assuming periodicity of the functions in $\tilde{\mathcal{U}}$. Another typical restriction that we will encounter is that $\tilde{\mathcal{U}}$ is finite dimensional, so that $u \in \tilde{\mathcal{U}}$ can be written as a linear combination of finitely many basis functions. While in theory such restrictions need not be placed on $\mathcal{V}$, for processing by a neural network, any $v \in \mathcal{V}$ will be approximated by some $\tilde{v} \in \tilde{\mathcal{V}}$ where $\tilde{\mathcal{V}}$ is isomorphic to some finite dimensional subspace of $\mathcal{V}$. Given these explicitly or implicitly restricted function spaces $\tilde{V}$ and $\tilde{U}$, the operator Ψ_θ is then *learned* by optimising its parameters θ iteratively by minimising a loss function with some variant of gradient descent. This loss function can be based on data samples or on a partial differential equation (PDE) that the target operator Ψ satisfies. We will focus on learning from data, and only rely on physics knowledge in computing the conserved quantity of interest, which will be discussed more in Chapter 3 and Chapter 4. A thorough survey of operator learning can be found in [8].

2.2.1 Learning from Data

Suppose we have a set of M input-output pairs $\{(v, u)_i\}_{i=1}^M$, s.t. $v \in \mathcal{V}$, $u \in \mathcal{U}$ and $\Psi(v) = u$. As discussed, v and u have to be encoded in a finite-dimensional manner, e.g. by evaluating v on a grid or mesh of $N_{\mathcal{V}}$ points $\{\boldsymbol{x}_j\}_{j=1}^{N_{\mathcal{V}}}$ in $\Omega_{\mathcal{V}}$, and by evaluating u on $N_{\mathcal{U}}$ points $\{\boldsymbol{y}_j\}_{j=1}^{N_{\mathcal{U}}}$ in $\Omega_{\mathcal{U}}$. The loss used throughout this paper can be specified as follows:

$$L_{\text{data}}(\theta) = \frac{1}{M} \sum_{i=1}^{M} \frac{1}{N_{\mathcal{U}}} \sum_{j=1}^{N_{\mathcal{U}}} |\Psi_\theta(v_i)(\boldsymbol{y}_j) - u_i(\boldsymbol{y}_j)|^2$$

2.3 Deep Operator Networks

The DeepONet, proposed in [1] exploits the universal *operator* approximation theorem due to [9], which can be seen as an analogue to the well-known universal *function* approximation theorem due to [10].

Theorem 2.3.1. *(adapted from [9]) Suppose $\sigma \in C(\mathbb{R})$ is non-polynomial, X is a Banach space and $\Omega_{\mathcal{V}} \subset X$, $\Omega_{\mathcal{U}} \subset \mathbb{R}^d$ are compact sets in X and $\mathbb{R}^d$, respectively. Suppose V is a compact set in $C(\Omega_{\mathcal{V}})$, $\Psi : V \to C(\Omega_{\mathcal{U}})$ is a linear continuous operator. Then for any $\epsilon > 0$ and $v \in V$, there are positive integers n, p, N, points $\{x_j\}_{j=1}^N \subset \Omega_{\mathcal{V}}$, vectors $\boldsymbol{c}_k, \boldsymbol{\theta}_k \in \mathbb{R}^n$ and a matrix $\boldsymbol{W}_{B,k} \in \mathbb{R}^{n \times N}$ for each $k = 1, \ldots, p$ as well as a matrix $\boldsymbol{W}_\alpha \in \mathbb{R}^{p \times d}$ and a vector $\boldsymbol{\zeta} \in \mathbb{R}^p$, s.t.*

$$\sup_{\boldsymbol{y} \in \Omega_{\mathcal{U}}} |\Psi(v)(\boldsymbol{y}) - \Psi_\theta(\boldsymbol{v})(\boldsymbol{y})| < \epsilon$$

where $\boldsymbol{v} \in \mathbb{R}^N$ is a vector of N function evaluations of v at $\{x_j\}$ and

$$\Psi_\theta(\boldsymbol{v})(\boldsymbol{y}) = \overbrace{\underbrace{\boldsymbol{B}_\theta(\boldsymbol{v})}_{1\times p}}^{branch} \overbrace{\underbrace{\boldsymbol{\alpha}_\theta(\boldsymbol{y})}_{p\times 1}}^{trunk}$$

with the k-th component of $\boldsymbol{B}_\theta(\boldsymbol{v})$ being $\boldsymbol{c}_k \cdot \sigma \circ (\boldsymbol{W}_{B,k}\boldsymbol{v} + \boldsymbol{\theta}_k)$ and $\boldsymbol{\alpha}_\theta(\boldsymbol{y}) = \sigma \circ (\boldsymbol{W}_\alpha \boldsymbol{y} + \boldsymbol{\zeta})$.

What the authors note is that this approximation relies essentially on $p + 1$ neural networks to approximate the output function arbitrarily close. Out of these, p are branch networks which can be interpreted as coefficients to p basis functions produced by one trunk network. This realisation inspires the design of DeepONets, where typically the p branch networks $\mathbb{R}^{N_{\mathcal{V}}} \to \mathbb{R}$ are 'stacked' into one network $\mathbb{R}^{N_{\mathcal{V}}} \to \mathbb{R}^p$. In addition, as the expressivity of neural networks scales exponentially with depth [11][12], the authors make all networks deep. If we want the output function to be vector valued, say with a codomain of dimension d_{out} we can simply increase the number of output channels from the branch network to d_{out}. Thus a DeepONet, in the sense considered for the remainder of this paper, consists of a branch network, $\boldsymbol{B}_\theta : \mathbb{R}^N \to \mathbb{R}^{d_{out} \times p}$ and a trunk network $\boldsymbol{\alpha}_\theta : \mathbb{R}^d \to \mathbb{R}^p$, where $N_{\mathcal{V}}$ is the number of 'sensor' points that are discretising the domain $\Omega_{\mathcal{V}}$ of the input. Note that we may represent the DeepONet output as a matrix-vector product

$$\Psi_\theta(\boldsymbol{v})(\boldsymbol{y}) = \underbrace{\boldsymbol{B}_\theta(\boldsymbol{v})}_{d_{out}\times p} \underbrace{\boldsymbol{\alpha}_\theta(\boldsymbol{y})}_{p\times 1}. \tag{2.4}$$

2.4 Fourier Neural Operators

FNOs essentially learn the mapping Ψ_θ in the spectral domain [2]. Thus, FNOs are intricately related to classical spectral methods for numerically solving PDEs (cf. [13]). The theory of FNOs is most neatly explained under the assumption that the functions in $\mathcal{V}$ and $\mathcal{U}$ share a domain, i.e. $\Omega_{\mathcal{V}} = \Omega_{\mathcal{U}} = \Omega \subset \mathbb{R}^n$, though in practice this can be relaxed. We will denote by d_{in} and d_{out} the dimensionality of the codomains of functions in $\mathcal{V}$ and $\mathcal{U}$ respectively. The FNO architecture is comprised of iterative updates to

a high-dimensional function representation $v_0 \mapsto v_1 \mapsto \ldots \mapsto v_T$. The first and final representation are related to the input function v and the output function u through lifting and projecting transformations, $P : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_v}$ and $Q : \mathbb{R}^{d_v} \to \mathbb{R}^{d_{out}}$, so that $v_0(\boldsymbol{x}) = P(v(\boldsymbol{x}))$ and $u(\boldsymbol{x}) = Q(v_T(\boldsymbol{x}))$. The iterative updates compose the sum of non-local integral operator $\mathcal{K}$ and a linear transformation W with a non-linear activation function σ. This can be summarised by the following update rule:

$$v_{t+1}(\boldsymbol{x}) := \sigma\Big(\boldsymbol{W}v_t(\boldsymbol{x}) + \big(\mathcal{K}_\theta v_t\big)(\boldsymbol{x})\Big), \qquad \forall \boldsymbol{x} \in \Omega$$

where $\boldsymbol{W} \in \mathbb{R}^{d_v \times d_v}$ and $\mathcal{K}_\theta$ is an integral kernel operator, which is typically chosen to be the convolution operator acting on v_t as follows:

$$\big(\mathcal{K}_\theta v_t\big)(\boldsymbol{x}) := \int_D \kappa\big(\boldsymbol{x} - \boldsymbol{y}\big)v_t(\boldsymbol{y})\mathrm{d}\boldsymbol{y}, \qquad \forall \boldsymbol{x} \in \Omega \tag{2.5}$$

To appreciate this choice of kernel operator, compare the role of κ to that of a fundamental solution[2]. Further to it being the natural choice for the kernel, Theorem 2.1.2 enables us to directly learn κ in its Fourier representation. With this in mind, the Fourier integral operator can be defined as follows.

Definition 2.4.1. *(adapted from [2]) For a given periodic function $\kappa : \bar{\Omega} \to \mathbb{R}^{d_v \times d_v}$ parameterised by θ with Fourier transform $\hat{\kappa}_\theta$ define the* ***Fourier integral operator*** *as*

$$\big(\mathcal{K}_\theta v_t\big)(\boldsymbol{x}) = \mathcal{F}^{-1}\Big(\hat{\kappa}_\theta \cdot (\mathcal{F}v_t)\Big)(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \Omega$$

If, in addition to periodicity, we assume that κ has at most d^n Fourier modes, then we may represent $\hat{\kappa}_\theta$ as d^n matrices with dimensions $d_v \times d_v$. This is useful in the case where we discretise v_t on a mesh $\mathcal{I}$ as introduced in Section 2.1, so that we can apply a discrete Fourier transform and its inverse to move between the physical and the spectral domain. In particular, if we assume access to N evenly spaced point evaluations of $v \in \mathcal{V}$ stacked as $\boldsymbol{v} \in \mathbb{R}^N \times \mathbb{R}^{d\mathcal{V}}$, our hidden representations become $\boldsymbol{v}_t \in \mathbb{R}^{N \times d_v}$ with non-zero Fourier coefficients $\hat{\boldsymbol{v}}_t \in \mathbb{C}^{N \times d_v}$. Truncating again to d^n modes we may then implement the convolution in (2.5) efficiently as a (tensorised) matrix multiplication on d^n channels.

2.5 Structure Preservation

We shall introduce minimal background on the quadratic conservation laws that are relevant for our purposes in the respective methodological sections. For more rigorous theoretical background on conservation laws in PDEs, which we will not rely on, see for example [14]. Various approaches to enforcing such conservation laws have been proposed in the context of operator learning. Approaches targeting conservation laws directly include ClawNO [15] which focuses on constructing a divergence-free output field, and

ProbConserv which takes a probabilistic perspective [16]. Perhaps most closely related to our work are the approaches proposed in [17], one of which is designed as a projection of the output from a DeepONet to a target manifold satisfying a constraint encoding the conservation law. We go beyond this by combining the DeepONet with spectral methods. Similar projection methods have been proposed for the FNO [18], however they focus on local properties imposed by the PDE in question instead of global integral quantities as we do in this paper.

Chapter 3

An Introductory Example

Many of the core methods proposed in this paper are best illustrated by considering the introductory example of the simple harmonic oscillator which we define in the following.

3.1 Harmonic Oscillator

Definition 3.1.1. *The **harmonic oscillator** is a function $x : \mathbb{R}^+ \to \mathbb{R}$ solving the second order ordinary differential equation (ODE) given by*

$$\begin{aligned} \frac{\mathrm{d}^2 x}{\mathrm{d}t^2} &= -\omega^2 x \\ \textit{with} \quad x(0) = q_0, &\quad \left.\frac{\mathrm{d}x}{\mathrm{d}t}\right|_{t=0} = p_0. \end{aligned}$$

The harmonic oscillator is a simple reflection of Hooke's law, and describes the movement of a mass attached to a spring. Note that the initial position q_0 and velocity p_0 of the mass determine the dynamics of the system, which may be visualised in the (v, x)-phase plane, $\mathbb{R}^2$. To describe the evolution in this phase plane, we may reformulate the equation as a first order coupled system given by

$$\begin{aligned} \frac{dx}{dt} &= v \\ \frac{dv}{dt} &= -\omega^2 x \\ \text{with} \quad x(0) = q_0, &\quad \left.\frac{\mathrm{d}x}{\mathrm{d}t}\right|_{t=0} = p_0. \end{aligned}$$

It is easy to see that the solution must take the general form

$$x(t) = a\sin(\omega t) + b\cos(\omega t) \tag{3.1}$$

$$v(t) = \omega(a\cos(\omega t) - b\sin(\omega t)) \tag{3.2}$$

where the initial conditions impose that $a = p_0/\omega$ and $b = q_0$. This solution not only specifies an analytical formula to generate training data for training the solution operator $\Psi_\theta : \mathcal{V} \to \mathcal{U}$. It also reveals that in this simple example, $\mathcal{V}$ is isomorphic to $\mathbb{R}^2$, and that $\mathcal{U}$

is a two-dimensional subspace of $\mathcal{C}^\infty(\mathbb{R}^+)$ spanned by the set $\{\cos, \sin\}$. The reason why this example is illustrative for our goal is that the total energy of the system is conserved under these dynamics. The energy is naturally given as the sum of kinetic and potential energy, also known as the Hamiltonian:

$$\mathcal{E}_t(\hat{\boldsymbol{u}}) \coloneqq \underbrace{\frac{1}{2}\omega^2 x(t)^2}_{\text{potential energy}} + \underbrace{\frac{1}{2}v(t)^2}_{\text{kinetic energy}} = \frac{1}{2}\omega^2 q_0^2 + \frac{1}{2}p_0^2 =: \mathcal{E}_0 \tag{3.3}$$

One important insight to extract from this example is that the energy may be viewed as the square of a norm induced by a weighted inner product.

Definition 3.1.2. *Let $\boldsymbol{W} \in \mathbb{R}^n \times \mathbb{R}^n$ be a given invertible diagonal matrix. Let $\langle \cdot, \cdot \rangle$ be an inner product on some n-dimensional vector space V over $\mathbb{C}$. We define the* ***$\boldsymbol{W}$-inner product*** *w.r.t. the standard basis $E = \{e_i\}_{i=1}^n$ of V, denoted $\langle \cdot, \cdot \rangle_{\boldsymbol{W}}$, by the bilinear form $(u, v) \mapsto \langle \boldsymbol{W}^{1/2}\hat{\boldsymbol{u}}, \boldsymbol{W}^{1/2}\hat{\boldsymbol{v}} \rangle$, where $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$ are the coordinates of u and v w.r.t E. Denote the corresponding norm as $|| \cdot ||_{\boldsymbol{W}}$.*

It is easy to check that $|| \cdot ||_{\boldsymbol{W}}$ remains valid inner product. Recall that we also used the notation $\hat{\boldsymbol{u}}$ for the vector of Fourier modes of a function u. We accept this abuse of notation here, as this will allow a neat parallel notation for the spectral version of this argument in Chapter 4. In the case of the harmonic oscillator, $\mathcal{V} = \mathbb{R}^2$, $\langle \cdot, \cdot \rangle = \sqrt{\hat{\boldsymbol{u}} \cdot \hat{\boldsymbol{v}}}$ and

$$\boldsymbol{W} = \frac{1}{2}\begin{pmatrix} \omega^2 & 0 \\ 0 & 1 \end{pmatrix}.$$

We can see that $\mathcal{E}_t(\hat{\boldsymbol{u}}) = ||\hat{\boldsymbol{u}}(t)||^2_{\boldsymbol{W}}$, writing $\hat{\boldsymbol{u}}(t) = \begin{pmatrix} x(t) & v(t) \end{pmatrix}^T$. Similar conservation laws, stating that certain quantities remain unchanged as the system evolves, exist for many systems described by ODEs or PDEs. Such conservation laws form the motivation behind this paper. To learn realistic, physical behaviour it is important to not only reduce the deviation between the ground truth and the prediction, but also to ensure that even if deviations occur, the predicted system viewed in isolation still behaves as the corresponding physical system would. The conservation of the relevant quantities is a necessary, through not sufficient condition for achieving this.

3.2 Proposed Method

Recall that in a DeepONet the output of the mapping prescribed by the learned operator $\Psi_\theta : \mathcal{V} \to \mathcal{U}$ can be described as the matrix vector product in (2.4). In the case of the harmonic oscillator, $\boldsymbol{v} = \hat{\boldsymbol{u}}_0 = (q_0, p_0) \in \mathbb{R}^2$, $\boldsymbol{y} = t \in \mathbb{R}^+$ and $d_{out} = 2$. We further require

$p \geq 2$, as we have established that $\mathcal{U}$ is two dimensional. For the predicted position and velocity we thus write:

$$\hat{\boldsymbol{u}} = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \Psi_\theta(\boldsymbol{u}_0)(t) = \underbrace{\boldsymbol{B}_\theta(\boldsymbol{u}_0)}_{2\times p} \underbrace{\boldsymbol{\alpha}_\theta(t)}_{p\times 1} \quad \text{for} \quad t \in \mathbb{R}^+ \tag{3.4}$$

We shall assume the dependence of $\boldsymbol{B}$ on $\boldsymbol{u}_0$ and of $\boldsymbol{\alpha}$ on t implicitly in what follows, to simplify notation.

3.2.1 Energy Conservation

Having defined $\langle \cdot, \cdot \rangle_{\boldsymbol{W}}$ as above and given the initial energy $\mathcal{E}_0$ we now note the following simple lemma that guarantees energy conservation.

Lemma 1. *Let the harmonic oscillator problem with associated energy as in* (3.3)*, initial data* $\boldsymbol{u}_0$ *with initial energy* $\mathcal{E}_0$ *be given. Then the DeepONet in* (3.4) *can be forced to conserve the learned energy by normalisation the output with the factor*

$$\lambda_1 = \sqrt{\frac{\mathcal{E}_0}{\mathcal{E}_t(\hat{\boldsymbol{u}})}}.$$

where $\mathcal{E}_t(\hat{\boldsymbol{u}}) = ||\boldsymbol{B}\boldsymbol{\alpha}||^2_{\boldsymbol{W}}$.

Proof. We have

$$\mathcal{E}_t(\lambda_1 \hat{\boldsymbol{u}}) = ||\lambda_1 \boldsymbol{B}\alpha||^2 = ||\frac{\sqrt{\mathcal{E}_0}}{||\boldsymbol{B}\boldsymbol{\alpha}||_{\boldsymbol{W}}} \boldsymbol{B}\alpha||^2_{\boldsymbol{W}} = \frac{\mathcal{E}_0}{||\boldsymbol{B}\boldsymbol{\alpha}||^2_{\boldsymbol{W}}} ||\boldsymbol{B}\alpha||^2_{\boldsymbol{W}} = \mathcal{E}_0$$

□

It should be noted that the same can be achieved by applying a similar normalisation w.r.t. the standard Euclidean norm to $\boldsymbol{\alpha}$, while ensuring the orthonormality of $\boldsymbol{B}$ w.r.t. $\langle \cdot, \cdot \rangle_{\boldsymbol{W}}$. While the implementation of this approach is theoretically more complicated (cf. Section 3.3.1.1), it may deliver different numerical results in practice. It is further interesting to note that the requirement for $\boldsymbol{B}$ to be orthonormal implicitly enforces the restriction $p \leq d_{out} = 2$. We briefly describe the argument here and compare the approaches in our numerical experiments.

Lemma 2. *Let the harmonic oscillator problem with associated energy as in* (3.3)*, initial data* $\boldsymbol{u}_0$ *with initial energy* $\mathcal{E}_0$ *be given. Then the DeepONet in* (3.4) *can be forced to conserve learned energy by normalising the trunk* $\boldsymbol{\alpha}$ *by a factor*

$$\lambda_2 = \frac{\sqrt{\mathcal{E}_0}}{||\boldsymbol{\alpha}||}$$

and ensuring that the branch $\boldsymbol{B}$ *has orthonormal columns w.r.t.* $||\cdot||_{\boldsymbol{W}}$.

Proof. Letting $\tilde{\boldsymbol{\alpha}} = \lambda_1 \boldsymbol{\alpha}$ and denoting by $\hat{\boldsymbol{u}}$ the post-processed output, we have

$$\begin{aligned}
\mathcal{E}_t(\hat{\boldsymbol{u}}) = ||\boldsymbol{B}\tilde{\boldsymbol{\alpha}}||^2_{\boldsymbol{W}} &= \langle \boldsymbol{B}\tilde{\boldsymbol{\alpha}}, \boldsymbol{B}\tilde{\boldsymbol{\alpha}} \rangle_{\boldsymbol{W}} \\
&= \langle \sum_{k=1}^{p} \tilde{\alpha}_k \boldsymbol{B}_{\cdot k}, \sum_{k=1}^{p} \tilde{\alpha}_k \boldsymbol{B}_{\cdot k} \rangle_{\boldsymbol{W}} \\
&= \sum_{i,j=1}^{p} \tilde{\alpha}_i \tilde{\alpha}_j \langle \boldsymbol{B}_{\cdot i}, \boldsymbol{B}_{\cdot j} \rangle_{\boldsymbol{W}} && \text{by bilinearity of } \langle \cdot, \cdot \rangle \\
&= \sum_{i=1}^{p} \tilde{\alpha}_i^2 && \text{by orthonormality} \\
&= ||\tilde{\boldsymbol{\alpha}}||^2 \\
&= ||\lambda_2 \boldsymbol{\alpha}||^2 \\
&= \mathcal{E}_0
\end{aligned}$$

□

3.2.2 Learned vs. Implicit Energy

When discussing the energy, $\mathcal{E}_t$, it is important to distinguish between two concepts: the *learned* energy and the *implicit* energy. The learned energy corresponds simply to the computation prescribed by (3.3), using the predicted values $(x(t), v(t)) = \Psi_\theta(\boldsymbol{u}_0)(t)$. Let us denote the first and second component component of $\Psi_\theta(\boldsymbol{u}_0)(t)$ by $\Psi^1_\theta(\boldsymbol{u}_0)(t)$ and $\Psi^2_\theta(\boldsymbol{u}_0)(t)$ respectively. The implicit energy is then obtained with the same formula, but computing the velocity *implicitly* by differentiating the first component w.r.t. the trunk input as $\partial_t x(t) = \partial_t \Psi_\theta(q_0, p_0)_1(t)$. In general,

$$v(t) = \Psi^2_\theta(q_0, p_0)(t) \neq \partial_t \Psi^1_\theta(q_0, p_0)_1(t) = \partial_t x(t).$$

We note that Lemma 1 and Lemma 2 hold only when the learned energy is used in the post-processing step and only when the learned energy is used to monitor energy preservation. All other combinations will, in general, not lead to observing energy conservation as is summarised in Table 3.1.

monitoring \ post-processing	**learned**	**implicit**
learned	✓	✗
implicit	✗	✗

Table 3.1: Exact conservation depending on energy monitored and energy used in post-processing.

Let us first consider the meaning of the learned energy. Here, the velocity is learned separately from the positional predictions, and the predicted velocity is not guaranteed

to reflect the infinitesimal changes in the positional prediction. Of course, once both positional and velocity components have converged to the true dynamics, we may take the learned energy as a valid approximation of the true energy.
The implicit energy on the other hand does not rely on the velocity predictions. The issue with achieving conservation of the implicit energy with the methods proposed in Lemma 1 and Lemma 2 is a non-linear feedback loop that ensues regardless of whether learned or implicit energy is used in the post-processing step. To see this, consider the post-processing prescribed by Lemma 1. Denote the final output by $\tilde{x} = \lambda_1 x$. Then, by the product rule, we have

$$\partial_t \tilde{x} = (\partial_t \lambda_1) x + \lambda_1 \partial_t x.$$

The final implicit energy, which is a function only of $\tilde{x}$, therefore becomes

$$\begin{aligned}\mathcal{E}_t(\tilde{x}) &= \frac{1}{2}\omega^2 \tilde{x}^2 + \frac{1}{2}(\partial_t \tilde{x})^2 \\ &= \lambda_1^2 (\frac{1}{2}\omega^2 x^2 + \frac{1}{2}(\partial_t x)^2) + \frac{1}{2}(\partial_t \lambda_1)^2 x^2 + \lambda_1 x (\partial_t \lambda_1)(\partial_t x).\end{aligned}$$

In the case where the implicit energy is used in the denominator of λ_1, this simplifies further to

$$\mathcal{E}_t(\tilde{x}) = \mathcal{E}_0 + \underbrace{\frac{1}{2}(\partial_t \lambda_1)^2 x^2 + \lambda_1 x (\partial_t \lambda_1)(\partial_t x)}_{non-linear\ \ feedback}.$$

It is easy to see that the case of the QR normalisation yields a similar non-linearity when requiring orthonormality w.r.t. the weighted inner product but using $\partial_t x(t)$ instead of $v(t)$.

3.3 Numerical Experiments

We now turn to the empirical performance of the augmented DeepONet approaches discussed for learning the dynamics of the harmonic oscillator. When monitoring the energy, $\mathcal{E}_t$, we monitor both the learned and implicit energies as discussed in Section 3.2.2.

3.3.1 Implementation

We implement both trunk and branch as neural networks with three hidden layers of size 128, tanh activations, Glorot normal initilisations [19] and optimise with Adam [20]. We set $p = 4$, noting that in the QR method this is implicitly reduced to $p = 2$.

3.3.1.1 Orthonormality of the Branch Matrix

To implement Lemma 2 we need to post-process the branch net to obtain a matrix orthonormal w.r.t. the weighted inner product $\langle \cdot, \cdot \rangle_{\boldsymbol{W}}$. Recall that, for a general matrix $\boldsymbol{M} \in \mathbb{C}^{m \times n}$, by the Gram-Schmidt process [21], there exist a unitary matrix $\boldsymbol{Q} \in \mathbb{C}^{m \times m}$, i.e. $\boldsymbol{Q}^\dagger \boldsymbol{Q} = \boldsymbol{I}$, and an upper triangular matrix $\boldsymbol{R} \in \mathbb{C}^{m \times n}$ such that $\boldsymbol{M} = \boldsymbol{Q}\boldsymbol{R}$. We exploit this by computing the QR decomposition of our branch matrix $\boldsymbol{B}$ (which we may view as complex in general), yielding such $\boldsymbol{Q}$ and $\boldsymbol{R}$ with $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{R}$. Note that this is w.r.t. the standard inner product, not $|| \cdot ||_{\boldsymbol{W}}$. It is easy to see that if $\boldsymbol{Q}$ is unitary, then $\boldsymbol{W}^{-1/2}\boldsymbol{Q}$ is unitary w.r.t. $|| \cdot ||_{\boldsymbol{W}}$, by construction. The `pytorch.linalg` package implements Householder QR [21], which is differentiable and hence can be added as a post-processing step to the branch network without preventing gradient computation. Given $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{R}$, we may obtain our final output implementing Lemma 2 as follows:

$$\hat{\boldsymbol{u}} = \hat{\boldsymbol{B}}\hat{\boldsymbol{\alpha}} = \underbrace{\boldsymbol{W}^{-\frac{1}{2}}\boldsymbol{Q}}_{=:\hat{\boldsymbol{B}}} \underbrace{\frac{\boldsymbol{S}\boldsymbol{R}\alpha}{||\boldsymbol{S}\boldsymbol{R}\alpha||}\sqrt{\mathcal{E}_0}}_{=:\hat{\boldsymbol{\alpha}}} \quad \text{where } \boldsymbol{S} = \boldsymbol{Q}^\dagger \boldsymbol{W}^{1/2}\boldsymbol{Q}$$

This implicitly defines $\tilde{\boldsymbol{B}}$ and $\tilde{\boldsymbol{\alpha}}$ that satisfy Lemma 2. It is obvious that $||\hat{\alpha}||^2 = \mathcal{E}_0$. Furthermore,

$$\begin{aligned}
\langle \tilde{\boldsymbol{B}}_{\cdot i}, \tilde{\boldsymbol{B}}_{\cdot j} \rangle_{\boldsymbol{W}} &= \langle (\boldsymbol{W}^{-1/2}\boldsymbol{Q})_{\cdot i}, (\boldsymbol{W}^{-1/2}\boldsymbol{Q})_{\cdot j} \rangle_{\boldsymbol{W}} \\
&= \langle \boldsymbol{W}^{-1/2}\boldsymbol{Q}_{\cdot i}, \boldsymbol{W}^{-1/2}\boldsymbol{Q}_{\cdot j} \rangle_{\boldsymbol{W}} \\
&= \langle \boldsymbol{Q}_{\cdot i}, \boldsymbol{Q}_{\cdot j} \rangle \\
&= \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} && \text{by orthonormality of } \mathbf{Q}
\end{aligned}$$

as required. The requirement that $d_{out} \geq p$ here is enforced automatically by the QR decomposition: Even if $\boldsymbol{B}$ is a wide matrix, $\boldsymbol{Q}$ will be square of dimension d_{out}, and $\boldsymbol{\alpha}$ is projected by $\boldsymbol{R}$ to match this dimensionality.

3.3.1.2 Implicit Energy

To compute the implicit energy, we make use of the differentiable architecture in `torch` and obtain gradients with respect to the trunk input via backward mode auto differentiation (AD). The computational overhead of this operation is roughly equivalent to an additional forward pass in our case since the dimensionality of the input (t) and the relevant output (x) are both 1 (cf. [22]).

3.3.2 Data

Training data is generated using the analytical solution in (3.1) and (3.2), for 50 initial conditions uniformly sampled with $q_0 \in [-1, 1]$, $p_0 \in [-1, 1]$ with ω fixed at 1. Separate

validation and test sets are obtained in the same way. The validation set is used to determine overfitting to the training data. We train the models up to $T = 10$ at a discretisation of $dt = 0.05$.

3.3.3 Results

The models which we compare against each other are the vanilla DeepONet, the QR DeepONet, the Normalised DeepONet and the Implicitly Normalised DeepONet. The latter two differ by the energy used in their normalisation, with the first relying on the learned energy and the second relying on the implicit energy (cf. Section 3.2.2). Our primary metric is the MSE on the test set, which includes both predictions on x and v, even where the implicit energy is used to ensure a fair comparison. We consider the test set MSE of the learned and implicit energies as metrics of successful energy conservation. Final results are shown in Table 3.2, averaged over five training runs. We note that both methods for the exact conservation of the learned energy achieve an improved test set performance compared to the vanilla method. Notably, the test set implicit energy MSE is highest for the QR method, despite outperforming the normalisation method in terms of test MSE. The normalisation method on the other hand reduces also the implicit energy MSE significantly compared to the vanilla method. The best model across metrics is the implicit normalisation method. While, as expected, it does not reduce the learned energy error to zero, it achieves marked improvements on both the test MSE and the implicit energy MSE with low standard deviation. We show training dynamics in Figure 3.1 and Figure 3.2. It is noteworthy that throughout training, training and validation MSE are almost identical for the exact conservation methods, with slight gaps appearing for the two other methods. Albeit at the expense of model performance, this is evidence supporting the hypothesis that enforcing energy conservation improves the models ability to generalise. Finally, predictions for an arbitrarily selected initial condition from the test set can be found in Figure 3.3.

Table 3.2: Test MSE and learned and implicit energy MSE on the test set (in 1×10^{-2}).

			Test Energy MSE			
	Test MSE		**Learned**		**Implicit**	
Vanilla	0.115	± 0.115	454.619	± 0.810	455.158	± 0.693
QR	0.139	± 0.115	**0.000**	**± 0.000**	832.810	± 1.819
Norm.	0.323	± 0.240	**0.000**	**± 0.000**	3.289	± 1.170
Imp. Norm.	**0.035**	**± 0.017**	0.078	± 0.025	**0.629**	**± 0.094**

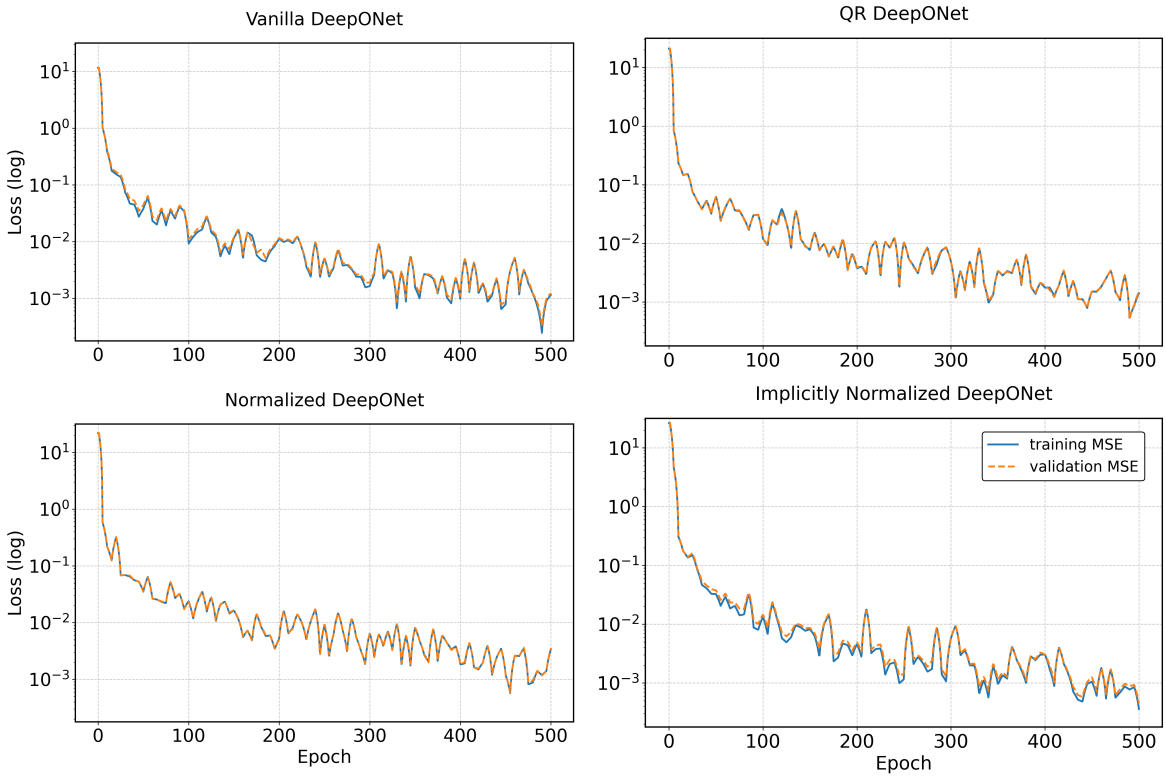

Figure 3.1: Prediction and energy evolution of DeepONet, QR DeepONet, Normalised DeepOnet and Implicitly Normalised DeepONet. Averaged over 5 training runs, with standard deviation.

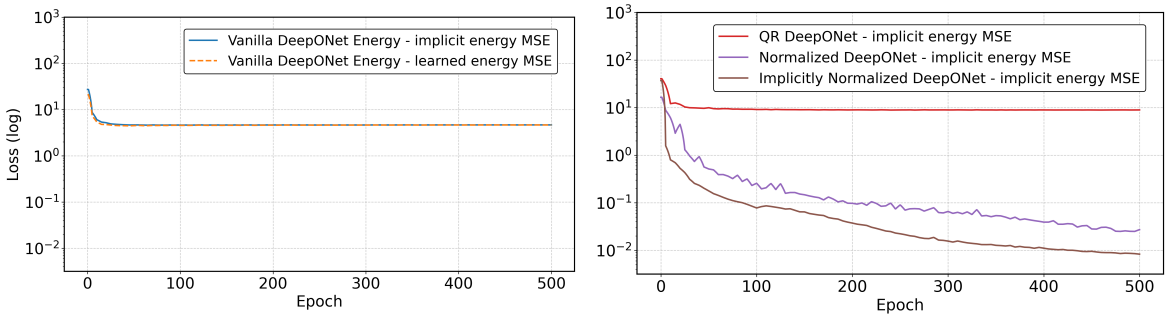

Figure 3.2: Left: Implicit and learned energies during training of vanilla DeepONet. Right: Implicit energies during training of QR DeepONet, Norm DeepOnet and Implicit Norm DeepONet.

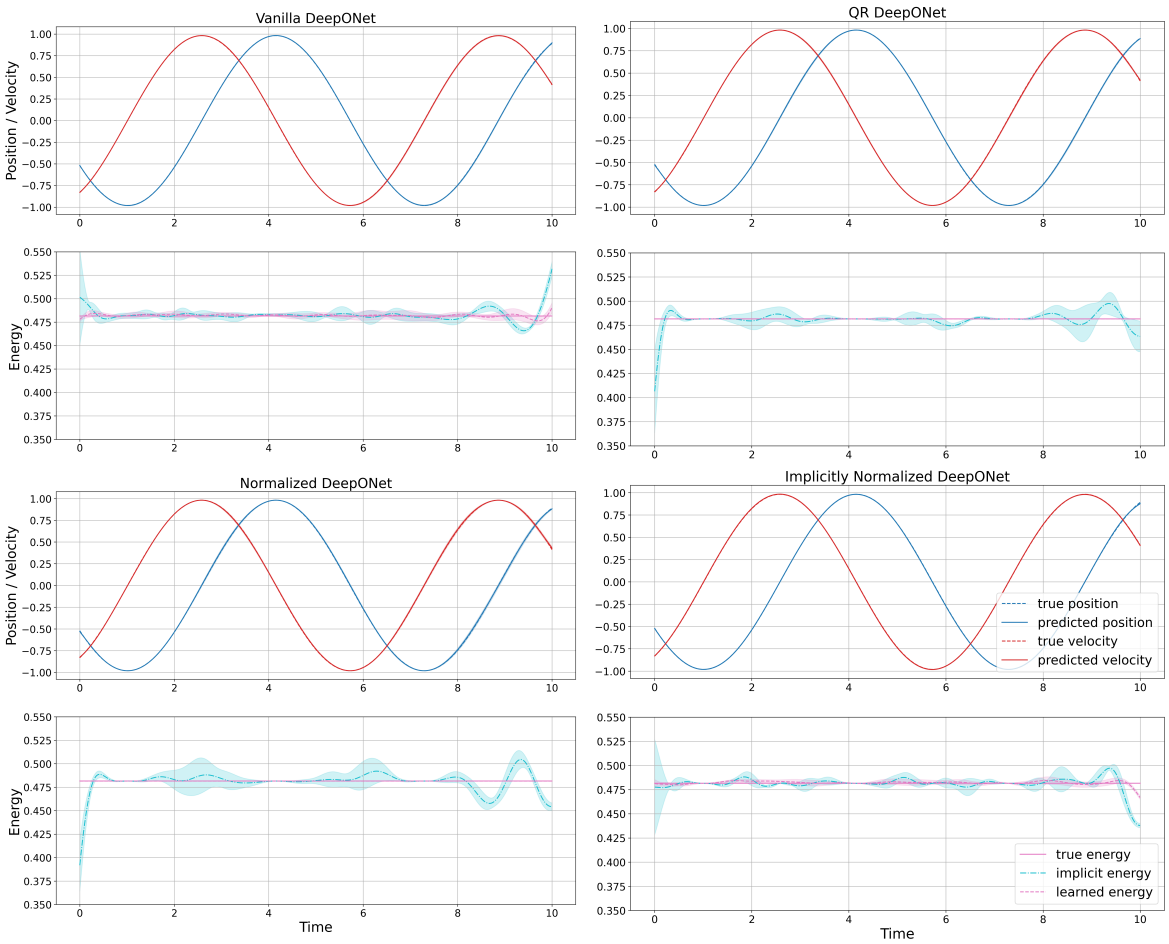

Figure 3.3: Predictions and energy evolution for Vanilla DeepONet, QR DeepONet, Norm DeepOnet and Implicit Norm DeepONet on an arbitrary test set example. Averaged over 5 training runs with standard deviation shaded.

Chapter 4

Spectral Methods

In this chapter we extend the insights gained from the harmonic oscillator example to more complex dynamics governed by PDEs which also conserve some known quantity over their evolution in time, now taking the form of an integral. We focus on the wave equation, whose solution is intricately related to the harmonic oscillator, but shall generalise where possible. The key to transferring the methodology to this setting lies in viewing the problem in the spectral domain, and we shall rely on key results from Section 2.1.

4.1 Wave Equation

The wave equation in one dimension is given by the following equation:

$$\partial_{tt} u = c^2 \partial_{xx} u \tag{4.1}$$

A solution to the wave equation conserves energy over time, which is computed as

$$\mathcal{E}(t) = \frac{1}{2} \int_\Omega \left((\partial_t u)^2 + c^2 (\partial_x u)^2 \right). \tag{4.2}$$

We shall work on a periodic domain $\Omega = [0, L_x] \subset \mathbb{R}$ and assume $u \in C^\infty([0, L_t] \times \Omega)$, where $L_t = L_x / c$ is the known time-periodicity of the solution to the wave equation. We may therefore assume periodic boundary conditions:

$$\partial_x^k u|_{x=0} = \partial_x^k u|_{x=L_x} \quad \text{and} \quad \partial_t^k u|_{t=0} = \partial_t^k u|_{t=L_t} \quad \forall\ k \geq 0$$

To better understand how the insights from the harmonic oscillator can be translated to the wave equation, let us derive the analytical solution under these conditions.

Theorem 4.1.1. *The wave equation* (4.1) *with periodic boundary conditions and initial data* $u(0, \cdot)$ *and* $\partial_t u(0, \cdot)$*, has solution*

$$u(x,t) = \sum_{k=-\infty}^{\infty} \hat{u}_k(t) e^{i\omega_k x}, \quad \omega_k = \frac{2\pi k}{L_x}$$

with

$$\hat{u}_k(t) = \begin{cases} \frac{\hat{u}_k'(0)}{c\omega_k}\sin(c\omega_k t) + \hat{u}_k(0)\cos(c\omega_k t) & \text{if} \quad k \neq 0 \\ \hat{u}_0'(0)t + \hat{u}_0(0) & \text{if} \quad k = 0 \end{cases}$$

where $\hat{u}(0)_k$ *and* $\hat{u}'(0)_k$ *are the Fourier coefficients of* $u(0,\cdot)$ *and* $\partial_t u(0,\cdot)$ *respectively.*

Proof. By periodicity, leveraging (2.1), we must look for solutions of the form

$$u(x,t) = \sum_{k=-\infty}^{\infty} \hat{u}_k(t) e^{i\omega_k x}$$

Then, differentiating to obtain both sides of (4.1), we find

$$\partial_{tt} u = \sum_{k=-\infty}^{\infty} \hat{u}_k''(t) e^{i\omega_k x} \quad \text{and} \quad \partial_{xx} u = -\sum_{k=-\infty}^{\infty} \omega_k^2 \hat{u}_k(t) e^{i\omega_k x}$$

Hence, we require

$$\sum_{k=-\infty}^{\infty} \hat{u}_k''(t) e^{i\omega_k x} = -c^2 \sum_{k=-\infty}^{\infty} \omega_k^2 \hat{u}_k(t) e^{i\omega_k x} \quad \text{for all } x$$

But the set $\{e^{i\omega_k x}\}_{k=-\infty}^{k=\infty}$ is pairwise orthogonal with respect to the inner product $\int u\bar{u}\,dx$ on $L^2([0, L_x])$. We thus may equate coefficients for each k. This yields,

$$\hat{u}_k''(t) + c^2\omega_k^2 \hat{u}_k(t) = 0 \quad \text{for} \quad k \in \mathbb{Z}$$

But this is just the harmonic oscillator, except when $k = 0$ (cf. Definition 3.1.1).

Case 1: $k \neq 0$ We saw in Chapter 3 that the harmonic oscillator has solution

$$\hat{u}_k(t) = a\sin(c\omega_k t) + b\cos(c\omega_k t)$$

with

$$a = \frac{p_0}{c\omega_k}, \quad b = q_0$$

In this case, $p_0 = \hat{u}_k'(0)$ and $q_0 = \hat{u}_k(0)$, obtained via Fourier transform on initial data $u(0,\cdot)$ and $\partial_t u(0,\cdot)$.

Case 2: $k = 0$ Noting that $\omega_0 = 0$, we need to solve $\hat{u}_k''(t) = 0$, so

$$\hat{u}_0(t) = at + b$$

for some $a, b \in \mathbb{R}$. From the initial conditions, we get

$$a = \hat{u}_0'(0), \quad b = \hat{u}_0(0)$$

which again can be obtained via the Fourier transform on the initial data $u(0,\cdot)$ and $\partial_t u(0,\cdot)$. □

Essentially, the solution to the wave equation in the periodic case evolves like infinitely many harmonic oscillators on the Fourier coefficients. Thus, by moving to the spectral domain, we may apply similar methods to the one used for the harmonic oscillator in Chapter 3. Before presenting the proposed architecture, we discuss the similarity of the conserved energies between the harmonic oscillator and the wave equation. To this end, note that for a periodic solution to (4.1), by Parceval's theorem (2.1.1), we may rewrite the preserved energy as

$$\mathcal{E}_t(u) = \frac{1}{2}\int_\Omega \left(v^2 + c^2(\partial_x u)^2\right) = \frac{L_x}{2}\sum_{k=-\infty}^{\infty}\left(|\hat{v}_k|^2 + c^2\omega_k^2|\hat{u}_k|^2\right) \tag{4.3}$$

where implicitly we defined $\partial_t u = v$ and we obtained the Fourier coefficients of the spatial partial derivative $\partial_x u$ in the spectral domain by multiplying by $i\omega_k$. We note how the energy Equation 4.3 can be computed in matrix form. For the RHS of (4.3) we may write

$$\mathcal{E}_t(u) = \frac{1}{2}\int_\Omega \begin{pmatrix} v & \partial_x u(t)\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & c^2\end{pmatrix}\begin{pmatrix} v \\ \partial_x u(t)\end{pmatrix} dx. \tag{4.4}$$

If we further assume that u and v have finitely many non-zero Fourier coefficients, say $\{\hat{v}_i\}_{i=-r}^{r}$ and $\{\hat{u}_i^{(\alpha_0)}\}_{i=-r}^{r}$. We may then write the RHS of (4.3) in block notation as

$$\mathcal{E}_t(u) = \frac{L_x}{2}\begin{pmatrix} v_{-r} \\ \vdots \\ v_r \\ \frac{2\pi(-r)}{L_x}\hat{u}_{-r} \\ \vdots \\ \frac{2\pi r}{L_x}\hat{u}_r\end{pmatrix}^\dagger \begin{pmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{c^2 I}\end{pmatrix}\begin{pmatrix} v_{-r} \\ \vdots \\ \hat{v}_r \\ \frac{2\pi(-r)}{L_x}\hat{u}_{-r} \\ \vdots \\ \frac{2\pi r}{L_x}\hat{u}_r\end{pmatrix}..$$

When learning these dynamics from data, as in the case of the harmonic oscillator, we may either obtain v *implicitly* by differentiating the architecture with respect to the input t, or *learn* a second output function v. The latter amounts to learning the equivalent coupled system given by:

$$\partial_t u = v$$
$$\partial_{xx} u = c^2 \partial_t v$$

For the wave equation, will will thus be looking to learn an operator $\Psi : \mathcal{V} \to \mathcal{U}$ with $\mathcal{V} = C^\infty(\{0\}\times\Omega)^2$ and $\mathcal{U} = C^\infty([0, L_t]\times\Omega)$ or $\mathcal{U} = C^\infty([0, L_t]\times\Omega)^2$ depending on whether we learn v separately.

4.2 General Setting

We now generalise the setting of the wave equation to that of more general PDEs that satisfy some quadratic conservation law. Let $\Omega = [0, L_1]\times\cdots\times[0, L_n] \subset \mathbb{R}^n$ be a periodic

domain, let $\mathcal{L}$ be some potentially non-linear differential operator and let $\beta \in \{1, 2, \dots\}$ be given. Suppose that whenever $u \in C^\infty(\mathbb{R}^+ \times \Omega, \mathbb{C})$ solves

$$\mathcal{L}[u] = \partial_t^\beta u, \tag{4.5}$$

u conserves a quadratic functional of the form

$$\mathcal{E}_t[u] = \int_\Omega \sum_{j \in \{0,\dots,k\}} c_j^2 |\partial_{i(j)}^{\alpha_j} u|^2 dx$$

where $k \leq n$, $c_j \neq 0$ for all $j \in [k]$, and $i : [k] \to [n]$ is injective with $i(0) = 0$. The component, and partial, indexed by 0 should be understood as the time dimension. Note that, analogously to (4.4), we may equivalently write

$$\mathcal{E}_t[u] = \int_\Omega (\mathcal{D}u)^\dagger \Sigma \mathcal{D}v \tag{4.6}$$

where we denote by $\mathcal{D}u$ the vector with entries $(\mathcal{D}u)_j = \partial_{i(j)}^{\alpha_j} u$ for a k-dimensional vector of partial derivatives of u, and $\Sigma \in \mathbb{R}^{(k+1)\times(k+1)}$ for a diagonal matrix containing the squared constants $c_0^2, \dots, c_k^2$. By construction Σ is invertible. For consistency with the harmonic oscillator and the wave equation we shall in general refer to $\mathcal{E}_t[u]$ as the energy, though the conserved quantity may correspond to other properties, e.g. total probability in the case of the Schrödinger equation (cf. Table 4.1). It should be clear that the energy derives from a symmetric bilinear form

$$\langle u, v \rangle_{\mathcal{D},\Sigma} = \int_\Omega (\mathcal{D}u)^\dagger \Sigma \mathcal{D}v.$$

By the periodicity of u in Ω, we now note that any $u \in C^\infty(\mathbb{R}^+ \times \Omega, \mathbb{C})$ may be expressed as a Fourier series with time-varying coefficients, i.e.

$$u(t, \boldsymbol{x}) = \sum_{\boldsymbol{m} \in \mathbb{Z}^n} \hat{u}_{\boldsymbol{m}}(t) e^{i\boldsymbol{\omega}_{\boldsymbol{m}} \cdot \boldsymbol{x}} \tag{4.7}$$

and for the α_0-th partial derivatives w.r.t. the time dimension, we have

$$\partial_0^{\alpha_0} u(t, \boldsymbol{x}) = \sum_{\boldsymbol{m} \in \mathbb{Z}^n} \hat{u}_{\boldsymbol{m}}^{(\alpha_0)}(t) e^{i\boldsymbol{\omega}_{\boldsymbol{m}} \cdot \boldsymbol{x}} \tag{4.8}$$

It follows again from Theorem 2.1.1, denoting by ℓ the product $\prod_{j=1}^n L_j$, that

$$\mathcal{E}_t(\partial_t^{\alpha_0} u, u) = \ell c_0^2 \sum_{\boldsymbol{m} \in \mathbb{Z}^n} |\hat{u}_{\boldsymbol{m}}^{(\alpha_0)}(t)|^2 + \ell \sum_{j \in [k] \setminus \{0\}} c_j^2 \sum_{\boldsymbol{m} \in \mathbb{Z}^n} |\big(\frac{2\pi m_j}{L_j}\big)^{\alpha_j} \hat{u}_{\boldsymbol{m}}(t)|^2 \tag{4.9}$$

To make this computation tractable, we will again assume that $\hat{u}_{\boldsymbol{m}}^{(\alpha_0)}(t) = 0$ and $\hat{u}(t)_{\boldsymbol{m}} = 0$ for all $\boldsymbol{m} \in R \coloneqq \{(m_1, \dots, m_n) | \max(\{|m_i|\}) > r\}$, and define $d = (2r + 1)$. To simplify notation, we will further re-index the sets $\{\hat{u}_{\boldsymbol{m}}^{(\alpha_0)}\}_{\boldsymbol{m} \in R}$ and $\{\hat{u}_{\boldsymbol{m}}\}_{\boldsymbol{m} \in R}$ to $\{\hat{u}_i^{(\alpha_0)}\}_{i=1}^{d^n}$ and

$\{\hat{u}_i\}_{i=1}^{d^n}$ respectively. We collect these as vectors $\hat{\boldsymbol{u}}^{(\alpha_0)}$ and $\hat{\boldsymbol{u}}$ in $\mathbb{C}^{d^n}$. Then we may express the energy equivalently in matrix notation as

$$\mathcal{E}_t(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}}) = \ell\hat{\mathcal{D}}(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}})^\dagger \hat{\Sigma}\hat{\mathcal{D}}(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}}) \tag{4.10}$$

where the i-th component of $\hat{\mathcal{D}}(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}}) \in \mathbb{C}^{d^n(k+1)}$ is given by

$$\hat{\mathcal{D}}(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}})_j = \begin{cases} \hat{u}_j^{(\alpha_0)}(t) & \text{for } j \leq d^n \\ \frac{2\pi([j \mod d]-1-r)}{L_{i(\lfloor j/d^n \rfloor)}} \hat{u}_{(j \mod d^n)}(t) & \text{otherwise} \end{cases}$$

and $\hat{\Sigma} \in \mathbb{R}^{(k+1)d^n \times (k+1)d^n}$ is defined as

$$\hat{\Sigma}_{ij} = \begin{cases} c^2_{\lfloor i/d^n \rfloor} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}.$$

Again, we note that this can be viewed to derive from a symmetric bilinear form

$$\langle (\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}}), (\hat{\boldsymbol{v}}^{(\alpha_0)}, \hat{\boldsymbol{v}}) \rangle_{\hat{\mathcal{D}}, \hat{\Sigma}} = \ell\hat{\mathcal{D}}(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}})^\dagger \hat{\Sigma}\hat{\mathcal{D}}(\hat{\boldsymbol{v}}^{(\alpha_0)}, \hat{\boldsymbol{v}})$$

It is easy to check that this is indeed linear in both arguments since it is a symmetric quadratic form and $\hat{\mathcal{D}}$ is a linear operator by construction.
Of course, when we are interested in learning the *solution operator* to (4.5), we must specify a suitable domain. Typically, we will assume smooth initial data of the form

$$u(0, \cdot), \ldots, \partial_t^l u(0, \cdot) \in C^\infty(\{0\} \times \Omega, \mathbb{C}) \tag{4.11}$$

where l is the lowest index such that u is uniquely defined. The operator of interest is therefore $\Psi : \mathcal{V} \to \mathcal{U}$ with $\mathcal{V} = C^\infty(\{0\} \times \Omega, \mathbb{C})^l$ and $\mathcal{U} = C^\infty(\mathbb{R}^+ \times \Omega, \mathbb{C})$ or $\mathcal{U} = C^\infty(\mathbb{R}^+ \times \Omega, \mathbb{C})^2$ – depending on whether we also *learn* some time derivative of u.
While the described generalisation is framed for quadratic functionals with non-zero contributions from the time derivative, it should be easy to see that this can be adapted to quadratic functionals with no contribution from the time derivative by adapting $\mathcal{D}$ and $\hat{\mathcal{D}}$ appropriately. Thus, several well known systems conserving L_2-norm fit immediately into this framework. Some examples are listed in Table 4.1.

Name	Equation
Schrödinger	$\partial_t u = i\Delta u + iVu$ $u : \mathbb{R}^+ \times \Omega \to \mathbb{C}$
Nonlinear Schrödinger	$\partial_t u = i\Delta u + i\lvert u\rvert^2 u$ $u : \mathbb{R}^+ \times \Omega \to \mathbb{C}$
Korteweg–De Vries	$\partial_t u = \frac{1}{2}\partial_x(u^2) - \partial_{xxx} u$ $u : \mathbb{R}^+ \times \Omega \to \mathbb{R}$

Table 4.1: Examples of PDEs conserving L_2-norm.

4.3 Proposed Methods

We propose two methods for conserving general integral quantities like the one described in Section 4.2. Both are spectral methods, in the sense that they learn Fourier modes of periodic functions. The first, which we refer to as DeepONet Spectral Operator (DSO) is inspired by the observation that the solution to the wave equation can be phrased as infinitely many harmonic oscillators in the spectral domain. The second, departs from the DeepONet architecture and extends the spectral perspective to the temporal domain, which unlocks a new way to obtain temporal partial derivatives easily in Fourier space. We refer to this method as the Augmented Fourier Neural Operator (AFNO) in reference to [2].

4.3.1 DeepONet Spectral Operator

In this method, we use a DeepONet to model the time evolution of Fourier modes of $u(t,\cdot)$ and $\partial_t^{(\alpha_0)}u(t,\cdot)$. Different from the Fourier integral operator in the FNO (Definition 2.4.1), the DSO is a non-linear transformation in the spectral domain which is conditional on a separate input, t. Note, however, that this does not generalises the FNO, which contains several layers of Fourier integral operators paired with a separate linear transformation in physical space.

Definition 4.3.1 (DeepONet Spectral Operator — DSO)**.** *Writing $\boldsymbol{u}_0 = (u_0, \dots, \partial_t^k u_0)$, the DeepONet spectral operator $\Psi_\theta : C^\infty(\{0\} \times \Omega, \mathbb{C})^k \to C^\infty(\mathbb{R}^+ \times \Omega, \mathbb{C})^2$ is a map $\boldsymbol{u}_0 \mapsto \Psi_\theta \boldsymbol{u}_0$ where $\Psi \boldsymbol{u}_0$ is parameterised in Fourier space as*

$$\big(\Psi_\theta \boldsymbol{u}_0\big)(t,x) := \begin{pmatrix} \mathcal{F}_x^{-1}\hat{\Psi}_\theta^1(t, \mathcal{F}_x \boldsymbol{u}_0)(x) \\ \mathcal{F}_x^{-1}\hat{\Psi}_\theta^2(t, \mathcal{F}_x \boldsymbol{u}_0)(x) \end{pmatrix} \qquad \forall\ (t,x) \in \Omega.$$

$\mathcal{F}_x \boldsymbol{u}_0$ should be understood to act separately on each component, and $\hat{\Psi}_\theta^1(t, \mathcal{F}_x \boldsymbol{u}_0)$ and $\hat{\Psi}_\theta^2(t, \mathcal{F}_x \boldsymbol{u}_0)$ mean two functions in Fourier space.

In practice, $\boldsymbol{u}_0$ is measured on an evenly spaced grid $\mathcal{I}$ so that each component in the second argument to Ψ_θ has N modes, which can be efficiently obtained with a FDFT (cf. Section 2.1). We parameterise Ψ_θ as a DeepONet in the spectral domain as follows:

$$\hat{\Psi}_\theta(t, \hat{\boldsymbol{u}}_0) = \begin{pmatrix} \hat{\Psi}_\theta^1(t, \hat{\boldsymbol{u}}_0) \\ \hat{\Psi}_\theta^2(t, \hat{\boldsymbol{u}}_0) \end{pmatrix} = \underbrace{\boldsymbol{B}_\theta(\hat{\boldsymbol{u}}_0)}_{2d^n \times p} \underbrace{\boldsymbol{\alpha}(t)}_{p \times 1}$$

where we denote by $\hat{\boldsymbol{u}}_0 \in \mathbb{C}^{kN}$ the vector of Fourier modes of the initial data and $\hat{\Psi}_\theta^i(t, \hat{\boldsymbol{u}}_0)$ is in $\mathbb{C}^{d^n}$ for $i = 1, 2$. As in the FNO [2], d^n may be chosen to be smaller than the desired resolution N_x and padded with zeros prior to applying $\mathcal{F}^{-1}$. Conservation of the learned energy can then be achieved analogously to Lemma 1 and Lemma 2 as follows.

4.3.2 Energy Conservation

The DSO architecture is designed to predict the target u itself, as well as its α_0-th time derivative $\partial_0^{(\alpha_0)} u$ analogously to the case of the Harmonic oscillator. We assign $\hat{\boldsymbol{u}}^{(\alpha_0)} = \hat{\Psi}^1_\theta(t, \hat{\boldsymbol{u}}_0)$ and $\hat{\boldsymbol{u}} = \hat{\Psi}^2_\theta(t, \hat{\boldsymbol{u}}_0)$ to match the notation in Section 4.2. This enables us to compute the learned energy which relies on the vector $\hat{\mathcal{D}}(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}})$.

Lemma 3. *Let Ω be a periodic domain. Let initial data of type* (4.11)*, an energy functional $\mathcal{E}_t(u)$ of type* (4.6) *and the initial energy $\mathcal{E}_0$ be given. Then by normalising with factor*

$$\lambda_3 = \sqrt{\frac{\mathcal{E}_0}{\mathcal{E}_t(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}})}}$$

the DSO can be forced to conserve the learned energy.

Here, of course $\mathcal{E}_t(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}}) = \mathcal{E}_t(\hat{\Psi}_\theta(t, \hat{\boldsymbol{u}}_\mathbf{0})) = \mathcal{E}((\Psi_\theta \boldsymbol{u}_0)(t, \cdot))$, computed according to (4.10).

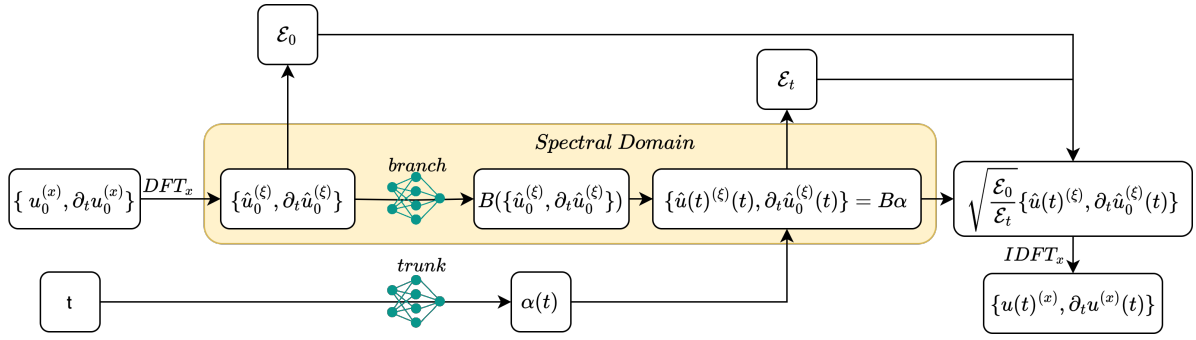

Figure 4.1: Normalised DSO on the example of the wave equation.

Proof. We are interested in the energy of the final, normalised output.

$$\begin{aligned}
\mathcal{E}_t(\lambda_3 \hat{\boldsymbol{u}}^{(\alpha_0)}, \lambda_3 \hat{\boldsymbol{u}}) &= \langle \lambda_3 \boldsymbol{B\alpha}, \lambda_3 \boldsymbol{B\alpha} \rangle_{\hat{\mathcal{D}}, \hat{\Sigma}} && \text{by bilinearity} \\
&= \lambda_3^2 \langle \boldsymbol{B\alpha}, \boldsymbol{B\alpha} \rangle_{\hat{\mathcal{D}}, \hat{\Sigma}} \\
&= \frac{\mathcal{E}_0}{\langle \boldsymbol{B\alpha}, \boldsymbol{B\alpha} \rangle_{\hat{\mathcal{D}}, \hat{\Sigma}}} \langle \boldsymbol{B\alpha}, \boldsymbol{B\alpha} \rangle_{\hat{\mathcal{D}}, \hat{\Sigma}} && \text{by definition of } \mathcal{E}_t(\cdot) \text{ and } \langle \cdot, \cdot \rangle_{\hat{\mathcal{D}}, \hat{\Sigma}} \\
&= \mathcal{E}_0
\end{aligned}$$

□

Note further the following alternative post-processing lemma, analogous to Lemma 2.

Lemma 4. *Let Ω be a periodic domain. Let initial data of type* (4.11)*, an energy functional $\mathcal{E}_t(u)$ of type* (4.6) *and the initial energy $\mathcal{E}_0$ be given. Then by normalising the trunk $\boldsymbol{\alpha}$ by a factor*

$$\lambda_4 = \frac{\sqrt{\mathcal{E}_0}}{||\boldsymbol{\alpha}||} \tag{4.12}$$

and by ensuring orthononormality of the columns of $\boldsymbol{B}_\theta(\boldsymbol{u}_0)$ w.r.t. $\langle\cdot,\cdot\rangle_{\hat{\mathcal{D}},\hat{\Sigma}}$ the DSO can be forced to conserve the learned energy.

Proof. We once again omit the dependence of $\boldsymbol{B}_\theta$ on $\boldsymbol{u}_0$ and of $\boldsymbol{\alpha}_\theta$ on t for notational clarity and assume $\boldsymbol{\alpha}_\theta \in \mathbb{R}^p$ and $\boldsymbol{B}_\theta \in \mathbb{C}^{2d^n \times p}$. We also drop θ in the subscript. Denote the post-processed trunk output by $\tilde{\boldsymbol{\alpha}} = \lambda_4 \boldsymbol{\alpha}$ and by $\hat{\boldsymbol{u}}^{(\alpha_0)}$ and $\hat{\boldsymbol{u}}$ the resulting final output. We compute:

$$\begin{aligned}
\mathcal{E}_t(\hat{\boldsymbol{u}}^{(\alpha_0)}, \hat{\boldsymbol{u}}) &= \langle \boldsymbol{B}\tilde{\boldsymbol{\alpha}}, \boldsymbol{B}\tilde{\boldsymbol{\alpha}} \rangle_{\hat{\mathcal{D}},\hat{\Sigma}} \\
&= \langle \sum_{k=1}^{p} \tilde{\alpha}_k \boldsymbol{B}_{\cdot k}, \sum_{k=1}^{p} \tilde{\alpha}_k \boldsymbol{B}_{\cdot k} \rangle_{\hat{\mathcal{D}},\hat{\Sigma}} \\
&= \sum_{i,j=1}^{p} \tilde{\alpha}_i \tilde{\alpha}_j \langle \boldsymbol{B}_{\cdot k}, \boldsymbol{B}_{\cdot k} \rangle_{\hat{\mathcal{D}},\hat{\Sigma}} && \text{by bilinearity} \\
&= \sum_{i,j=1}^{p} \tilde{\alpha}_i^2 && \text{by orthonormality} \\
&= \|\tilde{\boldsymbol{\alpha}}\|^2 \\
&= \|\lambda_4 \boldsymbol{\alpha}_\theta\|^2 \\
&= \mathcal{E}_0 && \text{by (4.12)}
\end{aligned}$$

□

4.3.2.1 Learned vs. Implicit Energy

The argument is largely analogous to that made in Section 3.2.2. While the learned energy is guaranteed to be constant after post-processing according to Lemma 3 or Lemma 4, there is no guarantee that the learned time derivative corresponds to the true rate of change of the system observed when tracing the spatial dynamics over time. Table 3.1 still holds, as in general:

$$\Psi_\theta^1 \boldsymbol{u}_0(t,x) \neq \partial_0^{\alpha_0} \Psi_\theta^2 \boldsymbol{u}_0(t,x)$$

As before we can see that a non-linear feedback invalidates conservation. Consider again the wave equation, where $\alpha_0 = 1$. When normalising via Lemma 3, irrespective of whether we use the implicit or learned energy in the denominator of λ_3, we find that

$$\partial_t(\lambda_3 u) = (\partial_t \lambda_3) u + \lambda_3 \partial_t u$$

The final implicit energy then becomes

$$\begin{aligned}
\mathcal{E}_t(\lambda_3 \hat{\boldsymbol{u}}) =& \frac{L_x}{2} \sum_{k=-r}^{r} (|(\partial_t \lambda_3)\hat{u}_k + \lambda_3 \partial_t \hat{u}_k|^2 \\
&+ c^2 \omega_k^2 |\lambda_3 \hat{u}_k|^2)
\end{aligned}$$

$$
\begin{aligned}
=&\frac{L_x}{2}\sum_{k=-r}^{r}(|(\partial_t\lambda_3)\hat{u}_k|^2+|\lambda_3\partial_t\hat{u}_k|^2+(\partial_t\lambda_3)\hat{u}_k\lambda_3\overline{\partial_t\hat{u}_k}+(\partial_t\lambda_3)\overline{\hat{u}_k}\lambda_3\partial_t\hat{u}_k\\
&+c^2\omega_k^2|\lambda_3\hat{u}_k|^2)\\
=&\frac{L_x}{2}\sum_{k=-r}^{r}(|\lambda_3\partial_t\hat{u}_k|^2+c^2\omega_k^2|\lambda_3\hat{u}_k|^2)\\
&+\frac{L_x}{2}\sum_{k=-r}^{r}(|(\partial_t\lambda_3)\hat{u}_k|^2+(\partial_t\lambda_3)\hat{u}_k\lambda_3\overline{\partial_t\hat{u}_k}+(\partial_t\lambda_3)\overline{\hat{u}_k}\lambda_3\partial_t\hat{u}_k)
\end{aligned}
$$

This simplifies further in the case where λ_3 contains the implicit energy of the unnormalised output:

$$
\mathcal{E}_t(\lambda_3\hat{\boldsymbol{u}})=\mathcal{E}_0+\underbrace{\frac{L_x}{2}\sum_{k=-r}^{r}(|(\partial_t\lambda_3)\hat{u}_k|^2+(\partial_t\lambda_3)\hat{u}_k\lambda_3\overline{\partial_t\hat{u}_k}+(\partial_t\lambda_3)\overline{\hat{u}_k}\lambda_3\partial_t\hat{u}_k)}_{non-linear\ \ feedback}
$$

The post-processing from Lemma 4 is inexact by the same token when normalising with or monitoring the implicit energy.

4.3.3 Augmented Fourier Neural Operator

Noting that a typical energy formulation, such as (4.3), contains spatial derivatives *and* time derivatives we further propose a fully spectral method, which removes the time-dependence of the Fourier coefficients learned by the architecture. Our method relies on a single layer Fourier integral operator (cf. Definition 2.4.1), with non-linear kernel in Fourier space. The method is a close relative of the FNO [2].

Definition 4.3.2 (Augmented Fourier Neural Operator – AFNO)**.** *Writing* $\boldsymbol{u}_0=(u_0,\dots,\partial_t^k u_0)$*, the augmented Fourier neural operator* $\Psi_\theta : C^\infty(\{0\}\times\Omega,\mathbb{C})^k\to C^\infty([0,L_0]\times\Omega,\mathbb{C})$ *is a map* $\boldsymbol{u}_0\mapsto\Psi_\theta\boldsymbol{u}_0$ *parameterised in Fourier space as*

$$
\big(\Psi_\theta\boldsymbol{u}_0\big)(t,x):=\mathcal{F}_{t,x}^{-1}\Big(\hat{\Psi}_\theta(\mathcal{F}_x\boldsymbol{u}_0)\Big)(t,x)\qquad\forall\ (t,x)\in[0,L_0]\times\Omega
$$

where $\mathcal{F}_x\boldsymbol{u}_0$ *should be understood to act separately on each component.*

Note that in this case we have to assume periodicity of the solution in time *and* space, which was not necessary for the DSO. Again, in practice we assume access to a finite number of measurements of u_0 so that Ψ_θ can be parameterised by a feed-forward neural network, acting on Fourier coefficients obtained via the FDFT in space from the initial data.

To achieve energy conservation in this case, it ought to be noted that we are interested in the integral over the spatial components for each hyperplane of fixed time t. The

energy for a time-slice can be obtained by 'marginalising' over the time dimension. To see this, we augment (4.7), (4.8) and (4.9) to obtain the following analogues, where now $\boldsymbol{m} = (m_0, \ldots, m_k) \in \mathbb{Z}^{n+1}$:

$$u(t,x) = \sum_{\boldsymbol{m}\in\mathbb{Z}^{n+1}} \hat{u}_{\boldsymbol{m}} e^{i\boldsymbol{\omega}_{\boldsymbol{m}}\cdot\boldsymbol{x}}$$

$$\partial_0^{\alpha_0} u(t,x) = \sum_{\boldsymbol{m}\in\mathbb{Z}^{n+1}} \big(i\frac{2\pi m_0}{L_0}\big)^{\alpha_0} \hat{u}_{\boldsymbol{m}} e^{i\boldsymbol{\omega}_{\boldsymbol{m}}\cdot\boldsymbol{x}} \tag{4.13}$$

$$\mathcal{E}_t(u) = \ell \sum_{j\in[k]} c_j^2 \sum_{\boldsymbol{m}\in\mathbb{Z}^{n+1}} |\big(\frac{2\pi m_j}{L_j}\big)^{\alpha_j} \hat{u}_{\boldsymbol{m}}|^2 \tag{4.14}$$

Here, the vector $\boldsymbol{x}$ now is also $n+1$ dimensional with the time component in the first coordinate. Note further that the Fourier coefficients are no longer time-dependent.

4.3.3.1 Energy Conservation

In the case of AFNO, the distinction between the learned and implicit energy cannot be made in the same way. Recall that both time and spatial derivatives are obtained simply by multiplying with a power of the respective wave number (times a power of i) in the Fourier domain, as can be seen in (4.13). Hence, the time derivative is learned as well as exact. The issue that arises when attempting to normalise AFNO is that the normalisation constant is necessarily local in time – i.e. specific to each hyperplane at a fixed value of t. The predictions, however, are inherently global, as the Fourier coefficients are shared for the entire space-time domain. Nevertheless, we propose to achieve *approximate* energy normalisation by normalising each time-slice in physical space analogously to Lemma 3 with the following factor:

$$c_5 = \sqrt{\frac{\mathcal{E}_0}{\mathcal{E}_t(u)}} \tag{4.15}$$

where the denominator is computed according to (4.14). While we would expect this normalisation to bring the energy close to the true energy, we cannot compute the final derivatives with (4.13) simply because there is no longer a coherent, global set of Fourier coefficients in space and time. The normalisation at t and $t+\delta$ may differ even for small δ invalidating any attempt to impute the time derivatives from the learned Fourier coefficients. This can be seen as a non-linear feedback loop similar to the ones discussed in Section 4.3.2, but it is harder to quantify. As a remedy, we approximate the true derivatives by applying a Fourier transform to the final output across the whole domain in physical space to once again obtain shared Fourier coefficients to compute the energies after normalisation. We may then compute the final energy with (4.14).

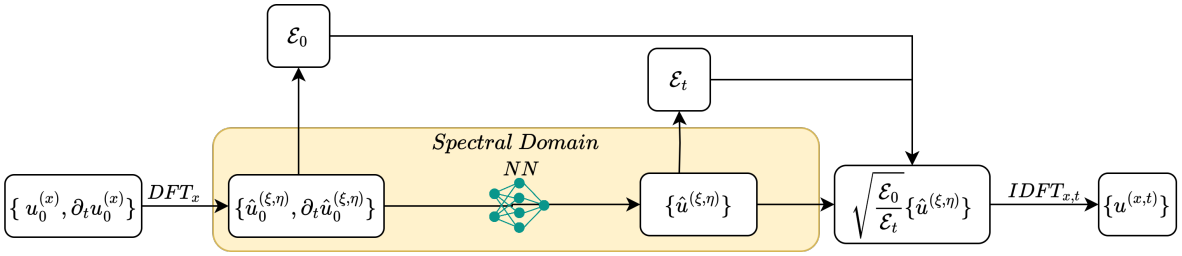

Figure 4.2: Normalised AFNO on the example of the wave equation.

4.4 Numerical Experiments

We will assess the performance of four DSO methods and two AFNO methods. The DSO methods are one vanilla baseline alongside an implementation of Lemma 4 and two implementations of Lemma 3, one based on the normalisation by the learned and one based on normalisation by the implicit energy. The AFNO methods consist of a baseline, as well as an implementation of the normalisation in (4.15).

4.4.1 Implementation

We implement the DSO with both trunk and branch having three hidden layers of size 128, and tanh activations. We set $p = 4$. The AFNO is implemented with a single neural network of the same dimensionality, but with swish activations [23]. The number of non-zero Fourier coefficients, determining the size of the output layer, is set to $d_x = 0.1 \times N_x$ for DSO and $d_x \times d_t$ with $d_x = 0.1 \times N_x$ and $d_t = 0.1 \times N_t$ for AFNO. We use Glorot normal initilisations [19] and optimise with Adam [20].

4.4.1.1 Learning Fourier Coefficients

To map between function representations in the spectral domain, we require the architectures to handle complex inputs and outputs. This is achieved by splitting input into real and imaginary parts, and assembling the complex values in a post-processing step. Note that in DSO it is sufficient to enable complex outputs in the branch network only. For real-valued functions we additionally must enforce Hermitian symmetry. It therefore suffices to learn $\lfloor d_x/2 \rfloor + 1$ coefficients in the DSO, while in the latter case it suffices to learn $(\lfloor d_t/2 \rfloor + 1)d_x$ coefficients in the AFNO.

4.4.1.2 Implementing Orthonormality of the Branch Matrix

For the implementation of Lemma 4 we once again rely on the QR composition, described in Section 3.3.1.1. The main difference is that we now use the matrix $\hat{\Sigma}$ in the weighted norm, and preprocess the output from the DeepONet with our linear operator $\hat{\mathcal{D}}$.

4.4.1.3 Obtaining Implicit Gradients

As before, the implicit gradients (relevant for all DSO methods) can be obtained by forward or backward-mode AD. As in this case the number of outputs is d_x which is significantly larger than the size relevant input (t) forward mode AD is the method of choice. Indeed, it is well known that the complexity of AD of a function $f : \mathbb{R}^k \to \mathbb{R}^s$ with evaluation cost(f) is given by $\mathcal{O}(k \times \text{cost}(f))$ in forward mode and $\mathcal{O}(s \times \text{cost}(f))$ in reverse mode (cf. [22]). While reverse mode AD is readily available through `torch`, we implemented the forward mode by augmenting the forward pass through all layers and transformations to keep track of the gradients.

4.4.2 Data

We generate training data for the wave equation using a Gaussian process with a symmetric kernel, which ensures periodicity. Let L_x and L_t be the desired periodicity in space and time respectively, and let N_x and N_t be the chosen resolution in space and time. Denote by $x_1, \ldots, x_{N_x} \in [0, L_x]$ the points corresponding to the spatial resolution. We further define a reduced spatial resolution M_x, with $M_x < N_x$ which we use to enforce a lower number of non-zero Fourier coefficients. Denote the corresponding points by $\tilde{x}_1, \ldots, \tilde{x}_{N_m}$. Following [24] which follows [25], we define a periodic kernel for the Gaussian process by

$$k_{\text{per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2\left(\pi(x - x')/p\right)}{s^2}\right)$$

where σ^2 is the variance controlling the amplitude, s is the lengthscale controlling smoothness and p is the period, which we set to L_x. For some small ϵ, we construct a covariance matrix

$$K_{ij} = k_{\text{per}}(\tilde{x}_i, \tilde{x}_j) + \epsilon\delta_{ij}$$

which we use to sample M_x-dimensional vectors from Gaussian distributions as

$$\begin{aligned} \boldsymbol{u}_0 &\sim \mathcal{N}(0, \boldsymbol{K}), & \boldsymbol{v}_0 &\sim \mathcal{N}(0, \boldsymbol{K}) \\ \hat{\boldsymbol{u}}_0 &= \mathcal{F}(\boldsymbol{u}_0), & \hat{\boldsymbol{v}}_0 &= \mathcal{F}(\boldsymbol{v}_0) \end{aligned}$$

Finally, we zero out the constant modes (at wave number zero) to ensure no drift across space or time, enforce Hermitian symmetry around the zero mode and transform back to the final N_x resolution by zero-padding additional $N_x - M_x$ modes. That is, the final initial conditions are given by:

$$u(0, \mathbf{x}) = \mathcal{F}^{-1}[\hat{\boldsymbol{u}}_0^{\text{padded}}] \cdot \frac{N_x}{M_x} \in \mathbb{R}^{N_x}, \quad v(0, \mathbf{x}) = \mathcal{F}^{-1}[\hat{\boldsymbol{v}}_0^{\text{padded}}] \cdot \frac{N_x}{M_x} \in \mathbb{R}^{N_x}.$$

We choose $s = 0.1$ and $\sigma = 1$. We evolve these up to time L_t using a fourth-order Runge-Kutta scheme [26], and test for periodicity. We reduce the discretisation to $N_x = N_t = 199$

grid points across a domain of width $L_x = 50$ and length $L_t = 5$. We generate training, validation and test sets of equal sizes, each with $N = 500$ initial conditions. The validation set is used for model selection.

4.4.3 Results

The main results are shown in Table 4.2. The table shows the MSE on the test set, as well as the MSE for the learned and implicit energies, except for AFNO based methods, where implicit energies are not available as per the discussion in Section 4.3.3.1.

Table 4.2: Test MSE (in 1×10^{-4}) and learned and implicit energy MSE on the test set (in 1×10^{0}) for DSO and AFNO models.

			Test Energy MSE	
		Test MSE	**Learned**	**Implicit**
DSO	**Vanilla**	8394.249	7366.470	7397.481
	QR	2177.627	**0.000**	5.467
	Norm.	2059.043	**0.000**	4.743
	Imp. Norm.	2990.589	2.214	2.431
AFNO	**Vanilla**	0.694	–	2274.808
	Norm.	**0.004**	–	**1.551**

We first note the stark differences in test set MSE between DSO and AFNO methods. At first glance, these may appear unreasonably large, but they are largely explained by large generalisation errors in the DSO methods evident from Figure 4.3. For the DSO, all normalisation methods achieve a worse fit on the training set throughout but attain a significantly reduced generalisation gap. This leads to an approximately four fold reduction in test set loss for all three methods, with the QR and learned normalisation achieving the lowest losses. This justifies the use of this post-processing technique, despite its limitations discussed in Section 4.3.2. This is contrary to the case of the harmonic oscillator (cf. Table 3.2), where the implicit normalisation performs best. However, as would be expected despite non-linearities preventing perfect normalisation, the implicit normalisation method still achieves the lowest error on the implicit energy. It should be noted that without any normalisation, the DSO method does not achieve energy conservation on the test set whatsoever. Similarly, the vanilla AFNO cannot ensure energy conservation on the test set, despite a low generalisation error as demonstrated by a small gap between the loss curves on the training set and the validation set (cf. Figure 4.3). The normalised AFNO, however, achieves almost perfect fit even on the test set and the lowest final implicit energy.

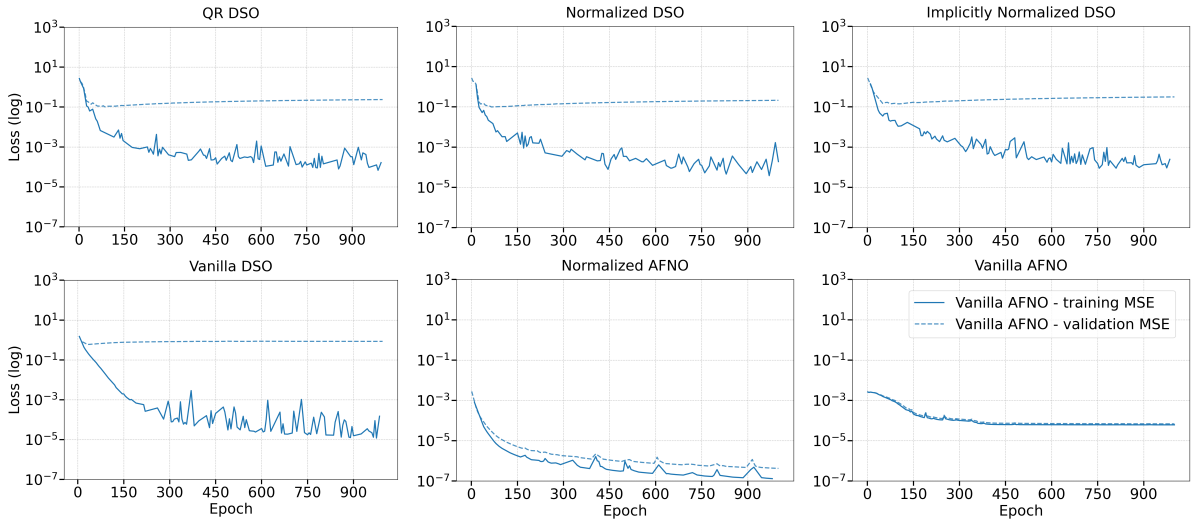

Figure 4.3: Training and validation MSE: QR DSO, Normalised DSO and Implicitly Normalised DSO, Vanilla DSO, Normalised AFNO and Vanilla AFNO.

We chose an arbitrary sample from the test to illustrate the predicted dynamics under each model in Figure 4.5 and Figure 4.6. Note that heatmap scales are shared, and the ground truth can be found in Figure 4.4. We see that all models predict the wave evolution reasonably well, but all normalised models show a strong reduction in the errors, reflecting the results in Table 4.2. Out of the DSO methods, it is apparent that the normalisation based on exact preservation of learned energy give the best prediction. Confirming the exceptional performance of the normalised AFNO (Figure 4.8), the absolute error for its prediction on the selected test set sample is lower than for the vanilla method by an order of magnitude. The time slices (rightmost image) confirm that the error is negligible.

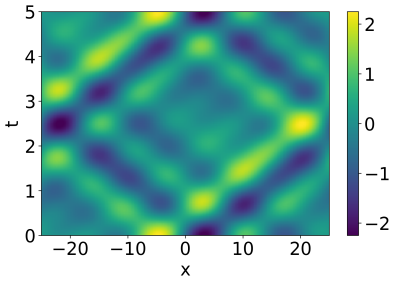

Figure 4.4: Ground truth evolution of selected test set example.

We also plot the evolution of the energy over time for the same example in Figure 4.7 and Figure 4.8. Alongside the total energy we plot the $\partial_x u$ and $\partial_t u$ components from Equation 4.3. For the vanilla DSO and AFNO methods, we see that that the predictions consistently underestimate the energy across time. This is true for both learned and implicit energy, which broadly align on average. This underestimation seems to be caused by underestimations in both the $\partial_x u$ and $\partial_t u$ components. This could be explained by low-amplitude predictions, as can be seen in the bottom plots of Figure 4.5 and

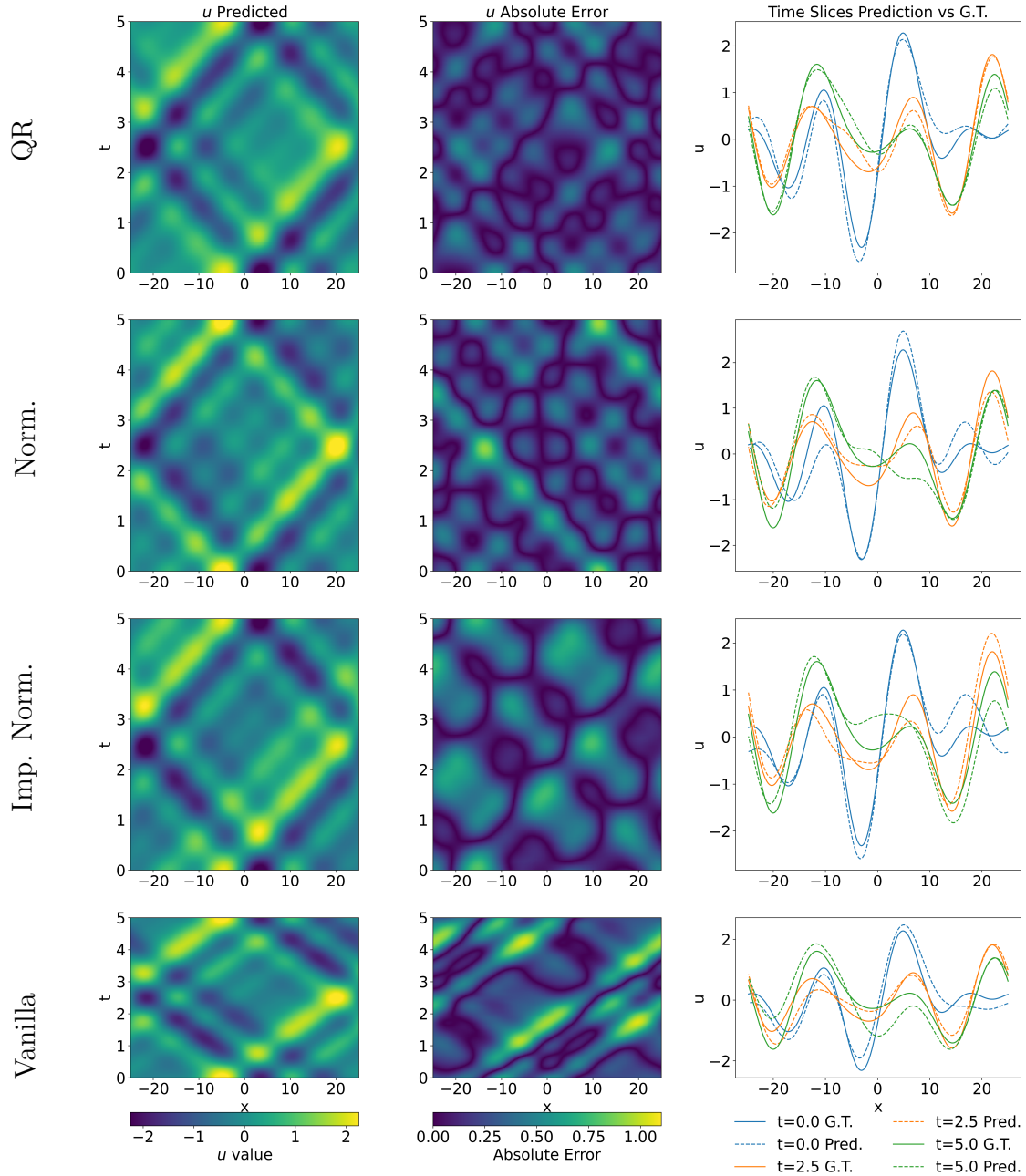

Figure 4.5: Prediction, absolute prediction error and prediction time slices of DSO models on selected test set initial condition.

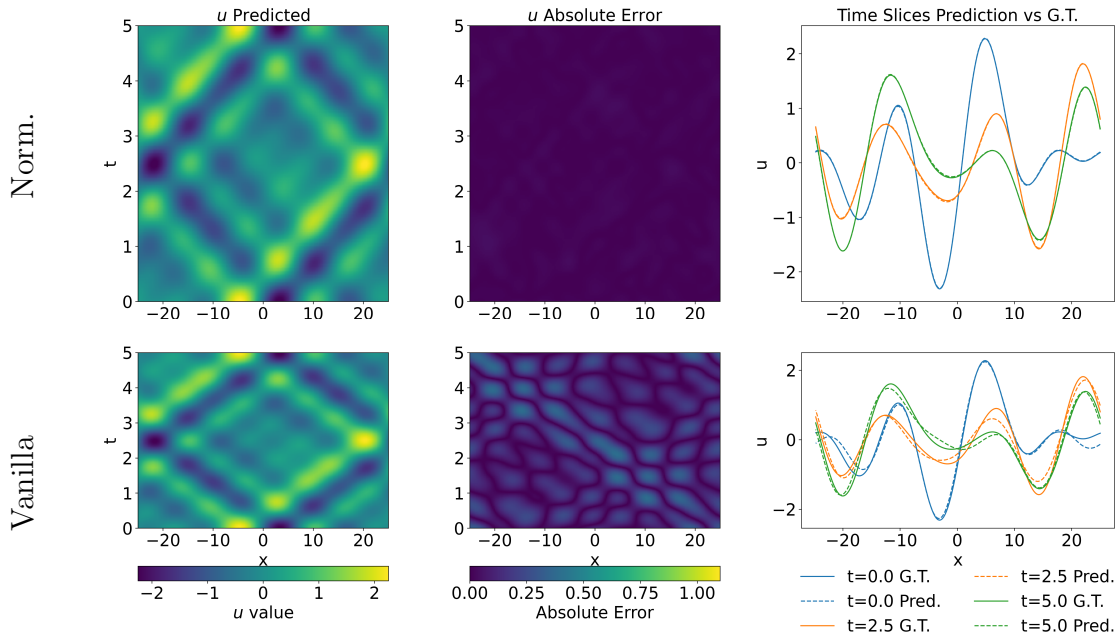

Figure 4.6: Prediction, absolute prediction error and prediction time slices of AFNO models on selected test set initial condition.

Figure 4.6. Meanwhile, all methods designed for energy conservation regularly over- and underestimate the true $\partial_x u$ and $\partial_t u$ components, leading to total implicit energies that meander around the true energy. This can be explained by the the conservation methods constraining the set of feasible models to those with sufficiently 'energetic' amplitudes – in the DSO this leads to higher training losses (as seen in Figure 4.3) but enforces predictions with more realistic distributions. The effect in AFNO appears to be more favourable training dynamics and faster convergence. Of course, learned energies are conserved exactly in the respective DSO methods ('QR' and 'Norm'). For the DSO model normalising with the implicit energy, we find that the final implicit energy is close to the truth, with some outliers. These could be explained by higher order derivatives in the non-linear feedback (cf. Section 4.3.2) which do not occur when normalising by a factor that does not already contain time derivatives.

Finally, the normalised AFNO achieves conservation of exact ('implicit') energy within an error that can be expected when working with imperfect (e.g. slightly aperiodic) training data and computations in the spectral domain. In the leftmost plots Figure 4.8 we include for comparison a 'Fourier Ground Truth' which is computed by performing a Fourier transform on the simulated test data, and computing the true energy based on (4.14). We see that the final energy computed based on our predictions moves in the same range.

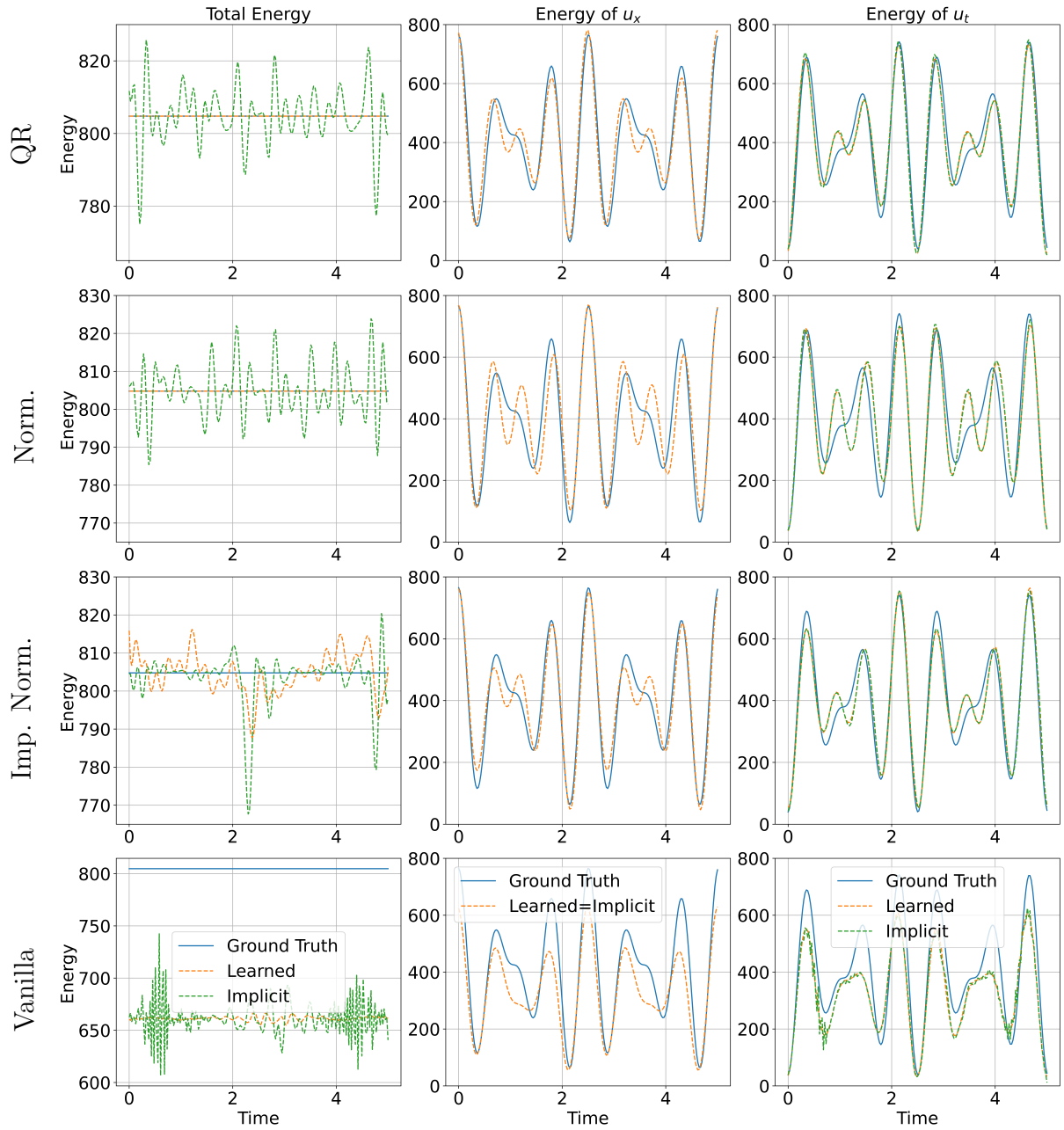

Figure 4.7: Total energy, and u_x and u_t energy components for the DSO models. Ground truth, learned energy and predicted energy.

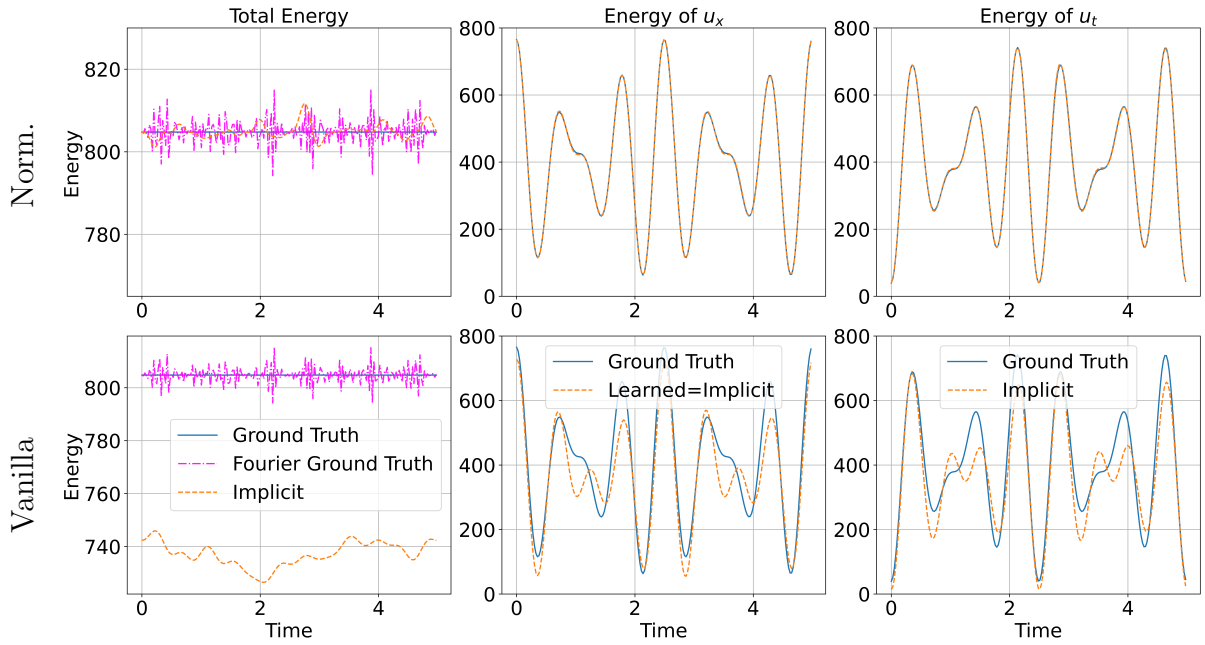

Figure 4.8: Total energy, and u_x and u_t energy components for AFNO models. Ground truth, learned energy and predicted energy. Additionally, energy of simulated test data computed in Fourier space for error scale comparison ('Fourier Ground Truth').

Chapter 5

Future Work and Conclusion

5.1 Future Work

One important direction for future work is to consider how more general conservation laws, which may not be quadratic, can be preserved. Our best model, the normalised AFNO, has demonstrated strong capabilities in conserving the energy of the wave equation. However, despite its strong performance, this model comes with the particular restriction that the solution be periodic in time. This is a strong restriction, since many real-world dynamics, such as atmospheric processes [27], do not exhibit exact periodicity. One approach for tackling this could be the method of Fourier continuation, as already used in several existing methods [28]. The task of adapting the normalisation method to solutions that are aperiodic in space is significantly more challenging, since Fourier continuations to a larger 'artificial' domain would invalidate the computation of the integral quantity in the spectral domain. Another interesting approach to overcoming the periodicity assumption would be the use of Chebychev polynomials as the basis for the latent space (cf. [13]). We have only briefly touched on the non-linearities emerging from the normalisation techniques in the DSO. Future work should attempt to rectify this with iterative numerical projection schemes, similar to [17] or efficiently in Fourier space as in [18]. Lastly, we hypothesise that the enforcement of conversation law also helps with handling distributional shifts, but further experiments are necessary to test this.

5.2 Conclusion

We have shown that by adjusting architectures to conserve a broad class of quadratic conservation laws satisfied by the equation of interest, models for learning PDE dynamics can be forced to behave closer to the physical truth. In doing so, we also proposed an innovative combination of DeepONet and spectral methods which, as opposed to the fully spectral AFNO method, is not restricted to learning time-periodic dynamics. Overall, the inductive hypothesis imposed by conservation laws improves training dynamics and helps all considered methods to generalise better from training data to unseen test datasets.

Appendix A

Code

This paper is accompanied by a comprehensive code base. Significant work has been channeled into implementing all methods and experiments presented herein, as well as data generation. Code generation tools (OpenAI GPT-4 [29], Anthropic Claude Sonnet 3.5 [30]) were used in a purely assistive manner, in line with the guidance from the supervisors. All code is available on an anonymous **GitHub**.

Bibliography

[1] Lu Lu et al. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature Machine Intelligence* 3.3 (Mar. 2021), pp. 218–229. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00302-5. URL: http://dx.doi.org/10.1038/s42256-021-00302-5.

[2] Zongyi Li et al. "Fourier Neural Operator for Parametric Partial Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=c8P9NQVtmnO.

[3] Thorsten Kurth et al. "FourCastNet: Accelerating Global High-Resolution Weather Forecasting Using Adaptive Fourier Neural Operators". In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC '23. Davos, Switzerland: Association for Computing Machinery, 2023. ISBN: 9798400701900. DOI: 10.1145/3592979.3593412. URL: https://doi.org/10.1145/3592979.3593412.

[4] Ahmad Peyvan and Varun Kumar. "Fusion DeepONet: A Data-Efficient Neural Operator for Geometry-Dependent Hypersonic Flows on Arbitrary Grids". In: *CoRR* abs/2501.01934 (2025). DOI: 10.48550/ARXIV.2501.01934. arXiv: 2501.01934. URL: https://doi.org/10.48550/arXiv.2501.01934.

[5] Elias M. Stein and Guido Weiss. *Introduction to Fourier Analysis on Euclidean Spaces (PMS-32)*. Princeton University Press, 1971, pp. 175–185. ISBN: 9780691080789. URL: http://www.jstor.org/stable/j.ctt1bpm9w6 (visited on 04/25/2025).

[6] Alan V. (Alan Victor) Oppenheim, Ronald W. (Ronald William) Schafer, and John R. Buck. *Discrete-time signal processing*. eng. 2nd ed. Prentice Hall international editions. Upper Saddle River, N.J. ; Prentice Hall, 1999. ISBN: 0137549202.

[7] James Cooley and John Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Mathematics of Computation* 19.90 (1965), pp. 297–301.

[8] Nicolas Boullé and Alex Townsend. "A Mathematical Guide to Operator Learning". In: (2023). arXiv: 2312.14688 [math.NA]. URL: https://arxiv.org/abs/2312.14688.

[9] T Chen and H Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems." eng. In: *IEEE Trans Neural Netw* 6.4 (1995), pp. 911–917. ISSN: 1045-9227 (Print); 1045-9227 (Linking). DOI: 10.1109/72.392253.

[10] T. Chen and H. Chen. "Approximations of continuous functionals by neural networks with application to dynamic systems". In: *IEEE Transactions on Neural Networks* 4.6 (1993), pp. 910–918. DOI: 10.1109/72.286886.

[11] Ben Poole et al. "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/148510031349642de5ca0c544f31b2ef-Paper.pdf.

[12] Maithra Raghu et al. "On the Expressive Power of Deep Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, June 2017, pp. 2847–2854. URL: https://proceedings.mlr.press/v70/raghu17a.html.

[13] Lloyd N. Trefethen. *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations*. Department of Computer Science and Center for Applied Mathematics, Upson Hall Cornell University, Ithaca, NY: Cornell University, 1996, pp. 235–259, 262–300.

[14] Lawrence C. Evans. *Partial Differential Equations*. 2nd ed. Vol. 19. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 2010, pp. 307–314, 660.

[15] Ning Liu et al. "Harnessing the power of neural operators with automatically encoded conservation laws". In: *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria: JMLR.org, 2024.

[16] Derek Hansen et al. "Learning physical models that can respect conservation laws". In: *Physica D: Nonlinear Phenomena* 457 (2024), p. 133952. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2023.133952. URL: https://www.sciencedirect.com/science/article/pii/S0167278923003068.

[17] Elsa Cardoso-Bihlo and Alex Bihlo. "Exactly conservative physics-informed neural networks and deep operator networks for dynamical systems". In: *Neural Networks* 181 (2025), p. 106826. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2024.106826. URL: https://www.sciencedirect.com/science/article/pii/S0893608024007500.

[18] Valentin Duruisseaux et al. "Towards Enforcing Hard Physics Constraints in Operator Learning Frameworks". In: *ICML 2024 AI for Science Workshop*. 2024. URL: https://openreview.net/forum?id=Zvxm14Rd1F.

[19] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[20] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[21] Gene H. Golub and Charles F. Van Loan. *Matrix Computations - 4th Edition*. Philadelphia, PA: Johns Hopkins University Press, 2013, pp. 254–256. DOI: 10.1137/1.9781421407944. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781421407944. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781421407944.

[22] Alan Edelman and Steven G. Johnson. *Forward and Reverse-Mode Automatic Differentiation*. 18.S096: Matrix Calculus for Machine Learning and Beyond, Independent Activities Period (IAP) 2023. Massachusetts Institute of Technology: MIT OpenCourseWare. License: CC BY-NC-SA. 2023. URL: https://ocw.mit.edu/courses/18-s096-matrix-calculus-for-machine-learning-and-beyond-january-iap-2023/mit18_s096iap23_lec08.pdf.

[23] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE]. URL: https://arxiv.org/abs/1710.05941.

[24] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006, p. 92. ISBN: 9780262182539. URL: http://www.gaussianprocess.org/gpml/.

[25] David J. C. MacKay. "Comparison of Approximate Methods for Handling Hyperparameters". In: *Neural Computation* 11.5 (1999), p. 110. DOI: 10.1162/089976699300016274.

[26] Wilhelm Kutta. "Beitrag zur Integration der Differentialgleichungen". In: *Zeitschrift für Mathematik und Physik* 46 (1901), pp. 435–453.

[27] M.A. Khodkar and Pedram Hassanzadeh. "A data-driven, physics-informed framework for forecasting the spatiotemporal evolution of chaotic dynamics with nonlinearities modeled as exogenous forcings". In: *Journal of Computational Physics* 440 (2021), p. 110412. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110412. URL: https://www.sciencedirect.com/science/article/pii/S0021999121003077.

[28] Haydn Maust et al. *Fourier Continuation for Exact Derivative Computation in Physics-Informed Neural Operators*. 2022. arXiv: 2211.15960 [cs.LG]. URL: https://arxiv.org/abs/2211.15960.

[29] OpenAI Collaboration. *GPT-4 Technical Report*. https://arxiv.org/abs/2303.08774. Accessed: 2025-04-24. 2024. arXiv: 2303.08774 [cs.CL].

[30] Anthropic Collaboration. *Claude 3.5 Sonnet Model Card Addendum*. https://www-cdn.anthropic.com/fed9cc193a14b84131812372d8d5857f8f304c52/Model_Card_Claude_3_Addendum.pdf. Accessed: 2025-04-24. 2024.