Advances in Symbolic Regression: From Generalized Formulation to Density Estimation and Inverse Problem

by

Tony Tohme

B.Eng., American University of Beirut, 2018
S.M., Massachusetts Institute of Technology, 2020

Submitted to the Department of Mechanical Engineering
and the Center for Computational Science & Engineering
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by: Tony Tohme
Department of Mechanical Engineering
Center for Computational Science & Engineering
May 3, 2024

Certified by: Kamal Youcef-Toumi
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by: Nicolas G. Hadjiconstantinou
Professor of Mechanical Engineering
Graduate Officer, Department of Mechanical Engineering
Co-Director, Center for Computational Science & Engineering

Accepted by: Youssef M. Marzouk
Professor of Aeronautics and Astronautics
Co-Director, Center for Computational Science & Engineering

Advances in Symbolic Regression: From Generalized Formulation to Density Estimation and Inverse Problem

by

Tony Tohme

Submitted to the Department of Mechanical Engineering
and the Center for Computational Science & Engineering
on May 3, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

In this thesis, we explore the field of Symbolic Regression (SR), a middle ground between simple linear regression and complex inscrutable black box regressors such as neural networks. In essence, SR searches the space of mathematical expressions to find a model that best captures the relationship between inputs and outputs of a given dataset. While SR has not gained mainstream popularity due to its computational intricacy and reliance on heuristics, its potential for generating explicit, concise, and interpretable mathematical models deserves further attention. This work presents a series of advancements in Symbolic Regression, extending its applicability and demonstrating its potential across diverse domains and problem settings.

Initially, we introduce GSR, a Generalized Symbolic Regression method that redefines the traditional SR optimization problem to discover analytical mappings from the input space to a transformed output space. The proposed GSR approach achieves promising performance compared to existing SR methods across established benchmark datasets, as well as a more challenging dataset introduced in this study, called SymSet.

Next, we delve into the task of recovering underlying partial differential equations (PDEs) from data through the use of the adjoint method. We begin by considering a family of parameterized PDEs encompassing linear, nonlinear, and spatial derivative candidate terms. We then formulate a PDE-constrained optimization problem aimed at minimizing the error of the PDE solution from data, and elegantly derive the corresponding adjoint equations. We showcase the efficacy of the proposed approach in selecting the appropriate candidate terms, thereby discovering the governing PDEs from data. We also compare its performance with a commonly employed method for PDE discovery.

Furthermore, we introduce MESSY Estimation, a Maximum-Entropy based Stochastic and Symbolic densitY estimation method. The proposed approach infers probability density functions symbolically from samples by leveraging the Maximum Entropy Distribution (MED) principle. We uncover three key contributions: (i) the Lagrange multipliers, inherent in the MED ansatz, can be efficiently computed by simply solving a linear system of equations, (ii) the density recovery task is enhanced through matching more unconventional low-order (symbolic) moments, rather than necessarily matching higher-order (raw) moments, and (iii) the proposed symbolic density estimation framework leads to increased interpretability and better conditioning.

Finally, we introduce ISR, an Invertible Symbolic Regression (ISR) approach, which bridges the concepts of SR and invertible maps. Specifically, ISR seamlessly combines the principles of Invertible Neural Networks (INNs) and Equation Learner (EQL), a neural network-based symbolic architecture for function learning. Demonstrating its versatility, ISR also serves as a symbolic normalizing flow for density estimation tasks. Additionally, we showcase its applicability in solving inverse problems, including a benchmark inverse kinematics problem, and notably, a geoacoustic inversion problem in oceanography aimed at inferring posterior distributions of underlying seabed parameters from acoustic signals.

The diverse findings of this thesis not only contribute to advancing the field of Symbolic Regression, but also underscore its versatility and potential across various domains. A shift to explicit symbolic models, as demonstrated in this thesis, could unveil hidden patterns within the plethora of datasets available today, offering new insights and directions in the evolving field of machine learning and data analysis.

Thesis supervisor: Kamal Youcef-Toumi
Title: Professor of Mechanical Engineering

This thesis is dedicated to my family

Acknowledgments

First and foremost, I would like to thank God for providing me with the strength, guidance, and perseverance to complete this thesis.

I extend my sincerest appreciation to my advisor, Kamal Youcef-Toumi, for believing in me and inspiring me. His invaluable support and mentorship have been instrumental in shaping my research and academic growth.

I would also like to thank Youssef Marzouk and Pierre Lermusiaux for serving on my committee and for their insightful feedback and discussions.

I was fortunate to have worked with great collaborators, including Mohsen Sadr, Mohammad Javad Khojasteh, Kevin Vanslette, Dehong Liu, Nicolas Hadjiconstantinou, and Florian Meyer. A special thanks to Mohsen Sadr, with whom I have had fruitful collaborations that significantly enriched my research experience. I honestly owe him tremendously. Also, thanks to my labmates for making the lab a fun place to work and learn.

My research was generously supported by the MathWorks Engineering Fellowship, the Al Ahdab Fellowship, and the Center for Complex Engineering Systems (CCES) at King Abdulaziz City for Science and Technology (KACST) and the Massachusetts Institute of Technology (MIT).

I gratefully acknowledge Professor Gilbert Strang and Associate Provost Philip Khoury, whose support and encouragement played a pivotal role throughout my journey at MIT.

To my friends in Boston – Alaa Khaddaj, Souhail and Judy Halaby, Mohamad Alrished, Bachar Al Hady, Rabih Salhab, Abhishek Patkar, Evan Massaro, Majed Almubarak, Rami Khaddaj, Abdullah Alomar, Hussein Mozannar, Mahmoud Ramadan, Georges and Christina Saade, Elias El Khoury, and others – thank you for always having my back and making Boston feel like a second home.

Special thanks to my friends – Joseph Chehade, Christelle Younes, Charbel Abdo, Tariq El Zein, Guitta Saade, Ahmad Noweiry, and Mohamad Rida Rammal – for always being there for me and celebrating every success with me.

Words will never be enough to express my gratitude and love to my parents, Mike and Nicole. Mom, your presence and support during this adventure were pivotal. You helped me navigate through my challenges with strength, and without you, I simply wouldn't be here now. To my aunts, Simone, Mona, and Arlette, my uncles, and my grandparents, your unconditional love and encouragement have also been the pillars of my strength.

I lovingly dedicate this thesis to my grandmother in heaven, Salma. Your memory has been a constant source of inspiration and strength.

Lastly, I am profoundly grateful to my wonderful wife, Maria, for her unwavering support and love throughout this journey. Your encouragement and belief in me have been my greatest motivation. I look forward to the next chapter of our lives together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the era of big data, the importance of computing, artificial intelligence (AI), machine learning (ML), and data analysis cannot be overstated. These technologies have revolutionized numerous fields, driving advancements in science, engineering, medicine, and beyond. AI and ML, in particular, have seen tremendous progress, with cutting-edge developments such as transformers and generative AI models pushing the boundaries of what machines can achieve. Transformers, introduced in [1], have become the foundation for many state-of-the-art models, including BERT [2], GPT [3], and T5 [4], due to their ability to handle long-range dependencies in data and perform exceptionally well on tasks such as natural language processing. Generative AI, epitomized by models like DALL-E [5] and ChatGPT [6], has demonstrated remarkable capabilities in generating human-like text and images, showcasing the power and versatility of modern AI systems.

Despite these advancements, a critical challenge persists: the interpretability of machine learning models. As ML models become more complex, their decision-making processes often become opaque, leading to the notorious "black box" problem. This lack of transparency can hinder trust and accountability, particularly in high-stakes domains such as healthcare, finance, and autonomous systems. Consequently, there is a growing demand for models that are not only accurate but also interpretable. Interpretability allows for greater confidence in model predictions, supporting more reliable and ethical deployment of AI systems.

This is where Symbolic Regression (SR) finds its niche. SR stands out by generating explicit mathematical expressions that describe the relationships within data, offering a level of interpretability that is often absent in traditional ML models. SR searches the space of mathematical expressions to find models that best capture the underlying patterns in the data, thus providing clear and concise representations of these relationships. This characteristic makes SR particularly valuable in scientific research, where the ability to derive interpretable models aligns with the core objective of uncovering fundamental laws and principles.

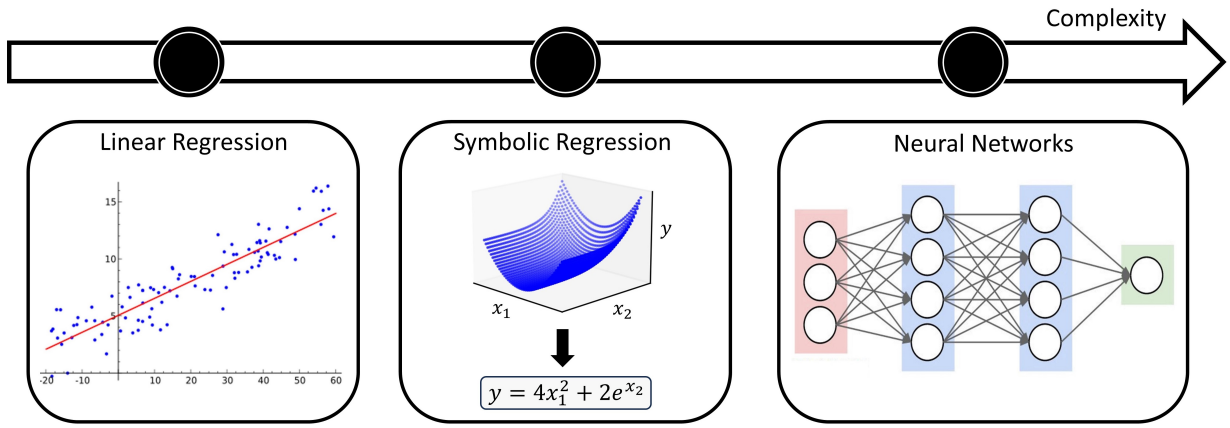


Figure 1.1: Complexity spectrum of regression models.

1.1 Background and Overview

Machine learning (ML) fundamentally revolves around the concept of *learning*. Learning problems in ML are typically categorized into three main types: (i) supervised learning, which involves training a model to map input examples to a target variable (e.g. regression and classification), (ii) unsupervised learning, which focuses on identifying patterns and relationships in datasets without labeled responses (e.g. clustering and dimensionality reduction), and (iii) reinforcement learning, where an agent interacts with an environment and learns to make decisions, using feedback from its actions to maximize a cumulative reward. In this thesis, we focus on the regression task.

Regression models range from simple linear regression to complex, often inscrutable black box neural networks (see Figure 1.1). While linear regression offers simplicity and interpretability, it often lacks the flexibility to model complex nonlinear relationships. On the other hand, neural networks, with their capacity to capture intricate patterns, often sacrifice transparency. This thesis explores Symbolic Regression (SR), a methodology that represents a middle ground between these two extremes and which has not received commensurate scholarly attention. In essence, SR searches the space of mathematical expressions to find a model that best captures the relationship between inputs and outputs of a given dataset. Unlike traditional regression models that fit data to predefined model structures, SR discovers the underlying mathematical structures themselves, offering unparalleled insight into the intrinsic relationships within the data. This pursuit aligns with the core scientific endeavor of formulating theories and laws that explain observable phenomena.

Historically, the concept of SR is rooted in the advent of genetic algorithms and genetic programming in the late 20^{th} century, where the idea was to simulate the process of natural evolution to solve optimization problems, including the discovery of mathematical expressions. This marked a significant departure from conventional model-fitting practices, opening new avenues for automated scientific discovery. Over the years, SR has been instrumental in identifying complex relationships in fields as diverse as physics, where it has been used to derive new formulations of physical laws, and biology, where it has helped model the intricate dynamics of biological systems. The historical journey of SR from a novel computational

technique to a powerful tool for scientific discovery underscores its transformative potential in unraveling the complexities of the natural world.

In recent years, the methodology underpinning Symbolic Regression has witnessed a remarkable diversification. Traditionally anchored in genetic algorithms and programming, the field has seen an infusion of novel computational approaches that have enriched the SR landscape. Techniques such as mixed integer optimization, neural network integration, and the application of transformer models have opened new pathways for SR applications. This broadening of techniques has enhanced the capability of SR to operate within more complex and computationally demanding environments, expanding its utility, and enabling researchers to tackle a broader array of problems with increased sophistication and efficacy.

While SR has not gained mainstream popularity due to its computational intricacy and reliance on heuristics, its potential for generating explicit, concise, and interpretable mathematical models deserves further attention. This work presents a series of advancements in Symbolic Regression, extending its applicability and demonstrating its potential across diverse domains and problem settings.

1.2 Contributions

Traditionally, SR has been predominantly utilized as a tool for fitting data to mathematical models, thereby facilitating regression tasks across various scientific and engineering disciplines. However, the inherent potential of SR extends far beyond mere curve fitting. In this thesis, we aim to broaden the horizons of SR by applying it to a variety of domains, including density estimation, inverse problems, and the discovery of partial differential equations.

In this section, we outline the key contributions of each chapter.

Generalized Symbolic Regression.
In Chapter 2, we present GSR, a Generalized Symbolic Regression approach, by modifying the conventional SR optimization problem formulation, while keeping the main SR objective intact. In GSR, we infer mathematical relationships between the independent variables and some transformation of the target variable. We constrain our search space to a weighted sum of basis functions, and propose a genetic programming approach with a matrix-based encoding scheme. We show that our GSR method is competitive with strong SR benchmark methods, achieving promising experimental performance on the well-known SR benchmark problem sets. Finally, we highlight the strengths of GSR by introducing SymSet, a new SR benchmark set which is more challenging relative to the existing benchmarks.

Data-Driven Discovery of Partial Differential Equations via the Adjoint Method.
In Chapter 3, we present an adjoint-based method for discovering the underlying governing partial differential equations (PDEs) given data. The idea is to consider a parameterized PDE in a general form, and formulate a PDE-constrained optimization problem aimed at minimizing the error of the PDE solution from data. Using variational calculus, we obtain an evolution equation for the Lagrange multipliers (adjoint equations) allowing us to compute the gradient of the objective function with respect to the parameters of PDEs given data in a straightforward manner. In particular, we consider a family of parameterized PDEs

encompassing linear, nonlinear, and spatial derivative candidate terms, and elegantly derive the corresponding adjoint equations. We show the efficacy of the proposed approach in identifying the form of the PDE up to machine accuracy, enabling the accurate discovery of PDEs from data. We also compare its performance with the famous PDE Functional Identification of Nonlinear Dynamics method known as PDE-FIND [7], on both smooth and noisy data. Even though the proposed adjoint method relies on forward/backward solvers, it outperforms PDE-FIND for large data sets thanks to the analytic expressions for gradients of the cost function with respect to each PDE parameter.

Maximum-Entropy based Stochastic and Symbolic densitY Estimation.
In Chapter 4, we introduce MESSY Estimation, a Maximum-Entropy based Stochastic and Symbolic densitY Estimation method. The proposed approach recovers probability density functions symbolically from samples using moments of a Gradient flow in which the ansatz serves as the driving force. In particular, we construct a gradient-based drift-diffusion process that connects samples of the unknown distribution function to a guess symbolic expression. We then show that when the guess distribution has the maximum entropy form, the parameters of this distribution can be found efficiently by solving a linear system of equations constructed using the moments of the provided samples. Furthermore, we use Symbolic regression to explore the space of smooth functions and find optimal basis functions for the exponent of the maximum entropy functional leading to good conditioning. The cost of the proposed method for each set of selected basis functions is linear with the number of samples and quadratic with the number of basis functions. However, the underlying acceptance/rejection procedure for finding optimal and well-conditioned bases adds to the computational cost. We validate the proposed MESSY estimation method against other benchmark methods for the case of a bi-modal and a discontinuous density, as well as a density at the limit of physical realizability. We find that the addition of a symbolic search for basis functions improves the accuracy of the estimation at a reasonable additional computational cost. Our results suggest that the proposed method outperforms existing density recovery methods in the limit of a small to moderate number of samples by providing a low-bias and tractable symbolic description of the unknown density at a reasonable computational cost.

ISR: Invertible Symbolic Regression.
In Chapter 5, we introduce an Invertible Symbolic Regression (ISR) method. It is a machine learning technique that generates analytical relationships between inputs and outputs of a given dataset via invertible maps (or architectures). The proposed ISR method naturally combines the principles of Invertible Neural Networks (INNs) and Equation Learner (EQL), a neural network-based symbolic architecture for function learning. In particular, we transform the affine coupling blocks of INNs into a symbolic framework, resulting in an end-to-end differentiable symbolic invertible architecture that allows for efficient gradient-based learning. The proposed ISR framework also relies on sparsity promoting regularization, allowing the discovery of concise and interpretable invertible expressions. We show that ISR can serve as a (symbolic) normalizing flow for density estimation tasks. Furthermore, we highlight its practical applicability in solving inverse problems, including a benchmark inverse kinematics problem, and notably, a geoacoustic inversion problem in oceanography aimed at inferring posterior distributions of underlying seabed parameters from acoustic signals.

1.3 Thesis Outline

The remainder of the thesis is organized as follows.

Chapter 2 presents a Generalized Symbolic Regression (GSR) method that redefines the traditional SR optimization problem to discover analytical mappings from the input space to a transformed output space. The material presented in this chapter is based on joint work with Dehong Liu and Kamal Youcef-Toumi [8].

Chapter 3 proposes an adjoint-based method for discovering governing laws/equations from data. The material presented in this chapter is based on joint work with Mohsen Sadr and Kamal Youcef-Toumi [9].

Chapter 4 introduces a Maximum-Entropy based Stochastic and Symbolic densitY (MESSY) Estimation method. The proposed approach infers probability density functions symbolically from samples by leveraging the Maximum Entropy Distribution (MED) principle. The material presented in this chapter is based on joint work with Mohsen Sadr, Kamal Youcef-Toumi, and Nicolas G. Hadjiconstantinou [10].

Chapter 5 introduces ISR, an Invertible Symbolic Regression method, which bridges the concepts of SR and invertible maps to produce invertible analytical models. The material presented in this chapter is based on joint work with Mohammad Javad Khojasteh, Mohsen Sadr, Florian Meyer, and Kamal Youcef-Toumi [11].

Each chapter is fairly self-contained, allowing readers to navigate the chapters independently. To maintain conciseness within the main body of the text, supplementary materials related to each chapter are provided in the appendices.

Finally, **Chapter** 6 summarizes the results of this work, and discusses some ideas for future research.

Chapter 2

GSR: A Generalized Symbolic Regression Approach

In this chapter, we present GSR, a Generalized Symbolic Regression approach, by modifying the conventional SR optimization problem formulation, while keeping the main SR objective intact. In GSR, we infer mathematical relationships between the independent variables and some transformation of the target variable. We constrain our search space to a weighted sum of basis functions, and propose a genetic programming approach with a matrix-based encoding scheme. We show that our GSR method is competitive with strong SR benchmark methods, achieving promising experimental performance on the well-known SR benchmark problem sets. Finally, we highlight the strengths of GSR by introducing SymSet, a new SR benchmark set which is more challenging relative to the existing benchmarks.

2.1 Introduction

Symbolic regression (SR) aims to find a mathematical expression that best describes the relationship between the independent variables and the target (or dependent) variable based on a given dataset. By inspecting the resulting expression, we may be able to identify nontrivial relations and/or physical laws which can provide more insight into the system represented by the given dataset. SR has gained tremendous interest and attention from researchers over the years for many reasons. First, many rules and laws in natural sciences (e.g. in physical and dynamical systems [12], [13]) are accurately represented by simple analytical equations (which can be explicit [14] or implicit [15], [16]). Second, in contrast to neural networks that involve complex input-output mapping, and hence are often treated as black boxes which are difficult to interpret, SR is very concise and interpretable. Finally, symbolic equations may outperform neural networks in out-of-distribution generalization (especially for physical problems) [17].

SR does not require *a priori* specification of a model. Conventional regression methods such as least squares [18], likelihood-based [19]–[21], and Bayesian regression techniques [22]–[26] use fixed-form parametric models and optimize for the model parameters only. SR seeks to find both a model structure and its associated parameters simultaneously.

Related Work. The SR problem has been widely studied in the literature [27], [28]. SR can be a very challenging problem and is thought to be NP-hard [29]–[32]. It can also be computationally expensive as the search space is very wide (or complex) containing expressions of any size and length [33], this issue being exacerbated with the dimension of the input feature vector (i.e. the number of independent variables). Several approaches have been suggested over the years. Most of the methods use genetic (or evolutionary) algorithms [12], [34]–[36]. Some more recent methods are Bayesian in nature [37], some are physics-inspired [30], and others use divide-and-conquer [38] and block building algorithms [39]–[41]. Lately, researchers proposed using machine learning algorithms and neural networks to solve the SR problem [31], [42]–[57]. Furthermore, some works suggested constraining the search space of functions to generalized linear space [58] (e.g. Fast Function eXtraction [59], Elite Bases Regression [60], etc.) which proved to accelerate the convergence of genetic algorithms significantly (at the expense of sometimes losing the generality of the solution [38]). Most of the SR methods use a tree-based implementation, where analytical functions are represented (or encoded) by expression trees. Some approaches suggested encoding functions as an integer string [61], others proposed representing them using matrices [33], [60], [62], [63]. As we will discuss in later sections, our implementation relies on matrices to encode expressions.

Our Contribution. We present *Generalized Symbolic Regression* (GSR), by modifying the conventional SR optimization problem formulation, while keeping the main SR objective intact. In GSR, we identify mathematical relationships between the independent variables (or features) and some transformation of the target variable. In other words, we learn the mapping from the feature space to a transformed target space (where the transformation applied to the target variable is also learned during this process). To find the appropriate functions (or transformations) to be applied to the features as well as to the targets, we constrain our search space to a weighted sum of basis functions. In contrast to conventional tree-based genetic programming approaches, we propose a matrix-based encoding scheme to represent the basis functions (and hence the full mathematical expressions). We run a series of numerical experiments on the well-known SR benchmark datasets and show that our proposed method is competitive with many strong SR methods. Finally, we introduce SymSet, a new SR benchmark problem set that is more challenging than existing benchmarks.

2.2 Notation and Problem Formulation

Consider the following regression task. We are given a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ consisting of N i.i.d. paired examples, where $\mathbf{x}_i \in \mathbb{R}^d$ denotes the i^{th} d-dimensional input feature vector and $y_i \in \mathbb{R}$ represents the corresponding continuous target variable. The goal of SR is to search the space of all possible mathematical expressions $\mathcal{S}$ defined by a set of given mathematical functions (e.g., exp, ln, sin, cos) and arithmetic operations (e.g., $+$, $-$, $\times$, $\div$), along with the following optimization problem:

$$f^* = \arg\min_{f \in \mathcal{S}} \sum_{i=1}^{N} \big[f(\mathbf{x}_i) - y_i\big]^2 \tag{2.1}$$

where f is the model function and f^* is the optimal model.

2.3 Generalized Symbolic Regression (GSR)

In this section, we introduce our Generalized Symbolic Regression (GSR) approach. We present its problem formulation, and discuss its solution and implementation.

2.3.1 Modifying the goal of symbolic regression

As highlighted in Section 2.2, the goal of SR is to search the function space to find the model that best fits the mapping between the independent variables and the target variable (i.e. the mapping between $\mathbf{x}_i$ and y_i, for all i). Since the main objective of SR is to recognize correlations and find non-trivial interpretable models (rather than making direct predictions), we modify the goal of SR; we instead search the function space to find the model that best describes the mapping between the independent variables and a transformation of the target variable (i.e. the mapping between $\mathbf{x}_i$ and some transformation or function of y_i, for all i). Formally, we propose modifying the goal of SR to search for appropriate (model) functions from a space of all possible mathematical expressions $\mathcal{S}$ defined by a set of given mathematical functions (e.g., exp, ln, sin, cos) and arithmetic operations (e.g., $+$, $-$, $\times$, $\div$), which can be described by the following optimization problem:

$$f^*, g^* = \arg\min_{f,\, g \in \mathcal{S}} \sum_{i=1}^{N} \big[f(\mathbf{x}_i) - g(y_i)\big]^2 \tag{2.2}$$

where f^* and g^* are the optimal analytical functions. In other words, instead of searching for mathematical expressions of the form $y = f(\mathbf{x})$ as is usually done in the SR literature, the proposed GSR approach attempts to find expressions of the form $g(y) = f(\mathbf{x})$. We illustrate this concept in Table 2.1.

Table 2.1: GSR finds analytical expressions of the form $g(y) = f(\mathbf{x})$ instead of $y = f(\mathbf{x})$.

Ground Truth Expression	**Learned Expression**
$y = \sqrt{x+5}$	$y^2 = x + 5$
$y = 1/(3x_1 + x_2^3)$	$y^{-1} = 3x_1 + x_2^3$
$y = (2x_1 + x_2)^{-\frac{2}{3}}$	$\ln(y) = -\frac{2}{3}\ln(2x_1 + x_2)$
$y = \ln(x_1^3 + 4x_1x_2)$	$e^y = x_1^3 + 4x_1x_2$
$y = e^{x_1^3 + 2x_2 + \cos(x_3)}$	$\ln(y) = x_1^3 + 2x_2 + \cos(x_3)$

Although the main goal of GSR is to find expressions of the form $g(y) = f(\mathbf{x})$, we may encounter situations where it is best to simply learn expressions of the form $y = f(\mathbf{x})$ (i.e. $g(y) = y$). For instance, consider the ground truth expression $y = \sin(x_1) + 2x_2$. In this case, we expect to learn the expression exactly as is (i.e. $g(y) = y$ and $f(\mathbf{x}) = \sin(x_1) + 2x_2$) as long as the right basis functions (i.e. $\sin(x)$ and x in this case) are within the search space, as we will see in the next sections.

Making predictions. Given a new input feature vector $\mathbf{x}_*$, predicting y_* with GSR is simply a matter of solving the equation $g(y) = f(\mathbf{x}_*)$ for y, or equivalently, $g(y) - f(\mathbf{x}_*) = 0$. Note that $f(\mathbf{x}_*)$ is a known quantity and y is the only unknown. If $g(\cdot)$ is an invertible function, then y_* can be easily found using $y_* = g^{-1}\big(f(\mathbf{x}_*)\big)$. If $g(\cdot)$ is not invertible, then y_* will be the root of the function $h(y) = g(y) - f(\mathbf{x}_*)$. Root-finding algorithms include Newton's method. Whether the function $g(\cdot)$ is invertible or not, we might end up with many solutions for y_* (an invertible function, which is not one-to-one, can lead to more than one solution). In this case, we choose y_* to be the solution that belongs to the range of y which can be determined from the training dataset.

2.3.2 A new problem formulation for symbolic regression

Now that we have presented the goal of our proposed GSR approach (summarized by Eq. (2.2)), we need to constrain the search space of functions $\mathcal{S}$ to reduce the computational challenges and accelerate the convergence of our algorithm. Inspired by [59], [60] as well as classical system identification methods [14], we confine $\mathcal{S}$ to generalized linear models, i.e. to functions that can be expressed as a linear combination (or as a weighted sum) of basis functions (which can be linear or nonlinear). In mathematical terms, for a given input feature vector $\mathbf{x}_i$ and a corresponding target variable y_i, the search space $\mathcal{S}$ is constrained to model functions of the form:

$$f(\mathbf{x}_i) = \sum_{j=1}^{M_\phi} \alpha_j \phi_j(\mathbf{x}_i), \qquad g(y_i) = \sum_{j=1}^{M_\psi} \beta_j \psi_j(y_i) \tag{2.3}$$

where $\phi_j(\cdot)$ and $\psi_j(\cdot)$ are the basis functions applied to the feature vector $\mathbf{x}_i$ and the target variable y_i, respectively, M_ϕ and M_ψ denote the corresponding number of basis functions involved, respectively. In matrix form, the minimization problem described in Eq. (2.2) is equivalent to finding the vectors of coefficients $\boldsymbol{\alpha} = [\alpha_1 \cdots \alpha_{M_\phi}]^T$ and $\boldsymbol{\beta} = [\beta_1 \cdots \beta_{M_\psi}]^T$ such that:

$$\boldsymbol{\alpha}^*, \boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\alpha},\boldsymbol{\beta}} ||\mathbf{X}\boldsymbol{\alpha} - \mathbf{Y}\boldsymbol{\beta}||^2 \tag{2.4}$$

where

$$\mathbf{X} = \begin{bmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_{M_\phi}(\mathbf{x}_1) \\ \vdots & \vdots & \ldots & \vdots \\ \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_{M_\phi}(\mathbf{x}_N) \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} \psi_1(y_1) & \psi_2(y_1) & \cdots & \psi_{M_\psi}(y_1) \\ \vdots & \vdots & \ldots & \vdots \\ \psi_1(y_N) & \psi_2(y_N) & \cdots & \psi_{M_\psi}(y_N) \end{bmatrix}. \tag{2.5}$$

Note that if we examine the minimization problem as expressed in Eq. (2.4), we can indeed minimize $||\mathbf{X}\boldsymbol{\alpha} - \mathbf{Y}\boldsymbol{\beta}||^2$ by simply setting $\boldsymbol{\alpha}^* = 0$ and $\boldsymbol{\beta}^* = 0$ which will not lead to a meaningful solution to our GSR problem. In addition, to avoid reaching overly complex mathematical expressions for $f(\cdot)$ and $g(\cdot)$, we are interested in finding sparse solutions for the weight vectors $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$ consisting mainly of zeros which results in simple analytical functions containing only the surviving basis functions (i.e. whose corresponding weights are nonzero). This is closely related to sparse identification of nonlinear dynamics (SINDy) methods [14]. To this end, we apply L_1 regularization, also known as *Lasso regression* [64],

by adding a penalty on the L_1 norm of the weights vector (i.e. the sum of its absolute values) which leads to sparse solutions with few nonzero coefficients. In terms of our GSR method, Lasso regression automatically performs basis functions selection from the set of basis functions that are under consideration.
Putting the pieces together, we reformulate the minimization problem in Eq. (2.4) as a constrained Lasso regression optimization problem defined as

$$\begin{aligned} \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \; & ||\boldsymbol{A}\boldsymbol{w}||_2^2 + \lambda||\boldsymbol{w}||_1 \\ \text{s.t. } & ||\boldsymbol{w}||_2 = 1 \end{aligned} \tag{2.6}$$

where $\lambda > 0$ is the regularization parameter, and

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{X} & -\boldsymbol{Y} \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = [\alpha_1 \; \cdots \; \alpha_{M_\phi} \; \beta_1 \; \cdots \; \beta_{M_\psi}]^T. \tag{2.7}$$

2.3.3 Solving the GSR problem

To solve the GSR problem, we first present our approach for solving the constrained Lasso problem in Eq. (2.6), assuming some particular sets of basis functions are given. We then outline our genetic programming (GP) procedure for finding the appropriate (or optimal) sets of these basis functions, before discussing our matrix-based encoding scheme (to represent the basis functions) that we will use in our GP algorithm.

Solving the Lasso optimization problem given particular sets of basis functions

We assume for now that, in addition to the dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, we are also given the sets of basis functions $\{\phi_j(\mathbf{x}_i)\}_{j=1}^{M_\phi}$ and $\{\psi_j(y_i)\}_{j=1}^{M_\psi}$ used with the input feature vector $\mathbf{x}_i$ and its corresponding target variable y_i, respectively, for all i. In other words, we assume for now that the matrix $\boldsymbol{A}$ in Eq. (2.7) is formed based on particular sets of basis functions $\big($i.e. $\{\phi_j(\mathbf{x}_i)\}_{j=1}^{M_\phi}$ and $\{\psi_j(y_i)\}_{j=1}^{M_\psi}\big)$, and we are mainly interested in solving the constrained optimization problem in Eq. (2.6). Applying the alternating direction method of multipliers (ADMM) [65], the optimization problem in Eq. (2.6) can be written as

$$\begin{aligned} \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \; & ||\boldsymbol{A}\boldsymbol{w}||_2^2 + \lambda||\boldsymbol{z}||_1 \\ \text{s.t. } & ||\boldsymbol{w}||_2 = 1 \\ & \boldsymbol{w} - \boldsymbol{z} = 0 \end{aligned} \tag{2.8}$$

where $\lambda > 0$ is the regularization parameter. The scaled form of ADMM (see [65] for details) for this problem is

$$\begin{aligned} \boldsymbol{w}_k &\leftarrow \arg\min_{||\boldsymbol{w}||_2=1} \mathcal{L}_\rho(\boldsymbol{w}, \boldsymbol{z}_{k-1}, \boldsymbol{u}_{k-1}) \\ \boldsymbol{z}_k &\leftarrow S_{\lambda/\rho}\big(\boldsymbol{w}_k + \boldsymbol{u}_{k-1}\big) \\ \boldsymbol{u}_k &\leftarrow \boldsymbol{u}_{k-1} + \boldsymbol{w}_k - \boldsymbol{z}_k \end{aligned} \tag{2.9}$$

Algorithm 1: Solving the constrained Lasso optimization problem using ADMM

Input: $\boldsymbol{A}$, λ, ρ, $\boldsymbol{w}_0$, $\boldsymbol{z}_0$, $\boldsymbol{u}_0$
Output: $\boldsymbol{w}$
function SolveADMM($\boldsymbol{A}$, λ, ρ, $\boldsymbol{w}_0$, $\boldsymbol{z}_0$, $\boldsymbol{u}_0$)
  Initialization: $\boldsymbol{w} \leftarrow \boldsymbol{w}_0, \boldsymbol{z} \leftarrow \boldsymbol{z}_0, \boldsymbol{u} \leftarrow \boldsymbol{u}_0$;
  while *Not Converge* **do**
    $\boldsymbol{w} \leftarrow \left(2\boldsymbol{A}^T\boldsymbol{A} + \rho I\right)^{-1} \cdot \rho\left(\boldsymbol{z} - \boldsymbol{u}\right)$;
    $\boldsymbol{w} \leftarrow \boldsymbol{w}/||\boldsymbol{w}||_2$;
    $\boldsymbol{z} \leftarrow S_{\lambda/\rho}\big(\boldsymbol{w} + \boldsymbol{u}\big)$;
    $\boldsymbol{u} \leftarrow \boldsymbol{u} + \boldsymbol{w} - \boldsymbol{z}$;
  end
end function

where $\boldsymbol{u}$ is the scaled dual vector, and

$$\mathcal{L}_\rho(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{u}) = ||\boldsymbol{A}\boldsymbol{w}||_2^2 + \frac{\rho}{2}\left|\left|\boldsymbol{w} - \boldsymbol{z} + \boldsymbol{u}\right|\right|_2^2 \tag{2.10}$$

where $\rho > 0$ is the penalty parameter and the *soft thresholding operator* S is defined as

$$S_\kappa(a) = \begin{cases} a - \kappa & a > \kappa \\ 0 & |a| \leq \kappa \\ a + \kappa & a < -\kappa \end{cases} \tag{2.11}$$

To find the minimizer $\boldsymbol{w}_k$ in the first step of the ADMM algorithm above (in Eq. (2.9)), we first compute the gradient of the function $\mathcal{L}_\rho(\boldsymbol{w}, \boldsymbol{z}_{k-1}, \boldsymbol{u}_{k-1})$ with respect to $\boldsymbol{w}$, set it to zero, and then normalize the resulting vector solution:

$$\begin{aligned} 0 &= \nabla_{\boldsymbol{w}}\mathcal{L}_\rho(\boldsymbol{w}, \boldsymbol{z}_{k-1}, \boldsymbol{u}_{k-1})\big|_{\boldsymbol{w}=\boldsymbol{w}_k} \\ &= 2\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{w}_k + \rho\left(\boldsymbol{w}_k - \boldsymbol{z}_{k-1} + \boldsymbol{u}_{k-1}\right) \end{aligned} \tag{2.12}$$

It follows that

$$\begin{aligned} \boldsymbol{w}_k &= \left(2\boldsymbol{A}^T\boldsymbol{A} + \rho I\right)^{-1} \cdot \rho\left(\boldsymbol{z}_{k-1} - \boldsymbol{u}_{k-1}\right) \\ \boldsymbol{w}_k &= \boldsymbol{w}_k/||\boldsymbol{w}_k||_2 \end{aligned} \tag{2.13}$$

Algorithm 1 outlines the overall process for solving the constrained Lasso optimization problem in Eq. (2.6), for a given matrix $\boldsymbol{A}$, regularization parameter λ, penalty parameter ρ, and initial guesses $\boldsymbol{w}_0$, $\boldsymbol{z}_0$, $\boldsymbol{u}_0$.

Finding the appropriate sets of basis functions using genetic programming

Now that we have presented Algorithm 1 that solves the constrained Lasso optimization problem in Eq. (2.6) for particular sets of basis functions, we go through our procedure for finding the optimal sets of basis functions $\big($and hence, the optimal analytical functions $f(\cdot)$ and $g(\cdot)\big)$.

Encoding Scheme

Most of the SR methods rely on expression trees in their implementation. That is, each mathematical expression is represented by a tree where nodes (including the root) encode arithmetic operations (e.g. $+$, $-$, $\times$, $\div$) or mathematical functions (e.g. cos, sin, exp, ln), and leaves contain the independent variables (i.e. $x_1, \ldots, x_d$) or constants. Inspired by [60], [62], we use matrices instead of trees to represent the basis functions. However, we propose our own encoding scheme that we believe is general enough to handle/recover a wide range of expressions.

We introduce the basis matrices $\boldsymbol{B}^{\phi}$ and $\boldsymbol{B}^{\psi}$ to represent the basis functions $\phi(\cdot)$ and $\psi(\cdot)$ used with the feature vector $\mathbf{x}$ and the target variable y, respectively. The basis matrices $\boldsymbol{B}^{\phi}$ and $\boldsymbol{B}^{\psi}$ are of sizes $n_{\boldsymbol{B}^{\phi}} \times m_{\boldsymbol{B}^{\phi}}$ and $n_{\boldsymbol{B}^{\psi}} \times 1$ respectively (i.e. $\boldsymbol{B}^{\psi}$ is a column vector), and take the form

$$\boldsymbol{B}^{\phi} = \begin{bmatrix} b^{\phi}_{1,1} & \cdots & b^{\phi}_{1,m_{\boldsymbol{B}^{\phi}}} \\ \vdots & \cdots & \vdots \\ b^{\phi}_{n_{\boldsymbol{B}^{\phi}},1} & \cdots & b^{\phi}_{n_{\boldsymbol{B}^{\phi}},m_{\boldsymbol{B}^{\phi}}} \end{bmatrix}, \qquad \boldsymbol{B}^{\psi} = \begin{bmatrix} b^{\psi}_{1,1} \\ \vdots \\ b^{\psi}_{n_{\boldsymbol{B}^{\psi}},1} \end{bmatrix}, \tag{2.14}$$

where the entries $b^{\phi}_{i,j}$ and $b^{\psi}_{i,1}$ are all integers. The first column $b^{\phi}_{\bullet,1}$ of $\boldsymbol{B}^{\phi}$ and the first (and only) column $b^{\psi}_{\bullet,1}$ of $\boldsymbol{B}^{\psi}$ indicate the mathematical function (or transformation) to be applied (on the input feature vector $\mathbf{x}$ and the target variable y, respectively). In $\boldsymbol{B}^{\phi}$, the second column $b^{\phi}_{\bullet,2}$ specifies the type of argument (see Table 2.2), and the remaining $n_v = m_{\boldsymbol{B}^{\phi}} - 2$ columns $b^{\phi}_{\bullet,3}, \cdots, b^{\phi}_{\bullet,m_{\boldsymbol{B}^{\phi}}}$ indicate which independent variables (or features) are involved (i.e. the active operands). The quantity n_v represents the maximum total multiplicity of all the independent variables included in the argument. Note that $n_{\boldsymbol{B}^{\phi}}$ and $n_{\boldsymbol{B}^{\psi}}$ specify the number of transformations to be multiplied together (i.e. each basis function $\phi(\cdot)$ and $\psi(\cdot)$ will be a product of $n_{\boldsymbol{B}^{\phi}}$ and $n_{\boldsymbol{B}^{\psi}}$ transformations, respectively).

The encoding/decoding process happens according to a table of mapping rules that is very straightforward to understand and employ. For instance, consider the mapping rules outlined in Table 2.2, where d is the dimension of the input feature vector. As we will see in our numerical experiments in Section 2.4, we will adopt this table for many SR benchmark problems. Other mapping tables are defined according to different benchmark problems[1] (more details about the SR benchmark problem specifications can be found in Appendix A.3). The encoding from the analytical form of a basis function to the basis matrix is straightforward. For example, for $d = 3$, $n_{\boldsymbol{B}^{\phi}} = 4$ and $m_{\boldsymbol{B}^{\phi}} = 5$ (i.e. $n_v = 3$), the basis function $\phi(\mathbf{x}) = x_2 \cos(x_1^2 x_2) \ln(x_1 + x_3)$ can be generated according to the encoding steps shown in Table 2.3.

[1]Each SR benchmark problem uses a specific set (or library) of allowable mathematical functions (e.g. cos, sin, exp, log), and hence, we mainly modify the first two rows of the mapping tables.

Table 2.2: Example table of mapping rules for a basis function. The identity operator is denoted by $\bullet^1$.

$b_{\bullet,1}$ / Transformation (T)	0	1	2	3	4	5
	1	$\bullet^1$	cos	sin	exp	ln
$b_{\bullet,2}$ / Argument Type (arg)	0	1	2			
	x	$\sum$	$\prod$			
$b_{\bullet,3}, \cdots, b_{\bullet,m_B}$ / Variable (v)	0	1	2	3	$\cdots$	d
	skip	x_1	x_2	x_3	$\cdots$	x_d

Table 2.3: Encoding steps corresponding to the basis function $\phi(\mathbf{x}) = x_2 \cos(x_1^2 x_2) \ln(x_1 + x_3)$.

Step	T	arg	v_1	v_2	v_3	Update
1	$\bullet^1$	x	x_2	—	—	$T_1(\mathbf{x}) = x_2$
2	cos	$\prod$	x_1	x_1	x_2	$T_2(\mathbf{x}) = \cos(x_1^2 x_2)$
3	ln	$\sum$	x_1	x_3	skip	$T_3(\mathbf{x}) = \ln(x_1 + x_3)$
4	1	—	—	—	—	$T_4(\mathbf{x}) = 1$
Final Update: $\phi(\mathbf{x}) = T_1(\mathbf{x}) \cdot T_2(\mathbf{x}) \cdot T_3(\mathbf{x}) \cdot T_4(\mathbf{x})$						

Based on the mapping rules in Table 2.2 and the encoding steps in Table 2.3, the basis function $\phi(\mathbf{x}) = x_2 \cos(x_1^2 x_2) \ln(x_1 + x_3)$ can be described by a 4×5 matrix as follows:

$$\boldsymbol{B}^{\phi} = \begin{bmatrix} 1 & 0 & 2 & \bullet & \bullet \\ 2 & 2 & 1 & 1 & 2 \\ 5 & 1 & 1 & 3 & 0 \\ 0 & \bullet & \bullet & \bullet & \bullet \end{bmatrix} \tag{2.15}$$

Remark 2.3.1. In Table 2.3, — denotes entries that are ignored during the construction of the basis function. The argument type in Step 1 is x which implies that we only select the first variable (encoded by $b_{\bullet,3}$) out of the n_v variables as an argument, and hence the entries corresponding to v_2 and v_3 are ignored. Similarly, the transformation in Step 4 is $T = 1$ which implies that the argument type and the n_v variables are all ignored. These are the only two cases where some entries are ignored during the construction process. The same ignored entries are reflected in the matrix $\boldsymbol{B}^{\phi}$ using $\bullet$. More encoding examples can be found in Appendix A.2.

Remark 2.3.2. The term 'skip' can be thought of as 0 or 1 when the argument type is summation $\sum$ or multiplication $\prod$ respectively. To account for the case where the argument type is x, we let $b_{\bullet,3} \in \{1, \ldots, d\}$ (i.e. we exclude 0) as $b_{\bullet,3}$ is the only entry considered in this case (see *Remark* 2.3.1).

Remark 2.3.3. The same basis function can be represented by several matrices for three reasons:
i) Each basis function is a product of transformations where each transformation is represented by a row in the basis matrix. Hence, a new basis matrix for the same basis function is formed by simply swapping rows.
ii) When the argument type is $\sum$ or $\prod$, the order of the n_v variables (including 'skip') starting from the third column of the matrix $\boldsymbol{B}^{\phi}$ does not affect the expression. Hence a new basis matrix for the same basis function is formed by simply swapping these columns.
iii) As mentioned in *Remark* 2.3.1, some entries are ignored in some cases. Hence a new basis matrix for the same basis function is formed by simply modifying these entries.

Remark 2.3.4. In the example above, we showed how we can produce the matrix $\boldsymbol{B}^{\phi}$ to represent a basis function $\phi(\mathbf{x})$. A similar (and even simpler) procedure can be applied to

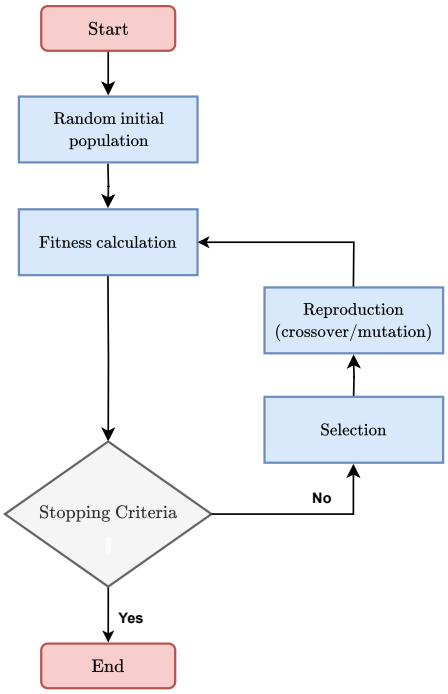


Figure 2.1: A flowchart of a generic evolutionary algorithm.

produce the matrix $\boldsymbol{B}^{\psi}$ that represents a basis function $\psi(y)$; we only need a mapping table corresponding to the set of allowable transformations (e.g. the first two rows of Table 2.2).

Note that the decoding from the basis matrix to the expression of a basis function is trivial; we go through the rows of the basis matrix and convert them into transformations according to a mapping table (e.g. Table 2.2), before finally multiplying them together. Also note that the search space of basis functions $\big($mainly $\phi(\cdot)\big)$ is huge in general which makes enumeration impractical, and hence, we will rely on GP for effective search process.

Genetic Programming (Evolutionary Algorithm)

The SR problem has been extensively studied in the literature, and a wide variety of methods has been suggested over the years to tackle it. Most of these methods are based on genetic programming (GP) [12], [34]–[36]. This is a heuristic search technique that tries to find the optimal mathematical expression (in the SR context) among all possible expressions within the search space. The optimal (or best) expression is found by minimizing some objective function, known as the fitness function.

GP is an evolutionary algorithm that solves the SR problem. It starts with an initial population (or first generation) of N_p randomly generated individuals (i.e. mathematical expressions), then recursively applies the *selection*, *reproduction (crossover)*, and *mutation* operations until *termination* (see a generic flowchart of a genetic algorithm in Fig. 2.1). During the selection operation, the fitness of each of the N_p individuals of the current generation is evaluated (according to the fitness function), and the n_p fittest (or best) individuals are selected for reproduction and mutation (the selected individuals are part of the new generation and can be thought of as parents). The reproduction (crossover) operation generates new individuals (offsprings) by combining random parts of two parent individ-

uals. The mutation operation produces a new individual by changing a random part of some parent individual. We illustrate the crossover and mutation operations in Fig. 2.2. Finally, the recursion terminates, when some individual reaches a predefined fitness level (i.e. until some stopping criterion is satisfied).

In our GSR approach, we use a slightly modified version of the GP algorithm described above. Each individual in the population initially consists of two sets of M_ϕ and M_ψ randomly generated basis functions encoded by basis matrices. Such matrices will form the functions $f(\cdot)$ and $g(\cdot)$ to be used with the input feature vector $\mathbf{x}$ and the target variable y, respectively. This is different from the GP algorithm described above where individuals typically represent the full mathematical expression or function as a whole. In addition, the N_p individuals in the population of a new generation consist of the n_p fittest individuals of the current generation in addition to $N_p - n_p$ individuals generated as follows. With probability $\frac{1}{4}$, a new individual is generated (reproduced) by randomly combining basis functions (i.e. basis matrices) from two parent individuals (i.e. crossover) selected from the n_p surviving individuals. With probability $\frac{1}{4}$, a new individual is generated by randomly choosing one of the n_p surviving individuals, and replacing (mutating) some of its basis functions (i.e. basis matrices) with completely new ones (i.e. randomly generated). With probability $\frac{1}{2}$, a completely new individual is randomly generated (in the same way we generate the individuals of the initial population). Randomly generating individuals enhances diversity in the basis functions and avoids reaching a plateau. Indeed, this is just one of many ways that can be followed to apply some sort of crossover/mutation on individuals defined by their sets of basis functions instead of their full mathematical expression. A pseudocode of our proposed GSR algorithm is provided in Appendix A.1.

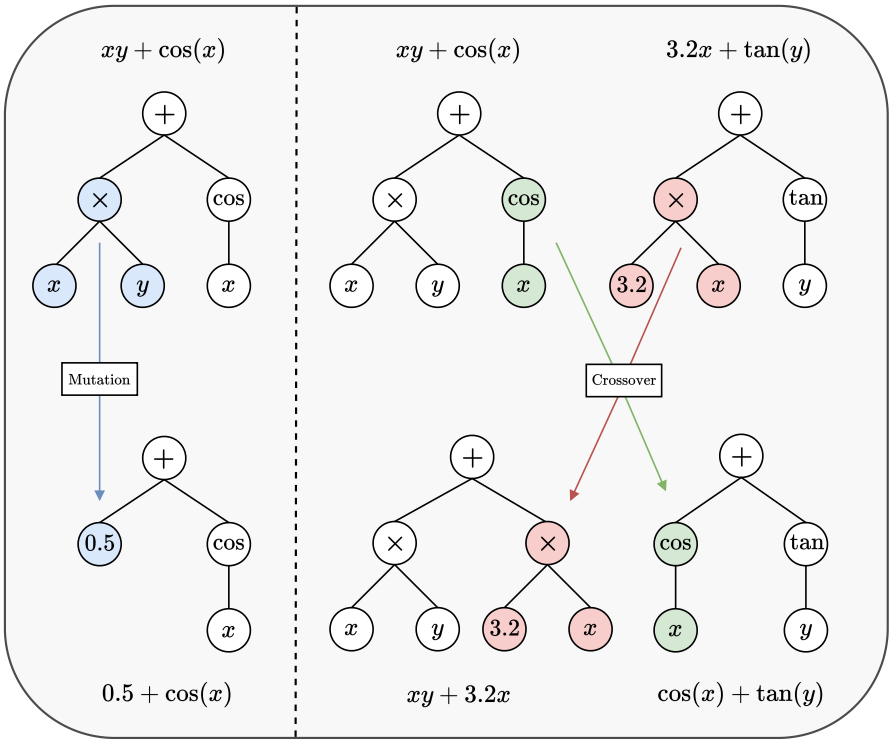


Figure 2.2: Illustration of the genetic programming mutation and crossover operations.

2.4 Experimental Results

We evaluate our proposed GSR method through a series of numerical experiments on a number of common SR benchmark datasets. In particular, we compare our approach to existing state-of-the-art methods using three popular SR benchmark problem sets: Nguyen [66], Jin [37], and Neat [67]. In addition, we demonstrate the benefits of our proposed method on the recently introduced SR benchmark dataset called Livermore [43], which covers problems with a wider range of difficulty compared to the other benchmarks. Finally, we introduce a new and more challenging set of SR benchmark problems, which we call SymSet, mainly for two reasons: i) Our GSR algorithm achieves perfect scores on Nguyen, and almost perfect scores on Jin, Neat, and Livermore, and hence we introduced a benchmark problem set that is more challenging, ii) The existing SR benchmark problem sets do not really reflect the strengths of our proposed method, and thus we designed SymSet to explicitly highlight the benefits we gain from using our proposed approach. SymSet contains benchmark problems with similar properties as Nguyen, Jin, Neat, and Livermore benchmarks, but with an additional function composition (or symbolic layer). Each SR benchmark problem consists of a ground truth expression, a training and test dataset, and a set (or libary) of allowable arithmetic operations and mathematical functions. Specifications of all the SR benchmark problems are described in Appendix A.3. Hyperparameters and additional experiment details are provided in Appendix A.1.

Across our experiments, we compare our GSR approach against several strong SR benchmark methods:
Neural-guided genetic programming population seeding (NGGPPS): A hybrid approach of neural-guided search and GP, which uses a recurrent neural network (RNN) to seed the starting population for GP [43]. NGGPPS achieves strong results on the well-known SR benchmarks.
Deep Symbolic Regression (DSR): A reinforcement learning method that proposes a risk-seeking policy gradient to train an RNN to produce better-fitting expressions [31]. DSR is the "RNN only" version of NGGPPS, and is also considered a strong performer on the common SR benchmarks.
Bayesian Symbolic Regression (BSR): A Bayesian framework which carefully designs prior distributions to incorporate domain knowledge (e.g. preference of basis functions or tree structure), and which employs efficient Markov Chain Monte Carlo (MCMC) methods to sample symbolic trees from the posterior distributions [37].
Neat-GP: A GP approach which uses the NeuroEvolution of Augmenting Topologies (NEAT) algorithm that greatly reduces the effects of bloat (i.e. controls growth in program size) [67].
PSTree: A piece-wise non-linear SR method based on decision tree and GP techniques [46]. PSTree can generate explainable models with high accuracy in a short period of time. PSTree is the current top performer on SRBench datasets [28], achieving state-of-the-art performance and beating other competitive SR methods such as Operon [55], [56] and AI Feynman [51].
PySR: A fast and parallelized SR method in Python/Julia [68], which uses evolutionary algorithms to search for symbolic expressions by optimizing a particular objective; the metric

Table 2.4: Recovery rate comparison of GSR against several algorithms on the Nguyen benchmark set over 100 independent runs. The formulas for these benchmarks are shown in Appendix Table A.11.

Benchmark	Expression	Recovery Rate (%) GSR	NGGPPS	DSR	Eureqa
Nguyen-1	$y = x^3 + x^2 + x$	100	100	100	100
Nguyen-2	$y = x^4 + x^3 + x^2 + x$	100	100	100	100
Nguyen-3	$y = x^5 + x^4 + x^3 + x^2 + x$	100	100	100	95
Nguyen-4	$y = x^6 + x^5 + x^4 + x^3 + x^2 + x$	100	100	100	70
Nguyen-5	$y = \sin(x^2)\cos(x) - 1$	100	100	72	73
Nguyen-6	$y = \sin(x) + \sin(x + x^2)$	100	100	100	100
Nguyen-7	$y = \ln(x + 1) + \ln(x^2 + 1)$	100	97	35	85
Nguyen-8	$y = \sqrt{x}$	100	100	96	0
Nguyen-9	$y = \sin(x_1) + \sin(x_2^2)$	100	100	100	100
Nguyen-10	$y = 2\sin(x_1)\cos(x_2)$	100	100	100	64
Nguyen-11	$y = x_1^{x_2}$	100	100	100	100
	Average	**100**	99.73	91.18	80.64

used for scoring equations is based on the work in [17].
gplearn: A Koza-style SR method in Python, which starts with a random population of models, and then iteratively performs tournament selection, crossover, and mutation [34].

We first compare GSR against NGGPPS, DSR, as well as Eureqa (a popular GP-based commercial software proposed in [12]) on the Nguyen benchmarks. We follow their experimental procedure and report the results in Table 2.4. We use *recovery rate* as our performance metric, defined as the fraction of independent training runs in which an algorithm's resulting expression achieves exact symbolic equivalence compared to the ground truth expression (as verified using a computer algebra system such as SymPy [69]). Table 2.4 shows that GSR significantly outperforms DSR and Eureqa in exactly recovering the Nguyen benchmark expressions. As NGGPPS achieves nearly perfect scores on the Nguyen benchmarks, GSR shows only a slight improvement (on Nguyen-7) compared to NGGPPS. However, GSR exhibits faster runtime than NGGPPS; by running each benchmark problem, GSR takes an average of 2.5 minutes per run on the Nguyen benchmarks compared to 3.2 minutes for NGGPPS. Runtimes on individual Nguyen benchmark problems are shown in Appendix Table A.3.

We next evaluate GSR on the Jin and Neat benchmark sets. The results are reported in Tables 2.5 and 2.6 respectively. A RMSE value of 0 indicates exact symbolic equivalence. From Table 2.5, we can clearly observe that GSR outperforms DSR and BSR and performs nearly as good as NGGPPS recovering all the Jin problems (accross all independent runs) except Jin-6. Table 2.6 shows that GSR outperforms all other methods (NGGPPS, DSR, and Neat-GP) on the Neat benchmarks. Note that expressions containing divisions (i.e. Neat-6, Neat-8, and Neat-9) are not exactly recovered by GSR (i.e. only approximations are recovered) since the division operator is not included in our scheme (see Appendix A.5 for details).

Table 2.5: Comparison of mean root-mean-square error (RMSE) for GSR against several methods on the Jin benchmark problem set over 50 independent runs. The formulas for these benchmarks are shown in Appendix Table A.11.

Benchmark	Mean RMSE GSR	NGGPPS	DSR	BSR
Jin-1	0	0	0.46	2.04
Jin-2	0	0	0	6.84
Jin-3	0	0	0.00052	0.21
Jin-4	0	0	0.00014	0.16
Jin-5	0	0	0	0.66
Jin-6	0.018	0	2.23	4.63
Average	0.0030	**0**	0.45	2.42

Table 2.6: Comparison of median RMSE for GSR against several methods on the Neat benchmark problem set over 30 independent runs. The formulas for these benchmarks are shown in Appendix Table A.11.

Benchmark	Median RMSE GSR	NGGPPS	DSR	Neat-GP
Neat-1	0	0	0	0.0779
Neat-2	0	0	0	0.0576
Neat-3	0	0	0.0041	0.0065
Neat-4	0	0	0.0189	0.0253
Neat-5	0	0	0	0.0023
Neat-6	2.0×10^{-4}	6.1×10^{-6}	0.2378	0.2855
Neat-7	0.0521	1.0028	1.0606	1.0541
Neat-8	4.0×10^{-4}	0.0228	0.1076	0.1498
Neat-9	8.1×10^{-9}	0	0.1511	0.1202
Average	**0.0059**	0.1139	0.1756	0.1977

We then run experiments on the Livermore benchmark set which contains problems with a large range of difficulty. In addition to NGGPPS and DSR, we compare against NGGPPS using the soft length prior (SLP) and hierarchical entropy regularizer (HER) recently introduced in [70]. We also compare against a recently proposed method, known by genetic expert-guided learning (GEGL) [52], which trains a molecule-generating deep neural network (DNN) guided with genetic exploration. Table 2.7 shows that our GSR method outperforms all other methods on both the Nguyen and Livermore benchmark sets, beating NGGPPS+SLP/HER which was the top performer on these two benchmark sets.

We highlight the strengths of GSR on the new SymSet benchmark problem set, and show the benefits of searching for expressions of the form $g(y) = f(\mathbf{x})$ instead of $y = f(\mathbf{x})$. Typical expressions, with exact symbolic equivalence, recovered by GSR are shown in Appendix Table A.18. The key feature of GSR lies in its ability to recover expressions of the form $g(y) = f(\mathbf{x})$. To better highlight the benefits offered by this feature, we disable it by constraining the search space in GSR to expressions of the form $y = f(\mathbf{x})$ (which is the most critical ablation). We refer to this special version of GSR as s-GSR. Note that most of the SymSet expressions cannot be exactly recovered by s-GSR (i.e. they can only be approximated). We compare the performance of GSR against s-GSR on the SymSet benchmarks in terms of accuracy and runtime (see Table 2.8). The results clearly show that GSR is faster than s-GSR, averaging around 2 minutes per run on the SymSet benchmarks compared to 2.27 minutes for s-GSR (i.e. $\sim 11\%$ runtime improvement). In addition, GSR is more accurate than s-GSR by two orders of magnitude. This is due to the fact that GSR exactly recovers the SymSet expressions across most of the runs, while s-GSR only recovers approximations for most of these expressions. This reflects the superiority of GSR over s-GSR, which demonstrates the benefits of learning expressions of the form $g(y) = f(\mathbf{x})$ in SR tasks. We further compare GSR against several strong SR methods with similar (or better) expression ability. In particular, we experiment on SymSet with NGGPPS, PSTree, PySR, and gplearn (see Table 2.8). GSR is more accurate than all these methods by three orders of magnitude, which further demonstrates the advantage of our proposed approach.

Table 2.7: Recovery rate comparison of GSR against several algorithms on the Nguyen and Livermore benchmark sets over 25 independent runs. Recovery rates on individual benchmark problems are shown in Appendix Table A.2.

	Recovery Rate (%)		
	All	Nguyen	Livermore
GSR	**90.59**	**100.00**	**85.45**
NGGPPS+SLP/HER	82.59	92.00	77.45
NGGPPS	78.59	92.33	71.09
GEGL	66.82	86.00	56.36
DSR	49.18	83.58	30.41

Table 2.8: Average performance in mean RMSE and runtime, along with their standard errors, for GSR against s-GSR and several strong SR methods on the SymSet benchmark problem sets over 25 independent runs. Mean RMSE and runtime values on individual benchmark problems are shown in Appendix Table A.4.

	SymSet Average	
	Mean RMSE	Runtime (sec)
GSR	$\mathbf{2.66 \times 10^{-4} \pm 1.59 \times 10^{-4}}$	120.84 ± 4.22
s-GSR	$2.56 \times 10^{-2} \pm 5.27 \times 10^{-3}$	136.19 ± 4.56
PSTree	$4.57 \times 10^{-1} \pm 7.98 \times 10^{-2}$	**16.23 ± 0.67**
NGGPPS	$4.65 \times 10^{-1} \pm 1.24 \times 10^{-1}$	158.57 ± 2.59
PySR	$4.99 \times 10^{-1} \pm 1.76 \times 10^{-1}$	87.07 ± 21.6
gplearn	$7.22 \times 10^{-1} \pm 1.64 \times 10^{-1}$	163.86 ± 2.94

As for the runtime, PSTree is the fastest method, averaging around 16 seconds per run on the SymSet expressions, while maintaining solid accuracies. This comes as no surprise given its state-of-the-art performance on SRBench datasets [28].

2.5 Discussion

Limitations. GSR, including state-of-the-art methods, have difficulty with expressions containing divisions. For GSR, this is due to the way we define our encoding scheme. Other methods fail even though the division is included in their framework. GSR can overcome this issue by modifying its encoding scheme to include divisions within the basis functions (at the expense of significantly increasing the complexity of the search space). Another limiting factor to GSR is that it cannot recover expressions containing composition of functions, such as $y = e^{\cos(x)} + \ln(x)$. This could be overcome by modifying the search space (e.g. one could expand the definition of a basis function to account for composition of functions up to some number of layers, or completely modify the search space to a symbolic neural network as in [42], [50], [54]). Another challenging task for GSR is to reach, although expressible, expressions containing multiple complex basis functions simultaneously. This can be due to the choice of the hyperparameters or the GP search process. A more elaborate discussion about the limitations of GSR can be found in Appendix A.5. These limitations will be addressed in future work. Indeed, there are plenty of expressions that still cannot be fully recovered by GSR. This is the case for all other SR methods as well.

Closely related work. There has been growing attention on the SR task with non explicit (or implicit) mathematical equations and several works have been attempted to address this interesting task. In particular, implicit sparse identification of nonlinear dynamics (implicit-SINDy) [15], [16] introduces the concept of identifying implicit expressions of the form $f(\mathbf{x}, y) = 0$ in the context of differential equations $\big($i.e. $y = \dot{x}_i = \frac{dx_i}{dt}$ for $i \in \{1, \ldots, d\}$ where $\mathbf{x} \in \mathbb{R}^d\big)$. Further, Eureqa [12], a well-established baseline SR algorithm, focuses on discovering invariants rather then trying to perform prediction directly. Inspired by the two aforementioned methods, and by the fact that the main objective of SR is to recognize correlations and define non-trivial interpretable models, GSR identifies relations between the input and a transformed output through searching for expressions of the form $g(y) = f(\mathbf{x})$, which keeps the possibility open for predicting the output y in a straightforward manner.

Computational complexity. Although genetic algorithms are inherently heuristic, understanding how our GSR algorithm operates and scales could still be valuable. Following Algorithm 7 from Appendix A.1, we can approximate the time complexity of GSR as:

$$O\Big(N_\epsilon \cdot N_p \cdot \big(M_\phi \cdot n_{\boldsymbol{B}^\phi} \cdot m_{\boldsymbol{B}^\phi} + M_\psi \cdot n_{\boldsymbol{B}^\psi} + N_\delta + N + \log N_p\big)\Big) \tag{2.16}$$

where N_ϵ is the number of generations until the GP algorithm converges, and N_δ is the number of iterations until the ADMM algorithm converges. Recall that N_p is the population size, M_ϕ and M_ψ denote the number of $n_{\boldsymbol{B}^\phi} \times m_{\boldsymbol{B}^\phi}$ and $n_{\boldsymbol{B}^\psi} \times 1$ basis matrices applied to $\mathbf{x}$ and y, respectively, and N is the number of paired training examples. More details about GSR's computational complexity can be found in Appendix A.1.

Compared to GSR, the special version s-GSR adopts a vanilla SR (where $g(y)$ is simply y) with the same GP algorithm and coefficient optimization process (through ADMM) as GSR. Hence, s-GSR's time complexity can be approximated as:

$$O\Big(N_\epsilon \cdot N_p \cdot \big(M_\phi \cdot n_{\boldsymbol{B}^\phi} \cdot m_{\boldsymbol{B}^\phi} + N_\delta + N + \log N_p\big)\Big) \tag{2.17}$$

Although GSR's computational complexity contains an extra term of $O\left(N_\epsilon \cdot N_p \cdot M_\psi \cdot n_{\boldsymbol{B}^\psi}\right)$, the number of GP generations N_ϵ produced by GSR is often much less than that of s-GSR, which explains the runtime advantage of GSR over s-GSR shown in Table 2.8.

GSR's expression ability. The term *Generalized* in GSR mainly stands for its ability to discover analytical mappings from the input space to a transformed output space through expressions of the form $g(y) = f(\mathbf{x})$. This generalizes the classical SR task of identifying expressions of the form $y = f(\mathbf{x})$ (i.e. the latter is simply a special case of GSR with $g(y) = y$). In addition, the term *Generalized* can denote the fact that we constrain the search space to generalized linear models, keeping in mind that the search space could be confined to other generalized spaces. Note that, by finding relations of the form $g(y) = f(\mathbf{x})$, the expression ability of GSR could resemble that of classical SR tasks which search for relations $y = f_c(x)$ where the composition function $f_c(\cdot)$ is defined as $f_c(\cdot) \coloneqq h \circ f(\cdot) = h(f(\cdot))$ with $h(\cdot) = g^{-1}(\cdot)$ if $g(\cdot)$ is invertable, or a function class of similar expression ability as $g^{-1}(\cdot)$ if $g(\cdot)$ is not invertable. However, GSR takes advantage of the fact that the target y is a scalar, and hence, we can apply many basis functions to y (through $g(\cdot)$) without much increasing the

complexity of the expression. In other words, we can avoid searching for functions equivalent to $g^{-1}(\cdot)$ in a space that could grow exponentially with the dimension of the input feature vector by simply searching for their corresponding inverse transformations applied to the scalar target variable. This concept, which happens implicitly in our algorithm provides an edge for GSR over traditional SR methods in terms of runtime, complexity, and smoothness of the search space. In short, GSR discovers simplified expressions by reducing redundancies in the search space, which greatly saves the computational complexity of the search process. It is worth mentioning that, in principle, GSR's concept of fitting $g(y) = f(\mathbf{x})$ (instead of $y = f(\mathbf{x})$) could be applied in conjunction with other classical SR methods; this may require some modifications to their parameter/coefficient optimization process.

GSR: a simple yet promising algorithm. GSR combines features and benefits from the usually disparate fields of system identification and genetic programming. On the one hand, SINDy methods use some LASSO-like approaches (or sequential thresholded least squares) to conduct their sparse non-linear regression for finding solutions that take the form of a linear combination of basis functions. On the other hand, evolutionary algorithms are effective in finding basis functions that achieve optimal solution. In other words, GSR combines well established evolutionary methods with more classical system identification methods. Although each of the algorithm components are relatively simple, the overall GSR algorithm achieves promising experimental performance, highlighting new insights, which can open up new research directions for future improvement.

2.6 Conclusion

We introduce GSR, a Generalized Symbolic Regression approach by modifying the formulation of the conventional SR optimization problem. In GSR, we identify mathematical relationships between the input features and some transformation of the target variable. That is, we infer the mapping from the feature space to a transformed target space, by searching for expressions of the form $g(y) = f(\mathbf{x})$ instead of $y = f(\mathbf{x})$. We confine our search space to a weighted sum of basis functions and use genetic programming with a matrix-based encoding scheme to extract their expressions. We perform several numerical experiments on well-known SR benchmark datasets and show that our GSR approach is competitive with strong SR benchmark methods. We further highlight the strengths of GSR by introducing SymSet, a new SR benchmark set which is more challenging relative to the existing benchmarks. In principle, GSR's concept of fitting $g(y) = f(\mathbf{x})$ could be extended to existing SR methods and could boost their performance.

Chapter 3

Data-Driven Discovery of Partial Differential Equations via the Adjoint Method

In this chapter, we present an adjoint-based method for discovering the underlying governing partial differential equations (PDEs) given data. The idea is to consider a parameterized PDE in a general form, and formulate a PDE-constrained optimization problem aimed at minimizing the error of the PDE solution from data. Using variational calculus, we obtain an evolution equation for the Lagrange multipliers (adjoint equations) allowing us to compute the gradient of the objective function with respect to the parameters of PDEs given data in a straightforward manner. In particular, we consider a family of parameterized PDEs encompassing linear, nonlinear, and spatial derivative candidate terms, and elegantly derive the corresponding adjoint equations. We show the efficacy of the proposed approach in identifying the form of the PDE up to machine accuracy, enabling the accurate discovery of PDEs from data. We also compare its performance with the famous PDE Functional Identification of Nonlinear Dynamics method known as PDE-FIND [7], on both smooth and noisy data. Even though the proposed adjoint method relies on forward/backward solvers, it outperforms PDE-FIND for large data sets thanks to the analytic expressions for gradients of the cost function with respect to each PDE parameter.

3.1 Introduction

A large portion of data-driven modelling of physical processes in literature is dedicated to deploying Neural Networks to obtain fast prediction given the training data set. The data-driven estimation methods include Physics-Informed Neural Networks [71], Pseudo-Hamiltonian neural networks [72], structure preserving [73], [74], and reduced order modelling [75]. These methods often provide efficient and somewhat "accurate" predictions when tested as an interpolation method in the space of input or boundary parameters. Such fast estimators are beneficial when many predictions of a dynamic system is needed, for example in the shape optimization task in fluid dynamics.

However, the data-driven estimators often fail to provide accurate solution to the dynamical system when tested outside the training space, i.e. for extrapolation. Furthermore, given the regression-based nature of these predictors, often they do not offer any error estimator in prediction. Since we already have access to an arsenal of numerical methods in solving traditional governing equations, it is attractive to learn the underlying governing equation given data instead. Once the governing equation is found, one can use the standard and efficient numerical methods for prediction. This way we guarantee the consistency with observed data, estimator for the numerical approximation, and interoperability. Hence, learning the underlying physics given data has motivated a new branch in the scientific machine learning for discovering the mathematical expression as the governing equation given data.

The wide literature of data-driven discovery of dynamical systems includes equation-free modeling [76], artificial neural networks [77], nonlinear regression [78], empirical dynamic modeling [79], [80], modeling emergent behavior [81], automated inference of dynamics [82]–[84], normal form identification in climate [85], nonlinear Laplacian spectral analysis [86] and Koopman analysis [87] among others. There has been a significant advancement in this field by combining symbolic regression wtih the evolutionary algorithms [8], [12], [88], which enable the direct extraction of nonlinear dynamical system information from data. Furthermore, the concept of sparsity [64] has recently been employed to efficiently and robustly deduce the underlying principles of dynamical systems [14], [15].

Related work. Next, we review several relevant works that have shaped the current landscape of discovering PDEs from data:

PDE-FIND [7]. This method has been developed to discover underlying partial differential equation by minimizing the L_2^2-norm point-wise error of the parameterized forward model from the data. Estimating all the possible derivatives using finite difference, PDE-FIND constructs a dictionary of possible terms and finds the underlying PDE by performing a sparse search using ridge regression problem with hard thresholding, also known as STRidge optimization method. Several further developments in the literature has been carried out based on this idea, namely [16], [89]. In these methods, as the size (or dimension) of the data set increases, the PDE discovery optimization problem based on point-wise error becomes extremely expensive, forcing the user to arbitrarily reduce the size of data by resampling, or compress the data using proper orthogonal decomposition. Needles to say, in case of nonlinear dynamics, such truncation of data can introduce bias in prediction leading to finding a wrong PDE.

PDE-Net [90], [91]. In this method, the PDE is learned from data using convolution kernels rather than brute-force use of finite differences, and apply neural networks to approximate nonlinear responses. Similar to PDE-FIND, the loss function of PDE-Net is the point-wise error from data which leads to a regression task that does not scale well with the size of the data set.

Hidden Physics Models [92]. This method assumes that the relevant terms of the governing PDE are already identified and finds its unknown parameters using Gaussian process regression (GPR). While GPR is an accurate interpolator which offers an estimate for the uncertainty in prediction [93], its training scales poorly with the size of the training data set as it requires inversion of the covariance matrix.

PINN-SR [94]. One of the issues with the PDE-FIND is the use of finite difference in estimating the derivatives. The idea of PINN-SR is to extend PDE-FIND's optimization problem to also find a PINN fit to the data in order to find smooth estimates for spatial derivatives. In particular, the training of PINN-SR combines the search for weights/biases of PINN approximation of the PDE with the sparse search in the space of possible terms to find the coefficients of the PDE given data. However, similar to PDE-FIND, the point-wise error from data is used as the loss for the regression task which does not scale well with the size of the data set.

Contributions. In this work, we introduce a novel approach for discovering PDEs from data based on the well-known adjoint method, i.e. PDE-constrained optimization method. The idea is to formulate the objective (or cost) functional such that the estimate function f minimizes the L_2^2-norm error from the data points f^* with the constraint that f is the solution to a parameterized PDE using the method of Lagrange multipliers. Here, we consider a parameterized PDE in a general form and the task is to find all the parameters including irrelevant ones. By finding the variational extremum of the cost functional with respect to the function f, we obtain a backward-in-time evolution equation for the Lagrange multipliers (adjoint equations). Next, we solve the forward parameterized PDE as well as the adjoint equations numerically. Having found estimates of the Lagrange multipliers and solution to the forward model f, we can numerically compute the gradient of the objective function with respect to the parameters of PDEs given data in a straightforward manner. In particular, for a family of parameterized and nonlinear PDEs, we show how the corresponding adjoint equations can be elegantly derived. We note that the adjoint method has been successfully used before as an efficient method for uncertainty quantification [95], shape optimization and sensitivity analysis method in fluid mechanics [96], [97] and plasma physics [98], [99]. Unlike the usual use of PDE-constrained adjoint optimization where the governing equation is known, in this work we are interested in finding the form along with the coefficients of the PDE given data.

The rest of the chapter is organized as follows. First in Section 3.2, we introduce and derive the proposed adjoint-based method of finding the underlying system of PDEs given data. Next in Section 3.3, we present our results on a wide variety of PDEs and compare the solution with the celebrated PDE-FIND in terms of error and computational/training time. In Section 3.6, we discuss the limitations for the current version of our approach and provide concluding remarks in Section 3.7.

3.2 Adjoint method for finding PDEs

In this section, we introduce the problem and derive the proposed adjoint method for finding governing equations given data.

Problem setup. Assume we are given a data set on a spatial/temporal grid $\mathcal{G} = \bigcup_{j=0}^{N_t} \mathcal{G}^{(j)}$ with $\mathcal{G}^{(j)} = \left\{ \left(\boldsymbol{x}^{(k)}, t^{(j)} \right) \;\middle|\; k = 1, ..., N_{\boldsymbol{x}} \right\}$ for the vector of functions $\boldsymbol{f}^*$ where k is the spatial

index and j the time index with $t^{(N_t)} = T$ being the final time. Here, $\boldsymbol{x}^{(k)} \in \Omega \subset \mathbb{R}^n$ is coordinates inside the solution domain Ω, $t^{(j)}$ denotes the j-th time that data is available, and output is a discrete map $\boldsymbol{f}^* : \mathcal{G} \to \mathbb{R}^N$. The goal is to find the governing equations that accurately estimates $\boldsymbol{f}^*$ at all points on $\mathcal{G}$. In order to achieve this goal, we formulate the problem using the method of Lagrange multipliers.

Adjoint method. For simplicity, let us first consider only the time interval $t \in \left[t^{(j)}, t^{(j+1)}\right]$. Consider a general a forward model $\mathcal{L}[\cdot]$ that evolves an N-dimensional vector of continuous functions $\boldsymbol{f}\big(\boldsymbol{x}, t = t^{(j)}\big)$ in $t \in \left(t^{(j)}, t^{(j+1)}\right]$ and $\boldsymbol{x} \in \Omega$ where the i-th PDE is given by

$$\mathcal{L}_i[\boldsymbol{f}] := \partial_t f_i + \sum_{\boldsymbol{d},\boldsymbol{p}} \alpha_{i,\boldsymbol{d},\boldsymbol{p}} \nabla_{\boldsymbol{x}}^{(\boldsymbol{d})}[\boldsymbol{f}^{\boldsymbol{p}}] = 0 \tag{3.1}$$

for $i = 1, \ldots, N$, resulting in a system of N-PDEs, i.e. the i-th PDE $\mathcal{L}_i$ predicts f_i. Here, $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$ is an n-dimensional (spatial) input vector, and $\boldsymbol{f} = [f_1, f_2, \ldots, f_N]$ is an N-dimensional vector of functions. We use the shorthand $f_i = f_i(\boldsymbol{x}, t)$ and $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x}, t)$. Furthermore, $\boldsymbol{p} = [p_1, \ldots, p_N]$ and $\boldsymbol{d} = [d_1, d_2, \ldots, d_n]$ are non-negative index vectors such that $\boldsymbol{f}^{\boldsymbol{p}} = f_1^{p_1} f_2^{p_2} \cdots f_N^{p_N}$ and

$$\nabla_{\boldsymbol{x}}^{(\boldsymbol{d})} = \nabla_{x_1}^{(d_1)} \nabla_{x_2}^{(d_2)} \cdots \nabla_{x_n}^{(d_n)} \ , \tag{3.2}$$

where $\nabla_{x_i}^{(d_i)}$ for $i = 1, ..., n$ indicates d_i-th derivative in x_i dimension, and $\partial_t f_i$ denotes the time derivative of the i-th function. We denote the vector of unknown parameters by $\boldsymbol{\alpha} = [\alpha_{i,\boldsymbol{d},\boldsymbol{p}}]_{(i,\boldsymbol{d},\boldsymbol{p})\in\mathcal{D}}$, where $\mathcal{D}$ represents the domain of all valid combinations of i, $\boldsymbol{d}$, and $\boldsymbol{p}$.

Having written the forward model (3.1) as general as possible, the goal is to find the parameters $\boldsymbol{\alpha}$ such that $\boldsymbol{f}$ approximates the data points of $\boldsymbol{f}^*$ at $t = t^{(j+1)}$ given the solution $\boldsymbol{f} = \boldsymbol{f}^*$ at $t = t^{(j)}$. To this end, we formulate a semi-discrete objective (or cost) functional that minimizes the L_2^2-norm error between what the model predicts and the data $\boldsymbol{f}^*$ on $\mathcal{G}^{(j+1)}$, with the constraint that $\boldsymbol{f}$ solves the forward model in Eq. (3.1), i.e.

$$\begin{aligned}\mathcal{C} = \sum_{i=1}^{N} \Bigg(\sum_{k} \Big(f_i^*\big(\boldsymbol{x}^{(k)}, t^{(j+1)}\big) - f_i\big(\boldsymbol{x}^{(k)}, t^{(j+1)}\big) \Big)^2 \\ + \frac{1}{\Delta\boldsymbol{x}\Delta t} \int \lambda_i(\boldsymbol{x}, t) \mathcal{L}_i[\boldsymbol{f}(\boldsymbol{x}, t)] d\boldsymbol{x} dt \Bigg) + \epsilon_0 ||\boldsymbol{\alpha}||_2^2 \ .\end{aligned} \tag{3.3}$$

where $\Delta\boldsymbol{x}$ and Δt denotes grid spacing in Ω and step size in t, respectively, $||.||_2$ denotes L_2-norm, and ϵ_0 is the regularization factor. We note that PDE discovery task is ill-posed since the underlying PDE is not unique and the regularization term helps us find the PDE with the least possible coefficients.

Clearly, given estimates of $\boldsymbol{f}$ and Lagrange multipliers $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_N)$, the gradient of the cost function with respect to model parameters can be simply computed via

$$\frac{\partial \mathcal{C}}{\partial \alpha_{i,\boldsymbol{d},\boldsymbol{p}}} = (-1)^{|\boldsymbol{d}|} \frac{1}{\Delta\boldsymbol{x}\Delta t} \int \boldsymbol{f}^{\boldsymbol{p}} \nabla_{\boldsymbol{x}}^{(\boldsymbol{d})}[\lambda_i] d\boldsymbol{x} dt + 2\epsilon_0 \alpha_{i,\boldsymbol{d},\boldsymbol{p}} \tag{3.4}$$

where $i = 1, ..., N$ and $|\boldsymbol{d}| = d_1 + ... + d_n$, where $|\cdot|$ denotes L_1-norm. Here, we used integration by parts and imposed the condition that $\boldsymbol{\lambda} \to \boldsymbol{0}$ on the boundaries of Ω at all time $t \in [0, T]$. The analytical expression (3.4) can be used for finding the parameters of PDE using in the gradient descent method with update rule

$$\alpha_{i,\boldsymbol{d},\boldsymbol{p}} \leftarrow \alpha_{i,\boldsymbol{d},\boldsymbol{p}} - \eta \frac{\partial \mathcal{C}}{\partial \alpha_{i,\boldsymbol{d},\boldsymbol{p}}} \tag{3.5}$$

for $i = 1, \ldots, N$, where $\eta = \beta \min(\Delta \boldsymbol{x})^{|\boldsymbol{d}|-d_{\max}}$ is the learning rate which includes a free parameter β and scaling coefficient for each term of the PDE, and $d_{\max} = \max(|\boldsymbol{d}|)$ for all considered $\boldsymbol{d}$. Let us also define $p_{\max} = \max(|\boldsymbol{p}|)$ as the highest order in the forward PDE model. We note that since the terms of the PDEs may have different scaling, the step size for the corresponding coefficient must be adjusted accordingly.

However, before we can use Eq. (3.4) and (3.5), we need to find $\boldsymbol{\lambda}$, hence the adjoint equation. This can be achieved by finding the functional extremum of the cost functional $\mathcal{C}$ with respect to $\boldsymbol{f}$. First, we note that the semi-descrete total variation of $\mathcal{C}$ can be derived as

$$\begin{aligned}\delta\mathcal{C} = &\sum_{i=1}^{N} \Bigg(- \sum_{k} 2\Big(f_i^*\big(\boldsymbol{x}^{(k)}, t^{(j+1)}\big) - f_i\big(\boldsymbol{x}^{(k)}, t^{(j+1)}\big)\Big)\delta f_{i,\boldsymbol{x}^{(k)},t^{(j+1)}} \\ &+ \frac{1}{\Delta \boldsymbol{x} \Delta t} \int \Bigg(- \frac{\partial \lambda_i}{\partial t} + \sum_{\boldsymbol{d},\boldsymbol{p}} (-1)^{|\boldsymbol{d}|} \alpha_{i,\boldsymbol{d},\boldsymbol{p}} \nabla_{f_i}[\boldsymbol{f}^{\boldsymbol{p}}] \nabla_{\boldsymbol{x}}^{(\boldsymbol{d})}[\lambda_i] \Bigg) \delta f_i d\boldsymbol{x} dt \\ &+ \sum_{k} \lambda_i\big(\boldsymbol{x}^{(k)}, t^{(j+1)}\big)\delta f_{i,\boldsymbol{x}^{(k)},t^{(j+1)}} \Bigg)\end{aligned} \tag{3.6}$$

where δf_i denotes variation with respect to $f_i(\boldsymbol{x}, t)$, and $\delta f_{i,\boldsymbol{x}^{(k)},t^{(j+1)}}$ variation with respect to $f_i\big(\boldsymbol{x} = \boldsymbol{x}^{(k)}, t = t^{(j+1)}\big)$. In this derivation, we descretized the last integral resulting from integration by parts in time using the same mesh as the one of data $\mathcal{G}^{(j+1)}$. Here again, we used integration by parts and imposed the condition that $\boldsymbol{\lambda} \to \boldsymbol{0}$ on the boundaries of Ω at all time $t \in \big[t^{(j)}, t^{(j+1)}\big]$ for $i = 1, ..., N$. Note that $f_i(\boldsymbol{x}, t)$ is the output of i-th PDE.

Next, we find the optimums of $\mathcal{C}$ (and hence the adjoint equations) by taking the variational derivatives with respect to f_i and $f_{i,\boldsymbol{x}^{(k)},t^{(j+1)}}$, i.e.

$$\frac{\delta \mathcal{C}}{\delta f_i} = 0 \implies \frac{\partial \lambda_i}{\partial t} = \sum_{\boldsymbol{d},\boldsymbol{p}} (-1)^{|\boldsymbol{d}|} \alpha_{i,\boldsymbol{d},\boldsymbol{p}} \nabla_{f_i}[\boldsymbol{f}^{\boldsymbol{p}}] \nabla_{\boldsymbol{x}}^{(\boldsymbol{d})}[\lambda_i] \tag{3.7}$$

and

$$\frac{\delta \mathcal{C}}{\delta f_{i,\boldsymbol{x}^{(k)},t^{(j+1)}}} = 0 \implies \lambda_i\big(\boldsymbol{x}^{(k)}, t^{(j+1)}\big) = 2\Big(f_i^*\big(\boldsymbol{x}^{(k)}, t^{(j+1)}\big) - f_i\big(\boldsymbol{x}^{(k)}, t^{(j+1)}\big)\Big) \tag{3.8}$$

for $i = 1, ..., N$ and $j = 0, ..., N_t - 1$. We note that the adjoint equation (3.7) for the system of PDEs is backward in time with the final condition at the time $t = t^{(j+1)}$ given by Eq. (3.8). In order to make the notation clear, we present examples for deriving the adjoint equations in Appendix B.1. The adjoint equation is in the continuous form, while the final condition is on the discrete points, i.e. on the grid $\mathcal{G}^{(j+1)}$. In order to obtain the Lagrange multipliers in $t \in \big[t^{(j+1)}, t^{(j)}\big)$, a numerical method appropriate for the forward (3.1) and adjoint equation (3.7) should be deployed. Furthermore, the adjoint equation should have the same or coarser spatial discretization as $\mathcal{G}^{(j+1)}$ to enforce the final condition (3.8).

Training with smooth data set. The training procedure follows the standard gradient descent method. We start by taking an initial guess for parameters $\boldsymbol{\alpha}$, e.g. here we take $\boldsymbol{\alpha} = 0$ initially. For each time interval $t \in \left[t^{(j)}, t^{(j+1)}\right]$, first we solve the forward model (3.1) numerically to estimate $\boldsymbol{f}\left(\boldsymbol{x}^{(k)}, t^{(j+1)}\right)$ given the initial condition

$$\boldsymbol{f}\left(\boldsymbol{x}^{(k)}, t^{(j)}\right) = \boldsymbol{f}^*\left(\boldsymbol{x}^{(k)}, t^{(j)}\right) . \tag{3.9}$$

Then, the adjoint Eq. (3.7) is solved backwards in time with the final time condition (3.8). Finally, the estimate for parameters of the model is updated using Eq. (3.5). We repeat this for all time intervals $j = 0, ..., N_t - 1$ until convergence. In order to improve the search for coefficients and enforce the PDE identification, we also deploy thresholding, i.e. set $\alpha_{i,\boldsymbol{d},\boldsymbol{p}} = 0$ if $|\alpha_{i,\boldsymbol{d},\boldsymbol{p}}| < \sigma$ where σ is a user-defined threshold, during and at the end of training, respectively. In Algorithm 2, we present a pseudocode for finding the parameters of the system of PDEs using the Adjoint method, as shown in the flowchart of Fig. 3.1.

We note that the type of guessed PDE may change during the training, which adds numerical complexity to the optimization and motivates the use of an appropriate solver for each type of guessed PDE, e.g. Finite Volume method for hyperbolic and Finite Element method for Elliptic PDEs. For simplicity, in this work we use the second order Finite Difference method across the board to estimate the spatial and Euler for the time derivative with small enough time step sizes in solving the forward/backward equations to avoid blowups due to possible instabilities.

Algorithm 2: Finding a system of PDEs using Adjoint method. Default threshold $\sigma = 10^{-3}$ applied after $N_{\text{thr}} = 100$ iterations, with tolerance $\gamma = 10^{-9}$ and regularization factor $\epsilon_0 = 10^{-9}$.

Input: data $\boldsymbol{f}^*$, learning rate η, tolerance γ, threshold σ applied after N_{thr}, and ϵ_0
Initialize the parameters $\boldsymbol{\alpha} = \mathbf{0}$;
repeat
  for $j = 0, \ldots, N_t - 1$ **do**
    Estimate $\boldsymbol{f}$ in $t \in \left(t^{(j)}, t^{(j+1)}\right]$ by solving the forward model in Eq. (3.1) given the initial condition Eq. (3.9);
    Find $\boldsymbol{\lambda}$ in $t \in \left[t^{(j)}, t^{(j+1)}\right)$ by solving the adjoint equation in Eq. (3.7);
    Compute the gradient using Eq. (3.4);
    Update parameters $\boldsymbol{\alpha}$ using Eq. (3.5);
  end
  if Epochs $> N_{\text{thr}}$ **then**
    Thresholding: set $\alpha_i = 0$ for all i that $|\alpha_i| < \sigma$;
  end
until *Convergence in* $\boldsymbol{\alpha}$ *with tolerance* γ;
Thresholding: set $\alpha_i = 0$ for all i that $|\alpha_i| < \sigma$;
Output: $\boldsymbol{\alpha}$

Training with noisy data set. Often the data set comes with some noise. There are several pre-processing steps that can be done to reduce the noise at the expense of introducing bias, for example removing high frequencies using Fast Fourier Transform or removing small singular values from data set using Singular Value Decomposition. However, we can also reduce the sensitivity of the training algorithm to the noise by averaging the gradients before updating the parameters. Assuming that the noise is martingale, the Monte Carlo averaging gives us the unbiased estimator for the expected value of the gradient over all the data set. We adapt the training procedure by averaging gradients over all available data points and then updating the parameters (see Algorithm 3 and the flowchart in Fig. 3.1 for more details). Clearly, this will make the algorithm more robust at higher cost since the update happens only after seeing all the data.

Algorithm 3: Finding a system of PDEs using Adjoint method with averaging for the computation of gradients over data set. Default threshold $\sigma = 10^{-3}$ applied after $N_{\text{thr}} = 100$ iterations, with tolerance $\gamma = 10^{-9}$ and regularization factor $\epsilon_0 = 10^{-9}$.

Input: data $\boldsymbol{f}^*$, learning rate η, tolerance γ, threshold σ applied after N_{thr}, and ϵ_0
Initialize the parameters $\boldsymbol{\alpha} = \mathbf{0}$;
repeat
  for $j = 0, \ldots, N_t - 1$ **do**
    Estimate $\boldsymbol{f}$ in $t \in \left(t^{(j)}, t^{(j+1)}\right]$ by solving the forward model in Eq. (3.1) given the initial condition in Eq. (3.9);
    Find $\boldsymbol{\lambda}$ in $t \in \left[t^{(j)}, t^{(j+1)}\right)$ by solving the adjoint equation in Eq. (3.7);
    Compute the gradient $\boldsymbol{g}^{(j)} = \partial C^{(j)}/\partial \boldsymbol{\alpha}$ using Eq. (3.4);
  end
  Average the gradient $\mathbb{E}[\partial C/\partial \boldsymbol{\alpha}] = \sum_j \boldsymbol{g}^{(j)}/N_t$;
  Update parameters $\boldsymbol{\alpha}$ with Eq. (3.5) using $\mathbb{E}[\partial C/\partial \boldsymbol{\alpha}]$;
  if Epochs $> N_{\text{thr}}$ **then**
    Thresholding: set $\alpha_i = 0$ for all i that $|\alpha_i| < \sigma$;
  end
until *Convergence in* $\boldsymbol{\alpha}$ *with tolerance* γ;
Thresholding: set $\alpha_i = 0$ for all i that $|\alpha_i| < \sigma$;
Output: $\boldsymbol{\alpha}$

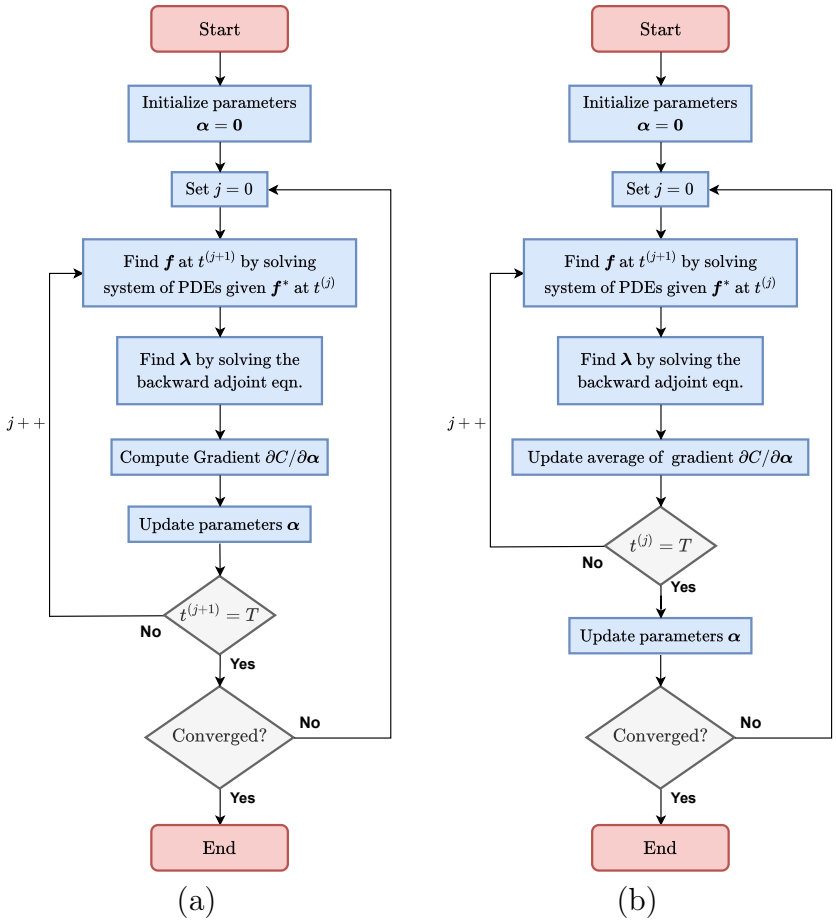


Figure 3.1: Training flowchart of the Adjoint method in finding PDEs (a) without and (b) with gradient averaging.

3.3 Results

We demonstrate the validity of our proposed adjoint-based method in discovering PDEs given measurements on a spatial-temporal grid. We consider data collected from a variety of systems. We mainly compare our approach to PDE-FIND in terms of error and time to convergence. All the results are obtained using a single core-thread of a 2.3 GHz Quad-Core Intel Core i7 CPU. We report the execution time τ obtained by averaging over 10 independent runs along with their error bars.

3.3.1 Heat equation

As a first example, let us consider measured data collected from the solution to the heat equation, i.e.

$$\frac{\partial f}{\partial t} + D\,\frac{\partial^2 f}{\partial x^2} = 0, \tag{3.10}$$

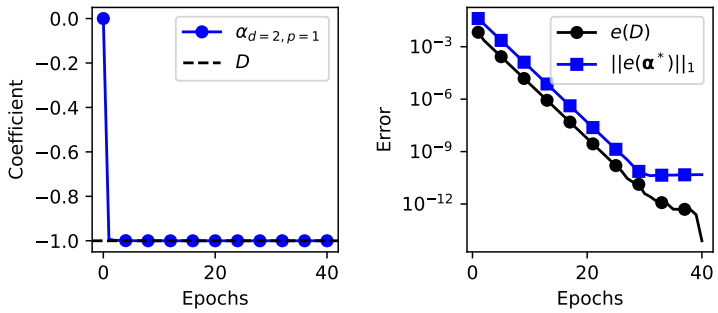


Figure 3.2: The estimated coefficient corresponding to D (left) and the L_1-norm error of all considered coefficients (right) given the discretized data of the heat equation during training.

with $D = -1$. The data is constructed using finite difference with initial condition $f(x, 0) = 5\sin(2\pi x)x(x - L)$ and a mesh with $N_x = 100$ nodes in x covering the domain $\Omega = [0, L]$ with $L = 1$ and $N_t = 100$ steps in t with final time $T = N_t\Delta t$ where $\Delta t = 0.05\Delta x^2/(1 + |D|)$ is the step size and $\Delta x = L/N_x$ is the mesh size in x.

We consider a system consisting of a single PDE (i.e. $N = 1$, $\boldsymbol{f} = f$, and $\boldsymbol{p} = p$) with one-dimensional input, i.e. $n = 1$ and $\boldsymbol{x} = x \subset \mathbb{R}$, and $\boldsymbol{d} = d \in \mathbb{N}$. In order to construct a general forward model, here we consider derivatives and polynomials with indices $d, p \in \{1, 2, 3\}$ as the initial guess for the forward model. This leads to 9 terms with unknown coefficients $\boldsymbol{\alpha}$ that we find using the proposed adjoint method (an illustrative derivation of the candidate terms can be found in Appendix B.1.1). While we expect to recover the coefficient that corresponds to D, we expect all the other coefficients (denoted by $\boldsymbol{\alpha}^*$) to become negligible. That is what we indeed observe in Fig. 3.2 where the error of the coefficient for each term is plotted against the number of epochs.
Next, we compare the solution obtained via the adjoint method against PDE-FIND with STRidge optimization method. Here, we test both methods in recovering the Heat equation given data on the grid with discretization $(N_t, N_x) \in \{(100, 100), (500, 100), (1000, 100), (1000, 1000)\}$. As shown in Fig. 3.3, the proposed adjoint method provides more accurate results across all data sizes. We also point out that as the size of the data set increases, PDE-FIND with STRidge regression method becomes more expensive, e.g. one order of magnitude more expensive than the adjoint method for the data on a grid size $(N_t, N_x) = (1000, 1000)$.

3.3.2 Burgers' equation

As a nonlinear test case, let us consider the data from Burgers' equation given by

$$\frac{\partial f}{\partial t} + \frac{\partial (Af^2)}{\partial x} = 0 \tag{3.11}$$

where $A = -1$. The data is obtained with similar simulation setup as for Heat equation (Section 3.3.1) except for the time step, i.e. $\Delta t = 0.05\Delta x/(1 + |A|)$.

Similar to Section 3.3.1, we adopt a system of one PDE with one-dimensional input. We also consider derivatives and polynomials with indices $d, p \in \{1, 2, 3\}$ in the construction of the forward model. This leads to 9 terms whose coefficients we find using the proposed adjoint

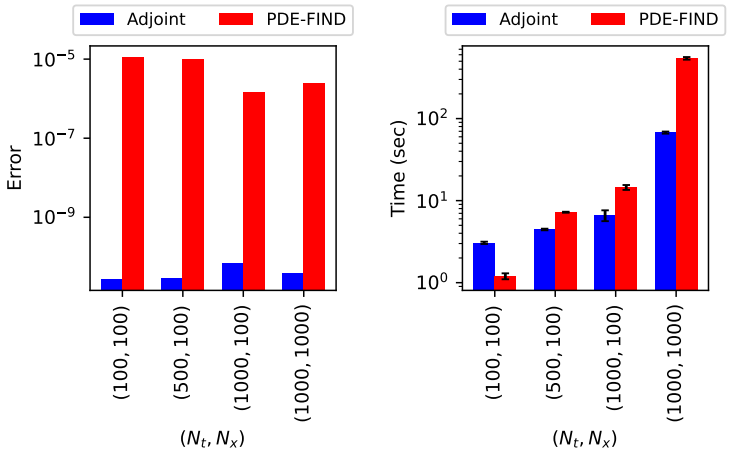


Figure 3.3: L_1-norm error of the estimated coefficients (left) and the execution time (right) for discovering the heat equation equation using the proposed Adjoint method (blue) and PDE-FIND method (red), given data on a grid with $N_t \in \{100, 500, 1000\}$ steps in t, and $N_x \in \{100, 1000\}$ nodes in x.

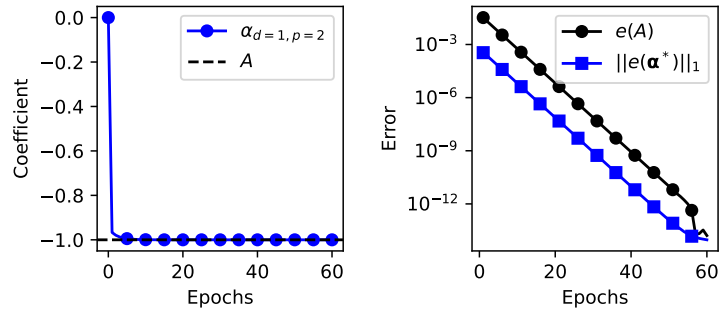


Figure 3.4: The estimated coefficient corresponding to A (left) and the L_1-norm error of all considered coefficients (right) given the discretized data of Burgers' equation during training.

method. As shown in Fig. 3.4, the proposed adjoint method finds the correct coefficients, i.e. $\alpha_{d=1,p=2}$ that corresponds to D as well as all the irrelevant ones denoted by $\boldsymbol{\alpha}^*$, up to machine accuracy in $\mathcal{O}(10)$ epochs.

Next, we compare the solution obtained from the adjoint method to the one from PDE-FIND using STRidge optimization method. Here, we compare the error on coefficients and computational time between the adjoint and PDE-FIND by repeating the task for the data set with increasing size, i.e. $(N_t, N_x) \in \{(100, 100),\ (1000, 100),\ (1000,\ 1000)\}$. As depicted in Fig. 3.5, the adjoint method provides us with more accurate solution across the different discretization sizes. Regarding the computational cost, while PDE-FIND seems faster on smaller data sets, as the size of the data grows, it becomes increasingly more expensive than the adjoint method. Similar to the heat equation, for the mesh size $(N_t, N_x) = (1000,\ 1000)$ we obtain one order of magnitude speed up compared to PDE-FIND.

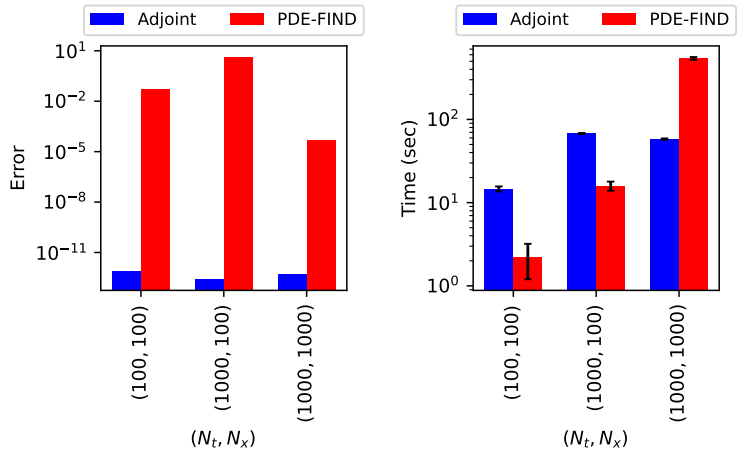


Figure 3.5: L_1-norm error of the estimated coefficients (left) and the execution time (right) for discovering the Burgers' equation equation using the Adjoint method (blue) and PDE-FIND method (red), given data on a grid with $N_t \in \{100, 1000\}$ steps in t, and $N_x \in \{100, 1000\}$ nodes in x.

3.3.3 Kuramoto Sivashinsky equation

As a more challenging test case, let us consider the recovery of the Kuramoto-Sivashinsky (KS) equation given by

$$\frac{\partial f}{\partial t} + A\frac{\partial f^2}{\partial x} + B\frac{\partial^2 f}{\partial x^2} + C\frac{\partial^4 f}{\partial x^4} = 0 \tag{3.12}$$

where $A = -1$, $B = 0.5$ and $C = -0.5$. The data is generated similar to previous sections except for the grid $(N_t, N_x) = (64, 256)$ and the time step size $\Delta t = 0.01\Delta x^4/(1 + |C|)$.

Here again, we adopt a system of one PDE with one-dimensional input. As a guess for the forward model, we consider terms consisting of derivatives with indices $d \in \{1, 2, 3, 4\}$ and polynomials with indices $b \in \{1, 2\}$, leading to 8 terms whose coefficients we find using the proposed adjoint method. As shown in Fig. 3.6, the adjoint method finds the coefficient with error of $\mathcal{O}(10^{-5})$, yet achieving machine accuracy seems not possible.

Again, in Fig. 3.7 we make a comparison between the predicted PDE using the adjoint method against PDE-FIND. In particular, we consider a data set on a temporal/spatial mesh of size $(N_t, N_x) = \{(64, 256),\ (128, 512),\ (256, 1024)\}$ and compare how the error and computational cost vary. Similar to previous sections, the error is reported by comparing the obtained coefficients against the coefficients of the exact PDE in L_1-norm. Interestingly, the PDE-FIND method has 3 to 4 orders of magnitude larger error compared to the adjoint method. Also, in terms of cost, the training time for PDE-FIND seems to grow at a higher rate than the adjoint method as the (data) mesh size increases.

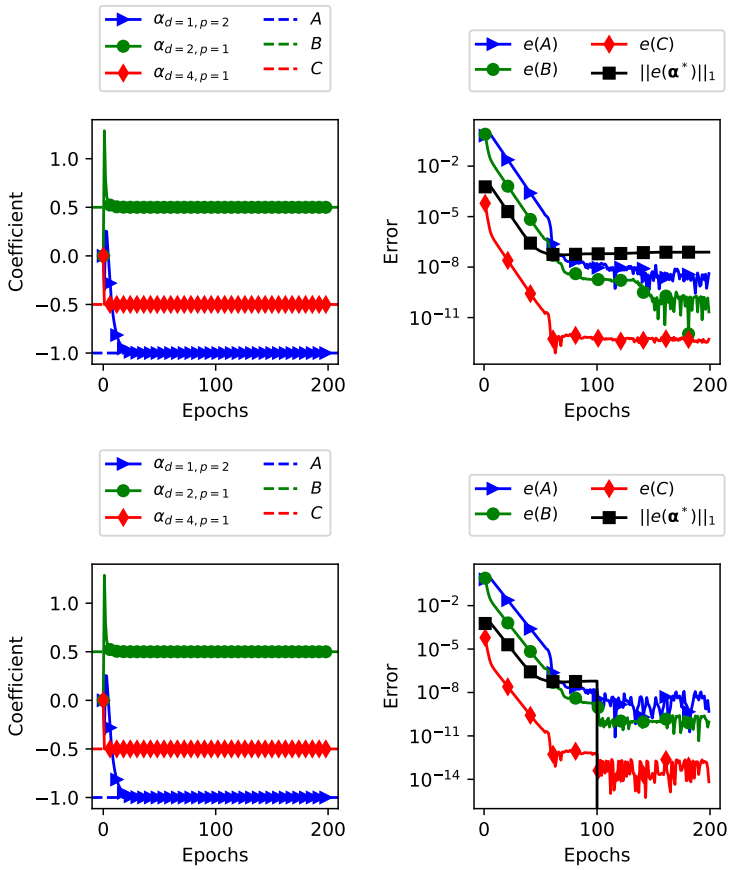


Figure 3.6: The estimated coefficients corresponding to A, B, C (left) and the L_1-norm error of all considered coefficients (right) given the discretized data of the KS equation during training without (top) and with (bottom) active thresholding for Epochs $> N_{\text{thr}} = 100$.

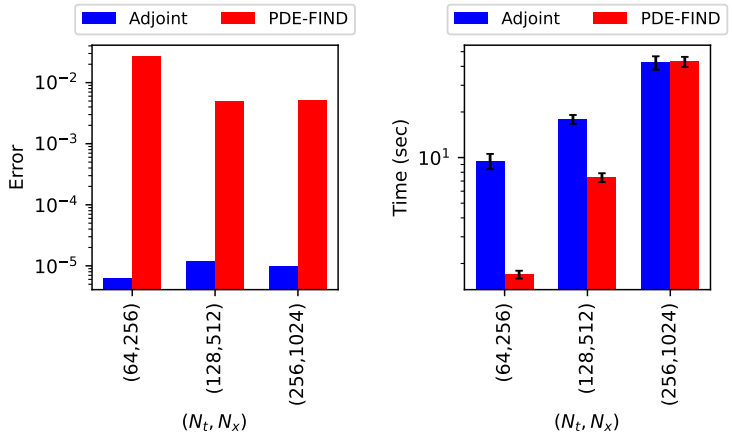


Figure 3.7: L_1-norm error of the estimated coefficients (left) and the execution time (right) for discovering the KS equation using the Adjoint method (blue) and PDE-FIND method (red), given data on a grid with $N_t \in \{64, 128, 256\}$ steps in t, $N_x \in \{256, 512, 1024\}$ nodes in x.

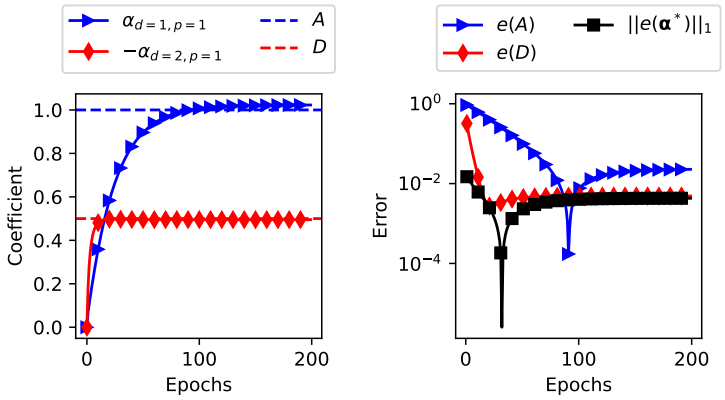


Figure 3.8: The estimated coefficients corresponding to A and D (left) and the L_1-norm error of all considered coefficients (right) of the Fokker-Planck equation as the governing law for the PDF associated with the random walk during training.

3.3.4 Random Walk

Next, let us consider the recovery of the governing equation on probability density function (PDF) given samples of its underlying stochastic process. As an example, we consider the Itô process

$$dX = Adt + \sqrt{2D}dW \tag{3.13}$$

where $A = 1$ is drift and $D = 0.5$ is the diffusion coefficient, and W denotes the standard Wiener process with $\text{Var}(dW) = \Delta t$. We generate the data set by simulating the random walk using Euler-Maruyama scheme starting from $X(t = 0) = 0$ for $N_t = 50$ steps with a time step size of $\Delta t = 0.01$. We estimate the PDF using histogram with $N_x = 100$ bins and $N_s = 1000$ samples.

Let us denote the distribution of X by f. Itô's lemma gives us the Fokker-Planck equation

$$\frac{\partial f}{\partial t} + A\frac{\partial f}{\partial x} - D\frac{\partial^2 f}{\partial x^2} = 0 \ . \tag{3.14}$$

Given the data set for f on a mesh of size (N_t, N_x), we can use finite difference to compute the contributions from derivatives of f in the governing law. Since this is one of the challenging test cases due to noise, here we only consider three possible terms in the forward model, consisting of derivatives with indices $d \in \{1, 2, 3\}$ and polynomial power $p = 1$. In Fig. 3.8, we show how the error of finding the correct coefficients evolves during training for the adjoint method. Clearly, the adjoint method seems to recover the true PDE with L_1 error of $\mathcal{O}(10^{-2})$ in its coefficients.

In Fig. 3.9, we make a comparison with PDE-FIND for the same number and order of terms as the initial guess for the PDE. We compare the two methods for a range of grid and sample sizes, i.e. $N_t \in \{50, 100\}$, $N_x = 100$, and $N_s \in \{10^3,\ 10^4\}$. It turns out that the proposed adjoint method overall provides more accurate estimate of the coefficients than

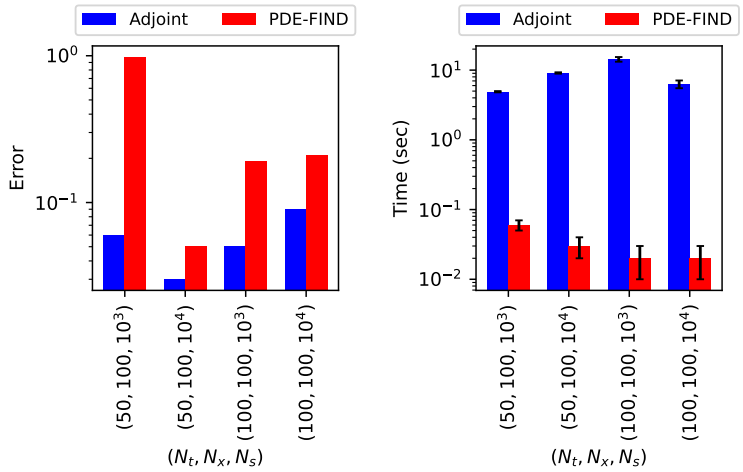


Figure 3.9: L_1-norm error of the estimated coefficients (left) and the execution time (right) for discovering the Fokker-Planck equation using the proposed Adjoint method (blue) and PDE-FIND method (red), given samples of its underlying stochastic process with $N_t \in \{50, 100\}$ steps in t, $N_x = 100$ histogram bins, and $N_s \in \{10^3, 10^4\}$ samples.

PDE-FIND, though at a higher cost. In Table 3.1, we show the discovered PDEs for both methods across the different discretizations.

Table 3.1: Recovery of the Fokker-Planck equation, i.e. $f_t + f_x - 0.5 f_{xx} = 0$, using the proposed adjoint method against PDE-FIND method given samples of the underlying stochastic process for various discretization parameters.

N_t	N_x	N_s	Method	Recovered PDE
50	100	1000	Adjoint	$f_t + 1.025 f_x - 0.465 f_{xx} = 0$
			PDE-FIND	$f_t + 0.798 f_x - 0.454 f_{xx} = 0$
		10000	Adjoint	$f_t + 1.022 f_x - 0.495 f_{xx} = 0$
			PDE-FIND	$f_t + 0.818 f_x - 0.496 f_{xx} = 0$
100	100	1000	Adjoint	$f_t + 1.010 f_x - 0.543 f_{xx} = 0$
			PDE-FIND	$f_t + 0.863 f_x - 0.560 f_{xx} = 0$
		10000	Adjoint	$f_t + 1.015 f_x - 0.589 f_{xx} = 0$
			PDE-FIND	$f_t + 0.894 f_x - 0.612 f_{xx} = 0$

3.3.5 Reaction Diffusion System of Equations

In order to show scalability and accuracy of the adjoint method for a system of PDEs in a higher dimensional space, let us consider a system of PDEs given by

$$\frac{\partial u}{\partial t} + c_0^u \nabla_{x_1}^2[u] + c_1^u \nabla_{x_2}^2[u] + R^u(u, v) = 0, \tag{3.15}$$

$$\frac{\partial v}{\partial t} + c_0^v \nabla_{x_1}^2[v] + c_1^v \nabla_{x_2}^2[v] + R^v(u, v) = 0 \tag{3.16}$$

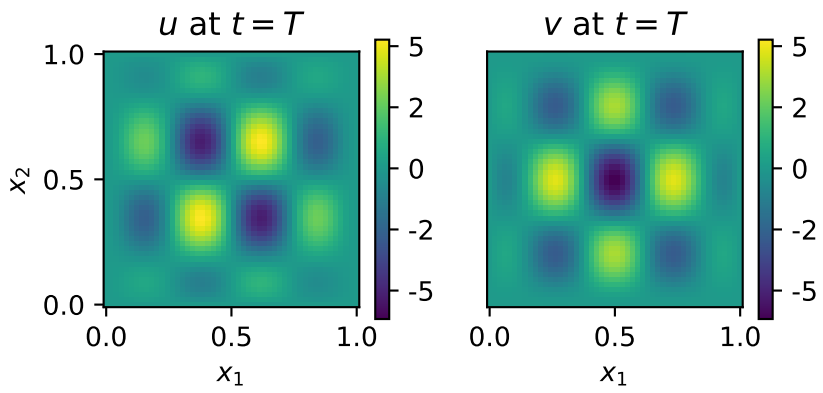


Figure 3.10: Solution to the reaction diffusion system of PDEs at time $t = T$ for u (left) and v (right).

where

$$R^u(u,v) = c_2^u u + c_3^u u^3 + c_4^u uv^2 + c_5^u u^2 v + c_6^u v^3 \tag{3.17}$$
$$R^v(u,v) = c_2^v v + c_3^v v^3 + c_4^v vu^2 + c_5^v v^2 u + c_6^v u^3 \tag{3.18}$$

We construct the data set by solving the system of PDEs Eqs. (3.15)-(3.16) using a 2nd order finite difference scheme with initial values

$$u_0 = a \sin\left(\frac{4\pi x_1}{L_1}\right) \cos\left(\frac{3\pi x_2}{L_2}\right) \left(L_1 x_1 - x_1^2\right)\left(L_2 x_2 - x_2^2\right)$$
$$v_0 = a \cos\left(\frac{4\pi x_1}{L_1}\right) \sin\left(\frac{3\pi x_2}{L_2}\right) \left(L_1 x_1 - x_1^2\right)\left(L_2 x_2 - x_2^2\right)$$

where $a = 100$, and the coefficients

$$\boldsymbol{c}^u = [c_i^u]_{i=0}^6 = [-0.1, -0.2, -0.3, -0.4, 0.1, 0.2, 0.3]$$
$$\boldsymbol{c}^v = [c_i^v]_{i=0}^6 = [-0.4, -0.3, -0.2, -0.1, 0.3, 0.2, 0.1].$$

We generate data by solving the system of PDEs Eqs. (3.15)-(3.16) using the finite difference method and forward Euler scheme for $N_t = 25$ steps with a time step size of $\Delta t = 10^{-6}$, and in the domain $\Omega = [0, L_1] \times [0, L_2]$ where $L_1 = L_2 = 1$ which is discretized using a uniform grid with $N_{x_1} \times N_{x_2} = 50^2$ nodes leading to mesh size $\Delta x_1 = \Delta x_2 = 0.02$. In Fig. 3.10 we show the solution to the system at time $T = N_t \Delta t$ for u and v.

We consider a system consisting of two PDEs, i.e. $N = \dim(\boldsymbol{f}) = \dim(\boldsymbol{p}) = 2$, with two-dimensional input, i.e. $n = \dim(\boldsymbol{x}) = \dim(\boldsymbol{d}) = 2$. Here, $\dim(\boldsymbol{f}) = \dim(\mathrm{Ima}(\boldsymbol{f}))$, where $\mathrm{Ima}(\cdot)$ denotes the image (or output) of a function.

In order to use the developed adjoint method, we construct a guess forward system of PDEs (or forward model) using derivatives up to 2nd order and polynomials of up to 3rd order. That is, $d_{\max} = 2$ and $p_{\max} = 3$. This leads to 90 terms whose coefficients we find using the proposed adjoint method (an illustrative derivation of the candidate terms can be found in Appendix B.1.2). The solution to the constructed model $\boldsymbol{f} \approx [u, v]$ as well as the adjoint equation for $\boldsymbol{\lambda}$ is found using the same discretization as the data set.

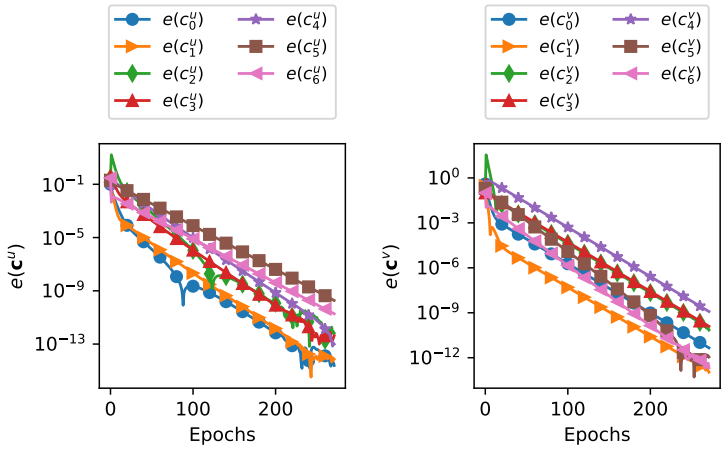


Figure 3.11: L_1-norm error in the estimated coefficients of the reaction diffusion system of PDEs during training.

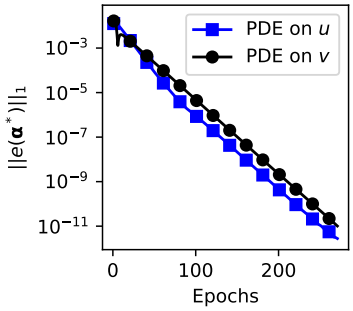


Figure 3.12: L_1-norm error in the estimated coefficients of the irrelevant terms compared to the true reaction diffusion system of PDEs during training, i.e. $||e(\boldsymbol{\alpha}^*)||_1 = ||\boldsymbol{\alpha}^*||_1$.

As shown in Fig. 3.11, the adjoint method finds the correct equations with error up to $\mathcal{O}(10^{-12})$. Furthermore, the coefficients corresponding to the irrelevant terms $\boldsymbol{\alpha}^*$ tend to zero with error of $\mathcal{O}(10^{-11})$, see Fig. 3.12.

Furthermore, we have compared the adjoint method against PDE-FIND for a range of grid sizes in Fig. 3.13. We observe that the cost of PDE-FIND grows with higher rate than adjoint method as the size of the data set increases.

3.4 Partial observations in time

Here, we investigate how the error of the discovery task increases when only a subset of the fine data set is available. Consider the heat equation presented in section 3.3.1 and consider a data set created by solving the exact PDE using the Finite Difference method with $\Delta t = T/N_t$ where $N_t = 1000$ and $\Delta x = L/N_x$ and $N_x = 1000$.

Let us assume that we are only provided with a subset of this data set. As a test, let us take every α time step as the input for the PDE discovery task, where $\alpha \in \{1, 2, 4, 8, 16\}$. This corresponds to using $\{100, 50, 25, 12.5, 6.25\}\%$ of the total data set. By doing so, the

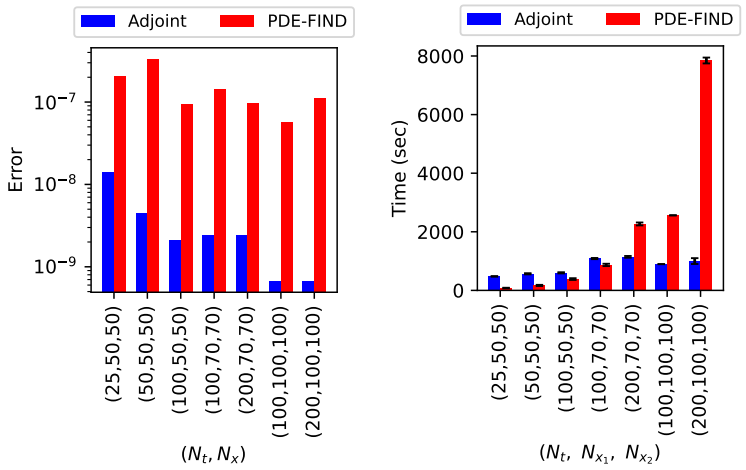


Figure 3.13: *Reaction diffusion system of PDEs.* Comparing the error and execution time of the adjoint method (blue) to PDE-FIND method (red) against the size of the data set for the tolerance of 10^{-7} in the discovered coefficients.

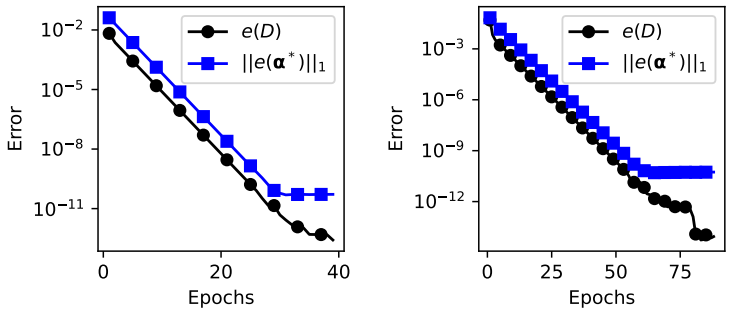


Figure 3.14: *Heat equation.* Evolution of the L_1-norm error in coefficients of all considered terms using adjoint method when only 50% (left) and 6.25% (right) of the data set is available. Even with fewer time observations, the proposed method accurately recovers the appropriate coefficients, though it requires more epochs.

accuracy of the Finite Difference method in estimating the time derivatives using the available data deteriorates, leading to large error in PDE discover task for PDE-FIND method.

However, the adjoint method can use a finer mesh in time compared to the data set in computing the forward and backward equations and only compare the solution to the data on the coarse mesh where data is available. We use $N_t = 1000$ for the forward and backward solvers in the adjoint method, and impose the final time condition where data is available. As shown in Table 3.2, Fig. 3.14, and Fig. 3.15, the proposed adjoint method is able to recover the exact PDE regardless of how sparse the data set is in time.

We emphasize that while adjoint method can use a finer discretization in time than the one for data on $\mathcal{G}$ in solving forward and backward equations, it is bound to use similar or coarser spatial discretization as $\mathcal{G}$. This is due to the fact that the data points $\boldsymbol{f}^*$ are used for the initial condition of the forward model eq. (3.9), and the final condition of the backward adjoint equation eq. (3.8).

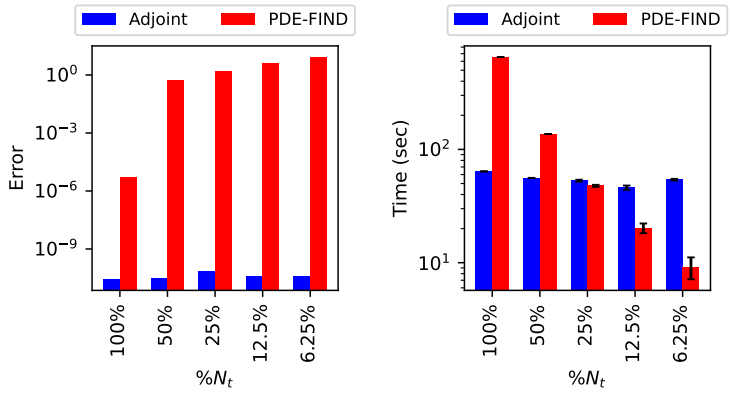


Figure 3.15: Error and execution time of the adjoint method (blue) and PDE-FIND method (red) in finding the coefficients of true heat equation given sparse data set in time. The proposed adjoint method consistently recovers the correct coefficients, even with limited observations in time.

Table 3.2: Comparing the proposed adjoint method and PDE FIND in recovering the Heat equation given sparse data set in time. Here we rounded the coefficients up to three decimal places. The proposed method consistently recovers the true PDE, even with fewer time observations.

$\%\boldsymbol{N_t}$	**Method**	**Recovered PDE**
100	Adjoint	$f_t - f_{xx} = 0$
	PDE-FIND	$f_t - f_{xx} = 0$
50	Adjoint	$f_t - f_{xx} = 0$
	PDE-FIND	$f_t - 0.999f_{xx} + 0.177f - 0.261f^3 - 0.089ff_x - 0.011f^3f_x - 0.003f^2f_{xx} - 0.001ff_{xxx} = 0$
25	Adjoint	$f_t - f_{xx} = 0$
	PDE-FIND	$f_t - 0.999f_{xx} + 0.532f - 0.778f^3 - 0.268ff_x - 0.035f^3f_x - 0.010f^2f_{xx} - 0.003ff_{xxx} = 0$
12.5	Adjoint	$f_t - f_{xx} = 0$
	PDE-FIND	$f_t - 0.999f_{xx} + 1.264f - 1.863f^3 - 0.638ff_x - 0.081f^3f_x - 0.025f^2f_{xx} - 0.007ff_{xxx} - 0.001f^3f_{xxx} = 0$
6.25	Adjoint	$f_t - f_{xx} = 0$
	PDE-FIND	$f_t - 0.999f_{xx} + 2.769f - 4.051f^3 - 1.398ff_x - 0.185f^3f_x - 0.055f^2f_{xx} - 0.016ff_{xxx} - 0.002f^3f_{xxx} = 0$

3.4.1 Sensitivity to noise

Here, we investigate how the error increases once noise is added to the data set. In particular, we add noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ to each point of the data set for $\boldsymbol{f}^*$, where $\mathcal{N}(0, \sigma^2)$ denotes a normal distribution with zero mean and standard deviation σ. As test cases, we revisit the heat equation (Section 3.3.1) and Burgers' equation (Section 3.3.2) with added noise of ϵ with $\sigma \in \{0.001,\ 0.005,\ 0.01,\ 0.1\}$ %. Before searching for the PDE, we first denoise the data set using Singular Value Decomposition and drop out terms with singular value below a threshold of $\mathcal{O}(10^{-4})$.

As shown in Figure 3.16, adding noise to the data set deteriorates the accuracy in finding the correct coefficients of the underlying PDE for both the adjoint method and PDE-FIND method. We observe that the adjoint method, both with and without gradient averaging, is less susceptible to noise compared to PDE-FIND, albeit at a higher computational cost.

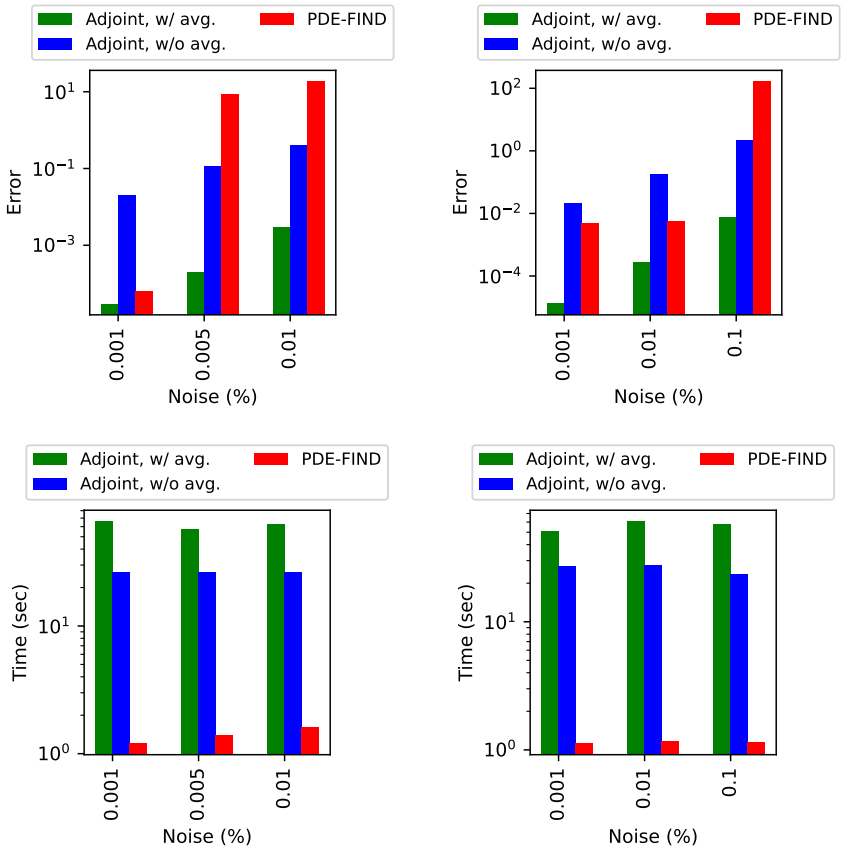


Figure 3.16: Error and execution time of the adjoint method with (green) and without averaging the gradients (blue), along with the PDE-FIND method (red) in finding the coefficients of the true PDE, i.e. the heat equation (left) and Burgers' equation (right), given noisy data. The adjoint method with gradient averaging is more robust to noise, though it comes at a higher computational cost.

Additionally, averaging the gradients in the adjoint method improves the accuracy around two orders of magnitude at higher computational cost.

3.5 Addressing ill-posedness

There may exist more than one PDE that replicates the data set. Therefore, the PDE discovery task is ill-posed due to the lack of uniqueness in the solution. This is an indication that further physically motivated constraints are needed to narrow the search space to find the desired PDE. However, among all possible PDEs, which PDE is found by the Adjoint method with the loss function defined as (3.3)?
To answer this question, let us consider a simple example of the wave equation

$$f(x,t) = \sin(x-t) \tag{3.19}$$

which is a solution to infinite PDEs. For example, one class of PDEs with solution f is

$$f_t + kf_x + (k-1)f_{xxx} + c(f_{xx} + f_{xxxx}) = 0 \quad \forall k \in \mathbb{N} \text{ and } c \in \mathbb{R}\,, \tag{3.20}$$

defined in a domain $x \in [0, 2\pi]$ and $T = 1$. We create a data set using the exact f on a grid with $N_t = 10$ time intervals and $N_x = 100$ spatial discretization points. Let us consider a similar setup as the heat equation example 3.3.1 with derivatives and polynomials indices $d \in \{1, ..., 6\}$ and $p = 1$ as the initial guess for the forward model. This leads to 6 terms with unknown coefficients $\boldsymbol{\alpha}$. Here, we enable averaging and use a finer discretization in time (100 steps for forward and backward solvers in each time interval) to cope with the instabilities of the Finite Difference solver due to the inclusion of the high-order derivatives. We also disable thresholding except at the end of the algorithm.

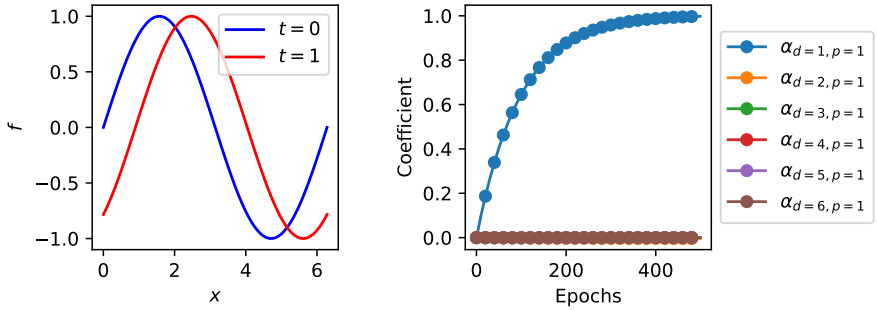


Figure 3.17: Profile of f at $t = 0$ and $t = 1$ (left) and the evolution of considered coefficients during adjoint optimization (right)

As shown in Fig. 3.17, the proposed Adjoint method returns the solution

$$f_t + f_x = 0 \tag{3.21}$$

which is the PDE with the least number of terms compared to all possible PDEs. We note that for the same problem setting, PDE-FIND identifies the same form of the PDE, i.e.

$$f_t + 0.9897 f_x = 0\,. \tag{3.22}$$

The identified form of PDE can be explained by the use of regularization term in the cost function 3.3, which enforces the minimization of the PDE coefficients. Clearly, the regularization term may be changed to find other possible solutions of this ill-posed problem.

3.6 Discussion

Below we highlight and discuss strengths and weaknesses of the proposed adjoint method.

Strengths. The proposed method has several strengths:

1. The proposed adjoint-based method of discovering PDEs can provides coefficients of the true governing equation with significant accuracy.

2. Since the gradient of the cost function with respect to parameters are derived analytically, the optimization problem converges fast. In particular, the adjoint method becomes cheaper than PDE-FIND as the size of the data set increases.

3. Since the adjoint method uses a PDE solver to find the underlying governing equation, there is a guarantee that the found PDE can be solved numerically with the same PDE solver as the one used by the adjoint method.

4. The adjoint method can use a finer mesh in time compared to the available discretization of the data set. This allows an accurate recovery of the underlying PDE compared to the PDE-FIND, where the error in the latter increases as the data set gets coarser since it estimates derivatives directly (either with Finite Difference or a polynomial fit) using the given data set.

Weaknesses. Our proposed method has some limitations:

1. In order to use the proposed adjoint method for discovering PDEs, a general solver of PDEs needs to be implemented. Here, we used Finite Differences which can be replaced with more advance solvers.

2. In this work, we used the same spatial discretization as the input data. If the spatial grid of input data is too coarse for the PDE solver, one has to use interpolation to estimate the data on a finer spatial grid that is more appropriate for the PDE solver.

3. In this work, we made the assumption that the underlying PDE can be solved numerically. This can be a limitation when there are no stable numerical methods to solve the true PDE. In this scenario, the proposed method may find another PDE that is solvable and fits to the data with a notable error.

3.7 Conclusion

In this work, we introduce a novel mathematical method for the discovery of partial differential equations given data using the adjoint method. By formulating the optimization problem in the variational form using the method of Lagrange multipliers, we find an analytic expression for the gradient of the cost function with respect to the parameters of the PDE as a function of the Lagrange multipliers and the forward model estimate. Then, using variational calculus, we find a backwards-in-time evolution equation for the Lagrange multipliers which incorporates the error with a source term (the adjoint equation). Hence, we can use the same solver for both forward model and the backward Lagrange equations. Here we used Finite Differences to estimate the spatial derivatives and forward Euler for the time derivatives, which indeed can be replaced with more stable and advanced solvers.

We compared the proposed adjoint method against PDE-FIND on several test cases. While PDE-FIND seems to be faster for small size problems, we observe that the adjoint method equipped with forward/backward solvers becomes faster than PDE-FIND as the size of the data set increases. Also, the adjoint method can provide machine-accuracy in identifying and finding the coefficients when the data set is noise-free. Furthermore, in the case of discovering PDEs for PDFs given its samples, both methods seem to suffer enormously form noise/bias associated with the finite number of samples and the Finite Difference on histogram. This motivates the use of smooth and least biased density estimator in these methods such as [10] in future work. In the future work, we intend to combine the adjoint-based method for the discovery of PDE with PINNs as the solver instead of Finite Difference method. This would allow us to handle noisy and sparse data as well as deploying larger time steps in estimating the forward and backward solvers.

Chapter 4

MESSY Estimation: Maximum-Entropy based Stochastic and Symbolic densitY Estimation

In this chapter, we introduce MESSY Estimation, a Maximum-Entropy based Stochastic and Symbolic densitY Estimation method. The proposed approach recovers probability density functions symbolically from samples using moments of a Gradient flow in which the ansatz serves as the driving force. In particular, we construct a gradient-based drift-diffusion process that connects samples of the unknown distribution function to a guess symbolic expression. We then show that when the guess distribution has the maximum entropy form, the parameters of this distribution can be found efficiently by solving a linear system of equations constructed using the moments of the provided samples. Furthermore, we use Symbolic regression to explore the space of smooth functions and find optimal basis functions for the exponent of the maximum entropy functional leading to good conditioning. The cost of the proposed method for each set of selected basis functions is linear with the number of samples and quadratic with the number of basis functions. However, the underlying acceptance/rejection procedure for finding optimal and well-conditioned bases adds to the computational cost. We validate the proposed MESSY estimation method against other benchmark methods for the case of a bi-modal and a discontinuous density, as well as a density at the limit of physical realizability. We find that the addition of a symbolic search for basis functions improves the accuracy of the estimation at a reasonable additional computational cost. Our results suggest that the proposed method outperforms existing density recovery methods in the limit of a small to moderate number of samples by providing a low-bias and tractable symbolic description of the unknown density at a reasonable computational cost.

4.1 Introduction

Recovering probability density functions from samples is one of the fundamental problems in statistics with many applications. For example, the traditional task of discovering the underlying dynamics governing the corresponding distribution function is strongly dependent on the quality of the density estimator [7]. Applications include particle physics [100],

boundary conditions for multi-scale kinetic problems [101], [102], and machine learning [103].

Broadly speaking, two categories of methods have been developed for this task: parametric and non-parametric estimators. While parametric methods assume a restrictive ansatz for the underlying distribution function, non-parametric methods provide a more flexible density estimate by performing a kernel integration locally using nearby samples. Although non-parametric methods do not need any prior knowledge of the underlying distribution, they suffer from the unclear choice of kernel and its support leading to bias and lack of moment matching. Examples of non-parametric density estimators include histogram and Kernel Density Estimation (KDE) [104]–[106].

On the other hand, parametric density estimators may allow matching of moments while introducing modeling error, since a guess for the distribution is required. Parametric distributions include Gaussian, orthogonal expansion with respect to Gaussian using Hermite polynomials (also known as Grad's ansatz in kinetic theory) [107]–[109], wavelet density estimation [110], and Maximum Entropy Distribution (MED) [111]–[114] function among others. Given only the mean and variance, information theory provides us with the Gaussian distribution function as the least biased density, which has been used extensively in the literature. However, including higher order moments in a similar way, i.e. *moment problem*, raises further complications. For example in the context of kinetic theory, Grad proposed a closure that incorporates higher-order moments by considering a deviation from Gaussian using Hermite polynomials. Even though the information from higher moments is incorporated as the parameters of the polynomial expansion in Grad's ansatz, such a formulation suffers from not guaranteeing positivity of the estimated density along with the introduction of bias.

Among parametric density estimators, the Maximum Entropy Distribution (MED) function has been proposed in information theory as the least biased density estimate given a number of moments of the unknown distribution [111]. While MED provides the least biased density estimate, it suffers from two limitations. First, the distribution parameters (Lagrange multipliers) can only be found by solving a convex optimization problem with ill-conditioned Hessian [115], [116]. The condition number increases either by increasing the order of the matching moments or approaching the limit of physical realizability which motivated the use of adaptive basis functions [117], [118]. Second, MED only exists and is unique in bounded domains. While existence/uniqueness is guaranteed for recovering the distribution in the subspace occupied by the samples, the computational complexity associated with the direct computation of Lagrange multipliers has prevented researchers from deploying MED in practice.

Related methods. The problem of recovering a distribution function from samples has been investigated and studied before. We briefly review some of the work most relevant to ours:

Data-driven maximum entropy distribution function: Several attempts have been made in the literature to speed up the computation of Lagrange multipliers for MED using Neural Networks [119]–[121] and Gaussian process regression [122]. Unfortunately, these approaches are data-dependent with support only on the trained subspace of distributions.

Similar to the standard MED and other related closures, the data-driven MED can only handle polynomial moments as input, even though the data may be better represented with moments of other basis functions.

Learning an invertible map: The idea is to train an invertible neural network that maps the samples to a known distribution function. Then the unknown distribution function is found by inverting the trained map with the known distribution as the input. This procedure is called the normalizing flow technique [123]–[129]. This method has been used for re-sampling unknown distributions, e.g. Boltzmann generators [130], as well as density recovery such as AI-Feynmann [131], [132]. We note that AI-Feynman does not obtain the density from the samples directly; instead it first fits a density to the samples using the normalizing flow technique, constructs an input/output data set, then finds a simpler expression using symbolic regression. While invertible maps can be used to accurately predict densities, they can become expensive since for each problem one has to learn the parameters of the considered map via optimization.

Diffusion map: Instead of training for an invertible map, the diffusion map [133], [134] constructs coordinates using eigenfunctions of Markov matrices. Using pairwise distances between samples, in this method a kernel matrix is constructed as a generator of the underlying Langevin diffusion process. As shown by [135], one can generate samples of the target distribution using Laplacian-adjusted Wasserstein gradient descent [136]. Unfortunately, this approach can become computationally expensive since it requires singular value decomposition of matrices of size equal to the number of samples.

Gradient flow: The gradient flow method has gained attention in recent years [103], [137], [138]. In particular, a class of sampling methods has been devised for drawing samples from a given distribution function using Langevin dynamics with the gradient of log-density as the driving force [139]–[141]. Yet, this approach does not provide the density of the samples by itself. In this work, we benefit from this formulation to recover the parameters of a density ansatz.

Stein Variational Gradient Descent method (SVGD): Given a target density function, the SVGD method as a deterministic and non-parameterized Gradient method generates samples of the target by carrying out a dynamic system on particles where the velocity field includes the grad-log of the target density (similar to the Gradient flow) and a kernel over particles [142]. SVGD has been derived by approximating a kernelized Wasserstein gradient flow of KL divergence [139]. While further steps have been taken to improve this method, e.g. SVGD with moment matching [143], similar to Gradient flow, this class of method cannot be used for estimating the density itself from samples.

Wavelet and Conditional Renormalization Group method: One of the powerful methods in signal processing is the wavelet method [144], which may be considered as an extension of the Fourier method and domain decomposition. The basic idea is to consider the data on a multiple grids/scales, where the contribution from the smallest frequencies are found on the coarsest mesh and the highest frequencies on the finest mesh. While this method has

been extended to finding high dimensional probability density functions [145], [146], it is not clear how much bias is introduced by the orthogonal wavelet bases.

KDE via diffusion: In this method, the bandwidth of the kernel density estimation is computed using the minimum of mean integrated squared error and the fact that the KDE is the fundamental solution to a heat (more precisely Fokker-Planck) equation [147], [148]. While improvement has been achieved in this direction, we note that the KDE-diffusion method suffers from smoothing effects which introduce bias. Moreover moments of the unknown distribution are not necessarily matched.

Symbolic Regression: Symbolic Regression (SR) is a challenging task in machine learning that aims to identify analytical expressions that best describe the relationship between inputs and outputs of a given dataset. SR does not require any prior knowledge about the model structure. Traditional regression methods such as least squares [18], likelihood-based [19], [20], and Bayesian regression techniques [22], [23], [25] use a fixed parametric model structure and only optimize for model parameters. SR optimizes for model structure and parameters simultaneously and hence is thought to be NP-hard, i.e. Non-deterministic Polynomial-time hard, [31], [32], [131]. The SR problem has gained significant attention over recent years [27], [28], and several approaches have been suggested in the literature. Most methods adopt genetic algorithms [8], [12], [34]. Lately, researchers proposed using machine learning algorithms (e.g. Bayesian optimization, nonlinear least squares, neural networks, transformers, etc.) to solve the SR problem [17], [31], [37], [43], [44], [46], [48], [50], [55]–[57], [132]. While most SR methods are concerned with finding a map from the input to the output, very few have addressed the problem of discovering probability density functions from samples [132].

Our Contributions. Our work improves the efficiency in determining the maximum entropy result for the unknown distribution. We specifically develop a new method for determining the unknown parameters (Lagrange multipliers) of this distribution without solving the optimization problem associated with this approach. This is achieved by relating the samples to the MED using Gradient flow, with the grad-log of the MED guess distribution serving as the drift. This results in a linear inverse problem for the Lagrange multipliers that is significantly easier to solve compared to the aforementioned optimization problem. We also propose a Monte Carlo search in the space of smooth functions for finding an optimal basis function for describing (the exponent of) the maximum entropy ansatz. As a selection criterion, we rate randomly created basis functions according to the condition number associated with the coefficient (Hessian) matrix of the inverse problem for the Lagrange multipliers. This helps to maintain good conditioning, which allows us to incorporate more degrees of freedom and recover the unknown density accurately. Discontinuous density functions are treated by considering only the domain supported by data and using a multi-level solution process.

The chapter is organized as follows. In Section 4.2 we review the concept of Gradient flow with grad-log of a known density as the drift. In Section 4.3, we show how parameters of a guess MED may be found by computing the relaxation rates of the corresponding Gradient

flow. Using the maximum entropy ansatz, in Section 4.4 we derive a linear inverse problem for finding the Lagrange multipliers without the need for solving an optimization problem. In Section 4.5, we propose a symbolic regression method for finding basis functions that can be used to increase degrees of freedom while maintaining good conditioning of the problem by construction. In Section 4.6, we propose a generalization of the maximum entropy ansatz that allows including further degrees of freedom in a multi-level fashion. Section 4.7 presents the complete MESSY algorithm. In Section 4.8, we validate MESSY by comparing its predictions to those of benchmark density estimators in recovering distributions featuring discontinuities and bi-modality, as well as distributions close to the limit of realizability. Finally, in Section 4.9, we offer our conclusions and outlook.

4.2 Gradient flow and theoretical motivation

Consider a set of samples of a random variable $\boldsymbol{X}$ from an unknown density distribution function $f(\boldsymbol{x})$. Let our guess for this distribution function, the "ansatz", be denoted by $\hat{f}(\boldsymbol{x})$.

Instead of constructing a non-parametric approximation of the target density numerically from samples of $\boldsymbol{X}$ (like histogram or KDE) and then calculating its difference from the guess density $\hat{f}$, in this work we suggest measuring the distance using transport. In particular, we use the fact that the steady-state distribution of $\boldsymbol{X}(t)$ which follows the stochastic differential equation (SDE)

$$d\boldsymbol{X} = \nabla_{\boldsymbol{x}} \big[\log \big(\hat{f} \big) \big] dt + \sqrt{2} d\boldsymbol{W}_t \tag{4.1}$$

is the distribution $\hat{f}$. Here, $\boldsymbol{W}_t$ is the standard Wiener process of dimension $\dim(\boldsymbol{x})$. We note that Eq. (4.1) is known as the gradient flow (or Langevin dynamics) with grad-log of density as the force. We note that this drift differs from the score-based generative model in [103] where the drift is a function of time, i.e. $\nabla_{\boldsymbol{x}} \big[\log(\hat{f}(t)) \big]$.

The distance of f from $\hat{f}$ may be measured by the time required for the SDE with $\boldsymbol{X}(t = 0) \sim f$ to reach steady state. Alternatively, one may compare the moments computed from the solution to $\boldsymbol{X}(t)$ against the input samples to measure this distance. Both these approaches are subject to numerical and statistical noise associated with the numerical scheme deployed in integrating Eq. (4.1). In the next section, we derive an efficient way of computing the parameters of our approximation $\hat{f}$ based on these ideas. We also show that the transition from f to $\hat{f}$ is monotonic.

4.3 Ansatz as the target density of Gradient flow

According to Ito's lemma [149] the transition of f to $\hat{f}$ is governed by the Fokker-Planck equation

$$\frac{\partial f}{\partial t} = \nabla_{\boldsymbol{x}} \Big[\hat{f}\, \nabla_{\boldsymbol{x}} [f/\hat{f}\,] \Big] \tag{4.2}$$

$$= -\nabla_{\boldsymbol{x}} \cdot \Big[\nabla_{\boldsymbol{x}} \big[\log\big(\hat{f}\,\big) \big] f \Big] + \nabla^2_{\boldsymbol{x}} \Big[f \Big]. \tag{4.3}$$

Proposition 4.3.1. *The distribution function $f(t)$ governed by the Fokker-Planck Eq. (4.2) converges to $\hat{f}$ as $t \to \infty$. Furthermore, the cross entropy distance between f and $\hat{f}$ monotonically decreases during this transition.*

Proof. Let us multiply both sides of Eq. (4.2) by $\log(f/\hat{f})$ and take the integral with respect to $\boldsymbol{x}$ in order to obtain the evolution of the cross-entropy $S = \int f \log(f/\hat{f}) d\boldsymbol{x}$. It follows that

$$\begin{aligned}
\frac{dS}{dt} &= \int \log(f/\hat{f})\, \nabla_{\boldsymbol{x}} \Big[\hat{f}\, \nabla_{\boldsymbol{x}} [f/\hat{f}] \Big]\, d\boldsymbol{x} \\
&= \int \nabla_{\boldsymbol{x}} \Big[\hat{f} \log(f/\hat{f}) \nabla_{\boldsymbol{x}} [f/\hat{f}] \Big] d\boldsymbol{x} - \int \hat{f}\, \nabla_{\boldsymbol{x}} [\log(f/\hat{f})] \cdot \nabla_{\boldsymbol{x}} [f/\hat{f}] d\boldsymbol{x} \\
&= \underbrace{\int \nabla_{\boldsymbol{x}} \Big[\hat{f} \log(f/\hat{f})\, \frac{f}{\hat{f}} \nabla_{\boldsymbol{x}} [\log(f/\hat{f})] \Big] d\boldsymbol{x}}_{=0} - \int \hat{f}\, \nabla_{\boldsymbol{x}} [\log(f/\hat{f})] \cdot \frac{f}{\hat{f}}\, \nabla_{\boldsymbol{x}} [\log(f/\hat{f})] d\boldsymbol{x} \\
&= -\sum_{i=1}^{\dim(\boldsymbol{x})} \int f \Big(\nabla_{x_i} [\log(f/\hat{f})] \Big)^2 d\boldsymbol{x} \leq 0\ .
\end{aligned} \tag{4.4}$$

Here, we use the regularity condition that $f \log(f/\hat{f}) \nabla_{\boldsymbol{x}} \log(f/\hat{f}) \to 0$ as $|\boldsymbol{x}| \to \infty$. Therefore, given any initial condition for f at $t = 0$, the cross-entropy distance between f and $\hat{f}$ following the Fokker-Planck in Eq. (4.2) monotonically decreases until it reaches the steady-state with the trivial fixed point $f \to \hat{f}$ as $t \to \infty$. For details, see [139]. □

Instead of seeking solutions of Eq. (4.2), our approach focuses on working with appropriate empirical moments of this equation, which can be evaluated from the available samples. As will be demonstrated below, this approach lends itself to a very effective method for determining $\hat{f}$.

Let us denote a vector of basis functions in $\mathbb{R}^{\dim(\boldsymbol{x})}$ by $\boldsymbol{H}(\boldsymbol{x})$. By multiplying both sides of Eq. (4.3) by $\boldsymbol{H}(\boldsymbol{x})$ and integrating with respect to $\boldsymbol{x}$, we obtain the evolution equation for the moments, also known as the relaxation rates,

$$\frac{d}{dt} \Big[\int \boldsymbol{H} f d\boldsymbol{x} \Big] = -\int \boldsymbol{H} \nabla_{\boldsymbol{x}} \cdot \Big[\nabla_{\boldsymbol{x}} [\log(\hat{f})] f \Big] d\boldsymbol{x} + \int \boldsymbol{H} \nabla^2_{\boldsymbol{x}} \Big[f \Big] d\boldsymbol{x}\ . \tag{4.5}$$

Assuming that the underlying density f is integrable in $\mathbb{R}^{\dim(\boldsymbol{x})}$ and $f\boldsymbol{H} \to \mathbf{0}$ as $\boldsymbol{x} \to \infty$, which is implied by the existence of moments, we use integration by parts to obtain

$$\frac{d}{dt}\Big[\int \boldsymbol{H} f d\boldsymbol{x}\Big] = \int \nabla_{\boldsymbol{x}}[\boldsymbol{H}] \cdot \nabla_{\boldsymbol{x}}[\log(\hat{f})] f d\boldsymbol{x} + \int \nabla^2_{\boldsymbol{x}}[\boldsymbol{H}] f d\boldsymbol{x} \ . \tag{4.6}$$

Given samples of f, one can compute the relaxation rates of moments represented by Eq. (4.6) as a measure of the difference between $\hat{f}$ and f. These relaxation rates can be used as the gradient in the search for parameters of a given ansatz, i.e.

$$\boldsymbol{g}(t) = \frac{d}{dt}\Big\langle \boldsymbol{H}(\boldsymbol{X}(t))\Big\rangle = \Big\langle \nabla_{\boldsymbol{x}}[\boldsymbol{H}(\boldsymbol{X}(t))] \cdot \nabla_{\boldsymbol{x}}[\log(\hat{f}(\boldsymbol{X}(t)))]\Big\rangle + \Big\langle \nabla^2_{\boldsymbol{x}}[\boldsymbol{H}(\boldsymbol{X}(t))]\Big\rangle \ . \tag{4.7}$$

In the above, $\langle \phi(\boldsymbol{X})\rangle$ denotes the unbiased empirical measure for the expectation of $\phi(\boldsymbol{X})$ which is computed using samples of $\boldsymbol{X}_i$, for $i = 1, ..., N$ via $\langle \phi(\boldsymbol{X})\rangle = \frac{1}{N}\sum_{i=1}^{N} \phi(\boldsymbol{X}_i)$.

In what follows we develop an approach that uses $g(t)$ as the gradient of an optimization problem to bring computational benefits to the solution of the maximum entropy problem.

4.4 Maximum Entropy Distribution as an ansatz for the gradient flow

In this work, we use the maximum entropy distribution function as our parameterized ansatz for $\hat{f}$, i.e.

$$\hat{f}(\boldsymbol{x}) = Z^{-1} \exp\big(\boldsymbol{\lambda} \cdot \boldsymbol{H}(\boldsymbol{x})\big) \tag{4.8}$$

where $Z = \int \exp(\boldsymbol{\lambda} \cdot \boldsymbol{H}(\boldsymbol{x}))d\boldsymbol{x}$ is the normalization constant. The motivation for choosing this family of distributions is the fact that this is the least-biased distribution for the moment problem, provided the given moments are matched.

Definition 4.4.1. *Moment problem*

The problem of finding a distribution function $f(\boldsymbol{x})$ given its moments $\int \boldsymbol{H}(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = \boldsymbol{\mu}$ for the vector of basis functions $\boldsymbol{H}(\boldsymbol{x})$ will be referred to as the moment problem.

In particular, the density in Eq. (4.8) is the extremum of the loss functional that minimizes the Shannon entropy with constraints on moments $\boldsymbol{\mu}$ using the method of Lagrange multipliers, i.e.

$$\hat{f}(\boldsymbol{x}) = \underset{\mathcal{F}\in\mathcal{K}}{\arg\min}\ \mathcal{C}[\mathcal{F}(\boldsymbol{x})] \tag{4.9}$$

$$\text{where} \quad \mathcal{C}[\mathcal{F}(\boldsymbol{x})] := \int \mathcal{F}(\boldsymbol{x}) \log(\mathcal{F}(\boldsymbol{x})) d\boldsymbol{x} - \sum_{i=1}^{N_b} \lambda_i \left(\int H_i(\boldsymbol{x}) \mathcal{F}(\boldsymbol{x}) d\boldsymbol{x} - \mu_i(\boldsymbol{x}) \right) \ . \tag{4.10}$$

Here $\mathcal{K}$ denotes the space of probability density functions with measurable moments; see [111] and Appendix C.1 for more details. In this work, we denote the number of considered

basis functions by N_b, while N_m denotes the highest order of these basis functions. For instance, in the case of traditional one-dimensional random variable where polynomial basis functions are deployed, i.e. $\boldsymbol{H} = \left[x, x^2, ..., x^{N_m}\right]$, we have $N_m = N_b$.

Here, we use the following definition for the growth rate of a basis function.

Definition 4.4.2. *Growth rate of n-th order*

A function $\psi(x)$ has the growth-rate of n-th order if $|\psi(x)| \leq Cx^n$ for all $x \geq x_0$ where $C \in \mathbb{R}^+$ and $x_0 \in \mathbb{R}$. This is often denoted by $\psi(x) = \mathcal{O}(x^n)$.

We note that the moment problem for the MED is reduced to the following optimization problem by substituting the extremum in Eq. (4.8) back in the objective functional of Eq. (4.10), see e.g. [150].

Definition 4.4.3. *Standard dual optimization problem of Maximum entropy distribution function*

Given moments $\boldsymbol{\mu}$, the Lagrange multipliers $\boldsymbol{\lambda}$ of the maximum entropy distribution are the solution to the following unconstrained optimization problem

$$\boldsymbol{\lambda} = _{\hat{\boldsymbol{\lambda}} \in \mathbb{R}_b^N} \left\{ \log \left[\int \exp(\hat{\boldsymbol{\lambda}} \cdot \boldsymbol{H}(\boldsymbol{x})) d\boldsymbol{x} \right] - \hat{\boldsymbol{\lambda}} \cdot \boldsymbol{\mu} \right\} . \tag{4.11}$$

Clearly, the standard optimization problem for finding Lagrange multipliers is nonlinear and requires deploying iterative methods, such as the Newton-Raphson method detailed in Appendix C.1.

Instead, using the Gradient flow, we find an alternative optimization problem for finding Lagrange multipliers which is linear and simple to compute from given samples. Substituting Eq. (4.8) for $\hat{f}$ in Eq. (4.7) results in the relaxation rate

$$\boldsymbol{g}(t) = \sum_{i=1}^{\dim(\boldsymbol{x})} \left\langle \nabla_{x_i} \left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right) \right] \otimes \nabla_{x_i} \left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right) \right] \right\rangle \boldsymbol{\lambda} + \sum_{i=1}^{\dim(\boldsymbol{x})} \left\langle \nabla_{x_i}^2 \left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right) \right] \right\rangle , \tag{4.12}$$

where $\otimes$ indicates the outer product. Let us define the matrix $\boldsymbol{L}^{\mathrm{ME}}$ as

$$\boldsymbol{L}^{\mathrm{ME}}(t) := \sum_{i=1}^{\dim(\boldsymbol{x})} \left\langle \nabla_{x_i} \left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right) \right] \otimes \nabla_{x_i} \left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right) \right] \right\rangle . \tag{4.13}$$

Definition 4.4.4. *Optimization problem of Maximum entropy distribution function via Gradient flow*

Given samples $\boldsymbol{X}$ of the target distribution f, Lagrange multipliers $\boldsymbol{\lambda}$ of maximum entropy distribution estimate $\hat{f}$ is the solution to the following unconstrained optimization problem

$$\boldsymbol{\lambda} = _{\hat{\boldsymbol{\lambda}} \in \mathbb{R}_b^N} \left\{ \boldsymbol{L}^{\mathrm{ME}}(t) : \frac{\hat{\boldsymbol{\lambda}} \otimes \hat{\boldsymbol{\lambda}}}{2} + \sum_{i=1}^{\dim(\boldsymbol{x})} \left\langle \nabla_{x_i}^2 \left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right) \right] \right\rangle \cdot \hat{\boldsymbol{\lambda}} \right\} , \tag{4.14}$$

where (:) denotes the Frobenius inner (or double dot) product, i.e. $\boldsymbol{A} : \boldsymbol{B} = \sum_{i,j} A_{ij} B_{ij}$ for given two-dimensional tensors $\boldsymbol{A}$ and $\boldsymbol{B}$.

We note that the matrix $\boldsymbol{L}^{\mathrm{ME}}$ is the Hessian of the optimization problem with gradient given by Eq. (4.12) which is positive definite, making the underlying optimization problem convex.

Proposition 4.4.5. *The Hessian matrix $\boldsymbol{L}^{\mathrm{ME}}$ is symmetric positive definite. As a result, the optimization problem with gradient given by Eq. (4.12) and Hessian matrix given by Eq. (4.13) is strictly convex.*

Proof. Clearly, the Hessian matrix defined by Eq. (4.13) is symmetric, i.e. $L^{\mathrm{ME}}_{i,j} = L^{\mathrm{ME}}_{j,i}\ \forall i, j = 1, ..., N_b$. We further note that this matrix is positive definite, i.e. for any non-zero vector $\boldsymbol{w} \in \mathbb{R}^{N_b}$ we can write

$$\boldsymbol{w}^T \boldsymbol{L}^{\mathrm{ME}}(t)\boldsymbol{w} = \sum_{i=1}^{\dim(\boldsymbol{x})} \left\langle \boldsymbol{w}^T \nabla_{x_i}\left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right)\right]\ \nabla_{x_i}\left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right)\right]^T \boldsymbol{w} \right\rangle \tag{4.15}$$

$$= \sum_{i=1}^{\dim(\boldsymbol{x})} \left\langle \left(\boldsymbol{w}^T \nabla_{x_i}\left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right)\right] \right)^2 \right\rangle > 0\ . \tag{4.16}$$

Given the Hessian is symmetric positive definite, we conclude that the underlying optimization problem is convex [151]. □

When the matrix $\boldsymbol{L}^{\mathrm{ME}}$ is well-conditioned, we can directly compute the Lagrange multipliers using samples, i.e. a linear solution to the optimization problem in Def. 4.4.4. This can be achieved by solving Eq. (4.12) for the Lagrange multipliers

$$\boldsymbol{L}^{\mathrm{ME}}(t)\boldsymbol{\lambda} = \boldsymbol{g}(t) - \sum_{i=1}^{\dim(\boldsymbol{x})} \left\langle \nabla^2_{x_i}\left[\boldsymbol{H}\left(\boldsymbol{X}(t)\right)\right] \right\rangle \tag{4.17}$$

for a given relaxation rate $\boldsymbol{g}$.

We proceed by noting that a convenient way for determining the parameters of $\hat{f}$ is to set $\hat{f} = f(t = 0)$ in the above formulation, or in other words, require that the given samples are also samples of $\hat{f}$ as given. This corresponds to the steady solution of Eq. (4.12), namely $\boldsymbol{g} \to \boldsymbol{0}$, which implies the remarkably simple result

$$\boldsymbol{\lambda} = -\left(\boldsymbol{L}^{\mathrm{ME}}\right)^{-1} \left(\sum_{i=1}^{\dim(\boldsymbol{x})} \left\langle \nabla^2_{x_i}\left[\boldsymbol{H}\left(\boldsymbol{X}(t=0)\right)\right] \right\rangle \right)\ , \tag{4.18}$$

which implies a closed-form solution for the Lagrange multipliers through the above linear problem.

While Eq. (4.18) analytically recovers the Lagrange multipliers $\boldsymbol{\lambda}$ directly from samples of $\boldsymbol{X}$, it still requires inverting the matrix $\boldsymbol{L}^{\mathrm{ME}}$ which may be ill-conditioned [152], [153]. This means that the resulting Lagrange multipliers may become sensitive to noise in the samples and the choice of the basis functions. In order to cope with this issue, we propose computing $\boldsymbol{\lambda}$ as outlined below.

Orthonormalizing the basis functions. We construct an orthonormal basis function with respect to $\boldsymbol{X} \sim f$ using the modified Gram-Schmidt algorithm as described in Algorithm 4. We deploy the orthonormal basis functions from the Gram-Schmidt procedure to construct $\nabla_{\boldsymbol{x}}[\boldsymbol{H}]^{\perp}$, i.e. $\nabla_{\boldsymbol{x}}[\boldsymbol{H}]$ is the input to Algorithm 4, and by integration we obtain $\boldsymbol{H}^{\perp}$. This leads to a well-conditioned matrix $\boldsymbol{L}^{\text{ME}}$, since the resulting matrix should be close to identity $\boldsymbol{L}^{\text{ME}} \approx \boldsymbol{I}$ with condition number $\text{cond}\big(\boldsymbol{L}^{\text{ME}}\big) \approx 1$ subject to round-off error. We note that the cost of this algorithm is quadratic with the number of basis functions and linear with the number of samples.

Algorithm 4: Modified Gram-Schmidt: Given a vector of basis functions $\boldsymbol{\phi}$, this algorithm constructs an orthonormal basis functions $\boldsymbol{\phi}^{\perp}$ with respect to f such that $\langle \boldsymbol{\phi}^{\perp}(\boldsymbol{X}) \otimes \boldsymbol{\phi}^{\perp}(\boldsymbol{X}) \rangle \approx \boldsymbol{I}$ using the modified Gram-Schmidt procedure [152], [154].

Input: $\boldsymbol{\phi}$
Initialize $\boldsymbol{\phi}^{\perp} \leftarrow \boldsymbol{\phi}$;
for $i = 1, ..., \dim(\boldsymbol{\phi})$ **do**
  $\phi_i^{\perp} = \phi_i^{\perp} / \sqrt{\langle (\phi_i^{\perp}(\boldsymbol{X}))^2 \rangle}$;
  for $j = i + 1, ..., \dim(\boldsymbol{\phi})$ **do**
    $\phi_j^{\perp} \leftarrow \phi_j^{\perp} - \langle \phi_i^{\perp}(\boldsymbol{X}) \phi_j^{\perp}(\boldsymbol{X}) \rangle \phi_i^{\perp}$;
  end
end
Return $\boldsymbol{\phi}^{\perp}$

4.4.1 Comparing the proposed formulation to standard Maximum Entropy Distribution

Here we point out several advantages of using the proposed loss function compared to the standard maximum entropy closure.

- **A closed-form solution.** By setting the relaxation rate of the moments to zero, the Lagrange multipliers can be computed directly from samples $\boldsymbol{X} \sim f$, i.e. by solving the system in Eq. (4.18), without the need for the line-search associated with the Newton method of solving the optimization problem of standard MED, i.e. Def. 4.4.3. This is a significant improvement compared to standard MED, where Lagrange multipliers are estimated iteratively — see Eq. (C.5) and Algorithm 8 in Appendix C.1.

- **Avoiding the curse of dimensionality in integration.** In the proposed method, the computational complexity associated with the integration only depends on the number of samples, and not on the dimension of the probability space. The proposed method takes full advantage of having access to the samples of the unknown distribution function. This is in contrast to the standard Newton-Raphson method of solving optimization problem 4.4.3 where samples of the initial guessed MED corresponding to the initial guessed $\boldsymbol{\lambda}$ is not available and one runs to the curse of dimensionality in computation of gradient and Hessian.

  In particular, we compute the orthonormal basis function, gradient, and Hessian using the samples of $\boldsymbol{X}$. This use of the Monte Carlo integration method

avoids the curse of high dimensionality associated with the conventional method for computing Lagrange multipliers. By deploying the Law of Large Numbers (LLN) in computing integrals, the proposed method benefits from the well-known result that the error of Monte Carlo integration is independent of dimension. In particular, given N independent, identically distributed d-dimensional random variables $\boldsymbol{X}_1, ..., \boldsymbol{X}_N$ with probability density $f(\boldsymbol{x})$, where $\boldsymbol{X} \sim f$, the variance of the empirical estimator of a moment $\phi(\boldsymbol{x})$ of f, i.e. $\mathbb{E}^{\Delta}[\phi(\boldsymbol{X})] = \sum_{i=1}^{N} \phi(\boldsymbol{X}_i)/N$, is

$$\mathrm{Var}(\mathbb{E}^{\Delta}[\phi(\boldsymbol{X})]) = \frac{\mathrm{Var}(\phi(\boldsymbol{x}))}{N} \tag{4.19}$$

which is independent of the dimension $d = \dim(\boldsymbol{x})$. Therefore, for a required ratio of variance in prediction and variance of the underlying random variable, i.e. $\mathrm{Var}(\mathbb{E}^{\Delta}[\phi(\boldsymbol{X})])/\mathrm{Var}(\phi(\boldsymbol{x}))$, the cost of integration is $\mathcal{O}(sN)$. The factor s denotes the cost of computing $\phi(\boldsymbol{X})$ for one sample.

This is a considerable advantage compared to the standard approach of finding the Lagrange multipliers of MED where the cost associated with integration is of order $\mathcal{O}(N^d)$ where N is the number of discretization points in each dimension. We remind the reader that in the standard Newton-Raphson method of finding the Lagrange multipliers, one updates the guessed $\boldsymbol{\lambda}$ by

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \boldsymbol{L}^{-1}(\boldsymbol{\lambda})\boldsymbol{g}(\boldsymbol{\lambda}) \tag{4.20}$$

where the gradient $\boldsymbol{g}$ and Hessian $\boldsymbol{L}$ need to be computed during each iteration, see Section C.1 and reference therein for details. Since samples of the guessed distribution, i.e. guessed $\boldsymbol{\lambda}$, are not available, computation of the gradient and Hessian can become expensive. Specifically, one has to either generate samples of the guess distribution in each intermediate step of the Newton-Raphson method which adds complexity, or deploy a deterministic integration method with cost $\mathcal{O}(N^{\mathrm{d}})$ where N is the number of discretization points in each dimension and $d = \dim(\boldsymbol{x})$.

We further point out that since in both the proposed solution and the standard iterative method a matrix needs to be inverted, the cost in both algorithms scales $\mathcal{O}(N_b^3)$ with number of basis functions (moments) regardless of the cost associated with integration. For example, in case of using monomials up to 2nd order in all dimensions as basis functions i.e. $\boldsymbol{H} = [x_1, ..., x_d, x_1^2, x_1x_2, ..., x_d^2]$, there are $|\boldsymbol{H}| = d + d(d+1)/2$ unique bases, leading to complexity $\mathcal{O}\left((d + d(d+1)/2)^3\right)$ for solving the linear system given by Eq. (4.18).

- **Relaxed existence requirements.** Since the proposed approach directly finds the Lagrange multipliers for the realizable moment problem linearly using Eq. (4.18), it avoids the problem of possible non-realizable distribution estimate with intermediate Lagrange multipliers that is present in the iterative methods. This is another advantage compared to the standard MED optimization problem, Eq. (4.4.3), where the line search Eq. (4.20) may fail as the distribution associated with the intermediate $\boldsymbol{\lambda}$ may

not exist (not integrable). This is a common problem when finding Lagrange multipliers for the moment problem close to the limit of realizability when the condition number of the Hessian becomes large and the iterative Newton-Raphson method fails, e.g. see [153].

4.5 Symbolic-Based Maximum Entropy Distribution

In the standard moment problem it is common to consider polynomials for the moment functions in $\boldsymbol{H}$, i.e. $\boldsymbol{H} = [x, x^2, \ldots]$, even though other basis functions may better represent the unknown distribution. Additionally, such polynomial basis functions are notorious for resulting in ill-conditioned solution processes. For these reasons, we introduce a symbolic regression approach to introduce some diversity and ultimately optimize over our use of basis functions. As we will see in the next section, adding the symbolic search to our MED description improves the accuracy, convergence, and robustness of the density recovery problem.

Before diving into the proposed method, we first briefly review the general task of symbolic regression.

Definition 4.5.1. *Symbolic Regression (SR) problem*

Given a metric $\mathcal{L}$ *and a dataset* $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ *consisting of* N *independent identically distributed (i.i.d.) paired samples, where* $\boldsymbol{x}_i \in \mathbb{R}^{\dim(\boldsymbol{x})}$ *and* $y_i \in \mathbb{R}$*, the SR problem searches in the space of functions* $\mathcal{S}$ *defined by a set of given mathematical functions (e.g.,* cos*,* sin*,* exp*,* ln*) and arithmetic operations (e.g.,* $+$*,* $-$*,* $\times$*,* $\div$*), for a function* $\psi^*(\boldsymbol{x})$ *which minimizes* $\sum_{i=1}^N \mathcal{L}\big(y_i, \psi(\boldsymbol{x}_i)\big)$ *where* $\psi \in \mathcal{S}$*.*

In order to deploy the SR method for the density recovery, we need to restrict the space of functions $\mathcal{S}$ to those which satisfy non-negativity, normalization and existence of moments with respect to the vector of linearly independent (polynomial) basis functions $\boldsymbol{R}$. The space of such distributions can be defined as

$$\mathcal{S}_{f|\boldsymbol{R}} := \left\{ f(\boldsymbol{x}) \in \mathcal{S} \;\middle|\; f(\boldsymbol{x}) \geq 0 \;\forall \boldsymbol{x} \in \mathbb{R}^{\dim(\boldsymbol{x})}, \int_{\mathbb{R}^{\dim(\boldsymbol{x})}} f(\boldsymbol{x})\, d\boldsymbol{x} = 1, \int_{\mathbb{R}^{\dim(\boldsymbol{x})}} \boldsymbol{R}(\boldsymbol{x}) f(\boldsymbol{x})\, d\boldsymbol{x} < +\infty \right\}. \tag{4.21}$$

In order to ensure non-negativity, motivated by the MED formulation, we consider $\hat{f}$ to be exponential, i.e.

$$\hat{f}(\boldsymbol{x}) \propto \exp\big(\mathcal{G}(\boldsymbol{x})\big) \qquad \Longleftrightarrow \qquad \log\Big(\hat{f}(\boldsymbol{x})\Big) \propto \mathcal{G}(\boldsymbol{x}) \;, \tag{4.22}$$

where $\mathcal{G}(\boldsymbol{x})$ is an analytical (or symbolic) function of $\boldsymbol{x} = \big[x_1, x_2, \ldots, x_{\dim(\boldsymbol{x})}\big]$. While the non-negativity is guaranteed, existence of moments needs to be verified when a test function for $\mathcal{G}(\boldsymbol{x})$ is considered. As our focus in this work is on the maximum entropy distribution function given by Eq. (4.8), we consider $\mathcal{G}(\boldsymbol{x})$ to have the form

$$\mathcal{G}(\boldsymbol{x}) = \boldsymbol{\lambda} \cdot \boldsymbol{H}(\boldsymbol{x}) = \sum_{i=1}^{N_b} \lambda_i H_i(\boldsymbol{x}) \;. \tag{4.23}$$

Now we proceed to provide a modified formulation for SR tailored to our MED problem.

Definition 4.5.2. *Symbolic Regression for the Maximum Entropy Distribution (SR-MED) problem*

Given a measure of difference between distributions $\mathcal{L}$ *(e.g. KL Divergence) and a dataset* $\mathcal{D} = \{\boldsymbol{X}_i\}_{i=1}^{N}$ *consisting of* N *i.i.d. samples, where* $\boldsymbol{X}_i \in \mathbb{R}^{\dim(\boldsymbol{x})}$*, the SR-MED problem searches in the space* $\mathcal{S}^{N_b}$ *for* N_b *basis functions subject to* $\hat{f} \in \mathcal{S}_{f|\boldsymbol{R}}$ *which minimizes* $\mathcal{L}$*.*

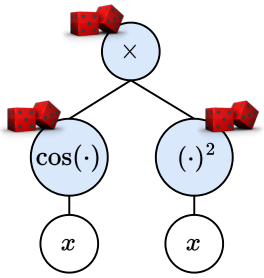


Figure 4.1: Expression tree for $x^2 \times \cos(x)$.

Here, we deploy continuous functions consisting of *binary* operators (e.g. $+$, $-$, $\times$, $\div$) or *unary* functions (e.g. cos, sin, exp, log) to fill the space S^{N_b}. As in most of the SR methods, we encode mathematical expressions using symbolic expression trees, a type of binary tree, where internal nodes contain operators or functions and terminal nodes (or leaves) contain input variables of constants. For instance, the expression tree in Figure 4.1 represents $x^2 \cos(x)$. In this work, we perform a Monte Carlo symbolic search in the space of smooth functions (by generating random expression trees) to find a vector of basis functions $\boldsymbol{H}$ that guarantees acceptable cond($\boldsymbol{L}^{\mathrm{ME}}$), by rejecting candidates that do not satisfy this condition. In our search, we do not consider test basis functions with odd growth rates which lead to non-realizable distributions.

4.6 Multi-level density recovery

We further improve our proposed method by introducing a multi-level process that improves our prediction as the distribution becomes more detailed. The goal is to obtain a more generalized MED estimate with the form

$$\hat{f}(\boldsymbol{x}) = \sum_{l=1}^{N_L} m^{[l]}\, \hat{f}^{[l]}(\boldsymbol{x}) \tag{4.24}$$

$$\text{where} \quad \hat{f}^{[l]}(\boldsymbol{x}) = \frac{1}{Z^{[l]}} \exp\left(\boldsymbol{\lambda}^{[l]} \cdot \boldsymbol{H}^{[l]}(\boldsymbol{x})\right), \tag{4.25}$$

$(.)^{[l]}$ denotes the level index, $Z^{[l]}$ is the normalization factor of density at level l, N_L is the number of levels considered and $m^{[l]}$ indicates the portion of total mass that is covered by $\hat{f}^{[l]}$. We note that this multi-level approach is recursive and can be described as follows:

- **Step 1: Find MED estimate $\hat{f}^{[l]}$ at level l.** At level l, first we pick a basis function $\boldsymbol{H}^{[l]}$ by solving the SR-MED problem detailed in Def. 4.5.2. Then, we orthonormalize

the basis function with respect to the distribution of the samples using Gram–Schmidt's procedure as outlined in Algorithm 4.

- **Step 2: Removing subset of samples covered by $\hat{f}^{[l]}$.** Here, we attempt to find and remove a subset of samples $\mathcal{D}^{[l]}_{\text{mask}}$ – representing a fraction of the *mass*, i.e. $m^{[l]} = |\mathcal{D}^{[l]}_{\text{mask}}|/|\mathcal{D}|$ – that can be estimated by our estimated $\hat{f}^{[l]}$ at this level. To this end, we deploy acceptance/rejection with probability $\hat{f}^{[l]}/\hat{f}^{\text{hist}}$ to find and remove $\mathcal{D}^{[l]}_{\text{mask}}$ from the remaining samples $\mathcal{D}^{[l]}$.

- **Step 3: Repeat steps 1-2 for the next level $l+1$ until almost no samples are left.** Repeat steps 1-2 with the remaining uncovered samples (which constitutes the next level) until there are (almost) no uncovered samples. The resulting total distribution is a weighted sum of the estimates from each level.

In Algorithm 5, we detail a pseudocode for our devised multi-level process. As we will see in the next section, our proposed multi-level recursive mechanism improves overall performance, and elegantly describe details of multi-mode distributions.

Algorithm 5: Multi-level, symbolic and recursive algorithm for density recovery. Here, $\mathcal{D}^{[l]}$ denotes the set of samples at level l and $\boldsymbol{u}$ is a random variable that is uniformly distributed in $(0,1)$, i.e. $\boldsymbol{u} \sim \mathcal{U}([0,1])$.

Input: $\mathcal{D}^{[1]} = \mathcal{D} = \{\boldsymbol{X}_i\}_{i=1}^{N}$, $N_L^{\text{tot}} = N_L$
for $l = 1, ..., N_L$ **do**
  Sample random basis functions $\boldsymbol{H}^{[l]}$ that satisfies Def. 4.5.2 starting from polynomials in level $l = 1$;
  Compute $\hat{f}^{[l]}(\boldsymbol{x})$ given $\mathcal{D}^{[l]}$ using Algorithm 4;
  $\mathcal{D}^{[l]}_{\text{mask}} \leftarrow \{\mathcal{D}^{[l]} \mid \hat{f}^{[l]}(\boldsymbol{X})/\hat{f}^{\text{hist}}(\boldsymbol{X}) > \boldsymbol{u}\}$ where $\boldsymbol{u} \sim \mathcal{U}([0,1])$;
  $m^{[l]} \leftarrow |\mathcal{D}^{[l]}_{\text{mask}}|/|\mathcal{D}|$;
  if $\sum_{j=1}^{l} |\mathcal{D}^{[j]}_{\text{mask}}| \approx |\mathcal{D}|$ **then**
    $\mathcal{D}^{[l]}_{\text{mask}} \leftarrow \mathcal{D}^{[l]}$; `// Mask all available samples`
    $m^{[l]} \leftarrow |\mathcal{D}^{[l]}_{\text{mask}}|/|\mathcal{D}|$;
    $N_L^{\text{tot}} \leftarrow l$;
    break; `// Terminate the process`
  else
    $\mathcal{D}^{[l+1]} \leftarrow \mathcal{D}^{[l]} \backslash \mathcal{D}^{[l]}_{\text{mask}}$; `// The uncovered samples are left for the next level`
  end
end
Return $\hat{f}(\boldsymbol{x}) = \sum_{l=1}^{N_L^{\text{tot}}} \hat{f}^{[l]}(\boldsymbol{x})\ |\mathcal{D}^{[l]}_{\text{mask}}|/|D|$; `//` N_L^{tot}`: total number of recursive calls`

4.7 Algorithm for MESSY estimation

The complete MESSY estimation algorithm is summarized in Algorithm 6. Within the iteration loop, following each application of the multi-level, symbolic, and recursive density

recovery summarized in Algorithm 5, we introduce a maximum-cross entropy distribution (MxED) correction step (see Appendix C.2 for details) to reduce any bias in our prediction for $\hat{f}$ from the former.

Finally, after completing the desired number of iterations, the algorithm returns the candidate density with the smallest KL Divergence given by

$$\mathrm{KL}\big(f \,||\, \hat{f}\big) = \int f(\boldsymbol{x}) \log\left(\frac{f(\boldsymbol{x})}{\hat{f}(\boldsymbol{x})}\right) d\boldsymbol{x} \tag{4.26}$$

$$= -\int f(\boldsymbol{x}) \log\big(\hat{f}(\boldsymbol{x})\big) d\boldsymbol{x} + \int f(\boldsymbol{x}) \log\big(f(\boldsymbol{x})\big) d\boldsymbol{x} \tag{4.27}$$

$$\approx -\left\langle \log\big(\hat{f}(\boldsymbol{X})\big)\right\rangle + \underbrace{\int f(\boldsymbol{x}) \log\big(f(\boldsymbol{x})\big) d\boldsymbol{x}}_{\text{constant with respect to } \hat{f}} . \tag{4.28}$$

In other words, we use $-\left\langle \log\big(\hat{f}(\boldsymbol{X})\big)\right\rangle$ as our selection criterion.

Algorithm 6: Pseudocode of the proposed `MESSY` estimation method. Here, $\boldsymbol{R}$ is the vector of linearly independent (polynomial) basis functions used in the moment matching procedure of MxED. Here, for MESSY-S the number of basis functions N_b is sampled uniformly from the sample space Ω_{N_b}, e.g. here we use $\Omega_{N_b} = \{2, ..., 8\}$ unless mentioned otherwise.

Input: $\mathcal{D} = \{\boldsymbol{X}_i\}_{i=1}^{N}$, Ω_{N_b}, N_m, N_{iters}
Initialize $\hat{f}^{(i)} = 0$ for $i = 1, ..., N_{\text{iters}}$;
for $i = 1$ **to** N_{iters} **do**
  if $i > 1$ **then**
    Sample $N_b \sim \mathcal{U}(\Omega_{N_b})$;
  end
  Find $\hat{f}$ using multi-level, symbolic and recursive Algorithm 5 for density recovery;
  Generate samples of $\boldsymbol{Y} \sim \hat{f}$;
  Apply boundary condition (bounded/unbounded) to $\hat{f}$;
  Correct $\hat{f}$ using MxED (Algorithm 9) given samples $\boldsymbol{Y}$ as prior and $\mathbb{E}[\boldsymbol{R}(\boldsymbol{X})]$ as target moments;
  $\hat{f}^{(i)} \leftarrow \hat{f}$;
end
$\hat{f}^{\text{MESSY-P}} = \hat{f}^{(1)}$;
$\hat{f}^{\text{MESSY-S}} = _{\hat{f} \in \{\hat{f}^{(i)}\}_{i=1}^{N_{\text{iters}}}} \left(\mathrm{KL}(f \,||\, \hat{f})\right)$;
Return $\hat{f}^{\text{MESSY-P}}$ and $\hat{f}^{\text{MESSY-S}}$.

The MESSY algorithm comes in two flavors: MESSY-P, which considers only polynomial basis functions for $\boldsymbol{H}$, and MESSY-S which includes optimization over basis functions using the SR algorithms outlined above. In fact, by convention, the SR algorithm in MESSY-S starts its first iteration using polynomial basis functions up to order N_m as the sample

space of smooth functions. In other words, MESSY-P is a special case of MESSY-S with $N_{\text{iter}} = 1$. In the remaining iterations of MESSY-S, we perform the symbolic search in the space of smooth functions of order N_m to find N_b bases that provide manageable cond($\boldsymbol{L}^{\text{ME}}$), as discussed in Section 4.6.

In addition, we provide the option to enforce boundedness of $\hat{f}$ on the support that is specified by the user, i.e. letting $\hat{f}(\boldsymbol{x}) = 0$ for all $\boldsymbol{x}$ outside the domain of interest. This allows us to recover distributions with discontinuity at the boundary which may have application in image processing.

We also provide an option to further reduce the bias by minimizing the cross-entropy given samples of bounded/unbounded multi-level estimate as prior and moments of input samples as the target moments (see Appendix C.2 for more details on the cross-entropy calculation). For this optional step, we generate samples of $\hat{f}$ and match the moments of polynomial basis functions up to order N_m. Since the solution at each level of $\hat{f}$ is close to the exact MED solution, the optimization problem associated with the moment matching procedure of the cross-entropy algorithm converges very quickly, i.e. in a few iterations, providing us with a correction that minimizes bias along with the weighted samples of our estimate as the by-product. We note that in general the order of the randomly created basis function during the MESSY-S procedure may be different from the one used in the cross-entropy moment matching procedure.

4.8 Results

In this section we demonstrate the effectiveness of the proposed MESSY estimation method in recovering distributions given samples, using a number of numerical experiments, involving a range of distributions ranging from multi-mode to discontinuous. For validation, we provide comparisons with the standard KDE with an optimal bandwidth obtained using K-fold cross-validation with $K = 5$. We consider a uniform grid with 20 elements for the cross-validation in a range that spans from 0.01 to 10 on a logarithmic scale. We also compare with cross-entropy closure with Gaussian as the prior (MxED) using Newton's method. We note that while the standard maximum entropy distribution function differs from MxED as the latter incorporates a prior, we intentionally use MxED as a benchmark instead because the standard approach can be extremely expensive.

Unless mentioned otherwise, we report error, time, and KL Divergence by ensemble averaging over 25 for different sets of samples. Furthermore, in the case of MESSY-S we perform $N_{\text{iters}} = 10$ iterations, and we consider $(+, -, \times)$ operators and (cos, sin) functions. Here we report the execution time using a single-core CPU for each method. Typical symbolic expressions of density functions recovered by MESSY for the test cases considered here can be found in Appendix C.5. Furthermore, in Appendix C.3, we perform an ablation study that shows the importance of each component of the MESSY algorithm.

4.8.1 Bi-modal distribution function

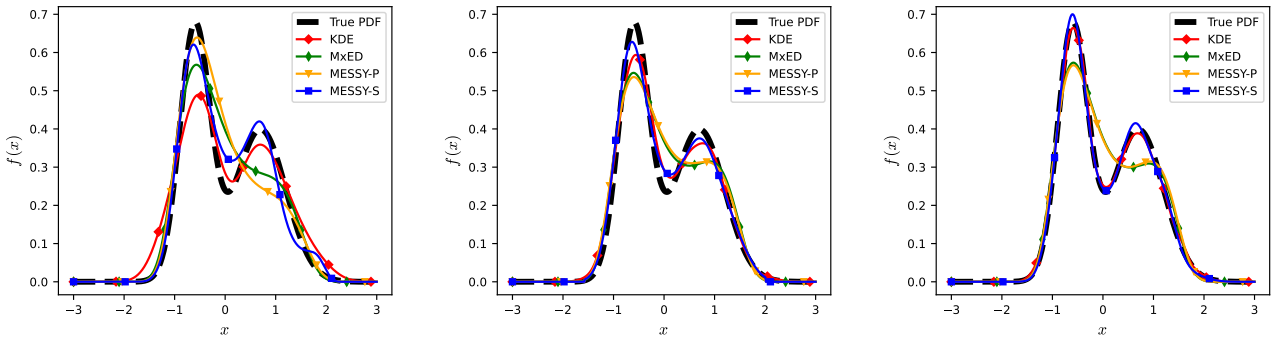
For our first test case, we consider a one-dimensional bi-modal distribution function constructed by mixing two Normal distribution functions $\mathcal{N}(x \,|\, \mu, \sigma)$, i.e.

$$f(x) = \alpha\,\mathcal{N}(x \,|\, \mu_1, \sigma_1) + (1-\alpha)\,\mathcal{N}(x \,|\, \mu_2, \sigma_2), \tag{4.29}$$

with $\alpha = 0.5$, means $\mu_1 = -0.6$ and $\mu_2 = 0.7$, and standard deviations $\sigma_1 = 0.3$ and $\sigma_2 = 0.5$.

Figure 4.2 compares results from MESSY, KDE and MxED for three different sample sizes, namely 100, 1000, and 10,000 samples of f. For MxED and MESSY-P, we use $N_b = N_m = 4$. In the case of MESSY-S, we randomly create N_b basis functions which are $\mathcal{O}(x^4)$ (where N_b is sampled uniformly within $\{2, \ldots, 8\}$). Both MESSY results are subject to a cross-entropy correction step with $N_b = 4$ polynomial moments. Clearly, the MxED and MESSY methods provide a reasonable estimate when a small number of samples is available; where KDE may suffer from bias introduced by the smoothing kernel. As a reference for the reader, we also compared the outcome density with a histogram in Appendix C.4.

In order to analyze the error further, Fig 4.3 presents the relative error in low and high order moments, KL Divergence, and single-core CPU time as the measure of computational cost for considered methods. Given optimal bandwidth is found, the KDE error can only be reduced by increasing the number of samples. However, maximum entropy based estimators such as MxED and MESSY provide more robust estimate when less samples are available. We point out that the convergence of the cases where only moments of polynomial basis functions are considered, i.e. MxED and MESSY-P, relies on the degree of the polynomials and not the number of samples. On the other hand, the additional search associated with MESSY-S returns more appropriate basis functions for a given upper bound on the order of the basis functions.



(a) 100 samples (b) 1,000 samples (c) 10,000 samples

Figure 4.2: Density estimation using KDE, MxED, MESSY-P, and MESSY-S given (a) 100, (b) 1,000, and (c) 10,000 samples.

Next, we perform a convergence study on 10,000 samples and show that the parametric description converges to the solution when its degrees of freedom are increased. In Fig. 4.4, we show that both MESSY-P and MESSY-S converge to the true solution by

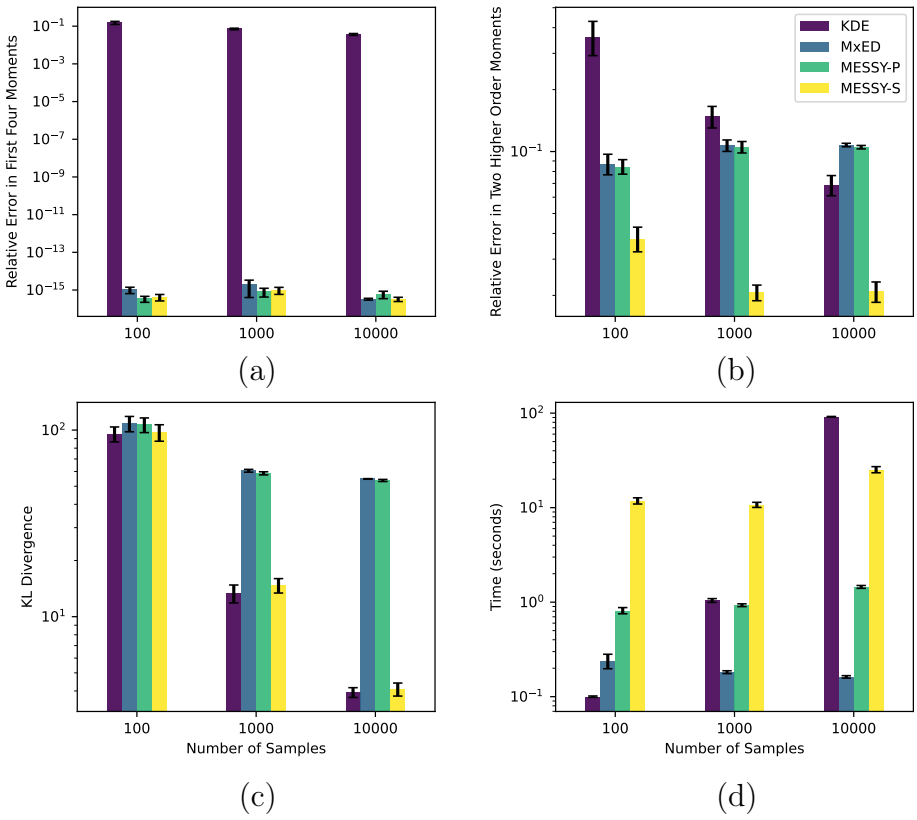


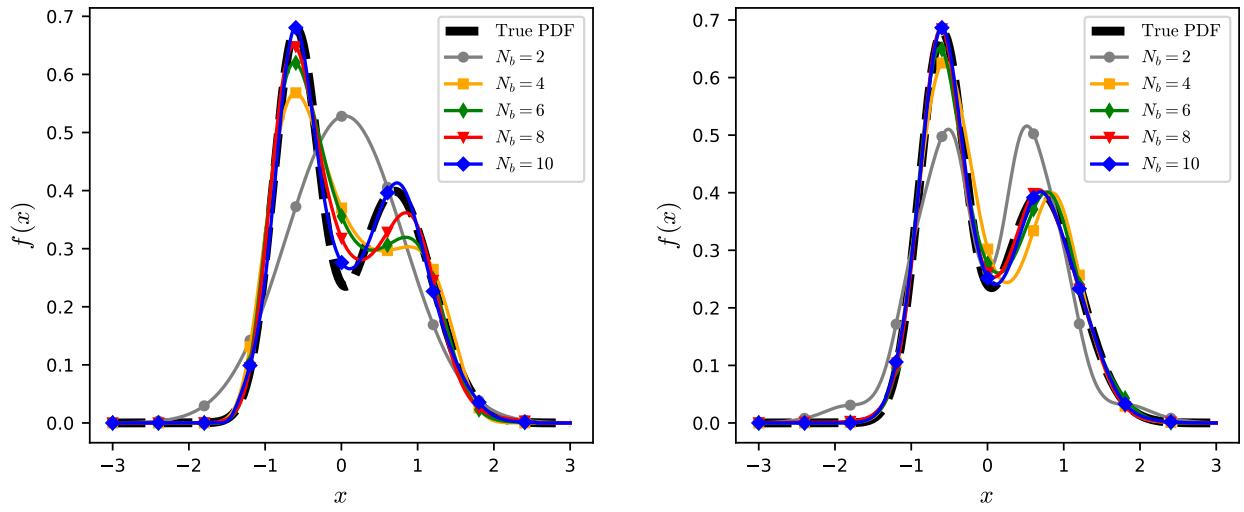
Figure 4.3: Comparing the relative error in (a) the first four moments, (b) two higher order moments (i.e. fifth and sixth moments), (c) KL Divergence, and (d) the execution time for KDE, MxED, MESSY-P, and MESSY-S in recovering distribution function for different sample sizes. Here, the error bar (in black) corresponds to the standard error of the empirical measurements.

increasing either the order of polynomial basis function, or the number of basis functions, respectively. In the case of MESSY-S, we generated symbolic expressions that are $\mathcal{O}(x^2)$. The improved agreement compared to the MESSY-P case highlights the benefit derived from non-traditional basis functions that may better represent the data.

As shown in Fig. 4.5, the MESSY-S procedure results in better-conditioned $\boldsymbol{L}^{\mathrm{ME}}$ matrices than the MESSY-P for the same degrees of freedom. However, the search for a *good* basis function increases the computational cost. In each iteration of the search for basis functions, the MESSY-S algorithm may reject symbolic basis candidates based on the condition number of the matrix $\boldsymbol{L}^{\mathrm{ME}}$. In other words, the improved performance associated with MESSY-S comes at some increased computational cost.

4.8.2 Limit of realizability

One of the challenging moment problems for maximum entropy methods is the one involving distributions near the border of physical realizability. In the one-dimensional case with moments of the first four monomials $[x, x^2, x^3, x^4]$ as the input, the moment problem is



(a) MESSY-P (b) MESSY-S

Figure 4.4: Convergence of MESSY estimation to target distribution function by (a) increasing the order of polynomial basis functions for MESSY-P or (b) increasing the number of randomly selected symbolic basis functions with $N_m = 2$ for MESSY-S.

physically realizable when

$$\int x^4 f(x)dx \geq \left(\int x^3 f(x)dx\right)^2 + 1. \tag{4.30}$$

The moment problem with moments approaching the equality in Eq. (4.30) is called *limit of realizablity* [155], [156]. We consider samples from a distribution in this limit as our test case here, since the standard MED cannot be solved due to an ill-conditioned Hessian (see [117], [153]).

In Fig. 4.6, we depict the estimated density of a bi-modal distribution in this limit given its samples with moments $\langle X \rangle = 0$, $\langle X^2 \rangle = 1$, $\langle X^3 \rangle = -2.10$ and $\langle X^4 \rangle = 5.42$. Here, we compare the density obtained using KDE, MxED, MESSY-P, and MESSY-S to the histogram of samples. In this example, we obtained the MESSY-S estimate by searching in the space of smooth functions with $N_b \in \{2, ..., 8\}$ basis functions and compare polynomial and symbolic basis functions of order 2 and 4.

In Fig. 4.7, we compare the KL Divergence, the execution time, and the condition number for each method. While KDE suffers from over-smoothing and MxED/MESSY-P require at least $N_b = 4$ (and consequently $N_m = 4$, resulting in a stiff problem with large condition number), MESSY-S can obtain accurate density estimates by using unconventional basis functions with $N_m = 2$, thus maintaining a manageable condition number.

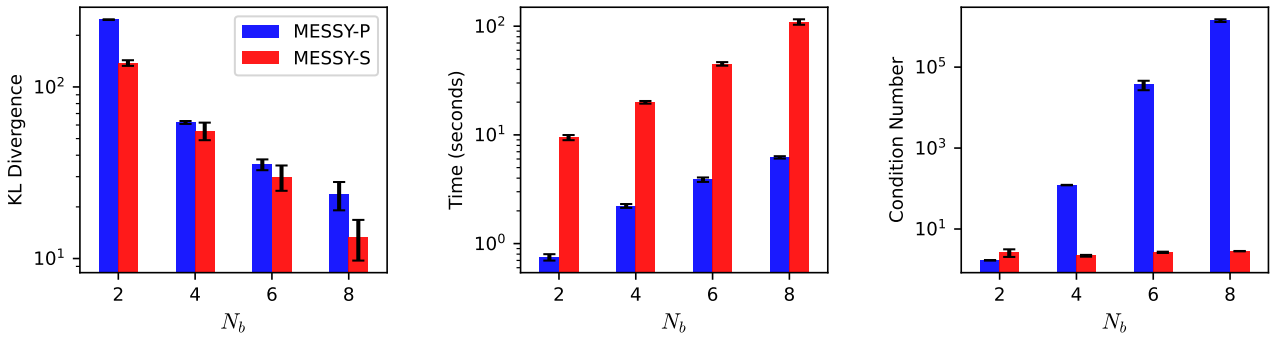


Figure 4.5: KL Divergence, execution time, and condition number against the degrees of freedom, i.e. the order of polynomial basis functions for MESSY-P or the number of symbolic basis functions with $N_m = 2$ for the MESSY-S estimate.

4.8.3 Discontinuous distributions

We now highlight the benefits of using MESSY estimation with *piecewise continuous* capability for recovering distributions with a discontinuity at the boundary. As an example, let us consider the exponential distribution with a probability density function given by

$$f(x) = \begin{cases} ae^{-ax} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.31}$$

with $a = 1$.

Given $10,000$ samples of this distribution, in Fig. 4.8 we compare KDE, MxED, and the proposed MESSY-P and MESSY-S methodologies. In the case of MxED and MESSY-P we consider second-order polynomial basis functions, and for MESSY-S we search the space of smooth functions for $N_b \in \{2, ..., 8\}$ symbolic basis functions of order $\mathcal{O}(x^2)$. For MESSY-P and MESSY-S, we apply the boundary condition

$$\hat{f}(x) = 0 \quad \forall x < \min(X). \tag{4.32}$$

By providing information about the boundedness of the expected distribution, we enable MESSY to accurately predict densities with discontinuity near the boundary. As it can be seen clearly from Fig. 4.8, in contrast to MxED, both MESSY-P and MESSY-S provide accurate predictions by taking advantage of the information about the boundedness of the target density.

The KL Divergence score and execution time for each method is shown in Fig. 4.9. These figures show that MESSY-P and MESSY-S provide a more accurate description compared to the KDE estimate, albeit at a higher computational cost.

4.8.4 Two-dimensional distributions

In this section the proposed MESSY methodology is applied to the estimation of 2-dimensional distributions. Let us consider a multivariate Gaussian distribution in two di-

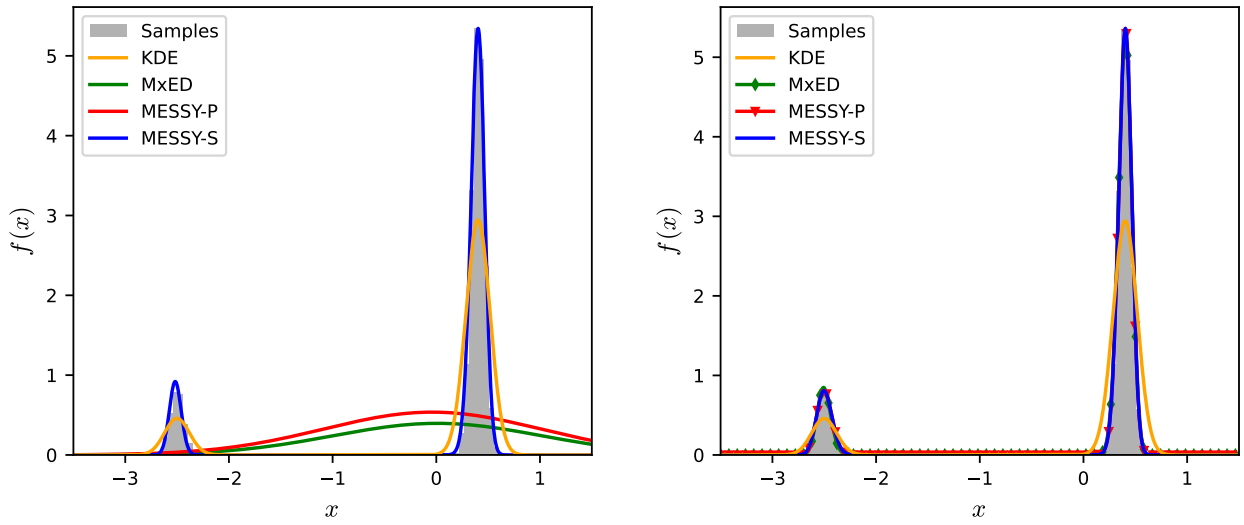


Figure 4.6: Estimating density for a case of distribution near the limit of realizability using KDE, MxED, MESSY-P, and MESSY-S. The solutions of MxED, MESSY-P, and MESSY-S are obtained using basis functions of second (left) and fourth (right) order.

mensions, i.e. $\boldsymbol{X} = (X_1, X_2) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with probability density

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}\sqrt{\det(\boldsymbol{\Sigma})}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\} \tag{4.33}$$

where $d = 2$, $\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$, $\boldsymbol{\mu}$ is the mean vector, and $\boldsymbol{\Sigma}$ is the covariance matrix. In this example,

$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}. \tag{4.34}$$

Let us also consider a more complex 2D distribution by multiplying two independent 1D distributions, namely the exponential and Gamma distributions, i.e. $X_1 \sim \text{Exp}(\lambda)$, $X_2 \sim \Gamma(k, \theta)$, and $\boldsymbol{X} = (X_1, X_2) \sim \text{Exp}(\lambda)\Gamma(k, \theta)$. In particular, we consider samples of the joint pdf given by

$$f(x_1, x_2; \lambda, k, \theta) = f(x_1; \lambda) f(x_2; k, \theta) = \begin{cases} \lambda e^{-\lambda x_1} \cdot \frac{x_2^{k-1} e^{-x_2/\theta}}{\Gamma(k)\theta^k} & x_1 \geq 0,\ x_2 \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.35}$$

where $\lambda > 0$ is the rate parameter, $k > 0$ is the shape parameter, $\theta > 0$ is the scale parameter, and $\Gamma(\cdot)$ is the gamma function. In our example, we use $\lambda = 2$, $k = 3$, and $\theta = 0.5$.

Fig. 4.10 compares the true distribution against the KDE, MESSY-P and MESSY-S estimate given 10^4 samples of the distribution. In cases of MESSY-P and MESSY-S, we consider basis functions up to 2nd order in each dimension. The figure shows that KDE,

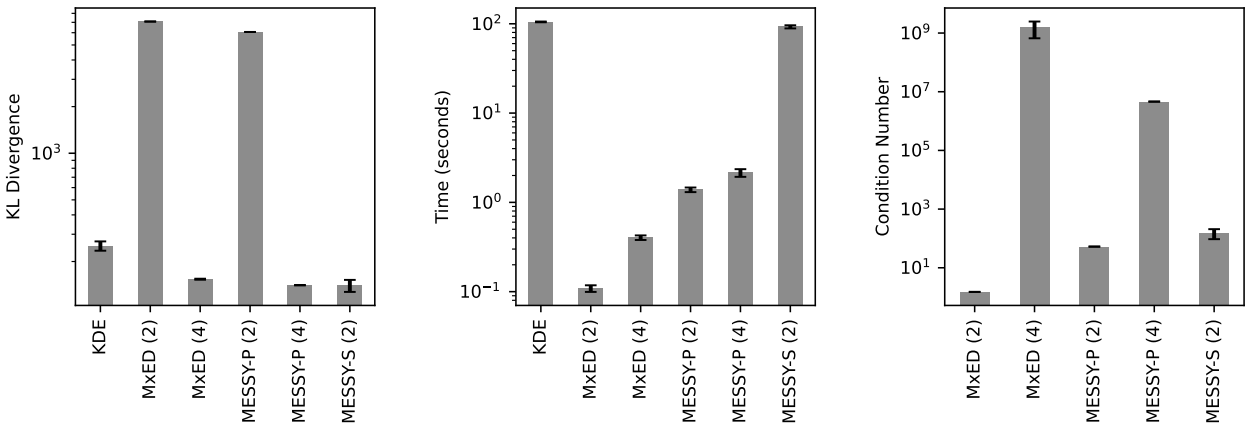


Figure 4.7: Comparing KL Divergence, execution time, and condition number of KDE, MxED, MESSY-P, and MESSY-S for an unknown distribution near the limit of the realizability. Here, we consider polynomial basis functions of second and fourth order for MxED and MESSY-P denoted by MxED (2), MxED (4), MESSY-P (2) and MESSY-P (4), respectively. In MESSY-S, we consider symbolic basis functions of second order only which we denote by MESSY-S (2).

MESSY-P and MESSY-S provide a reasonable estimate for the multivariate normal distribution. However, among all considered estimators for the case of the Gamma-exponential distribution, MESSY-S seems to provide a slightly better estimate, i.e. it captures the peak in the most probable region of the distribution.

4.8.5 Cost of MESSY-P with respect to dimension

As discussed in section 4.4.1, for a given moment problem (fixed vector of basis functions), the cost of evaluating Lagrange multipliers in the proposed method is linear with respect to number of samples and independent of dimension — a benefit of Monte Carlo integration. However, similar to the standard MED approach, the proposed MESSY estimate requires the inversion of a Hessian matrix $\boldsymbol{L} \in \mathbb{R}^{N_b \times N_b}$. Therefore, in addition to integration, a MESSY cost estimate also includes solution of a linear system of equations with cost of order $\mathcal{O}(N_b^3)$. In this example, we consider the MESSY-P estimator with basis functions $\boldsymbol{H} = \left[x_1, ..., x_d, x_1^2, x_1 x_2, ..., x_d^2\right]$ for estimating a target d-dimensional multivariate normal distribution Eq. (4.33) with mean $\boldsymbol{\mu}$

$$\mu \sim \mathcal{N}(\mathbf{0}, \frac{1}{2}\boldsymbol{I}) \tag{4.36}$$

and covariance matrix $\boldsymbol{\Sigma}$

$$\boldsymbol{\Sigma} = \boldsymbol{I} + \boldsymbol{U}\boldsymbol{U}^T \tag{4.37}$$

$$\text{where} \quad U_{i,j} = \begin{cases} 0 & i < j \\ \xi & \text{otherwise} \end{cases} \quad \text{and} \quad \xi \sim \mathcal{N}\left(0, \frac{1}{2}\right). \tag{4.38}$$

Using 2nd order polynomial basis functions, the number of unique basis functions is $N_b = d + d(d+1)/2$, leading to a cost of order $\mathcal{O}((d + d(d+1)/2)^3)$ for a d-dimensional

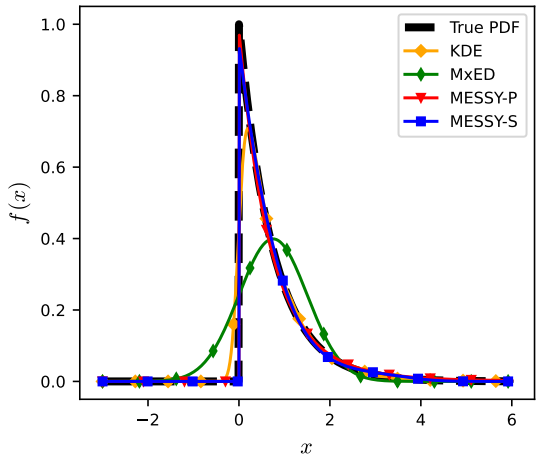


Figure 4.8: Estimating density of exponential distribution function from its samples using KDE, MxED, MESSY-P, and MESSY-S. For MxED, MESSY-P and MESSY-S, with $N_m = 2$.

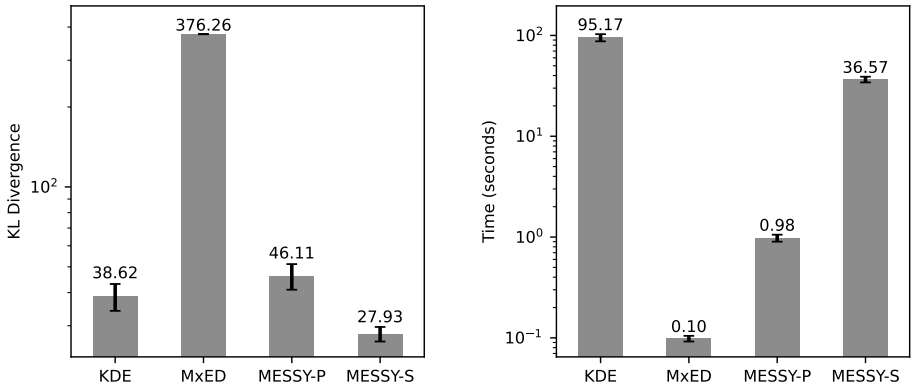


Figure 4.9: KL Divergence and execution time for KDE, MxED, MESSY-P, and MESSY-S estimation of exponential distribution function given 10,000 samples.

probability space. In Fig. 4.11-(a), we illustrate this by showing the normalized execution time (normalized by the time for $d = 1$ and given sample size) against the dimension of the target density given $N \in \{100,\ 200,\ 400,\ 800,\ 1600,\ 3200\}$ samples. Furthermore, in order to highlight the linear dependence of cost with number of samples N for a fixed moment problem, in Fig. 4.11-(b) we also report the normalized execution time of computing Lagrange multipliers (normalized by the time for $N = 100$ and dimension d of target density) as a function of number of samples.

Since the most expensive component of MESSY, i.e. the computation of Lagrange multipliers, has been carried out on a single computer core with a direct solver, we confirm that MESSY algorithm can be used for density recovery in large number of dimensions at a feasible computational cost, regardless of the scaling with dimension. The limiting factor appears to be the storage associated with solving the linear system. In case a larger system than the one studied here are of interest, one may use iterative solvers, see [157].

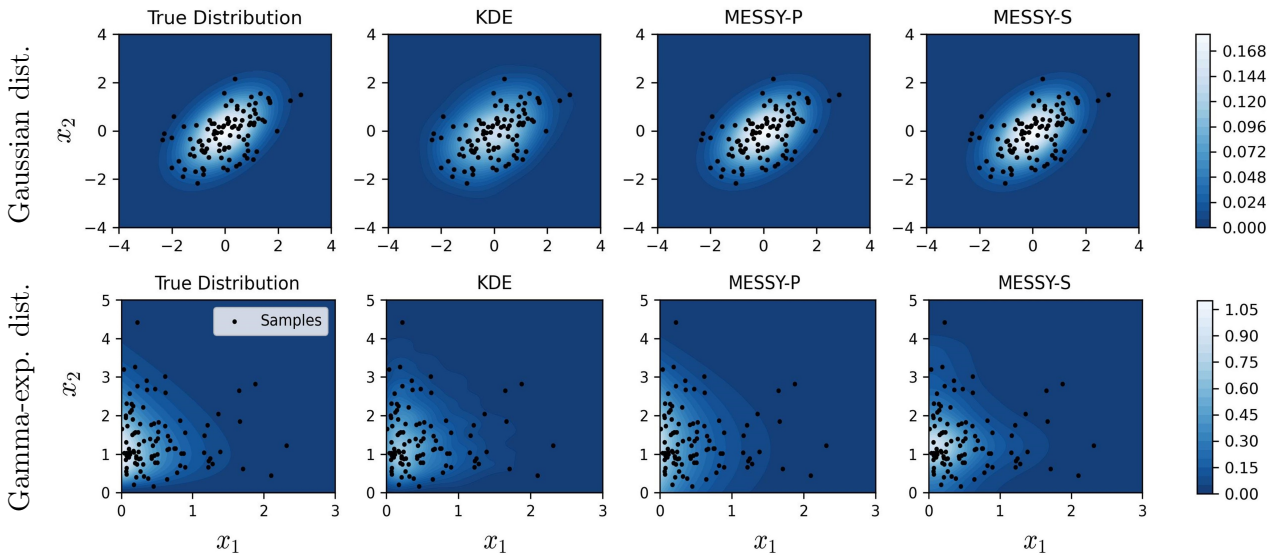
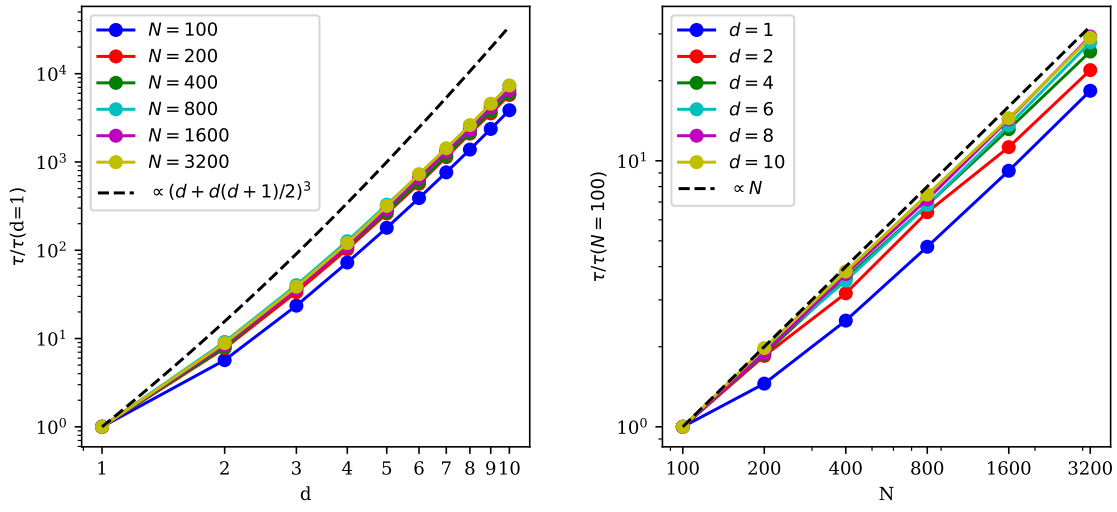


Figure 4.10: Estimating the density of samples in two dimensions using KDE, MESSY-P, and MESSY-S. We use 10,000 samples from a Gaussian (top) and the Gamma-exponential density defined in Eq. (4.35) (bottom). For better visualization, we only show 100 random samples. We also report that the KL-divergence for MESSY-S, MESSY-P and KDE are on the same order.

4.9 Conclusion and Outlook

We present a new method for symbolically recovering the underlying probability density function of a given finite number of its samples. The proposed method uses the maximum entropy distribution as an ansatz thus guaranteeing positivity and least bias. We devise a method for finding parameters of this ansatz by considering a Gradient flow in which the ansatz serves as the driving force. One main takeaway from this work is that the parameters of the MED ansatz can be computed efficiently by solving a linear problem involving moments calculated from the given samples. We also show that the complexity associated with finding Lagrange multipliers, the most expensive part of the algorithm, scales linearly with the number of samples in all dimensions. On the other hand, the cost of inverting the Hessian matrix with dimension $\mathbb{R}^{N_b \times N_b}$ is $\mathcal{O}(N_b^3)$, where the number of basis functions N_b grows with dimension. Overall, our numerical experiments show that densities of dimension 10 can be recovered using a single core of a typical workstation. It is worth noting that since the whole probability space is represented with a few parameters, the MESSY representation can reduce the training cost associated with the PDE-FIND [7] and SINDy [14] method, where in the latter the regression is done on a data set that discretizes the whole space with a large number of parameters that need to be fit. Further detailed analysis of the trade-off between cost and accuracy in inferring densities in high-dimensional probability spaces will be addressed in future work.

The second main takeaway from this work is that accurate density recovery does not necessarily require the use of high-order moments. In fact, increasing the number of complex but low-order basis functions leads to superior expressiveness and better assimilation of the



(a) (b)

Figure 4.11: Relative execution time τ of computing Lagrange multipliers given $N \in \{100, 200, 400, 800, 1600, 3200\}$ samples of d-dimensional multivariate normal distribution function where $d = 1, ..., 10$. Left (a) shows the normalized execution time versus dimension and right (b) the normalized execution time versus number of samples. The execution time is computed on a single core and single thread of 2.3GHz Quad-Core Intel Core i7 processor and averaged over 5 ensembles.

data. For this reason, the proposed method is equipped with a Monte Carlo search in the space of smooth functions for finding basis functions to describe the exponent of the MED ansatz, using KL Divergence, calculated from the unknown-distribution samples, as an optimality criterion. Discontinuous densities are treated by considering piece-wise continuous functions with support on the space covered by samples.

We validate and test the proposed MESSY estimation approach against benchmark non-parametric (KDE) and parametric (MxED) density estimation methods. In our experiments, we consider three canonical test cases; a bi-modal distribution, a distribution close to the limit of realizability, and a discontinuous distribution function. Our results suggest that MESSY estimation exhibits several positive attributes compared to existing methods. Specifically, while retaining some of the most desirable features associated with MED, namely non-negativity, least bias, and matching the moments of the unknown distribution, it outperforms standard maximum-entropy-based approaches for two reasons. First, it uses samples of the target distribution in the evaluation of the Hessian, which has a linear cost with respect to the dimension of the random variable. Second, the resulting linear problem for finding the Lagrange multipliers from moments is significantly more efficient than the Newton line search used by the classical MED approach. Moreover, our multi-level algorithm allows for recovery of more complex distributions compared to the standard MED approach. Combining the efficient approach of finding maximum entropy density via a linear system with the symbolic exploration for the optimal basis functions paves the way for achieving low bias, consistent, and expressive density recovery from samples.

Although the proposed MESSY estimate alleviates a number of numerical challenges associated with finding the MED given samples, it cannot expand the space of the MED solution. In other words, if the moment problem does not permit any MED solutions, e.g. [158], the MESSY estimate of MED fails to infer any densities since there is no MED solution to be found. Secondly, there are no guarantees that the coefficient of the leading term in the exponent of the MED will be negative. Hence, in unbounded domains the MESSY estimate may diverge in the tails of the distribution. Both of these issues can be resolved by regularizing the MED ansatz, e.g. via incorporating a Wasserstein distance from a prior distribution as suggested by [159].

The most important, perhaps, distinguishing characteristic of the proposed methodology from non-parametric approaches, such as KDE, is the ability to recover a tractable symbolic estimator of the unknown density. This can be beneficial in a number of applications, such as, for example, finding the underlying dynamics of a stochastic process. In such a case, where it is known that the transition probability takes an exponential form, this method can be used for recovering the drift and diffusion terms of an Ito process given a sequence of random variables, see e.g. [160], [161]. Furthermore, the proposed method may be used to find a relation between moments of interest which could be helpful in determining the parameters of a statistical model, such as a closure for a hierarchical moment description of fluid flow far from equilibrium. That said, exploring the efficiency of our algorithm in high-dimensional contexts is a critical next step, considering the complexity of modern machine learning datasets. Other possible directions for future work include: (i) data-driven discovery of governing dynamical laws from samples; and (ii) applications to variance reduction.

Chapter 5

ISR: Invertible Symbolic Regression

In this chapter, we introduce an Invertible Symbolic Regression (ISR) method. It is a machine learning technique that generates analytical relationships between inputs and outputs of a given dataset via invertible maps (or architectures). The proposed ISR method naturally combines the principles of Invertible Neural Networks (INNs) and Equation Learner (EQL), a neural network-based symbolic architecture for function learning. In particular, we transform the affine coupling blocks of INNs into a symbolic framework, resulting in an end-to-end differentiable symbolic invertible architecture that allows for efficient gradient-based learning. The proposed ISR framework also relies on sparsity promoting regularization, allowing the discovery of concise and interpretable invertible expressions. We show that ISR can serve as a (symbolic) normalizing flow for density estimation tasks. Furthermore, we highlight its practical applicability in solving inverse problems, including a benchmark inverse kinematics problem, and notably, a geoacoustic inversion problem in oceanography aimed at inferring posterior distributions of underlying seabed parameters from acoustic signals.

5.1 Introduction

In many applications in engineering and science, experts have developed theories about how measurable quantities result from system parameters, known as forward modeling. In contrast, the *Inverse Problem* aims to find unknown parameters of a system that lead to desirable observable quantities. A typical challenge is that numerous configurations of these parameters yield the same observable quantity, especially with underlying complicated nonlinear governing equations and where hidden parameters outnumber the observable variables. To tackle challenging and ill-posed inverse problems, a common method involves estimating a posterior distribution on the unknown parameters, given the observations. Such a probabilistic approach facilitates the uncertainty quantification by analyzing the diversity of potential inverse solutions.

An established computationally expensive approach in finding the posterior distribution is to directly generate samples using acceptance/rejection. In this scope, the Markov Chain Monte Carlo (MCMC) methods [162]–[169] offer a strong alternative for achieving near-optimum Bayesian estimation [170]–[172]. However, MCMC methods can be inefficient [173] as the number of unknown parameter increases.

When the likelihood is unknown or intractable, the Approximate Bayesian Computation (ABC) [174] is often used to estimate the posterior distribution. However, similar to MCMC, this method also suffers from poor scalability [175], [176]. A more efficient alternative is to approximate the posterior using a tractable distribution, i.e. the variational method [177]–[179]. However, the performance of the variational method deteriorates as the true posterior becomes more complicated.

Since direct sampling of the posterior distribution requires many runs of the forward map, often a trained and efficient surrogate model is used instead of the exact model. Surrogate models are considered as an efficient representation of the forward map trained on the data. Popular approaches include recently introduced Physics-Informed Neural Networks [71]. However, due to the black-box nature of neural networks, it is beneficial to express the forward map symbolically instead for several reasons. First, Symbolic Regression (SR) can provide model interpretability, while understanding the inner workings of deep Neural Networks is challenging [54], [180]. Second, studying the symbolic outcome can lead to valuable insights and provide nontrivial relations and/or physical laws [131], [132], [181], [182]. Third, they may achieve better results than Neural Networks in out-of-distribution generalization [17]. Fourth, unlike conventional regression methods, such as least squares [18], likelihood-based [19]–[21], and Bayesian regression techniques [22]–[26], SR does not rely on a fixed parametric model structure.

The attractive properties of SR, such as interpretability, often come at high computational cost compared to standard Neural Networks. This is because SR optimizes for model structure and parameters simultaneously. Therefore, SR is thought to be NP-hard [31], [32]. However tractable solutions exist, that can approximate the optimal solution suitable for applications. For instance, genetic algorithms [8], [12], [27], [28], [34] and machine learning algorithms, such as neural networks, and transformers [17], [31], [37], [43], [44], [46], [48], [50], [55]–[57], [132], are used to solve SR efficiently.

Related Works. In recent years, a branch of Machine Learning methods has emerged and is dedicated to finding data-driven invertible maps. While they are ideal for data generation and inverse problem, they lack interpretablity. On the other hand, several methods have been developed to achieve interpretablity in representing the forward map via Symbolic Regression. Hence, it is natural to incorporate SR in the invertible map for the inverse problem. Next, we review the related works in the scope of this work.

Normalizing Flows: The idea of this class of methods is to train an invertible map such that in the forward problem, the input samples are mapped to a known distribution function, e.g. the normal distribution function. Then, the unknown distribution function is found by inverting the trained map with the normal distribution as the input. This procedure is called the normalizing flow technique [123]–[129]. This method has been used for re-sampling unknown distributions, e.g. Boltzmann generators [130], as well as density recovery such as AI-Feynmann [131], [132].

Invertible Neural Networks (INNs): This method can be categorized in the class of normalizing flows. The invertibility of INN is rooted in their architecture. The most popular design is constructed by concatenating affine coupling blocks [124], [125], [183], which limits the architecture of the neural network. INNs have been shown to be effective in estimating the

posterior of the probabilistic inverse problems while outperforming MCMC, ABC, and variational methods. Applications include epidemiology [184], astrophysics [185], medicine [185], optics [186], geophysics [187], [188], and reservoir engineering [189]. However, similar to standard neural networks, INNs lead to a model that cannot be evaluated with interpretable mathematical formula.
Equation Learner (EQL): Among SR methods, the EQL network is one of the attractive methods since it incorporates gradient descent in the symbolic regression task for better efficiency [42], [50], [54]. EQL devises a neural network-based architecture for SR task by replacing commonly used activation functions with a dictionary of operators and use back-propagation for training. However, in order to obtain a symbolic estimate for the inverse problem efficiently, it is necessary to merge such efficient SR method with an invertible architecture, which is the goal of this work.

Our Contributions. We present Invertible Symbolic Regression (ISR), a machine learning technique that identifies mathematical relationships that best describe the forward and inverse map of a given dataset through the use of invertible maps. ISR is based on an invertible symbolic architecture that bridges the concepts of Invertible Neural Networks (INNs) and Equation Learner (EQL), i.e. a neural network-based symbolic architecture for function learning. In particular, we transform the affine coupling blocks of INNs into a symbolic framework, resulting in an end-to-end differentiable symbolic inverse architecture. This allows for efficient gradient-based learning. The symbolic invertible architecture is easily invertible with a tractable Jacobian, which enables explicit computation of posterior probabilities. The proposed ISR method, equipped with sparsity promoting regularization, captures complex functional relationships with concise and interpretable invertible expressions. In addition, as a byproduct, we naturally extend ISR into a conditional ISR (cISR) architecture by integrating the EQL network within conditional INN (cINN) architectures present in the literature. We further demonstrate that ISR can also serve as a symbolic normalizing flow (for density estimation) in a number of test distributions. We demonstrate the applicability of ISR in solving inverse problems, and compare it with INN on a benchmark inverse kinematics problem, as well as a geoacoustic inversion problem in oceanography (see [190] for further information). Here, we aim to characterize the undersea environment, such as water-sediment depth, sound speed, etc., from acoustic signals. To the best of our knowledge, this work is the first attempt towards finding interpretable solutions to general nonlinear inverse problems by establishing analytical relationships between measurable quantities and unknown variables via symbolic invertible maps.

The remainder of the chapter is organized as follows. In Section 5.2, we go through an overall background about Symbolic Regression (SR) and review the Equation Learner (EQL) network architecture. In Section 5.3, we introduce and present the proposed Invertible Symbolic Regression (ISR) method. We then show our results in Section 5.4, where we demonstrate the versatility of ISR as a density estimation method on a variety of examples (distributions), and then show its applicability in inverse problems on an inverse kinematics benchmark problem and through a case study in ocean geoacoustic inversion. Finally in Section 5.5, we provide our conclusions and outlook.

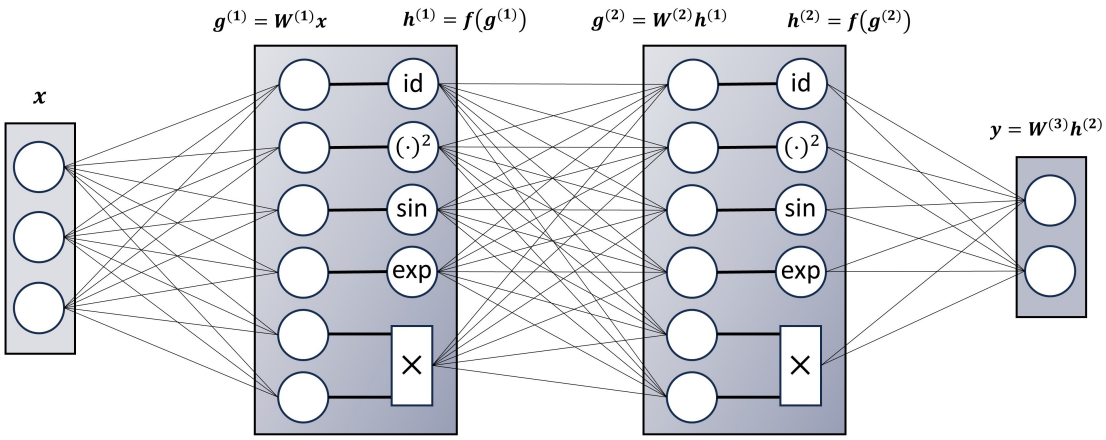


Figure 5.1: EQL network architecture for symbolic regression. For visual simplicity, we only show 2 hidden layers and 5 activation functions per layer (identity or "id", square, sine, exponential, and multiplication).

5.2 Background

Before diving into the proposed ISR method, we first delve into a comprehensive background of the Symbolic Regression (SR) task, as well as the Equation Learner (EQL) network architecture.

Symbolic Regression. Given a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$ consisting of N independent and identically distributed (i.i.d.) paired examples, where $\mathbf{x}_i \in \mathbb{R}^{d_\mathbf{x}}$ represents the input variables and $\mathbf{y}_i \in \mathbb{R}^{d_\mathbf{y}}$ the corresponding output for the i-th observation, the objective of SR is to find an analytical (symbolic) expression f that best maps inputs to outputs, i.e. $\mathbf{y}_i \approx f(\mathbf{x}_i)$. SR seeks to identify the functional form of f from the space of functions $\mathcal{S}$ defined by a set of given arithmetic operations (e.g. $+$, $-$, $\times$, $\div$) and mathematical functions (e.g. sin, cos, exp, etc.) that minimizes a predefined loss function $\mathcal{L}(f, \mathcal{D})$, which measures the discrepancy between the true outputs $\mathbf{y}_i$ and the predictions $f(\mathbf{x}_i)$ over all observations in the dataset. Unlike conventional regression methods that fit parameters within a predefined model structure, SR dynamically constructs the model structure itself, offering a powerful means to uncover underlying physical laws and/or nontrivial relationships.

Equation Learner Network. The Equation Learner (EQL) network is a multi-layer feed-forward neural network that is capable of performing symbolic regression by substituting traditional nonlinear activation functions with elementary functions. The EQL network was initially introduced by [42] and [50], and further explored by [54]. As shown in Figure 5.1, the EQL network architecture is based on a fully connected neural network where the ouput $\mathbf{h}^{(i)}$ of the i-th layer is given by

$$\mathbf{g}^{(i)} = \mathbf{W}^{(i)}\mathbf{h}^{(i-1)} \tag{5.1}$$

$$\mathbf{h}^{(i)} = f\big(\mathbf{g}^{(i)}\big) \tag{5.2}$$

where $\mathbf{W}^{(i)}$ is the weight matrix of the i-th layer, f denotes the nonlinear (symbolic) activation functions, and $\mathbf{h}^{(0)} = \mathbf{x}$ represents the input data. In regression tasks, the final layer does not typically have an activation function, so the output for a network with L hidden layers is given by

$$\mathbf{y} = \mathbf{h}^{(L+1)} = \mathbf{g}^{(L+1)} = \mathbf{W}^{(L+1)}\mathbf{h}^{(L)}. \tag{5.3}$$

In traditional neural networks, activation functions such as ReLU, tanh, or sigmoid are typically employed. However, for the EQL network, the activation function $f(\mathbf{g})$ may consist of a separate primitive function for each component of $\mathbf{g}$ (e.g. the square function, sine, exponential, etc.), and may include functions that take multiple arguments (e.g. the multiplication function). In addition, the primitive functions may be duplicated within each layer (to reduce the training's sensitivity to random initializations).

It is worth mentioning that, for visual simplicity, the schematic shown in Figure 5.1, shows an EQL network with only two hidden layers, where each layer has only five primitive functions, i.e., the activation function

$$f(\mathbf{g}) \in \big\{\text{identity, square, sine, exponential, multiplication}\big\}.$$

However, the EQL network can in fact include other functions or more hidden layers to fit a broader range (or class) of functions. Indeed, the number of hidden layers can dictate the complexity of the resulting symbolic expression and plays a similar role to the maximum depth of expression trees in genetic programming techniques. Although the EQL network may not offer the same level of generality as traditional symbolic regression methods, it is adequately capable of representing the majority of functions commonly encountered in scientific and engineering contexts. Crucially, the parametrized nature of the EQL network enables efficient optimization via gradient descent (and backpropagation).

After training the EQL network, the identified equation can be directly derived from the network weights. To avoid reaching overly complex symbolic expressions and to maintain interpretability, it is essential to guide the network towards learning the simplest expression that accurately represents the data. In methods based on genetic programming, this simplification is commonly achieved by restricting the number of terms in the expression. For the EQL network, this is attained by applying sparsity regularization to the network weights, which sets as many of these weights to zero as possible (e.g. L_1 regularization [64], $L_{0.5}$ regularization [191]). In this work, we use a smoothed $L_{0.5}$ regularization [192], [193], which was also adopted by [54].

5.3 Invertible Symbolic Regression

In this section, we delineate the problem setup for inverse problems, and then describe the proposed ISR approach.

5.3.1 Problem Specification

In various engineering and natural systems, the theories developed by experts describe how measurable (or observable) quantities $\mathbf{y} \in \mathbb{R}^{d_\mathbf{y}}$ result from the unknown (or hidden) properties $\mathbf{x} \in \mathbb{R}^{d_\mathbf{x}}$, known as the *forward process* $\mathbf{x} \to \mathbf{y}$. The goal of inverse prediction is to

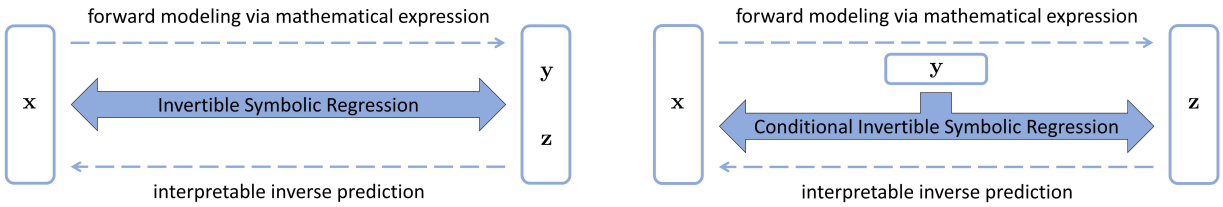


Figure 5.2: (Left) The proposed ISR framework learns a bijective symbolic transformation that maps the (unknown) variables $\mathbf{x}$ to the (observed) quantities $\mathbf{y}$ while transforming the lost information into latent variables $\mathbf{z}$. (Right) The conditional ISR (cISR) framework learns a bijective symbolic map that transforms $\mathbf{x}$ directly to a latent representation $\mathbf{z}$ given the observation $\mathbf{y}$. As we will show, both the forward and inverse mappings are efficiently computable and possess a tractable Jacobian, allowing explicit computation of posterior probabilities.

predict the unknown variables $\mathbf{x}$ from the observable variables $\mathbf{y}$, through the *inverse process* $\mathbf{y} \to \mathbf{x}$. As critical information is lost during the forward process (i.e. $d_{\mathbf{x}} \geq d_{\mathbf{y}}$), the inversion is usually intractable. Given that $f^{-1}(\mathbf{y})$ does not yield a uniquely defined solution, an effective inverse model should instead estimate the *posterior* probability distribution $p(\mathbf{x} \,|\, \mathbf{y})$ of the hidden variables $\mathbf{x}$, conditioned on the observed variable $\mathbf{y}$.

Invertible Symbolic Regression (ISR). Assume we are given a training dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, collected using forward model $\mathbf{y} = s(\mathbf{x})$ and prior $p(\mathbf{x})$. To counteract the loss of information during the forward process, we introduce latent random variables $\mathbf{z} \in \mathbb{R}^{d_{\mathbf{z}}}$ drawn from a multivariate standard normal distribution, i.e. $\mathbf{z} \sim p_{\boldsymbol{Z}}(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{d_{\mathbf{z}}})$, where $d_{\mathbf{z}} = d_{\mathbf{x}} - d_{\mathbf{y}}$. These latent variables are designed to capture the information related to $\mathbf{x}$ that is not contained in $\mathbf{y}$ [185]. In ISR, we aim to learn a bijective symbolic function $f : \mathbb{R}^{d_{\mathbf{x}}} \to \mathbb{R}^{d_{\mathbf{y}}} \times \mathbb{R}^{d_{\mathbf{z}}}$ from the space of functions defined by a set of mathematical functions (e.g. sin, cos, exp, log) and arithmetic operations (e.g. $+$, $-$, $\times$, $\div$), and such that

$$[\mathbf{y}, \mathbf{z}] = f(\mathbf{x}) = \big[f_{\mathbf{y}}(\mathbf{x}), f_{\mathbf{z}}(\mathbf{x})\big], \qquad \mathbf{x} = f^{-1}(\mathbf{y}, \mathbf{z}) \tag{5.4}$$

where $f_{\mathbf{y}}(\mathbf{x}) \approx s(\mathbf{x})$ is an approximation of the forward process $s(x)$. As discussed later, we will learn f (and hence f^{-1}) through an invertible symbolic architecture with bi-directional training. The solution of the inverse problem (i.e. the posterior $p(\mathbf{x} \,|\, \mathbf{y}^*)$) can then be found by calling f^{-1} for a fixed observation $\mathbf{y}^*$ while randomly (and repeatedly) sampling the latent variable $\mathbf{z}$ from the same standard Gaussian distribution.

Conditional Invertible Symbolic Regression (cISR). Inspired by works on conditional invertible neural networks (cINNs) [186], [194]–[196], instead of training ISR to predict $\mathbf{y}$ from $\mathbf{x}$ while transforming the lost information into latent variables $\mathbf{z}$, we train them to transform $\mathbf{x}$ directly to latent variables $\mathbf{z}$ given the observed variables $\mathbf{y}$. This is achieved by incorporating $\mathbf{y}$ as an additional input within the bijective symbolic architecture during both the forward and inverse passes (see Figure 5.2). cISR works with larger latent spaces than ISR since $d_{\mathbf{z}} = d_{\mathbf{x}}$ regardless of the dimension $d_{\mathbf{y}}$ of the observed quantities $\mathbf{y}$. Further details are provided in the following section.

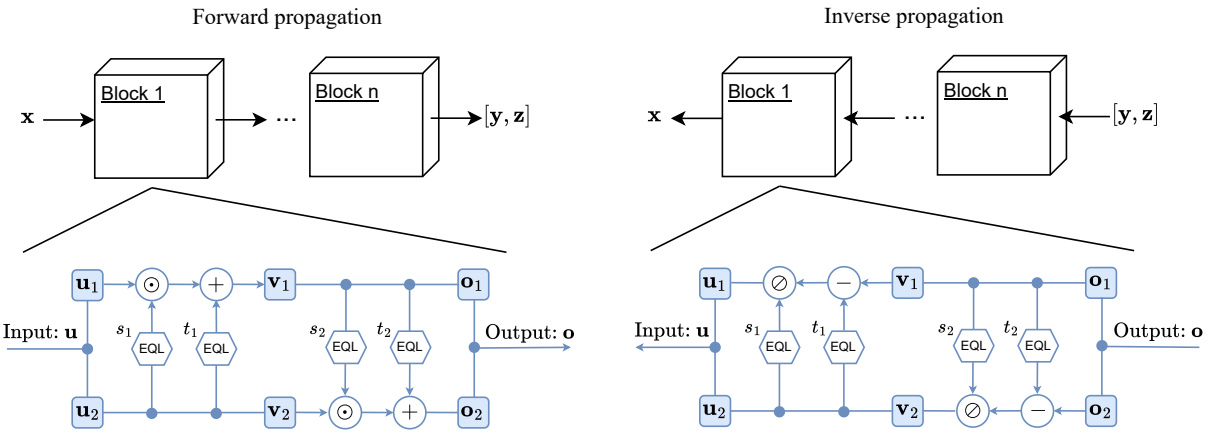


Figure 5.3: The proposed ISR method integrates EQL within the affine coupling blocks of the INN invertible architecture.[1] This results in a bijective symbolic transformation that is both easily invertible and has a tractable Jacobian. Indeed, the forward and inverse directions both possess identical computational cost. Here, $\odot$ and $\oslash$ denote element-wise multiplication and divison, respectively.

In addition to approximating the forward model via mathematical relations, ISR also identifies an interpretable inverse map via analytical expressions (see Figure 5.2). Such interpretable mappings are of particular interest in physical sciences, where an ambitious objective involves creating intelligent machines capable of generating novel scientific findings[131], [132], [181], [182], [197]. As described next, the ISR architecture is both easily invertible and has a tractable Jacobian, allowing for explicit computation of posterior probabilities.

5.3.2 Invertible Symbolic Architecture

Inspired by the architectures proposed by [124], [125], [185], we adopt a fully invertible architecture mainly defined by a sequence of n reversible blocks where each block consists of two complementary affine coupling layers. In particular, we first split the block's input $\mathbf{u} \in \mathbb{R}^{d_\mathbf{u}}$ into $\mathbf{u}_1 \in \mathbb{R}^{d_{\mathbf{u}_1}}$ and $\mathbf{u}_2 \in \mathbb{R}^{d_{\mathbf{u}_2}}$ (where $d_{\mathbf{u}_1} + d_{\mathbf{u}_2} = d_\mathbf{u}$), which are fed into the coupling layers as follows:

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 \odot \exp\left(s_1(\mathbf{u}_2)\right) + t_1(\mathbf{u}_2) \\ \mathbf{u}_2 \end{bmatrix}, \qquad \begin{bmatrix} \mathbf{o}_1 \\ \mathbf{o}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \odot \exp\left(s_2(\mathbf{v}_1)\right) + t_2(\mathbf{v}_1) \end{bmatrix}, \tag{5.5}$$

where $\odot$ denotes the Hadamard product or element-wise multiplication. The outputs $[\mathbf{o}_1, \mathbf{o}_2]$ are then concatenated again and passed to the next coupling block. The internal mappings s_1 and t_1 are functions from $\mathbb{R}^{d_{\mathbf{u}_2}} \to \mathbb{R}^{d_{\mathbf{u}_1}}$, and s_2 and t_2 are functions from $\mathbb{R}^{d_{\mathbf{u}_1}} \to \mathbb{R}^{d_{\mathbf{u}_2}}$. In general, s_i and t_i can be arbitrarily complicated functions (e.g. neural networks as in [185]). In our proposed ISR approach, they are represented by EQL networks (see Figure 5.3), resulting in a fully symbolic invertible architecture. Moving forward, we shall refer to them as EQL *subnetworks* of the block.

[1] As direct division can lead to numerical issues, we apply the exponential function to s_i (after clipping its extreme values) in the formulation described in Eq. (5.5). This also guarantees non-zero diagonal entries

The transformations above result in upper and lower triangular Jacobians:

$$J_{\mathbf{u}\mapsto\mathbf{v}} = \begin{bmatrix} \operatorname{diag}\big(\exp\big(s_1(\mathbf{u}_2)\big)\big) & \frac{\partial \mathbf{v}_1}{\partial \mathbf{u}_2} \\ 0 & I \end{bmatrix}, \qquad J_{\mathbf{v}\mapsto\mathbf{o}} = \begin{bmatrix} I & 0 \\ \frac{\partial \mathbf{o}_2}{\partial \mathbf{v}_1} & \operatorname{diag}\big(\exp\big(s_2(\mathbf{v}_1)\big)\big) \end{bmatrix}. \quad (5.6)$$

Hence, their determinants can be trivially computed:

$$\begin{aligned} \det\big(J_{\mathbf{u}\mapsto\mathbf{v}}\big) &= \textstyle\prod_{i=1}^{d_{\mathbf{u}_1}} \exp\big(\,[s_1(\mathbf{u}_2)]_i\,\big) = \exp\Big(\textstyle\sum_{i=1}^{d_{\mathbf{u}_1}} [s_1(\mathbf{u}_2)]_i\Big), \\ \det\big(J_{\mathbf{v}\mapsto\mathbf{o}}\big) &= \textstyle\prod_{i=1}^{d_{\mathbf{u}_2}} \exp\big(\,[s_2(\mathbf{v}_1)]_i\,\big) = \exp\Big(\textstyle\sum_{i=1}^{d_{\mathbf{u}_2}} [s_2(\mathbf{v}_1)]_i\Big)\,. \end{aligned} \quad (5.7)$$

Then, the resulting Jacobian determinant of the coupling block is given by

$$\begin{aligned} \det\big(J_{\mathbf{u}\mapsto\mathbf{o}}\big) &= \det\big(J_{\mathbf{u}\mapsto\mathbf{v}}\big)\cdot\det\big(J_{\mathbf{v}\mapsto\mathbf{o}}\big) \\ &= \exp\Big(\textstyle\sum_{i=1}^{d_{\mathbf{u}_1}} [s_1(\mathbf{u}_2)]_i\Big)\cdot\exp\Big(\textstyle\sum_{i=1}^{d_{\mathbf{u}_2}} [s_2(\mathbf{v}_1)]_i\Big) \\ &= \exp\Big(\textstyle\sum_{i=1}^{d_{\mathbf{u}_1}} [s_1(\mathbf{u}_2)]_i + \sum_{i=1}^{d_{\mathbf{u}_2}} [s_2(\mathbf{v}_1)]_i\Big) \\ &= \exp\Big(\textstyle\sum_{i=1}^{d_{\mathbf{u}_1}} [s_1(\mathbf{u}_2)]_i + \sum_{i=1}^{d_{\mathbf{u}_2}} \big[s_2\big(\mathbf{u}_1\odot\exp\big(s_1(\mathbf{u}_2)\big)+t_1(\mathbf{u}_2)\big)\big]_i\Big) \end{aligned} \quad (5.8)$$

which can be efficiently calculated. Indeed, the Jacobian determinant of the whole map $\mathbf{x}\to[\mathbf{y},\mathbf{z}]$ is the product of the Jacobian determinants of the n underlying coupling blocks (see Figure 5.3).
Given the output $\mathbf{o}=[\mathbf{o}_1,\mathbf{o}_2]$, the expressions in Eqs. (5.5) are clearly invertible:

$$\mathbf{u}_2 = \big(\mathbf{o}_2 - t_2(\mathbf{o}_1)\big)\oslash\exp\big(s_2(\mathbf{o}_1)\big), \qquad \mathbf{u}_1 = \big(\mathbf{o}_1 - t_1(\mathbf{u}_2)\big)\oslash\exp\big(s_1(\mathbf{u}_2)\big) \quad (5.9)$$

where $\oslash$ denotes element-wise division. Crucially, even when the coupling block is inverted, the EQL subnetworks s_i and t_i need *not* themselves be invertible; they are only ever evaluated in the forward direction. We denote the whole ISR map $\mathbf{x}\to[\mathbf{y},\mathbf{z}]$ as $f(\mathbf{x};\theta)=\big[f_{\mathbf{y}}(\mathbf{x};\theta),f_{\mathbf{z}}(\mathbf{x};\theta)\big]$ parameterized by the EQL subnetworks parameters θ, and the inverse as $f^{-1}(\mathbf{y},\mathbf{z};\theta)$.

Remark 5.3.1. The proposed ISR architecture consists of a sequence of these symbolic reversible blocks. To enhance the model's predictive and expressive capability, we can: i) increase the number of reversible coupling blocks, ii) increase the number of hidden layers in each underlying EQL network, iii) increase the number of hidden neurons per layer in each underlying EQL network, or iv) increase the complexity of the symbolic activation functions used in the EQL network. However, it is worth noting that these enhancements come with a trade-off, as they inevitably lead to a decrease in the model's interpretability.

Remark 5.3.2. To further improve the model capacity, as in [185], we incorporate (random, but fixed) permutation layers between the coupling blocks, which shuffles the input elements for subsequent coupling blocks. This effectively randomizes the configuration of splits $\mathbf{u}=[\mathbf{u}_1,\mathbf{u}_2]$ across different blocks, thereby enhancing interplay between variables.

in the Jacobian matrices.

Remark 5.3.3. Inspired by [185], we split the coupling block's input vector $\mathbf{u} \in \mathbb{R}^{d_\mathbf{u}}$ into two halves, i.e. $\mathbf{u}_1 \in \mathbb{R}^{d_{\mathbf{u}_1}}$ and $\mathbf{u}_2 \in \mathbb{R}^{d_{\mathbf{u}_2}}$ where $d_{\mathbf{u}_1} = \frac{d_\mathbf{u}}{2}$ and $d_{\mathbf{u}_2} = d_\mathbf{u} - d_{\mathbf{u}_1}$. In the case where $\mathbf{u}$ is one-dimensional (or scalar), i.e. $d_\mathbf{u} = 1$ and $\mathbf{u} \in \mathbb{R}$, we pad it with an extra zero (so that $d_\mathbf{u} = 2$) along with a loss term that prevents the encoding of information in the extra dimension (e.g. we use the L_2 loss to maintain those values near zero).

Remark 5.3.4. The proposed ISR architecture is also compatible with the conditional ISR (cISR) framework proposed in the previous section. In essence, cISR identifies a bijective symbolic transformation directly between $\mathbf{x}$ and $\mathbf{z}$ given the observation $\mathbf{y}$. This is attained by feeding $\mathbf{y}$ as an extra input to each coupling block, during both the forward and inverse passes. In particular, and as suggested by [194]–[196], we adapt the same coupling layers given by Eqs. (5.5) and (5.9) to produce a conditional coupling block. Since the subnetworks s_i and t_i are never inverted, we enforce the condition on the observation by concatenating $\mathbf{y}$ to their inputs without losing the invertibility, i.e. we replace $s_1(\mathbf{u}_2)$ with $s_1(\mathbf{u}_2, \mathbf{y})$, etc. In complex settings, the condition $\mathbf{y}$ is first fed into a separate feed-forward conditioning network, resulting in higher-level conditioning features that are then injected into the conditional coupling blocks. Although cISR can have better generative properties [196], it leads to more complex symbolic expressions and less interpretability as it explicitly conditions the map on the observation within the symbolic formulation. We denote the entire cISR forward map $\mathbf{x} \to \mathbf{z}$ as $f(\mathbf{x}; \mathbf{y}, \theta)$ parameterized by θ, and the inverse as $f^{-1}(\mathbf{z}; \mathbf{y}, \theta)$.

5.3.3 Maximum Likelihood Training of ISR

We train the proposed ISR model to learn a bijective symbolic transformation $f : \mathbb{R}^{d_\mathbf{x}} \to \mathbb{R}^{d_\mathbf{y}} \times \mathbb{R}^{d_\mathbf{z}}$. There are various choices to define the loss functions with different advantage and disadvantages [185], [195], [196], [198], [199]. As reported in [196], there are two main training approaches:

i) A standard supervised L_2 loss for fitting the model's $\mathbf{y}$ predictions to the training data, combined with a Maximum Mean Discrepancy (MMD) [185], [200] for fitting the latent distribution $p_{\boldsymbol{Z}}(\mathbf{z})$ to $\mathcal{N}(\mathbf{0}, \boldsymbol{I}_{d_\mathbf{z}})$, given samples.

ii) A Maximum Likelihood Estimate (MLE) loss that enforces $\mathbf{z}$ to be standard Gaussian, i.e. $\mathbf{z} \sim p_{\boldsymbol{Z}}(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{d_\mathbf{z}})$ and by approximating the distribution on $\mathbf{y}$ with a Gaussian distribution around the ground truth values $\mathbf{y}_{\text{gt}}$ with very low variance σ^2 [124], [196], [199].

Given that MLE is shown to perform well as reported in the literature [185], we apply it here. Next, we demonstrate how this approach is equivalent to minimizing the forward Kullback-Leibler (KL) divergence as the cost (cf. [201]). We note that given the map $f(\mathbf{x}; \theta) \mapsto [\mathbf{z}, \mathbf{y}]$, parameterized by θ, and assuming $\mathbf{y}$ and $\mathbf{z}$ are independent, the density $p_\mathbf{x}$ relates to $p_\mathbf{y}$ and $p_\mathbf{z}$ through the change-of-variables formula

$$p_{\boldsymbol{X}}(\mathbf{x}; \theta) = p_{\boldsymbol{Y}}\big(\mathbf{y} = f_\mathbf{y}(\mathbf{x}; \theta)\big)\; p_{\boldsymbol{Z}}\big(\mathbf{z} = f_\mathbf{z}(\mathbf{x}; \theta)\big) \cdot \Big|\det\big(J_{\mathbf{x} \mapsto [\mathbf{z}, \mathbf{y}]}(\mathbf{x}; \theta)\big)\Big| \,. \tag{5.10}$$

where $J_{\mathbf{x} \mapsto [\mathbf{z}, \mathbf{y}]}(\mathbf{x}; \theta)$ denotes the Jacobian of the map f parameterized by θ. This expression is then used to define the loss function, which we derive by following the work in [201].

In particular, we aim to minimize the forward KL divergence between a target distribution $p^*_{\mathbf{X}}(\mathbf{x})$ and our *flow-based* model $p_{\mathbf{X}}(\mathbf{x};\theta)$, given by

$$\begin{aligned}\mathcal{L}(\theta) &= D_{\mathrm{KL}}\big[p^*_{\mathbf{X}}(\mathbf{x}) \,\big|\big|\, p_{\mathbf{X}}(\mathbf{x};\theta)\big] \\ &= -\mathbb{E}_{p^*_{\mathbf{X}}(\mathbf{x})}\big[\log p_{\mathbf{X}}(\mathbf{x};\theta)\big] + \text{const.} \\ &= -\mathbb{E}_{p^*_{\mathbf{X}}(\mathbf{x})}\big[\log p_{\mathbf{Y}}\big(f_{\mathbf{y}}(\mathbf{x};\theta)\big) + \log p_{\mathbf{Z}}\big(f_{\mathbf{z}}(\mathbf{x};\theta)\big) + \log\big|\det\big(J_{\mathbf{x}\mapsto[\mathbf{z},\mathbf{y}]}(\mathbf{x};\theta)\big)\big|\big] + \text{const.}\end{aligned} \tag{5.11}$$

The forward KL divergence is particularly suitable for cases where we have access to samples from the target distribution, but we cannot necessarily evaluate the target density $p^*_{\mathbf{X}}(\mathbf{x})$. Assuming we have a set of samples $\{\mathbf{x}_i\}_{i=1}^N$ from $p^*_{\mathbf{X}}(\mathbf{x})$, we can approximate the expectation in Eq. (5.11) using Monte Carlo integration as

$$\mathcal{L}(\theta) \approx -\frac{1}{N}\sum_{i=1}^{N}\Big(\log p_{\mathbf{Y}}\big(f_{\mathbf{y}}(\mathbf{x}_i;\theta)\big) + \log p_{\mathbf{Z}}\big(f_{\mathbf{z}}(\mathbf{x}_i;\theta)\big) + \log\big|\det\big(J_{\mathbf{x}\mapsto[\mathbf{z},\mathbf{y}]}(\mathbf{x}_i;\theta)\big)\big| + \text{const.}\Big)\,. \tag{5.12}$$

As we can see, minimizing the above Monte Carlo approximation of the KL divergence is equivalent to maximizing likelihood (or minimizing negative log-likelihood). Assuming $p_{\mathbf{Z}}$ is standard Gaussian and $p_{\mathbf{Y}}$ is a multivariate normal distribution around $\mathbf{y}_{\mathrm{gt}}$, the negative log-likelihood (NLL) loss in Eq. (5.12) becomes

$$\mathcal{L}_{\mathrm{NLL}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\cdot\frac{\big(f_{\mathbf{y}}(\mathbf{x}_i;\theta)-\mathbf{y}_{\mathrm{gt}}\big)^2}{\sigma^2} + \frac{1}{2}\cdot f_{\mathbf{z}}(\mathbf{x}_i;\theta)^2 - \log\big|\det\big(J_{\mathbf{x}\mapsto[\mathbf{z},\mathbf{y}]}(\mathbf{x}_i;\theta)\big)\big|\right)\,. \tag{5.13}$$

In other words, we find the optimal ISR parameters θ by minimizing the NLL loss in Eq. (5.13), and the resulting bijective symbolic expression can be directly extracted from the these optimal parameters.

Remark 5.3.5. We note that cISR is also suited for maximum likelihood training. Given the conditioning observation $\mathbf{y}$, the density $p_{\mathbf{X}\,|\,\mathbf{Y}}$ relates to $p_{\mathbf{Z}}$ through the change-of-variables formula

$$p_{\mathbf{X}\,|\,\mathbf{Y}}(\mathbf{x}\,|\,\mathbf{y},\theta) = p_{\mathbf{Z}}\big(\mathbf{z} = f(\mathbf{x};\mathbf{y},\theta)\big)\cdot\big|\det\big(J_{\mathbf{x}\mapsto\mathbf{z}}(\mathbf{x};\mathbf{y},\theta)\big)\big|\,, \tag{5.14}$$

where $J_{\mathbf{x}\mapsto\mathbf{z}}(\mathbf{x};\mathbf{y},\theta)$ indicates the Jacobian of the map f conditioned on $\mathbf{y}$ and parameterized by θ. Following the same procedure as above, the cISR model can be trained by minimizing the following NLL loss function

$$\mathcal{L}_{\mathrm{NLL}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\cdot f(\mathbf{x}_i;\mathbf{y}_i,\theta)^2 - \log\big|\det\big(J_{\mathbf{x}\mapsto\mathbf{z}}(\mathbf{x}_i;\mathbf{y}_i,\theta)\big)\big|\right)\,. \tag{5.15}$$

As we will show in the next section, if we ignore the condition on the observation $\mathbf{y}$, the loss in Eq. 5.15 can also be used for training ISR as a normalizing flow for the unsupervised learning task of approximating a target probability density function from samples (cf. Eq. 5.18).

5.4 Results

We evaluate our proposed ISR method on a variety of problems. We first show how ISR can serve as a normalizing flow for density estimation tasks on several test distributions. We then demonstrate the capabilities of ISR in solving inverse problems by considering two synthetic problems and then a more challenging application in ocean acoustics [172], [202]–[207]. We mainly compare our ISR approach against INN [185] throughout our experiments. Further experimental details can be found in Appendix D.2.

5.4.1 Leveraging ISR for Density Estimation via Normalizing Flow

Given N independently and identically distributed (i.i.d.) samples, i.e. $\{\mathbf{X}_i\}_{i=1}^N \sim p_{\mathbf{x}}^{\text{target}}$, we would like to estimate the target density $p_{\mathbf{x}}^{\text{target}}$ and generate new samples from it. This problem categorizes as the density estimation where non-parametric, e.g. Kernel Density Estimation [106], and parametric estimators, e.g. Maximum Entropy Distribution as the least biased estimator [10], are classically used. In recent years, this problem has been approached using normalizing flow equipped with invertible map, which has gained a great deal of interest in the generative AI task. In an attempt to introduce interpretability in the trained model, we extend invertible normalizing flow to symbolic framework using the proposed ISR architecture.

In order to use the proposed ISR method as the normalizing flow for the unsupervised task of resampling from an intractable target distribution, we drop out $\mathbf{y}$ and enforce $d_{\mathbf{x}} = d_{\mathbf{z}}$.[2] In this case, we aim to learn a an invertible and symbolic map $f : \mathbb{R}^{d_{\mathbf{x}}} \to \mathbb{R}^{d_{\mathbf{z}}}$, parameterized by θ such that

$$\mathbf{z} = f(\mathbf{x}; \theta), \qquad \mathbf{x} = f^{-1}(\mathbf{z}; \theta), \tag{5.16}$$

where $\mathbf{z} \sim p_{\mathbf{z}}$ is the standard normal distribution function, which is easy to sample from. Using the change-of-variables formula, the density $p_{\mathbf{x}}$ relates to the density $p_{\mathbf{z}}$ via

$$p_{\mathbf{x}}(\mathbf{x}; \theta) = p_{\mathbf{z}}\big(\mathbf{z} = f(\mathbf{x}; \theta)\big) \cdot \left|\det\big(J_{\mathbf{x} \mapsto \mathbf{z}}(\mathbf{x}; \theta)\big)\right|, \tag{5.17}$$

where $J_{\mathbf{x} \mapsto \mathbf{z}}$ indicates the Jacobian of the map f parameterized by θ. Following the same procedure outlined in Section 5.3.3, the model can be trained by minimizing the following NLL loss function

$$\mathcal{L}_{\text{NLL}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{2} \cdot f(\mathbf{x}_i; \theta)^2 - \log \left|\det\big(J_{\mathbf{x} \mapsto \mathbf{z}}(\mathbf{x}_i; \theta)\big)\right| \right) . \tag{5.18}$$

We compare the proposed ISR approach with INN in recovering several two-dimensional target distributions (i.e. $d_{\mathbf{x}} = d_{\mathbf{z}} = 2$). First, we consider a fairly simple multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with mean $\boldsymbol{\mu} = [0, 3]$ and covariance matrix $\boldsymbol{\Sigma} = \frac{1}{10} \cdot \boldsymbol{I}_2$ as the target density. Then, we consider more challenging distributions: the "Banana,"

[2] In the absence of $\mathbf{y}$, cISR and ISR are equivalent, so we simply refer to them as ISR. Similarly, INN and cINN become equivalent, and we simply refer to them as INN.

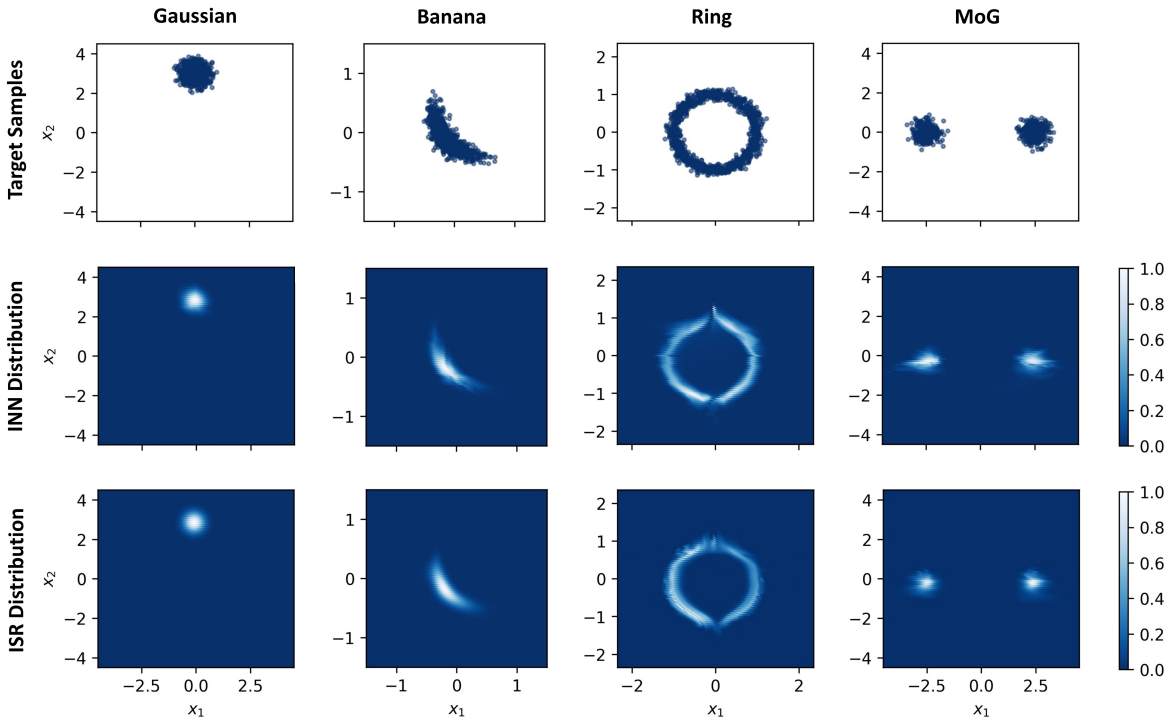


Figure 5.4: Samples from four different target densities (first row), and their estimated distributions using INN (second row) and the proposed ISR method (third row).

"Mixture of Gaussians (MoG)," and "Ring" distributions that are also considered in [208], [209]. For each of these target distributions, we draw $N_s = 10^4$ i.i.d. samples and train an invertible map that transport the samples to a standard normal distribution. This is called normalizing flow, where we intend to compare the standard INN with the proposed ISR architecture. Here, we use a single coupling block for the Gaussian and banana cases, and two coupling blocks for the ring and MoG test cases.

As shown in Figure 5.4, the proposed ISR method finds and generates samples of the considered target densities with slightly better accuracy than INN. We report the invertible symbolic expressions in Table D.1 of Appendix D.1. For instance, the first target distribution considered in Figure 5.4 is the two-dimensional multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with mean $\boldsymbol{\mu} = [0, 3]$ and covariance matrix $\boldsymbol{\Sigma} = \frac{1}{10} \cdot \boldsymbol{I}_2$. This is indeed a shifted and scaled standard Gaussian distribution where we know the analytical solution to the true map:

$$\begin{aligned} \mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} &\sim \mathcal{N}\left(\boldsymbol{\mu}, \frac{1}{10} \cdot \boldsymbol{I}_2\right) \\ &\sim \boldsymbol{\mu} + \sqrt{\frac{1}{10}} \cdot \mathcal{N}\left(\mathbf{0}, \boldsymbol{I}_2\right) \\ &= \boldsymbol{\mu} + \frac{1}{\sqrt{10}} \cdot \mathbf{Z} = \begin{bmatrix} 0 \\ 3 \end{bmatrix} + 0.316 \cdot \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} 0.316\, Z_1 \\ 3 + 0.316\, Z_2 \end{bmatrix} . \end{aligned} \tag{5.19}$$

As shown in Table D.1 of Appendix D.1, for this Gaussian distribution example, the proposed

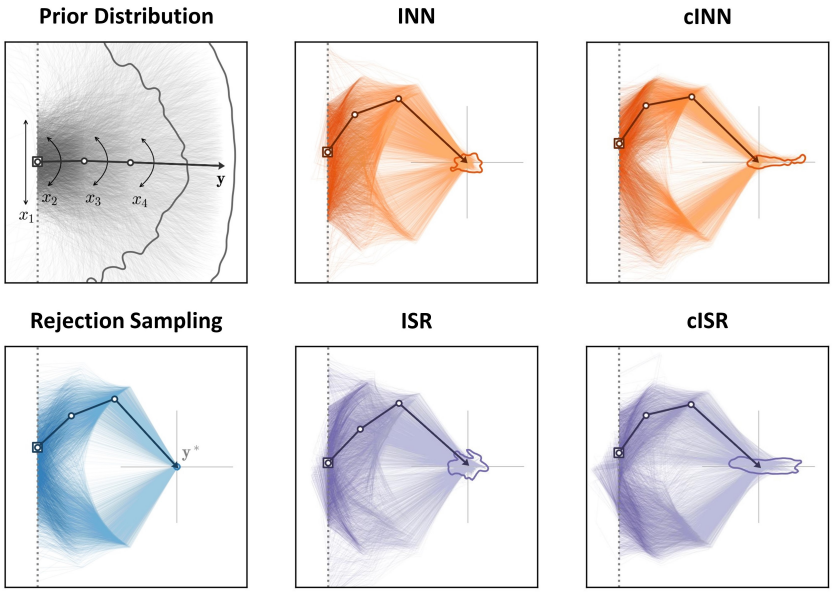


Figure 5.5: Results for the inverse kinematics benchmark problem. The faint colored lines indicate sampled arm configurations $\mathbf{x}$ taken from each model's predicted posterior $\hat{p}(\mathbf{x} \,|\, \mathbf{y}^*)$, conditioned on the target end point $\mathbf{y}^*$, which is indicated by a gray cross. The contour lines around the target end point enclose the regions containing 97% of the sampled arms' end points. We emphasize the arm with the highest estimated likelihood as a bold line.

ISR method finds the following bijective expression:

$$z_1 = x_1 \cdot e^{1.16} = 3.19\, x_1 \qquad \Longleftrightarrow \qquad x_1 = \frac{1}{3.19}\, z_1 = 0.313\, z_1$$

$$z_2 = x_2 \cdot e^{1.14} - 9.39 = 3.13\, x_2 - 9.39 \qquad \Longleftrightarrow \qquad x_2 = \frac{1}{3.13}\,(z_2 + 9.39) = 0.319\, z_2 + 3.00$$

In other words, the proposed ISR method identifies the true underlying transformation given by Eq. (5.19) with a high accuracy.

5.4.2 Inverse Kinematics

We now consider a geometrical benchmark example used by [185], [196], which simulates an inverse kinematics problem in a two-dimensional space: A multi-jointed 2D arm moves vertically along a rail and rotates at three joints. In this problem, we are interested in the configurations (i.e. the four degrees of freedom) of the arm that place the arm's end point at a given position. The forward process computes the coordinates of the end point $\mathbf{y} \in \mathbf{R}^2$, given a configuration $\mathbf{x} \in \mathbf{R}^4$ (i.e. $d_\mathbf{x} = 4$, $d_\mathbf{y} = 2$, and hence $d_\mathbf{z} = 2$). In particular, the forward process takes $\mathbf{x} = [x_1, x_2, x_3, x_4]$ as argument, where x_1 denotes the arm's starting height, and x_2, x_3, x_4 are its three joint angles, and returns the coordinates of its end point $\mathbf{y} = [y_1, y_2]$ given by

$$\begin{aligned} y_1 &= \ell_1 \sin(x_2) + \ell_2 \sin(x_2 + x_3) + \ell_3 \sin(x_2 + x_3 + x_4) + x_1 \\ y_2 &= \ell_1 \cos(x_2) + \ell_2 \cos(x_2 + x_3) + \ell_3 \cos(x_2 + x_3 + x_4) \end{aligned} \tag{5.20}$$

where the segment lengths $\ell_1 = 0.5$, $\ell_2 = 0.5$, and $\ell_3 = 1$. The parameters $\mathbf{x}$ follow a Gaussian prior $\mathbf{x} \sim \mathcal{N}\big(\mathbf{0}, \boldsymbol{\sigma}^2 \cdot \boldsymbol{I}_4\big)$ with $\boldsymbol{\sigma}^2 = [0.25^2, 0.25, 0.25, 0.25]$, which favors a configuration with a centered origin and 180° joint angles (see Figure 5.5). We consider a training dataset of size 10^6, constructed using this Gaussian prior and the forward process in Eq. (5.20). The inverse problem here asks to find the posterior distribution $p(\mathbf{x} \,|\, \mathbf{y}^*)$ of all possible configurations (or parameters) $\mathbf{x}$ that result in the arm's end point being positioned at a given $\mathbf{y}^*$ location. This inverse kinematics problem, being low-dimensional, offers computationally inexpensive forward (and backward) process, which enables fast training, intuitive visualizations, and an approximation of the true posterior estimates via rejection sampling.[3]

An example of a challenging end point $\mathbf{y}^*$ is shown in Figure 5.5, where we compare the proposed ISR method against the approximate true posterior (obtained via rejection sampling), as well as INN. The chosen $\mathbf{y}^*$ is particularly challenging, since this end point is unlikely under the prior $p(\mathbf{x})$, and results in a strongly bi-modal posterior $p(\mathbf{x} \,|\, \mathbf{y}^*)$ [185], [196]. As we can observe in Figure 5.5, compared to rejection sampling, all the considered architectures (i.e. INN, cINN, ISR, and cISR) are able to capture the two symmetric modes well. However, we can clearly see that they all generate $\mathbf{x}$-samples such that their resulting end points miss the target $\mathbf{y}^*$ by a wider margin. Quantitative results are also provided in Appendix D.3.

5.4.3 Application: Geoacoustic Inversion

Predicting acoustic propagation at sea is vital for various applications, including sonar performance forecasting and mitigating noise pollution at sea. The ability to predict sound propagation in a shallow water environment depends on understanding the seabed's geoacoustic characteristics. Inferring those characteristics from ocean acoustic measurements (or signals) is known as geoacoustic inversion (GI). GI involves several components: (i) representation of the ocean environment, (ii) selection of the inversion method, including the forward propagation model implemented, and (iii) quantification of the uncertainty related to the parameters estimates.

We start by describing the ocean environment. We consider the setup of SWellEx-96 [210], [211], which was an experiment done off the coast of San Diego, CA, near Point Loma. This experimental setting is one of the most used, documented, and understood studies in the undersea acoustics community.[4] As depicted in Figure 5.6, the data is collected via a vertical line array (VLA). The specification of the 21 hydrophones of the VLA and sound speed profile (SSP) in the water column is provided in the SWellEx-96 documentation. The SSP and sediment parameters are considered to be range-independent. Water depth refers to the depth of the water at the array. The source is towed by a research vessel which consists

[3] **Rejection sampling.** Assume we require N_s samples of $\mathbf{x}$ from the posterior $p(\mathbf{x} \,|\, \mathbf{y}^*)$ given some observation $\mathbf{y}^*$. After setting some acceptance threshold ϵ, we iteratively generate $\mathbf{x}$-samples from the prior. For each sample, we simulate the corresponding $\mathbf{y}$-values and only keep those with $\text{dist}(\mathbf{y}, \mathbf{y}^*) < \epsilon$. The process is repeated until N_s samples are collected (or accepted). Indeed, the smaller the threshold ϵ, the more $\mathbf{x}$-samples candidates (and hence the more simulations) have to be generated. Hence, we adopt this approach in this low-dimensional inverse kinematics problem, where we can afford to run the forward process (or simulation) a huge number of times.

[4] see http://swellex96.ucsd.edu/

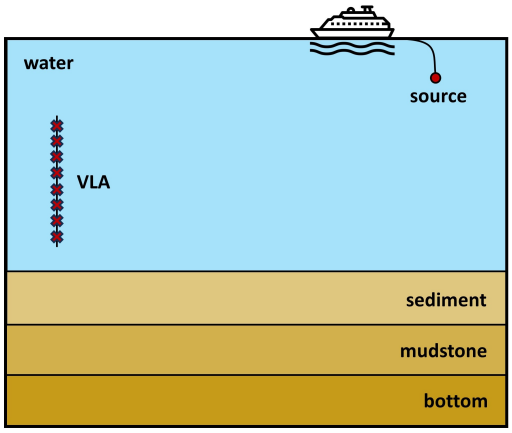


Figure 5.6: The SWellEx-96 experiment environment. The acoustic source is towed by a research vessel and transmits signals at various frequencies. The acoustic sensor consists of a vertical line array (VLA). Based on the measurements collected at the VLA, the objective is to estimate posterior distributions over parameters of interest (e.g. water depth, sound speed at the water-sediment interface, source range and depth, etc.).

of a comb signal comprising frequencies of $49, 79, 112, 148, 201, 283,$ and 388 Hz. While in the SWellEx-96 experiment the position of the source changes with time, for this task we consider the instant when the source depth is 60 m and the distance (or range) between the source and the VLA is 3 km.

The sediment layer is modeled with the following properties. The seabed consists initially of a sediment layer that is 23.5 meters thick, with a density of 1.76 g/cm^3, and an attenuation of 0.2 dB/kmHz. The sound speed at the bottom of this layer is assumed to be 1593 m/s. The second layer is mudstone that is 800 meters thick, possessing a density of 2.06 g/cm^3, and an attenuation of 0.06 dB/kmHz. The top and bottom sound speeds of this layer are 1881 m/s and 3245 m/s respectively. The description of the geoacoustic model of the SWellEx-96 experiment is complemented by a half-space featuring a density of 2.66 g/cm^3, an attenuation of 0.020 dB/kmHz, and a sound speed of 5200 m/s.

Here, we consider two geoacoustic inversion tasks:

Task 1. Based on the measurements at the VLA, the objective of this task is to infer the posterior distribution over the water depth as well as the sound speed at the water-sediment interface. For this task, we assume all the quantities above to be known. The unknown parameters m_1 (the water depth) and m_2 (the sound speed at the water-sediment interface) follow a uniform prior in $[200.5, 236.5]$ m and $[1532, 1592]$ m/s, i.e. $m_1 \sim \mathcal{U}([200.5, 236.5])$ and $m_2 \sim \mathcal{U}([1532, 1592])$, where $\mathcal{U}(\Omega)$ denotes a uniform distribution in the domain Ω.

Task 2. In addition to the two parameters considered in *Task 1* (i.e. the water depth m_1 and the sound speed at the water-sediment interface m_2), we also estimate the posterior distribution over the VLA tilt m_3, as well as the thickness of the first (sediment) layer m_4. All other quantities provided above are assumed to be known. As in *Task 1*, the unknown parameters follow a uniform prior, i.e. $m_1 \sim \mathcal{U}([200.5, 236.5])$, $m_2 \sim \mathcal{U}([1532, 1592])$, $m_3 \sim \mathcal{U}([-2, 2])$, and $m_4 \sim \mathcal{U}([18.5, 28.5])$.

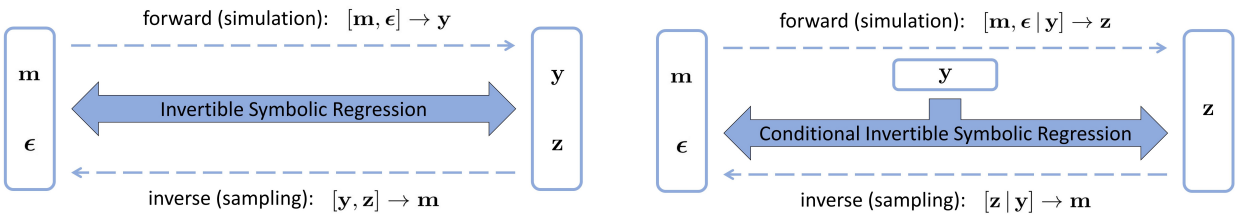


Figure 5.7: A conceptual figure of ISR (left) and cISR (right) for the geoacoustic inversion task. The posterior distribution of the parameters of interest $\mathbf{m}$ can be obtained by sampling $\mathbf{z}$ (e.g. from a standard Gaussian distribution) for a fixed observation $\mathbf{y}^*$ and running the trained bijective model backwards. To appropriately account for noise in the data, we include random data noise $\boldsymbol{\epsilon}$ as additional model parameters.

The received pressure $\mathbf{y}$ on each hydrophone and for each frequency is a function of unknown parameters $\mathbf{m}$ (e.g. water depth, sound speed at the water-sediment interface, etc.) and additive noise $\boldsymbol{\epsilon}$ as follows

$$\mathbf{y} = s(\mathbf{m}, \boldsymbol{\epsilon}) = F(\mathbf{m}) + \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}) \tag{5.21}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of data noise. Here, $s(\mathbf{m}, \boldsymbol{\epsilon})$ is a known forward model that, assuming an additive noise model, can be rewritten as $F(\mathbf{m}) + \boldsymbol{\epsilon}$, where $F(\mathbf{m})$ represents the undersea acoustic model [202]. The SWellEx-96 experiment setup involves a complicated environment and no closed from analytical solution is available for $F(\mathbf{m})$. In this case, $F(\mathbf{m})$ can only be evaluated numerically, and we use the normal-modes program KRAKEN [212] for this purpose.

Recently, machine learning algorithms have gained attention in the ocean acoustics community [205] for their notable performance and efficiency, especially when compared to traditional methods such as MCMC. In this work, for the first time, we use the concept of invertible networks to estimate posterior distributions in GI. Particularly appealing is that invertible architectures can replace both the forward propagation model as well as the inversion method.

We now discuss the training of the invertible architectures. We use the uniform prior on parameters $\mathbf{m}$ (described in *Task 1* and *Task 2* above) and the forward model in Eq. (5.21) to construct a synthetic data set for the SWellEx-96 experimental setup using the normal-modes program KRAKEN. For each parameter $\mathbf{m}$, we have $7 \times 21 = 147$ values for the pressure $\mathbf{y}$ received at hydrophones corresponding to the source's 7 different frequencies and the 21 active hydrophones. For inference, we use a test acoustic signal $\mathbf{y}^*$ that corresponds to the actual parameters values from the SWellEx-96 experiment, where the source is 60 m deep and its distance from the VLA is 3 km (i.e. $m_1^* = 216.5$, $m_2^* = 1572.368$, $m_3^* = 0$, $m_4^* = 23.5$). Also, the signal-to-noise ratio (SNR) is 15 dB.

Inspired by [187], for the invertible architectures, we include data noise $\boldsymbol{\epsilon}$ as additional model parameters to be learned. In this context, as depicted in the Figure 5.7, the input of the network is obtained by augmenting the unknown parameters $\mathbf{m}$ with additive noise $\boldsymbol{\epsilon}$. There are several ways to use the measurements collected across the 21 hydrophones for training. For instance, one can stack all hydrophones' data and treat them as a single quantity at the network's output. Alternatively, one can treat each hydrophone measurement

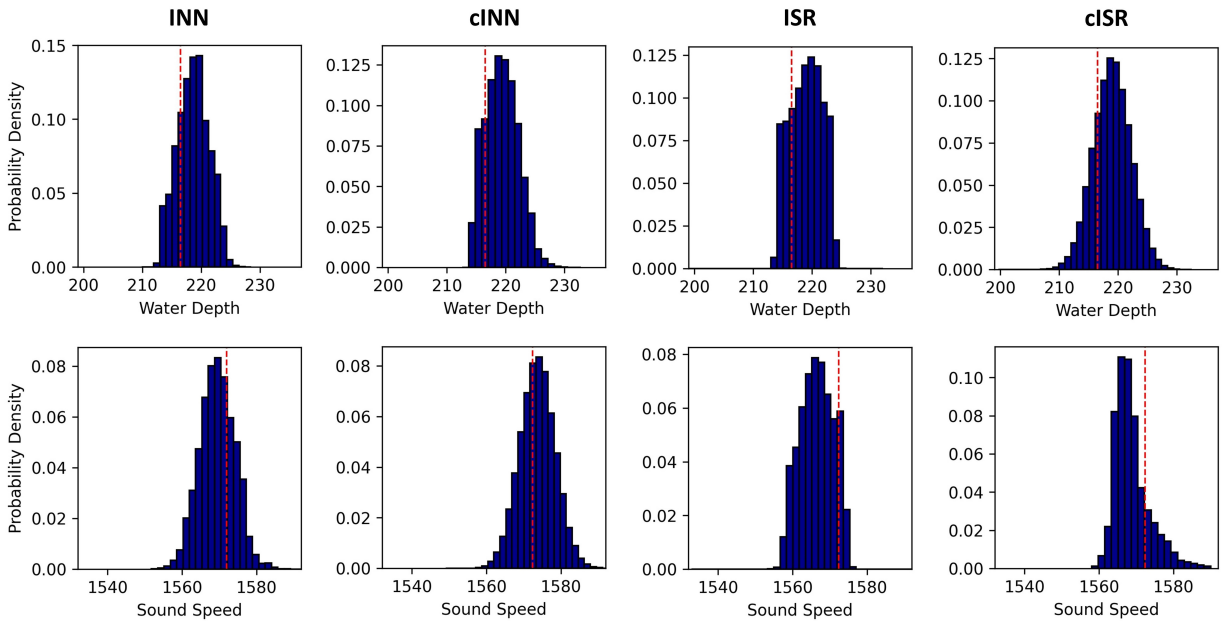


Figure 5.8: *Task 1.* For a fixed observation $\mathbf{y}^*$, we compare the estimated posteriors $\hat{p}(\mathbf{x} \,|\, \mathbf{y}^*)$ of INN, cINN, and the proposed ISR and cISR methods. Vertical dashed red lines show the ground truth values $\mathbf{x}^*$.

independently as an individual training example. For the former, the additive noise will be learned separately for each hydrophone pressure $\mathbf{y}$, while for the latter, we essentially learn the effective additive noise over all hydrophones simultaneously. In this experiment, we adopt the latter training approach, which disregards the inter-hydrophone variations, thereby reducing computational overhead.

The pressures received on the hydrophones are considered in the frequency domain, and hence they can be complex numbers. While the invertible architectures can be constructed to address complex numbers, in this case study, we stack the real and imaginary parts of the pressure field at the network's output. That is, the pressure $y = \mathrm{Re}\{y\} + i\,\mathrm{Im}\{y\}$ will be represented as $\big[\mathrm{Re}\{y\}, \mathrm{Im}\{y\}\big]$ at the network's output. In short, the 7 pressures (corresponding to the 7 source's frequencies) received at each hydrophone are replaced by 14 real numbers at the output of the network. Also, the 14 corresponding additive noises are concatenated with the parameters $\mathbf{m}$ at the network's input. Since the dimension of the network's input is $14{+}d_{\mathbf{m}}$, the latent variables $\mathbf{z}$ at the network output will be $d_{\mathbf{m}}$-dimensional for ISR and INN, and $(14{+}d_{\mathbf{m}})$-dimensional for cINN and cISR. We compare the performance of the proposed ISR and cISR algorithms against INN and cINN in solving GI.

The inferred posterior distributions via INN, cINN, ISR, and cISR, for the GI *Task 1* and *Task 2* are depicted in Fig. 5.8 and Fig. 5.9, respectively. The performance of the ISR and cISR architectures is similar to that of the INN and cINN architectures. All methods produce point estimates – Maximum a Posteriori (MAP) estimates – close to the ground truth values, showcasing the efficacy of invertible architectures in addressing cumbersome inversion tasks.

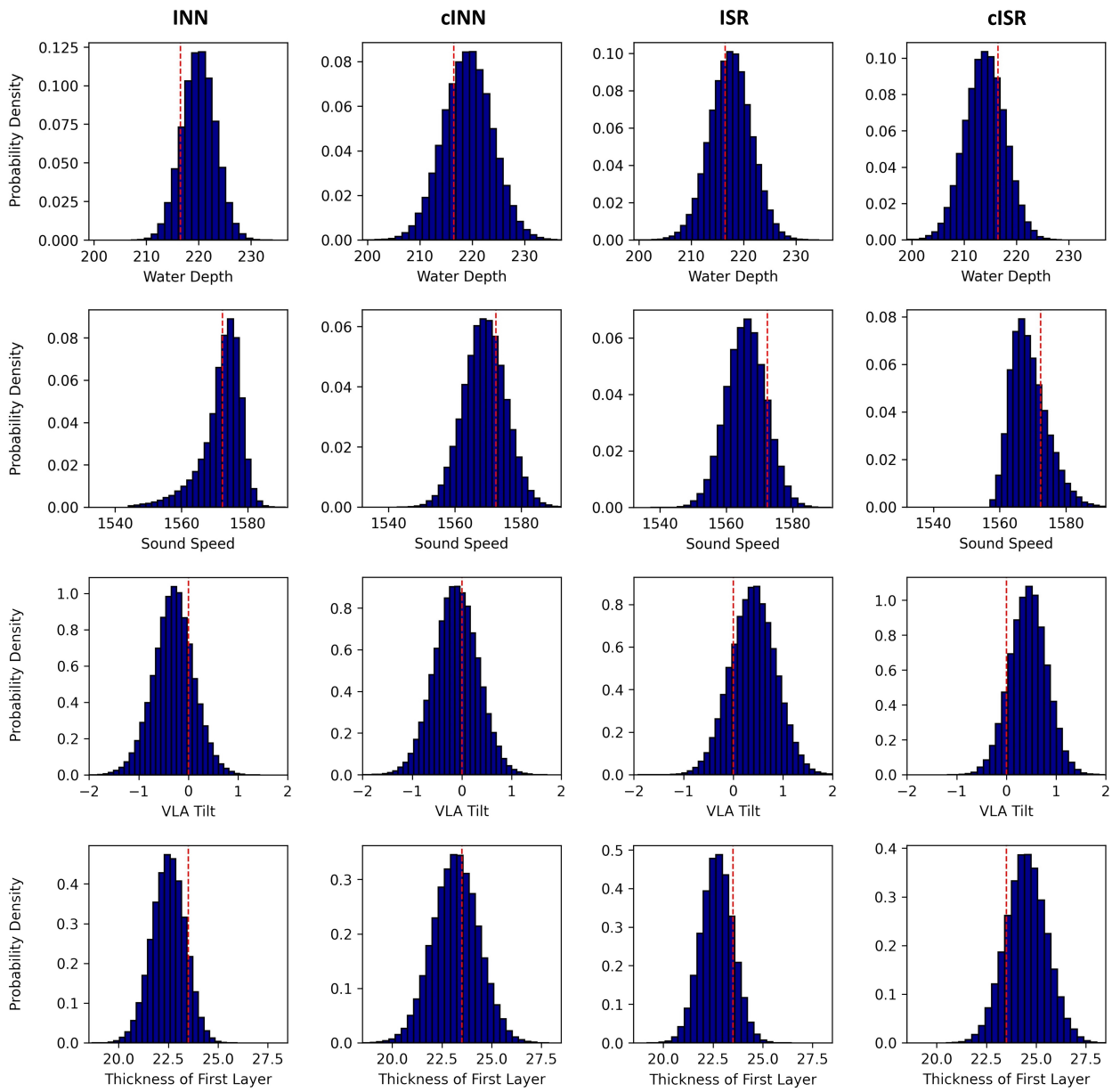


Figure 5.9: *Task 2.* For a fixed observation $\mathbf{y}^*$, we compare the estimated posteriors $\hat{p}(\mathbf{x} \mid \mathbf{y}^*)$ of INN, cINN, and the proposed ISR and cISR methods. Vertical dashed red lines show the ground truth values $\mathbf{x}^*$.

5.5 Conclusion

In this work, we introduce Invertible Symbolic Regression (ISR), a novel technique that identifies the relationships between the inputs and outputs of a given dataset using invertible architectures. This is achieved by bridging and integrating concepts of Invertible Neural Networks (INNs) and Equation Learner (EQL). This integration transforms the affine coupling blocks of INNs into a symbolic framework, resulting in an end-to-end differentiable symbolic inverse architecture that allows for efficient gradient-based learning. The proposed ISR method, equipped with sparsity promoting regularization, has the ability to not only capture complex functional relationships but also yield concise and interpretable invertible expressions. We demonstrate the versatility of ISR as a normalizing flow for density estimation and its applicability in solving inverse problems, particularly in the context of ocean acoustics, where it shows promising results in inferring posterior distributions of underlying parameters. This work is a first attempt toward creating interpretable symbolic invertible maps. While we mainly focused on introducing the ISR architecture and showing its applicability in density estimation tasks and inverse problems, an interesting research direction would be to explore the practicality of ISR in challenging generative modeling tasks (e.g. image or text generation, etc.).

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

This thesis presents a series of advancements in the field of Symbolic Regression (SR), extending its applicability beyond traditional regression tasks and demonstrating its potential across diverse domains and problem settings.

We first propose a Generalized Symbolic Regression (GSR) method, which reformulates the traditional SR optimization problem to discover analytical mappings from the input space to a transformed output space. The proposed GSR approach achieves promising performance compared to existing SR methods across well-known benchmark datasets, as well as the more challenging SymSet dataset that we introduce in this study.

We then present an adjoint-based method for discovering the underlying partial differential equations (PDEs) given data. We consider a family of parameterized PDEs in general form, and formulate a PDE-constrained optimization problem that minimizes the error between the PDE solution and the data. We then show how the corresponding adjoint equations can be elegantly derived. We show the effectiveness of the proposed approach in accurately discovering the governing PDEs from data. We also compare its performance with a commonly used method for PDE discovery, on both smooth and noisy data.

Furthermore, we introduce MESSY Estimation, a Maximum-Entropy based Stochastic and Symbolic densitY estimation method. In this work, we explore the task of inferring probability density functions symbolically from samples by leveraging the Maximum Entropy Distribution (MED) principle. We uncover three main contributions: (i) the Lagrange multipliers, inherent in the MED ansatz, can be efficiently computed by simply solving a linear system of equations, (ii) the density recovery task improves by matching more unconventional low-order (symbolic) moments, rather than necessarily matching higher-order (raw) moments, and (iii) the proposed symbolic density estimation framework leads to enhanced interpretability and better conditioning.

Finally, we propose ISR, an Invertible Symbolic Regression (ISR) approach that bridges the concepts of SR and invertible architectures. In particular, ISR naturally combines the principles of Invertible Neural Networks (INNs) and Equation Learner (EQL), a neural network-based symbolic architecture for function learning. In addition, ISR serves as a symbolic normalizing flow for density estimation tasks. We show ISR's applicability in

solving inverse problems, including a benchmark inverse kinematics problem, and notably, a geoacoustic inversion problem for inferring posterior distributions of underlying seabed parameters from acoustic signals.

In conclusion, this thesis not only expands the toolkit of SR but also sets the stage for new research directions in SR and its application to diverse scientific disciplines.

6.2 Recommendations

We offer recommendations for researchers interested in building upon the contributions of this thesis to further advance this field of study. The following directions are proposed to expand the scope and enhance the efficacy of Symbolic Regression (SR) approaches.

For Generalized Symbolic Regression (GSR), we primarily explored its performance on noise-free datasets. Future work should investigate its robustness and adaptability to noisy data, which is prevalent in real-world scenarios. Additionally, GSR's utility for identifying physical and dynamical systems governed by implicit laws could be further explored. Extending the GSR concept of mapping inputs to transformed outputs to existing SR methods could potentially enhance their overall performance and applicability.

In the area of PDE discovery using adjoint methods, an exciting research direction would involve integrating the adjoint-based method with Physics-Informed Neural Networks (PINNs) as the solver, rather than relying solely on the finite difference method. This integration could improve the handling of noisy and sparse data and facilitate the use of larger time steps in the estimation processes of both forward and backward solvers, thereby enhancing computational efficiency and accuracy.

For MESSY Estimation, further research could delve into optimizing the algorithm's efficiency in high-dimensional contexts, a critical aspect considering the complexity of modern machine learning datasets. Other possible direction includes the application of MESSY Estimation to variance reduction tasks. Additionally, investigating the integration of the maximum entropy formalism instead of the maximum likelihood approach in the training of transport-based density estimation methods could open new avenues for research and application.

The Invertible Symbolic Regression (ISR) presents a promising framework for challenging generative modeling tasks, such as image or text generation. Exploring the practical application of ISR in these areas could provide significant insights in generative techniques.

Moreover, a broader research direction would involve developing SR techniques compatible with real-time (or online) learning. Bridging the concepts between parametrized symbolic methods (e.g. Equation Learners [42], [50], [54], Symbolic Metamodels [213] based on the Kolmogorov-Arnold representation theorem [214] and the Meijer G-function [215], etc.) with dynamic approaches like the nonlinear recursive least squares technique, could significantly advance real-time adaptive and interpretable modeling.

To further bridge the gap between theoretical advancements and practical applications, it is crucial to consider the deployment of SR methods in real-world domains such as robotics, control, oceanography, mechatronics, etc. Immediate applications could involve using SR to derive control laws for robotic systems or model ocean dynamics based on sensor data. Challenges in these applications include handling noisy, incomplete, and high-dimensional

data, as well as ensuring computational efficiency in real-time environments. Future work should focus on developing robust methods for learning from real-world data, including advanced data preprocessing techniques and domain-specific adaptations of SR algorithms. Identifying the types of data that best support SR, such as high-fidelity sensor measurements and extensive simulation data, will be crucial for successful implementation. Collaborations with domain experts will also be essential to tailor SR approaches to the unique requirements and constraints of these fields, ultimately enhancing the practical utility and impact of SR in solving cumbersome, real-world tasks.

These recommendations aim to extend the reach and deepen the impact of SR in scientific research and practical applications, pushing the boundaries of current methodologies and introducing novel approaches to solving complex, real-world problems.

Appendix A

Supplementary Material for Chapter 2

A.1 Implementation, Hyperparameters, and Additional Experiment Details

Implementation. Our GSR method discovers expressions of the form $g(y) = f(\mathbf{x})$ where $y \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^d$, and where the search space for $f(\cdot)$ and $g(\cdot)$ is constrained to a weighted sum of M_ϕ and M_ψ basis functions $\big($namely $\phi(\cdot)$ and $\psi(\cdot)\big)$, respectively. We use a matrix-based encoding scheme to represent $\phi(\cdot)$ and $\psi(\cdot)$ using basis matrices $\boldsymbol{B}^\phi$ and $\boldsymbol{B}^\psi$ of sizes $n_{\boldsymbol{B}^\phi} \times m_{\boldsymbol{B}^\phi}$ and $n_{\boldsymbol{B}^\psi} \times 1$, respectively (where $m_{\boldsymbol{B}^\phi} = n_v + 2$ and n_v is defined in Section 2.3.3). Hence, in addition to the number of basis functions M_ϕ and M_ψ, the parameters $n_{\boldsymbol{B}^\phi}$, n_v, and $m_{\boldsymbol{B}^\phi}$ affect the complexity of the evolved expressions, and hence can be controlled to confine the search space (although d also affects the complexity of the expression, it is given by the problem and cannot be controlled). Although more than one basis matrix can lead to the same basis function (see *Remark* 2.3.3), the search space of basis functions $\big($mainly $\phi(\cdot)\big)$ is still huge in general, and thus, enumerating all the possible basis functions is not practical. Hence, we will rely on genetic programming (GP) for effective search process. A pseudocode of our GP-based GSR algorithm is outlined in Algorithm 7.

The main inputs to our GP-based algorithm are N_p, n_p, M_ϕ, M_ψ, $\mathcal{L}_\mathbf{x}$, and $\mathcal{L}_y$. Recall that N_p is the population size and n_p is the number of surviving individuals per generation. $\mathcal{L}_\mathbf{x}$ and $\mathcal{L}_y$ are the libraries of allowable transformations that can be used with $\mathbf{x}$ and y, respectively. These libraries form the first two rows of mapping tables, e.g. Table A.5, and are defined by the benchmark problem. Note that the division operator is not part of our GSR architecture. That is, the main arithmetic operations used by GSR are $\{+, -, \times\}$. For example, for the Nguyen benchmark dataset, the library of allowable operations is $\mathcal{L}_0 = \{+, -, \times, \div, \cos, \sin, \exp, \ln\}$ as shown in Table A.11. In this case, we define $\mathcal{L}_\mathbf{x} = \{1, \bullet^1, \cos, \sin, \exp, \ln\}$ (resulting in the mapping Table 2.2) and $\mathcal{L}_y = \{1, \bullet^1, \exp, \ln\}$. Regarding the stopping criterion, common terminating conditions for GP include: i) a solution reaches minimum criterion (e.g. error threshold), ii) the algorithm reaches a fixed number of generations (or iterations), iii) the algorithm generates a fixed number of individuals (or candidates expressions), iv) the algorithm reaches a plateau such that new generations no longer improve results, v) combinations of the above conditions. In our case,

the algorithm terminates when the solution hits a minimum root-mean-square error (RMSE) threshold. To accelerate termination, we slowly relax the error threshold by gradually increasing it. To avoid reaching a plateau and since we are dealing with a small population size as shown in Table A.1, we enhance diversity (in the basis functions) by producing completely new individuals with probability 1/2 per generation (while performing crossover and mutation with probability 1/4 each per generation). To speed up the search process, we employ sublibraries $\mathcal{L}_{\mathbf{x}}^s \subseteq \mathcal{L}_{\mathbf{x}}$ and $\mathcal{L}_y^s \subseteq \mathcal{L}_y$ of allowable transformations, used when generating completely new individuals (or completely new basis functions in the case of mutation). For $\mathbf{x}$, we mainly rely on three sublibraries which are the most common: a polynomial sublibrary $\mathcal{L}_{\text{poly}}$, a trigonometric sublibrary $\mathcal{L}_{\text{trig}}$, and the original library $\mathcal{L}_{\mathbf{x}}$ itself. For the Nguyen benchmark example above, $\mathcal{L}_{\text{poly}} = \{1, \bullet^1\}$ and $\mathcal{L}_{\text{trig}} = \{1, \bullet^1, \cos, \sin\}$. Note that power operators such as $\bullet^2$, $\bullet^3$ would be included in these sublibraries if they were part of the original library defined by the benchmark problem. The function ChooseSublibrary( ) works according to some cycle. For example, assuming k is the generation (or iteration) counter, if $k \leq 1,500$, each cycle consists of 70 iterations broken into three stages, the first stage consists of 15 iterations and assigns $\mathcal{L}_{\mathbf{x}}^s \leftarrow \mathcal{L}_{\mathbf{x}}$, the second stage consists of 25 iterations and assigns $\mathcal{L}_{\mathbf{x}}^s \leftarrow \mathcal{L}_{\text{poly}}$, and the third and final stage consists of the remaining 35 iterations and assigns $\mathcal{L}_{\mathbf{x}}^s \leftarrow \mathcal{L}_{\text{trig}}$. This cycle repeats until $k = 1,500$, after which the cycle's size becomes $1,500$ iterations broken into three equal stages (i.e. 500 iterations per sublibrary). For y, the cycle consists of 20 iterations, in which we equally alternate between the polynomial sublibrary $\mathcal{L}_{\text{poly}}$ and the original library $\mathcal{L}_y$ itself (i.e. 10 iterations for each sublibrary). Indeed, the use of sublibraries is only possible when the corresponding operations are included in the original library defined by the benchmark problem (e.g. Neat-6 and Neat-8 cannot use trigonometric sublibraries since $\{\cos, \sin\}$ are not included in their corresponding original libraries, as shown in Table A.11). In addition, it is up to the user to specify the cycle's size and how to alternate between sublibraries, or even decide whether to use sublibraries in the first place.

Hyperparameters. Throughout our experiments, we adopt the following hyperparameter values. For GP, we use a population size $N_p = 30$, and we allow for $n_p = 10$ surviving individuals per generation. We perform crossover with probability $P_c = \frac{1}{4}$ and allow for only 2 parents to be involved in the process (i.e. new individuals are formed by combing basis functions from two randomly chosen parent individuals). We apply mutation with probability $P_m = \frac{1}{4}$ and allow for 3 basis functions (randomly selected from an individual) to be mutated (i.e. to be discarded and replaced by completely new basis functions). We generate a (completely new) random individual with probability $P_r = \frac{1}{2}$. For ADMM, we use a regularizer $\lambda = 0.4$, a penalty $\rho = 0.1$. The algorithm terminates when the ℓ^2-norm of the difference between the weight vectors from two consecutive iterations falls below a threshold of $\delta = 10^{-5}$. Regarding initial conditions, we use $\boldsymbol{w}_0 = \widehat{\mathbf{1}} = \frac{[\frac{1}{2}\cdots\frac{1}{2}]^T}{\sqrt{\frac{1}{4}+\cdots+\frac{1}{4}}}$ (where "$\widehat{\ }$" denotes a normalized vector), $\boldsymbol{z}_0 = \mathbf{1} = [1\cdots1]^T$, $\boldsymbol{u}_0 = \mathbf{0} = [0\cdots0]^T$. For GSR, we allow for a maximum of $M_\phi = 15$ basis functions $\phi(\cdot)$ for each expression of $f(\cdot)$ (this is the maximum number since some of the M_ϕ basis functions will be multiplied by 0, i.e. at most we get M_ϕ nonzero coefficients multiplying the basis fcuntions). To avoid overfitting and overly complex expressions, we allow for a maximum of $M_\psi = 1$ basis function $\psi(\cdot)$ for each expression of $g(\cdot)$ (in this case the maximum and minimum are both 1 and $g(\cdot)$

will consist of a single basis function). It is worth noting that we use $M_\psi = 2$ for SymSet-11. Each basis $\psi(\cdot)$ will consist of a single transformation $n_{\boldsymbol{B}^\psi} = 1$. Each basis $\phi(\cdot)$ will be a product of N_t transformations, where N_t is a random integer between 1 and 3, i.e. $n_{\boldsymbol{B}^\phi} \in \{1, 2, 3\}$. For each of these N_t transformations, the maximum total multiplicity of all the independent variables (or features) in an argument is a random integer between 2 and 5, i.e. $n_v \in \{2, 3, 4, 5\}$. GSR terminates when a candidate expression achieves a RMSE lower than a threshold with a starting value of $\epsilon = 10^{-6}$ (recall that this threshold is slowly relaxed during the process, e.g. by progressively multiplying it by a factor of $\sqrt{10}$ for every $1,500$ iterations). All hyperparameter values are summarized in Table A.1.

Computational complexity. Although genetic algorithms are inherently heuristic, understanding how our GSR algorithm operates and scales could still be valuable. Following Algorithm 7 above, we can approximate the time complexity of GSR as:

$$
\begin{aligned}
O\Big(& N_p \cdot \Big(M_\phi \cdot n_{\boldsymbol{B}^\phi} \cdot m_{\boldsymbol{B}^\phi} + M_\psi \cdot n_{\boldsymbol{B}^\psi} \cdot 1 + N_\delta + O\left(\text{fitness}\right) \Big) + O\left(N_p \log N_p\right) \\
& + N_\epsilon \cdot \Big((N_p - n_p) \cdot \big(P_c \cdot O\left(\text{crossover}\right) + P_m \cdot O\left(\text{mutation}\right) \\
& + P_r \cdot \left(M_\phi \cdot n_{\boldsymbol{B}^\phi} \cdot m_{\boldsymbol{B}^\phi} + M_\psi \cdot n_{\boldsymbol{B}^\psi} \cdot 1\right) + N_\delta + O\left(\text{fitness}\right) \big) + O\left(N_p \log N_p\right) \Big) \Big)
\end{aligned}
\tag{A.1}
$$

where N_ϵ is the number of generations until the GP algorithm hits the tolerance ϵ, and N_δ is the number of iterations until the ADMM algorithm hits the tolerance δ. Note that performing crossover or mutation operations takes $O(1)$ time (i.e. a constant amount of time), and computing the fitness (which calculates RMSE on N paired training examples) takes $O(N)$ time. Also note that P_c, P_m, and P_r are probabilities which can be treated as constants. Hence, GSR's time complexity reduces to:

$$
O\Big(N_\epsilon \cdot N_p \cdot \big(M_\phi \cdot n_{\boldsymbol{B}^\phi} \cdot m_{\boldsymbol{B}^\phi} + M_\psi \cdot n_{\boldsymbol{B}^\psi} + N_\delta + N + \log N_p \big) \Big) \tag{A.2}
$$

Compared to GSR, the special version s-GSR adopts a vanilla SR (where $g(y)$ is simply y) with the same GP algorithm and coefficient optimization process (through ADMM) as GSR. Thus, s-GSR's time complexity can be approximated as:

$$
O\Big(N_\epsilon \cdot N_p \cdot \big(M_\phi \cdot n_{\boldsymbol{B}^\phi} \cdot m_{\boldsymbol{B}^\phi} + N_\delta + N + \log N_p \big) \Big) \tag{A.3}
$$

Although GSR's computational complexity contains an extra term of $O\left(N_\epsilon \cdot N_p \cdot M_\psi \cdot n_{\boldsymbol{B}^\psi}\right)$, the number of GP generations N_ϵ produced by GSR is often much less than that of s-GSR, which explains the runtime advantage of GSR over s-GSR shown in Table A.4.

Algorithm 7: GP Procedure for GSR

Input: N_p, n_p, M_ϕ, M_ψ, $\mathcal{L}_\mathbf{x}$, $\mathcal{L}_y$
Output: $\mathcal{I}^*$
function SolveGSR(N_p, n_p, M_ϕ, M_ψ, $\mathcal{L}_\mathbf{x}$, $\mathcal{L}_y$)
  Initialize population:
    // $\boldsymbol{\mathcal{I}}(k) \leftarrow \{\mathcal{I}_1(k), \mathcal{I}_2(k), \ldots, \mathcal{I}_{N_p}(k)\}$
  $k \leftarrow 0$; // Initialize the generation (or iteration) counter
  for $i = 1$ **to** N_p **do**
    $\mathcal{I}_i(k) \leftarrow$ GenerateRandomIndividual($M_\phi, M_\psi, \mathcal{L}_\mathbf{x}, \mathcal{L}_y$);
    /* Each individual $\mathcal{I}_i(k)$ contains two randomly generated sets of M_ϕ and M_ψ basis matrices respectively */
  end
  Evaluate each individual $\mathcal{I}_i(k)$ with respect to the fitness function;
  /* For each individual $\mathcal{I}_i(k)$, form the matrix $\boldsymbol{A}_i(k)$, solve for the optimal coefficients vector $\boldsymbol{w}_i(k) \leftarrow$ SolveADMM($\boldsymbol{A}_i(k), \cdots$), then compute its fitness */
  $\boldsymbol{\mathcal{I}}(k) \leftarrow$ sorted($\boldsymbol{\mathcal{I}}(k)$); // in ascending order of fitness
  while *Stopping Criterion not Satisfied* **do**
    $k \leftarrow k + 1$; // Increment the generation (or iteration) counter
    UpdateCriterion( );
    /* Start with a strict stopping criterion (e.g. very low error threshold) and slowly relax it (e.g. gradually increase the error threshold) */
    $\mathcal{L}^s_\mathbf{x}, \mathcal{L}^s_y \leftarrow$ ChooseSublibrary($\mathcal{L}_\mathbf{x}, \mathcal{L}_y$);
    /* Choose sublibraries $\mathcal{L}^s_\mathbf{x} \subseteq \mathcal{L}_\mathbf{x}$ and $\mathcal{L}^s_y \subseteq \mathcal{L}_y$ of allowable operations to be used with $\mathbf{x}$ and y respectively */
    $\boldsymbol{\mathcal{I}}_{[1:n_p]}(k) \leftarrow \boldsymbol{\mathcal{I}}_{[1:n_p]}(k-1)$;
    /* The n_p fittest individuals of the previous generation are copied to the current new one */
    for $i = n_p + 1$ **to** N_p **do**
      $u \leftarrow$ GenerateRandomInteger(1, 4);
      if $u = 1$ **then**
        $\mathcal{I}_i(k) \leftarrow$ Reproduce($\boldsymbol{\mathcal{I}}_{[1:n_p]}(k), \mathcal{L}^s_\mathbf{x}, \mathcal{L}^s_y$);
        /* Crossover based on the surviving individuals */
      else if $u = 2$ **then**
        $\mathcal{I}_i(k) \leftarrow$ Mutate($\boldsymbol{\mathcal{I}}_{[1:n_p]}(k), \mathcal{L}^s_\mathbf{x}, \mathcal{L}^s_y$);
        /* Mutation based on the surviving individuals */
      else
        $\mathcal{I}_i(k) \leftarrow$ GenerateRandomIndividual($M_\phi, M_\psi, \mathcal{L}^s_\mathbf{x}, \mathcal{L}^s_y$);
        /* Randomly generate a completely new individual */
      end
    end
    Evaluate each individual $\mathcal{I}_i(k)$ with respect to the fitness function;
    $\boldsymbol{\mathcal{I}}(k) \leftarrow$ sorted($\boldsymbol{\mathcal{I}}(k)$); // in ascending order of fitness
  end
  $\mathcal{I}^* \leftarrow \mathcal{I}_1(k)$; // return the fittest individual
end function

Table A.1: Hyperparameter values for GSR for all experiments, unless otherwise specified.

Hyperparameter	**Symbol**	**Value**
GP Parameters		
Population size	N_p	30
Number of survivors per generation	n_p	10
Crossover probability	P_c	1/4
Number of parents involved in crossover	—	2
Mutation probability	P_m	1/4
Number of bases to mutate	—	3
Randomly generated individual probability	P_r	1/2
ADMM Parameters		
Regularization parameter	λ	0.4
Penalty parameter	ρ	0.1
Tolerance on the solution error	δ	10^{-5}
Initial guesses	$\boldsymbol{w}_0$, $\boldsymbol{z}_0$, $\boldsymbol{u}_0$	$\widehat{\frac{\mathbf{1}}{\mathbf{2}}}, \mathbf{1}, \mathbf{0}$
GSR Parameters		
Maximum number of basis functions $\phi(\cdot)$ for each expression of $f(\cdot)$	M_ϕ	15
Maximum number of basis functions $\psi(\cdot)$ for each expression of $g(\cdot)$	M_ψ	1
Maximum total multiplicity of all features in an argument	n_v	$\{2, 3, 4, 5\}$
Number of tranformations multiplied together per basis $\phi(\cdot)$	$n_{\boldsymbol{B}^\phi}$	$\{1, 2, 3\}$
Number of tranformations multiplied together per basis $\psi(\cdot)$	$n_{\boldsymbol{B}^\psi}$	1
Tolerance on the solution error (RMSE)	ϵ	10^{-6}

Additional experiment details. For all benchmark problems, we run GSR for multiple independent trials using different random seeds (following the experimental procedure in [31], [43]). Table A.2 shows the recovery rates of GSR against literature-reported values from several algorithms on the Nguyen and Livermore individual benchmark problems. We first note that, due to the wide domain of sampled input points imposed by Livermore-1 (i.e. $[-10, 10]$), we observed some instabilities in the solution due to the presence of the exponential function, which we decided to exclude from the library of allowable operations during the search process for this benchmark. In what follows, we provide explanations for the results shown in Table A.2. Note that Livermore-5 is difficult to recover by NGGPPS+SLP/HER and the remaining methods as it contains subtractions. Subtraction is more difficult than addition since it is not cumulative. This is not an issue for GSR since both additions and subtractions are equally recovered through the sign of the optimal coefficients multiplying the basis functions. Livermore-10 and Livermore-17 are more challenging than Nguyen-10 since they require adding the same basis function many more times (which is apparent through the poor recovery rates of the different methods). Fortunately, this is also not a problem for GSR since it can be easily solved by finding the right coefficient multiplying the basis function. Livermore-18 is more challenging than Livermore-2 and Nguyen-5 since it requires recovering the constant 5 without a constant optimizer $\left(\text{which can be recovered as } \frac{x+x+x+x+x}{x}\right)$. For GSR, this can be recovered by naturally solving for the real-valued coefficient. The problem of the different methods on Livermore-22 lies in the constant 0.5, which requires finding $\frac{x}{x+x}$ compared to GSR which simply solves for the optimal parameter multiplying x^2. We observe that GSR performs poorly on Livermore-9 and Livermore-21 compared to NGGPPS+SLP/HER. This can be due to the choice of hyperparameters (e.g. M_ϕ, $n_{\boldsymbol{B}^\phi}$, and n_v) as well as the GP-based search process. These two benchmarks require finding the first 9 and 8 powers of x simultaneously, respectively, which can be difficult to achieve by GSR, especially that we only consider $M_\phi = 15$ basis functions $\phi(\cdot)$ per expression of $f(\cdot)$, as mentioned earlier. Note that polynomials were not an issue for GSR up to the 6^{th} order (i.e. Nguyen-4). We also tried experimenting with a 7^{th} order polynomial (i.e. $y = x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$) and GSR achieved 100% recovery rate. We started observing a decline in the recovery rate when we added the 8^{th} power of x. In other words, Livermore-21 (the 8^{th} order polynomial) seems to be the limit that GSR can reach with polynomials while Livermore-9 (the 9^{th} order polynomial) becomes very difficult to recover. It is worth noting that if the libraries for the Livermore-9 and Livermore-21 problems contained the square and cube operators $\{\bullet^2, \bullet^3\}$ (as is the case for the Jin benchmarks described in Table A.11), then GSR would easily recover these two problems. Finally, GSR is not able to recover Livermore-7 $\left(y = \sinh(x)\right)$ and Livermore-8 $\left(y = \cosh(x)\right)$ since both benchmarks require finding the basis function $e^{-\bullet}$,[1] which cannot be expressed using our current encoding scheme unless it is available as a transformation by itself. That is, the exponential operator $e^\bullet$ is not enough to recover $\frac{1}{e^\bullet} = e^{-\bullet}$ using our current encoding scheme. Had the negative exponential operator $e^{-\bullet}$ been part of the library of allowable operations defined by Livermore-7 and Livermore-8, GSR would easily recover these two benchmarks. As we can see for SymSet-1 $\left(y = x\sinh(x) - \frac{4}{5}\right)$, we added the operator $e^{-\bullet}$ to the library of allowable operations (see Table A.12), which made GSR's mission much simpler and it

[1] Livermore-7 and Livermore-8 can be expressed as $\sinh(x) = \frac{e^x - e^{-x}}{2}$ and $\cosh = \frac{e^x + e^{-x}}{2}$ respectively.

was able to recover the corresponding ground truth expression as shown in Table A.18. It is worth mentioning that, although ground truth expressions are not expressible, GSR was naturally able to recover the best approximations possible for Livermore-7 and Livermore-8, which turned out to be their Taylor expansions around 0. GSR's typical output expressions were as follows:

Livermore-7:

$$\begin{aligned}
0.51655\,y &= +0.51655x + 0.48165x(x \times x) + 0.48165x(x \times x) \\
&\quad - 0.048738(x + x + x)(x + x + x)(x + x) + 0.0022335(x + x)(x)(x \times x \times x) \\
\iff \quad y &\approx x + 0.166x^3 + 0.00865x^5 \\
&\approx x + \frac{x^3}{3!} + \frac{x^5}{5!} \qquad \big(\text{the first three terms of the Taylor series of } \sinh(x) \text{ around } 0\big) \\
&\approx \sinh(x)
\end{aligned}$$

Livermore-8:

$$\begin{aligned}
-0.8616\,y &= -0.2154 - 0.042956(x + x)x - 0.035877(x \times x)(x \times x) - 0.2154 - 0.2154 \\
&\quad - 0.0012368(x \times x)(x \times x \times x \times x) - 0.042956x(x + x) - 0.25898x(x) - 0.2154 \\
\iff \quad y &\approx 1 + 0.5x^2 + 0.0416x^4 + 0.00144x^6 \\
&\approx 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} \qquad \big(\text{the first four terms of the Taylor series of } \cosh(x) \text{ around } 0\big) \\
&\approx \cosh(x)
\end{aligned}$$

We next perform a runtime comparison between GSR and NGGPPS on the Nguyen benchmark problem set. We run each benchmark problem and report the runtimes in Table A.3. We find that GSR exhibits faster runtime than NGGPPS, averaging 2.5 minutes per run on the Nguyen benchmarks compared to 3.2 minutes for NGGPPS. It is worth noting that although GSR is, on average, faster than NGGPPS, it still exhibits slower runtime on some problems (e.g. Nguyen-6 and Nguyen-9 in Table A.3). This is due to the randomness of the search process as well as the use of sublibraries as mentioned earlier in the Appendix. Indeed, the runtime depends on the stopping criterion or condition. For example, one can shorten the runtime further if the interest is just in an approximation rather than an exact recovery. Our GSR method recovers exact expressions in the order of few minutes.

We further highlight the strengths of GSR on the new SymSet benchmark problem set, and show the benefits of searching for expressions of the form $g(y) = f(\mathbf{x})$ instead of $y = f(\mathbf{x})$. Typical expressions, with exact symbolic equivalence, recovered by GSR are shown in Table A.18. The key feature of GSR lies in its ability to recover expressions of the form $g(y) = f(\mathbf{x})$. To better highlight the benefits offered by this feature, we disable it by constraining the search space in GSR to expressions of the form $y = f(\mathbf{x})$ (which is the most critical ablation). We refer to this special version of GSR as s-GSR. Note that most of the SymSet expressions cannot be exactly recovered by s-GSR (i.e. they can only be approximated). We compare the performance of GSR against s-GSR on the SymSet benchmarks in terms of accuracy and runtime (see Table A.4). The results clearly show that

GSR is faster than s-GSR, averaging around 2 minutes per run on the SymSet benchmarks compared to 2.27 minutes for s-GSR (i.e. $\sim 11\%$ runtime improvement). In addition, GSR is more accurate than s-GSR by two orders of magnitude. This is due to the fact that GSR exactly recovers the SymSet expressions across most of the runs, while s-GSR only recovers approximations for most of these expressions. It is worth mentioning that on SymSet-1, SymSet-4, SymSet-5, SymSet-10, and SymSet-12, we observe mean RMSE values of the same order of magnitude between GSR and s-GSR, since these expressions can be exactly recovered by simply learning expressions of the form $y = f(\mathbf{x})$. As GSR has to perform a search to discover that $g(y)$ is simply y for these expressions, it exhibits slower runtime than s-GSR in recovering these expressions (see Table A.4).

In addition, we compare GSR against several strong SR methods with similar (or better) expression ability. In particular, we experiment on SymSet with NGGPPS, PSTree, PySR, and gplearn (see Table A.4). GSR is more accurate than all these methods by three orders of magnitude, which further demonstrates the advantage of our proposed approach. As for the runtime, PSTree is the fastest method, averaging around 16 seconds per run on the SymSet expressions, while maintaining solid accuracies. This comes as no surprise given its state-of-the-art performance on SRBench datasets [28]. It is worth mentioning that on SymSet-16, all the methods (i.e. NGGPPS, PSTree, PySR, and gplearn) exhibited some instabilities in the solution over all independent runs. Hence, we excluded SymSet-16 for these methods in Table A.4.

Table A.2: Recovery rate comparison of GSR against literature-reported values from several algorithms on the Nguyen and Livermore benchmark problem sets over 25 independent runs. The ground truth expressions for these benchmarks are shown in Tables A.11 and A.12.

	Recovery Rate (%)				
Benchmark	**GSR**	**NGGPPS + SLP/HER**	**NGGPPS**	**GEGL**	**DSR**
Nguyen-1	100	100	100	100	100
Nguyen-2	100	100	100	100	100
Nguyen-3	100	100	100	100	100
Nguyen-4	100	100	100	100	100
Nguyen-5	100	100	100	92	72
Nguyen-6	100	100	100	100	100
Nguyen-7	100	100	96	48	35
Nguyen-8	100	100	100	100	96
Nguyen-9	100	100	100	100	100
Nguyen-10	100	100	100	92	100
Nguyen-11	100	100	100	100	100
Nguyen-12*	100	4	12	0	0
Nguyen Average	**100**	92.00	92.33	86.00	83.58
Livermore-1	100	100	100	100	3
Livermore-2	100	100	100	44	87
Livermore-3	100	100	100	100	66
Livermore-4	100	100	100	100	76
Livermore-5	100	40	4	0	0
Livermore-6	100	100	88	64	97
Livermore-7	0	4	0	0	0
Livermore-8	0	0	0	0	0
Livermore-9	4	88	24	12	0
Livermore-10	100	8	24	0	0
Livermore-11	100	100	100	92	17
Livermore-12	100	100	100	100	61
Livermore-13	100	100	100	84	55
Livermore-14	100	100	100	100	0
Livermore-15	100	100	100	96	0
Livermore-16	100	100	92	12	4
Livermore-17	100	36	68	4	0
Livermore-18	100	48	56	0	0
Livermore-19	100	100	100	100	100
Livermore-20	100	100	100	100	98
Livermore-21	76	88	24	64	2
Livermore-22	100	92	84	68	3
Livermore Average	**85.45**	77.45	71.09	56.36	30.41
All Average	**90.59**	82.59	78.59	66.82	49.18

Table A.3: Runtimes of GSR vs. NGGPPS on the Nguyen benchmarks. The ground truth expressions for these benchmarks are shown in Table A.11.

	Runtime (sec)	
Benchmark	**GSR**	**NGGPPS**
Nguyen-1	18.66	27.05
Nguyen-2	25.96	59.79
Nguyen-3	38.77	151.06
Nguyen-4	63.82	268.88
Nguyen-5	447.01	501.65
Nguyen-6	465.79	43.96
Nguyen-7	33.35	752.32
Nguyen-8	93.89	123.21
Nguyen-9	391.79	31.17
Nguyen-10	68.05	103.72
Nguyen-11	38.86	66.50
Average	**153.27**	193.57

Table A.4: Average performance in mean RMSE and runtime, along with their standard errors, for GSR against s-GSR and several strong SR methods on the SymSet benchmark problem sets over 25 independent runs. The ground truth expressions for these benchmarks are shown in Table A.12.

Benchmark	Mean RMSE		Runtime (sec)	
	GSR	**s-GSR**	**GSR**	**s-GSR**
SymSet-1	$2.52 \times 10^{-5} \pm 9.29 \times 10^{-6}$	$3.69 \times 10^{-5} \pm 1.62 \times 10^{-5}$	49.94 ± 2.61	43.81 ± 1.95
SymSet-2	$4.28 \times 10^{-7} \pm 3.35 \times 10^{-8}$	$2.34 \times 10^{-3} \pm 1.21 \times 10^{-3}$	75.24 ± 2.83	91.95 ± 4.26
SymSet-3	$6.58 \times 10^{-6} \pm 1.79 \times 10^{-6}$	$1.22 \times 10^{-3} \pm 8.17 \times 10^{-4}$	66.34 ± 2.51	88.14 ± 3.49
SymSet-4	$7.94 \times 10^{-4} \pm 5.19 \times 10^{-4}$	$1.37 \times 10^{-4} \pm 1.46 \times 10^{-3}$	428.23 ± 8.46	405.54 ± 6.04
SymSet-5	$3.63 \times 10^{-5} \pm 9.92 \times 10^{-5}$	$4.98 \times 10^{-5} \pm 7.37 \times 10^{-5}$	71.86 ± 4.12	64.19 ± 3.82
SymSet-6	$4.38 \times 10^{-5} \pm 8.09 \times 10^{-6}$	$8.12 \times 10^{-2} \pm 1.93 \times 10^{-2}$	76.59 ± 3.72	94.18 ± 4.16
SymSet-7	$2.39 \times 10^{-4} \pm 1.71 \times 10^{-5}$	$6.83 \times 10^{-2} \pm 6.56 \times 10^{-3}$	93.61 ± 2.54	109.94 ± 4.98
SymSet-8	$6.83 \times 10^{-4} \pm 1.95 \times 10^{-5}$	$7.88 \times 10^{-3} \pm 3.88 \times 10^{-3}$	68.07 ± 2.47	90.61 ± 3.06
SymSet-9	$3.57 \times 10^{-6} \pm 3.41 \times 10^{-5}$	$6.32 \times 10^{-3} \pm 1.39 \times 10^{-3}$	57.36 ± 2.68	83.18 ± 2.09
SymSet-10	$2.12 \times 10^{-4} \pm 4.43 \times 10^{-4}$	$3.24 \times 10^{-4} \pm 7.58 \times 10^{-5}$	393.62 ± 5.47	379.56 ± 6.86
SymSet-11	$1.43 \times 10^{-3} \pm 9.97 \times 10^{-4}$	$9.39 \times 10^{-2} \pm 6.78 \times 10^{-3}$	124.38 ± 9.65	187.49 ± 8.35
SymSet-12	$1.22 \times 10^{-5} \pm 5.37 \times 10^{-5}$	$7.09 \times 10^{-5} \pm 2.88 \times 10^{-4}$	78.53 ± 4.02	69.72 ± 2.97
SymSet-13	$2.31 \times 10^{-5} \pm 1.83 \times 10^{-5}$	$9.13 \times 10^{-2} \pm 3.28 \times 10^{-2}$	96.14 ± 5.48	114.85 ± 4.61
SymSet-14	$2.18 \times 10^{-5} \pm 9.47 \times 10^{-6}$	$6.14 \times 10^{-2} \pm 7.78 \times 10^{-3}$	112.32 ± 4.57	137.61 ± 4.53
SymSet-15	$1.81 \times 10^{-6} \pm 4.75 \times 10^{-5}$	$6.71 \times 10^{-3} \pm 3.57 \times 10^{-4}$	46.97 ± 2.33	85.07 ± 4.14
SymSet-16	$7.24 \times 10^{-4} \pm 3.58 \times 10^{-4}$	$9.86 \times 10^{-3} \pm 2.19 \times 10^{-3}$	98.37 ± 3.71	126.16 ± 5.94
SymSet-17	$2.57 \times 10^{-4} \pm 7.03 \times 10^{-5}$	$3.41 \times 10^{-3} \pm 4.54 \times 10^{-3}$	116.78 ± 4.49	143.31 ± 6.28
Average	$\mathbf{2.66 \times 10^{-4} \pm 1.59 \times 10^{-4}}$	$2.56 \times 10^{-2} \pm 5.27 \times 10^{-3}$	120.84 ± 4.22	136.19 ± 4.56
Benchmark	**NGGPPS**	**PSTree**	**NGGPPS**	**PSTree**
SymSet-1	$3.66 \times 10^{-1} \pm 7.81 \times 10^{-3}$	$7.92 \times 10^{-3} \pm 2.23 \times 10^{-3}$	175.32 ± 1.54	42.98 ± 3.23
SymSet-2	$4.75 \times 10^{-1} \pm 5.92 \times 10^{-2}$	$1.96 \times 10^{-1} \pm 2.72 \times 10^{-2}$	171.46 ± 2.03	24.68 ± 1.05
SymSet-3	$1.34 \times 10^{-2} \pm 1.91 \times 10^{-3}$	$3.73 \times 10^{-3} \pm 4.01 \times 10^{-4}$	167.11 ± 1.69	21.68 ± 1.89
SymSet-4	$1.76 \times 10^{0} \pm 9.89 \times 10^{-1}$	$1.32 \times 10^{0} \pm 1.61 \times 10^{-1}$	171.85 ± 2.01	24.14 ± 1.09
SymSet-5	$4.21 \times 10^{-1} \pm 3.58 \times 10^{-2}$	$3.19 \times 10^{-1} \pm 9.94 \times 10^{-2}$	177.98 ± 1.57	13.09 ± 0.37
SymSet-6	$2.08 \times 10^{0} \pm 4.80 \times 10^{-1}$	$1.42 \times 10^{0} \pm 3.72 \times 10^{-1}$	179.34 ± 1.75	13.28 ± 0.36
SymSet-7	$2.43 \times 10^{-2} \pm 9.62 \times 10^{-3}$	$5.25 \times 10^{-1} \pm 7.62 \times 10^{-2}$	133.89 ± 6.95	11.78 ± 0.24
SymSet-8	$3.93 \times 10^{-1} \pm 5.08 \times 10^{-2}$	$4.02 \times 10^{-1} \pm 6.42 \times 10^{-2}$	169.46 ± 1.81	11.54 ± 0.41
SymSet-9	$1.34 \times 10^{-1} \pm 3.55 \times 10^{-2}$	$1.23 \times 10^{-1} \pm 1.90 \times 10^{-2}$	175.04 ± 1.04	12.23 ± 0.29
SymSet-10	$3.32 \times 10^{-1} \pm 3.09 \times 10^{-2}$	$9.91 \times 10^{-1} \pm 9.59 \times 10^{-2}$	178.27 ± 1.75	11.06 ± 0.17
SymSet-11	$2.49 \times 10^{-1} \pm 2.71 \times 10^{-2}$	$1.11 \times 10^{-1} \pm 2.24 \times 10^{-2}$	166.91 ± 1.66	21.26 ± 0.53
SymSet-12	$8.76 \times 10^{-1} \pm 6.89 \times 10^{-2}$	$8.32 \times 10^{-1} \pm 1.39 \times 10^{-1}$	178.56 ± 1.53	11.65 ± 0.32
SymSet-13	$2.19 \times 10^{-1} \pm 1.77 \times 10^{-1}$	$8.69 \times 10^{-1} \pm 1.71 \times 10^{-1}$	126.39 ± 7.88	9.92 ± 0.23
SymSet-14	$4.93 \times 10^{-17} \pm 2.85 \times 10^{-18}$	$8.24 \times 10^{-2} \pm 8.50 \times 10^{-3}$	99.85 ± 3.81	9.95 ± 0.21
SymSet-15	$2.86 \times 10^{-17} \pm 4.49 \times 10^{-18}$	$6.22 \times 10^{-2} \pm 1.28 \times 10^{-2}$	94.42 ± 2.84	9.93 ± 0.18
SymSet-17	$9.16 \times 10^{-2} \pm 6.52 \times 10^{-3}$	$5.44 \times 10^{-2} \pm 5.51 \times 10^{-3}$	171.25 ± 1.55	10.46 ± 0.19
Average	$4.65 \times 10^{-1} \pm 1.24 \times 10^{-1}$	$4.57 \times 10^{-1} \pm 7.98 \times 10^{-2}$	158.57 ± 2.59	**16.23 ± 0.67**
Benchmark	**PySR**	**gplearn**	**PySR**	**gplearn**
SymSet-1	$1.10 \times 10^{-3} \pm 3.01 \times 10^{-4}$	$4.45 \times 10^{-2} \pm 2.91 \times 10^{-3}$	61.21 ± 19.72	140.44 ± 7.27
SymSet-2	$5.75 \times 10^{-2} \pm 2.28 \times 10^{-2}$	$3.42 \times 10^{-1} \pm 5.12 \times 10^{-2}$	119.55 ± 39.39	145.84 ± 1.67
SymSet-3	$1.71 \times 10^{-3} \pm 2.02 \times 10^{-4}$	$1.72 \times 10^{-2} \pm 9.41 \times 10^{-3}$	12.62 ± 0.69	169.38 ± 1.19
SymSet-4	$6.14 \times 10^{-1} \pm 6.12 \times 10^{-1}$	$2.89 \times 10^{0} \pm 5.77 \times 10^{-1}$	79.39 ± 1.98	248.12 ± 9.71
SymSet-5	$5.91 \times 10^{-2} \pm 1.28 \times 10^{-2}$	$2.64 \times 10^{-1} \pm 1.76 \times 10^{-2}$	69.47 ± 0.49	190.83 ± 1.41
SymSet-6	$6.57 \times 10^{0} \pm 1.96 \times 10^{0}$	$3.94 \times 10^{0} \pm 1.31 \times 10^{0}$	67.45 ± 0.67	215.11 ± 6.69
SymSet-7	$5.22 \times 10^{-2} \pm 4.64 \times 10^{-2}$	$8.93 \times 10^{-1} \pm 1.50 \times 10^{-1}$	117.47 ± 28.85	169.94 ± 1.91
SymSet-8	$3.57 \times 10^{-1} \pm 3.85 \times 10^{-2}$	$4.35 \times 10^{-1} \pm 3.16 \times 10^{-2}$	135.13 ± 30.82	167.64 ± 0.98
SymSet-9	$2.58 \times 10^{-2} \pm 8.51 \times 10^{-3}$	$2.19 \times 10^{-1} \pm 4.17 \times 10^{-2}$	162.11 ± 45.01	159.22 ± 1.86
SymSet-10	$4.14 \times 10^{-2} \pm 1.72 \times 10^{-2}$	$1.63 \times 10^{0} \pm 3.38 \times 10^{-1}$	48.17 ± 0.43	192.92 ± 1.72
SymSet-11	$2.81 \times 10^{-2} \pm 3.40 \times 10^{-3}$	$1.88 \times 10^{-1} \pm 2.91 \times 10^{-2}$	49.88 ± 0.63	184.38 ± 1.34
SymSet-12	$2.53 \times 10^{-2} \pm 1.11 \times 10^{-2}$	$2.62 \times 10^{-1} \pm 3.11 \times 10^{-2}$	136.14 ± 41.77	187.32 ± 2.90
SymSet-13	$8.44 \times 10^{-2} \pm 6.39 \times 10^{-2}$	$1.07 \times 10^{-16} \pm 1.29 \times 10^{-17}$	16.01 ± 2.98	12.52 ± 0.77
SymSet-14	$1.59 \times 10^{-2} \pm 4.51 \times 10^{-3}$	$9.82 \times 10^{-2} \pm 1.59 \times 10^{-2}$	57.54 ± 9.81	142.12 ± 5.79
SymSet-15	$6.90 \times 10^{-3} \pm 4.02 \times 10^{-3}$	$2.05 \times 10^{-1} \pm 1.36 \times 10^{-2}$	93.54 ± 56.54	147.58 ± 1.12
SymSet-17	$3.94 \times 10^{-2} \pm 5.21 \times 10^{-3}$	$1.18 \times 10^{-1} \pm 8.20 \times 10^{-3}$	167.45 ± 65.81	148.42 ± 0.67
Average	$4.99 \times 10^{-1} \pm 1.76 \times 10^{-1}$	$7.22 \times 10^{-1} \pm 1.64 \times 10^{-1}$	87.07 ± 21.60	163.86 ± 2.94

A.2 More Examples on Our Matrix-Based Encoding Scheme

The encoding process happens according to a table of mapping rules that is very straightforward to understand and use. For example, consider the mapping rules shown in Table A.5 below, where d is the dimension of the input feature vector. Note that Table A.5 involves more transformations than Table 2.2.

Table A.5: An example table of mapping rules for a basis function. Placeholder operands are denoted by $\bullet$, e.g. $\bullet^2$ corresponds to the square operator. The identity operator is denoted by $\bullet^1$.

$b_{\bullet,1}$	0	1	2	3	4	5	6	7	8	9
Transformation (T)	1	$\bullet^1$	$\bullet^{-1}$	$\bullet^2$	$\bullet^3$	cos	sin	exp	ln	$\sqrt{\bullet}$
$b_{\bullet,2}$	0	1	2							
Argument Type (arg)	x	$\sum$	$\prod$							
$b_{\bullet,3}, \cdots, b_{\bullet,m_{\boldsymbol{B}}}$	0	1	2	3	$\cdots$	d				
Variable (v)	skip	x_1	x_2	x_3	$\cdots$	x_d				

Example 1. For $d = 2$, $n_{\boldsymbol{B}^\phi} = 2$ and $m_{\boldsymbol{B}^\phi} = 4$ (i.e. $n_v = 2$), the basis function $\phi(\mathbf{x}) = x_1^2 e^{x_1 x_2}$ can be generated according to the encoding steps shown in Table A.6.

Table A.6: Encoding steps corresponding to the basis function $\phi(\mathbf{x}) = x_1^2 e^{x_1 x_2}$.

Step	T	arg	v_1	v_2	Update
1	$\bullet^2$	x	x_1	—	$T_1(\mathbf{x}) = x_1^2$
2	exp	$\prod$	x_1	x_2	$T_2(\mathbf{x}) = e^{x_1 x_2}$
Final Update: $\phi(\mathbf{x}) = T_1(\mathbf{x}) \cdot T_2(\mathbf{x})$					

Based on the mapping rules in Table A.5 and the encoding steps in Table A.6, the basis function $\phi(\mathbf{x}) = x_1^2 e^{x_1 x_2}$ can be encoded by a 2×4 matrix as follows:

$$\boldsymbol{B}^\phi = \begin{bmatrix} 3 & 0 & 1 & \bullet \\ 7 & 2 & 1 & 2 \end{bmatrix} \tag{A.4}$$

Example 2. For $d = 3$, $n_{\boldsymbol{B}^\phi} = 5$ and $m_{\boldsymbol{B}^\phi} = 5$ (i.e. $n_v = 3$), the basis function $\phi(\mathbf{x}) = \frac{x_2^3 \sin(x_2 x_3)\sqrt{x_2 + 2x_3}}{2x_1 + x_2}$ can be generated according to the encoding steps in Table A.7.

Based on the mapping rules in Table A.5 and the encoding steps in Table A.7, the basis function $\phi(\mathbf{x}) = \frac{x_2^3 \sin(x_2 x_3)\sqrt{x_2 + 2x_3}}{2x_1 + x_2}$ can be encoded by a 5×5 matrix as follows:

$$\boldsymbol{B}^\phi = \begin{bmatrix} 2 & 1 & 1 & 1 & 2 \\ 4 & 0 & 2 & \bullet & \bullet \\ 6 & 2 & 2 & 3 & \bullet \\ 9 & 1 & 2 & 3 & 3 \\ 0 & \bullet & \bullet & \bullet & \bullet \end{bmatrix} \tag{A.5}$$

Table A.7: Encoding steps corresponding to the basis function $\phi(\mathbf{x}) = \frac{x_2^3 \sin(x_2 x_3)\sqrt{x_2+2x_3}}{2x_1+x_2}$.

Step	T	arg	v_1	v_2	v_3	Update
1	$\bullet^{-1}$	$\sum$	x_1	x_1	x_2	$T_1(\mathbf{x}) = (2x_1 + x_2)^{-1}$
2	$\bullet^3$	x	x_2	—	—	$T_2(\mathbf{x}) = x_2^3$
3	sin	$\prod$	x_2	x_3	—	$T_3(\mathbf{x}) = \sin(x_2 x_3)$
4	$\sqrt{\bullet}$	$\sum$	x_2	x_3	x_3	$T_4(\mathbf{x}) = \sqrt{x_2 + 2x_3}$
5	1	—	—	—	—	$T_5(\mathbf{x}) = 1$
Final Update: $\phi(\mathbf{x}) = T_1(\mathbf{x}) \cdot T_2(\mathbf{x}) \cdot T_3(\mathbf{x}) \cdot T_4(\mathbf{x}) \cdot T_5(\mathbf{x})$						

Example 3. For $n_{\boldsymbol{B}^\psi} = 2$, the basis function $\psi(y) = y^3\sqrt{y}$ can be generated according to the encoding steps shown in Table A.8.

Table A.8: Encoding steps corresponding to the basis function $\psi(y) = y^3\sqrt{y}$.

Step	T	Update
1	$\bullet^3$	$T_1(y) = y^3$
2	$\sqrt{\bullet}$	$T_2(y) = \sqrt{y}$
Final Update: $\psi(y) = T_1(y) \cdot T_2(y)$		

Based on the mapping rules in Table A.5 and the encoding steps in Table A.8, the basis function $\psi(y) = y^3\sqrt{y}$ can be encoded by a 2×1 matrix as follows:

$$\boldsymbol{B}^\psi = \begin{bmatrix} 4 \\ 9 \end{bmatrix} \tag{A.6}$$

Example 4. For $n_{\boldsymbol{B}^\psi} = 3$, the basis function $\psi(y) = \ln(y)$ can be generated according to the encoding steps shown in Table A.9.

Table A.9: Encoding steps corresponding to the basis function $\psi(y) = \ln(y)$.

Step	T	Update
1	1	$T_1(y) = 1$
2	ln	$T_2(y) = \ln(y)$
3	1	$T_3(y) = 1$
Final Update: $\psi(y) = T_1(y) \cdot T_2(y) \cdot T_3(y)$		

Based on the mapping rules in Table A.5 and the encoding steps in Table A.9, the basis function $\psi(y) = \ln(y)$ can be encoded by a 3×1 matrix as follows:

$$\boldsymbol{B}^\psi = \begin{bmatrix} 0 \\ 8 \\ 0 \end{bmatrix} \tag{A.7}$$

Example 5. For $n_{\boldsymbol{B}^\psi} = 1$, the basis function $\psi(y) = e^y$ can be generated according to the encoding steps shown in Table A.10.

Table A.10: Encoding steps corresponding to the basis function $\psi(y) = e^y$.

Step	T	Update
1	exp	$T_1(y) = e^y$
Final Update: $\psi(y) = T_1(y)$		

Based on the mapping rules in Table A.5 and the encoding steps in Table A.10, the basis function $\psi(y) = e^y$ can be encoded by a 1×1 matrix as follows:

$$\boldsymbol{B}^\psi = \begin{bmatrix} 7 \end{bmatrix} \tag{A.8}$$

A.3 Symbolic Regression Benchmark Problem Sets

Table A.11: Specifications of the Symbolic Regression (SR) benchmark problems. Input variables are denoted by x for 1-dimensional problems, and by (x_1, x_2) for 2-dimensional problems. $U(a, b, c)$ indicates c random points uniformly sampled between a and b for every input variable; different random seeds are used for the training and test sets. $E(a, b, c)$ indicates c evenly spaced points between a and b for every input variable; the same points are used for the training and test sets $\big($except Neat-6, which uses $E(1, 120, 120)$ as test set, and the Jin tests, which use $U(-3, 3, 30)$ as test set$\big)$. To simplify the notation, libraries (of allowable arithmetic operators and mathematical functions) are defined relative to a 'base' library $\mathcal{L}_0 = \{+, -, \times, \div, \cos, \sin, \exp, \ln\}$. Placeholder operands are denoted by $\bullet$, e.g. $\bullet^2$ corresponds to the square operator.

Benchmark	**Expression**	**Dataset**	**Library**
Nguyen-1	$y = x^3 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Nguyen-2	$y = x^4 + x^3 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Nguyen-3	$y = x^5 + x^4 + x^3 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Nguyen-4	$y = x^6 + x^5 + x^4 + x^3 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Nguyen-5	$y = \sin(x^2)\cos(x) - 1$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Nguyen-6	$y = \sin(x) + \sin(x + x^2)$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Nguyen-7	$y = \ln(x + 1) + \ln(x^2 + 1)$	$U(0, 2, 20)$	$\mathcal{L}_0$
Nguyen-8	$y = \sqrt{x}$	$U(0, 4, 20)$	$\mathcal{L}_0$
Nguyen-9	$y = \sin(x_1) + \sin(x_2^2)$	$U(0, 1, 20)$	$\mathcal{L}_0$
Nguyen-10	$y = 2\sin(x_1)\cos(x_2)$	$U(0, 1, 20)$	$\mathcal{L}_0$
Nguyen-11	$y = x_1^{x_2}$	$U(0, 1, 20)$	$\mathcal{L}_0$
Nguyen-12	$y = x_1^4 - x_1^3 + \frac{1}{2}x_2^2 - x_2$	$U(0, 1, 20)$	$\mathcal{L}_0$
Nguyen-12*	$y = x_1^4 - x_1^3 + \frac{1}{2}x_2^2 - x_2$	$U(0, 10, 20)$	$\mathcal{L}_0$
Jin-1	$y = 2.5x_1^4 - 1.3x_1^3 + 0.5x_2^2 - 1.7x_2$	$U(-3, 3, 100)$	$\mathcal{L}_0 - \{\ln\} \cup \{\bullet^2, \bullet^3, \text{const}\}$
Jin-2	$y = 8x_1^2 + 8x_2^3 - 15$	$U(-3, 3, 100)$	$\mathcal{L}_0 - \{\ln\} \cup \{\bullet^2, \bullet^3, \text{const}\}$
Jin-3	$y = 0.2x_1^3 + 0.5x_2^3 - 1.2x_2 - 0.5x_1$	$U(-3, 3, 100)$	$\mathcal{L}_0 - \{\ln\} \cup \{\bullet^2, \bullet^3, \text{const}\}$
Jin-4	$y = 1.5e^{x_1} + 5\cos(x_2)$	$U(-3, 3, 100)$	$\mathcal{L}_0 - \{\ln\} \cup \{\bullet^2, \bullet^3, \text{const}\}$
Jin-5	$y = 6\sin(x_1)\cos(x_2)$	$U(-3, 3, 100)$	$\mathcal{L}_0 - \{\ln\} \cup \{\bullet^2, \bullet^3, \text{const}\}$
Jin-6	$y = 1.35x_1x_2 + 5.5\sin((x_1 - 1)(x_2 - 1))$	$U(-3, 3, 100)$	$\mathcal{L}_0 - \{\ln\} \cup \{\bullet^2, \bullet^3, \text{const}\}$
Neat-1	$y = x^4 + x^3 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0 \cup \{1\}$
Neat-2	$y = x^5 + x^4 + x^3 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0 \cup \{1\}$
Neat-3	$y = \sin(x^2)\cos(x) - 1$	$U(-1, 1, 20)$	$\mathcal{L}_0 \cup \{1\}$
Neat-4	$y = \ln(x + 1) + \ln(x^2 + 1)$	$U(0, 2, 20)$	$\mathcal{L}_0 \cup \{1\}$
Neat-5	$y = 2\sin(x_1)\cos(x_2)$	$U(-1, 1, 100)$	$\mathcal{L}_0$
Neat-6	$y = \sum_{k=1}^{x} \frac{1}{k}$	$E(1, 50, 50)$	$\{+, \times, \div, \bullet^{-1}, -\bullet, \sqrt{\bullet}\}$
Neat-7	$y = 2 - 2.1\cos(9.8x_1)\sin(1.3x_2)$	$E(-50, 50, 10^5)$	$\mathcal{L}_0 \cup \{\tan, \tanh, \bullet^2, \bullet^3, \sqrt{\bullet}\}$
Neat-8	$y = \frac{e^{-(x_1-1)^2}}{1.2+(x_2-2.5)^2}$	$U(0.3, 4, 100)$	$\{+, -, \times, \div, \exp, e^{-\bullet}, \bullet^2\}$
Neat-9	$y = \frac{1}{1+x_1^{-4}} + \frac{1}{1+x_2^{-4}}$	$E(-5, 5, 21)$	$\mathcal{L}_0$

Table A.12: Specifications of the Symbolic Regression (SR) benchmark problems. Input variables are denoted by x for 1-dimensional problems, by (x_1, x_2) for 2-dimensional problems, and by (x_1, x_2, x_3) for 3-dimensional problems. $U(a, b, c)$ indicates c random points uniformly sampled between a and b for every input variable; different random seeds are used for the training and test sets. To simplify the notation, libraries (of allowable arithmetic operators and mathematical functions) are defined relative to 'base' libraries $\mathcal{L}_0 = \{+, -, \times, \div, \cos, \sin, \exp, \ln\}$ or $\mathcal{L}_0^c = \mathcal{L}_0 \cup \{\text{const}\}$. Placeholder operands are denoted by $\bullet$, e.g. $\bullet^2$ corresponds to the square operator.

Benchmark	**Expression**	**Dataset**	**Library**
Livermore-1	$y = \frac{1}{3} + x + \sin(x^2)$	$U(-10, 10, 1000)$	$\mathcal{L}_0$
Livermore-2	$y = \sin(x^2)\cos(x) - 2$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-3	$y = \sin(x^3)\cos(x^2) - 1$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-4	$y = \ln(x + 1) + \ln(x^2 + 1) + \ln(x)$	$U(0, 2, 20)$	$\mathcal{L}_0$
Livermore-5	$y = x_1^4 - x_1^3 + x_1^2 - x_2$	$U(0, 1, 20)$	$\mathcal{L}_0$
Livermore-6	$y = 4x^4 + 3x^3 + 2x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-7	$y = \sinh(x)$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-8	$y = \cosh(x)$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-9	$y = x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-10	$y = 6\sin(x_1)\cos(x_2)$	$U(0, 1, 20)$	$\mathcal{L}_0$
Livermore-11	$y = \frac{x_1^2 x_1^2}{x_1 + x_2}$	$U(-1, 1, 50)$	$\mathcal{L}_0$
Livermore-12	$y = \frac{x_1^5}{x_2^3}$	$U(-1, 1, 50)$	$\mathcal{L}_0$
Livermore-13	$y = x^{\frac{1}{3}}$	$U(0, 4, 20)$	$\mathcal{L}_0$
Livermore-14	$y = x^3 + x^2 + x + \sin(x) + \sin(x^2)$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-15	$y = x^{\frac{1}{5}}$	$U(0, 4, 20)$	$\mathcal{L}_0$
Livermore-16	$y = x^{\frac{2}{5}}$	$U(0, 4, 20)$	$\mathcal{L}_0$
Livermore-17	$y = 4\sin(x_1)\cos(x_2)$	$U(0, 1, 20)$	$\mathcal{L}_0$
Livermore-18	$y = \sin(x^2)\cos(x) - 5$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-19	$y = x^5 + x^4 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-20	$y = e^{-x^2}$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-21	$y = x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	$U(-1, 1, 20)$	$\mathcal{L}_0$
Livermore-22	$y = e^{-0.5x^2}$	$U(-1, 1, 20)$	$\mathcal{L}_0$
SymSet-1	$y = x\sinh(x) - \frac{4}{5}$	$U(-1, 1, 20)$	$\mathcal{L}_0^c - \{\ln\} \cup \{e^{-\bullet}\}$
SymSet-2	$y = (x^5 - 3x^4 - 2.8x + 5)^{-1}$	$U(-1, 1, 20)$	$\mathcal{L}_0^c \cup \{\bullet^{-1}\}$
SymSet-3	$y = (x^4 - 1.2x^2 + 11.5)^{\frac{1}{3}}$	$U(-1, 1, 20)$	$\mathcal{L}_0^c \cup \{\bullet^2, \bullet^3\}$
SymSet-4	$y = 0.8 - \cos(x) + 4.2e^x \sin(x^2)$	$U(-3, 3, 20)$	$\mathcal{L}_0^c$
SymSet-5	$y = 4.5x_1^2 + x_1x_2^3 - 1.7x_2 - 3.1$	$U(-1, 1, 20)$	$\mathcal{L}_0^c$
SymSet-6	$y = \frac{5}{3x_1 - x_2^3}$	$U(-1, 1, 20)$	$\mathcal{L}_0^c \cup \{\bullet^{-1}\}$
SymSet-7	$y = \ln(x_1^3 + 4x_1x_2)$	$U(0, 2, 20)$	$\mathcal{L}_0^c$
SymSet-8	$y = \sqrt{5x_1^5 + 14x_1^3x_2^4 - 2x_2 + 7}$	$U(-1, 1, 20)$	$\mathcal{L}_0^c \cup \{\bullet^2, \bullet^3\}$
SymSet-9	$y = (2x_1 + x_2)^{-\frac{2}{3}}$	$U(0, 2, 20)$	$\mathcal{L}_0^c$
SymSet-10	$y = 1.5\cos(x_1)\ln(x_1x_2) - 2.5$	$U(0, 1, 20)$	$\mathcal{L}_0^c$
SymSet-11	$y = \sqrt{2\cos(x_1) + 30e^{x_2}} + 4$	$U(-1, 1, 20)$	$\mathcal{L}_0^c \cup \{\bullet^2\}$
SymSet-12	$y = 0.4x_1^4 + 6.2x_2 - 3.5x_1x_3 - 4.5$	$U(-1, 1, 20)$	$\mathcal{L}_0^c$
SymSet-13	$y = \frac{2x_2}{x_1 + x_3}$	$U(0, 1, 20)$	$\mathcal{L}_0^c$
SymSet-14	$y = \frac{x_1x_2x_3}{x_1 + x_2 + x_3}$	$U(0, 2, 20)$	$\mathcal{L}_0^c$
SymSet-15	$y = (x_1 + x_2)^{x_3}$	$U(0, 1, 20)$	$\mathcal{L}_0^c$
SymSet-16	$y = e^{2.6x_1 - \ln(x_2) + 9.8\cos(x_3)}$	$U(0, 1, 20)$	$\mathcal{L}_0^c$
SymSet-17	$y = \ln\left(0.2e^{x_1 + x_2} + 0.5\cos(x_3^2)\right)$	$U(0, 1, 20)$	$\mathcal{L}_0^c$

A.4 Typical Recovered Expressions

Table A.13: Typical expressions (with exact symbolic equivalence) recovered by GSR for the Nguyen benchmark set. Note that the coefficients in the GSR expressions form a unit vector due to the normalization constraint imposed by the Lasso problem in Eq. (2.6). It is easy to verify (by simplification) that the GSR expressions are symbolically equivalent to the ground truth expressions.

Benchmark		Expression
Nguyen-1	Truth	$y = x^3 + x^2 + x$
	GSR	$-0.558\, y = -0.1395(x \times x \times x) - 0.1395(x \times x \times x) - 0.2697x - 0.1395(x \times x \times x)$ $-0.2697x - 0.2697x + 0.0651(x + x + x + x) + 0.0651(x + x + x + x)$ $-0.1395(x \times x \times x) - 0.2697x - 0.558(x \times x)$
Nguyen-2	Truth	$y = x^4 + x^3 + x^2 + x$
	GSR	$-0.58554\, y = -0.09759x(x + x) - 0.58554x - 0.19518(x \times x)x - 0.09759(x + x)x$ $-0.19518(x \times x \times x) - 0.29277x(x \times x \times x) - 0.29277x(x \times x \times x)$ $-0.09759(x + x)x - 0.19518(x \times x)x$
Nguyen-3	Truth	$y = x^5 + x^4 + x^3 + x^2 + x$
	GSR	$0.5\, y = +0.125x + 0.25(x \times x \times x \times x \times x) + 0.125(x)x + 0.5(x \times x)x$ $+0.125x + 0.5x(x \times x \times x \times x) + 0.125x + 0.125(x \times x) + 0.125x$ $+0.125(x)x + 0.125(x \times x) + 0.25x(x \times x \times x)$
Nguyen-4	Truth	$y = x^6 + x^5 + x^4 + x^3 + x^2 + x$
	GSR	$-0.48318\, y = -0.096636x - 0.48318x(x \times x)(x \times x) - 0.16106(x \times x) - 0.16106(x \times x)$ $-0.24159(x \times x)x - 0.096636x - 0.48318(x \times x \times x)(x \times x)x$ $-0.24159(x \times x \times x) - 0.24159(x \times x \times x)(x + x) - 0.096636x$ $-0.096636x - 0.096636x - 0.16106(x \times x)$
Nguyen-5	Truth	$y = \sin(x^2)\cos(x) - 1$
	GSR	$0.63246\, y = 0.63246\cos(x)\sin(x \times x) - 0.31623 - 0.31623$
Nguyen-6	Truth	$y = \sin(x) + \sin(x + x^2)$
	GSR	$0.5\, y = 0.5\cos(x)\sin(x \times x) + 0.5\sin(x) + 0.5\cos(x \times x)\sin(x)$
Nguyen-7	Truth	$y = \ln(x + 1) + \ln(x^2 + 1)$
	GSR	$0.70956\, e^y = +0.1095x(x \times x \times x) + 0.1095(x \times x \times x \times x) + 0.17082x + 0.23652$ $+0.17082x + 0.12702(x \times x)(x + x) + 0.17082x - 0.1095x(x + x)(x \times x)$ $+0.22776x(x \times x) + 0.23652 + 0.22776(x \times x)x + 0.23652$ $+0.23652(x + x + x)x + 0.17082x + 0.01314(x + x)$
Nguyen-8	Truth	$y = \sqrt{x}$
	GSR	$0.83654\, \ln(y) = -0.032175\ln(x \times x \times x \times x \times x) + 0.54697\ln(x)$
Nguyen-9	Truth	$y = \sin(x_1) + \sin(x_2^2)$
	GSR	$-0.57735\, y = -0.57735\sin(x_1) - 0.57735\sin(x_2 \times x_2)$
Nguyen-10	Truth	$y = 2\sin(x_1)\cos(x_2)$
	GSR	$0.44721\, y = 0.89442\sin(x_1)\cos(x_2)$
Nguyen-11	Truth	$y = x_1^{x_2}$
	GSR	$0.70711\, \ln(y) = 0.70711\, x_2\ln(x_1)$
Nguyen-12	Truth	$y = x_1^4 - x_1^3 + \frac{1}{2}x_2^2 - x_2$
	GSR	$-0.4\, y = -0.2(x_2 \times x_2) - 0.4x_1 - 0.4x_1 + 0.4(x_1 + x_2 + x_1) + 0.4(x_1 \times x_1)x_1$ $-0.4x_1(x_1 \times x_1 \times x_1)$
Nguyen-12*	Truth	$y = x_1^4 - x_1^3 + \frac{1}{2}x_2^2 - x_2$
	GSR	$-0.6\, y = -0.1(x_2 + x_2 + x_2)x_2 - 0.3(x_1 + x_1)(x_1 \times x_1 \times x_1) + 0.3(x_1 \times x_1 \times x_1)$ $+0.3x_1(x_1 \times x_1) + 0.6x_2$

Table A.14: Typical expressions (with exact symbolic equivalence) recovered by GSR for the Jin benchmark set. Note that the coefficients in the GSR expressions form a unit vector due to the normalization constraint imposed by the Lasso problem in Eq. (2.6). It is easy to verify (by simplification) that the GSR expressions are symbolically equivalent to the ground truth expressions. Although GSR does not exactly recover Jin-6, it recovers approximations with very low RMSE, as shown in Table 2.5.

Benchmark		**Expression**
Jin-1	Truth	$y = 2.5x_1^4 - 1.3x_1^3 + 0.5x_2^2 - 1.7x_2$
	GSR	$0.30664\, y = +0.15332x_2^2 - 0.398632x_1^3 - 0.260644x_2 - 0.260644x_2 + 0.7666x_1x_1^3$
Jin-2	Truth	$y = 8x_1^2 + 8x_2^3 - 15$
	GSR	$0.06909\, y = -0.518175 + 0.55272x_2^3 - 0.518175 + 0.27636x_1^2 + 0.27636x_1^2$
Jin-3	Truth	$y = 0.2x_1^3 + 0.5x_2^3 - 1.2x_2 - 0.5x_1$
	GSR	$-0.7943\, y = -0.11914x_2 - 0.15886x_1^3 + 0.39715(x_2 + x_2 + x_2 + x_1) - 0.11915x_2 - 0.39715x_2^3$
Jin-4	Truth	$y = 1.5e^{x_1} + 5\cos(x_2)$
	GSR	$-0.25198\, y = -0.37797e^{x_1} - 0.62995\cos(x_2) - 0.62995\cos(x_2)$
Jin-5	Truth	$y = 6\sin(x_1)\cos(x_2)$
	GSR	$0.13484\, y = -0.80904\sin(x_2)\cos(x_1) + 0.40452\sin(x_2 + x_1) + 0.40452\sin(x_2 + x_1)$
Jin-6	Truth	$y = 1.35x_1x_2 + 5.5\sin((x_1 - 1)(x_2 - 1))$
	GSR	Not exactly recovered

Table A.15: Typical expressions (with exact symbolic equivalence) recovered by GSR for the Neat benchmark set. Note that the coefficients in the GSR expressions form a unit vector due to the normalization constraint imposed by the Lasso problem in Eq. (2.6). It is easy to verify (by simplification) that the GSR expressions are symbolically equivalent to the ground truth expressions. Although GSR does not exactly recover Neat-6, Neat-7, Neat-8, and Neat-9, it recovers approximations with very low RMSE, as shown in Table 2.6.

Benchmark		**Expression**
Neat-1	Truth	$y = x^4 + x^3 + x^2 + x$
	GSR	$-0.64952\, y = -0.32476x(x+x)(x \times x) - 0.064952(x+x) + 0.082996(x+x+x)$ $-0.32476(x \times x) - 0.2129x - 0.32476(x \times x) - 0.18764x(x+x)x$ $-0.064952(x+x) - 0.2129x - 0.2129x - 0.27424(x \times x \times x)$
Neat-2	Truth	$y = x^5 + x^4 + x^3 + x^2 + x$
	GSR	$0.3914\, y = +0.3914x(x \times x \times x \times x) + 0.18274x + 0.27676x(x) + 0.18274x$ $+0.02794(x+x) + 0.3914x(x \times x) - 0.02702(x+x+x)(x+x)$ $+0.18274x + 0.27676(x \times x) - 0.12682(x+x+x) + 0.3914(x \times x \times x)x$ $+0.18274x + 0.18274x - 0.12682(x+x+x) + 0.18274x$
Neat-3	Truth	$y = \sin(x^2)\cos(x) - 1$
	GSR	$-0.57735\, y = -0.57735 \sin(x \times x)\cos(x) + 0.57735$
Neat-4	Truth	$y = \ln(x+1) + \ln(x^2+1)$
	GSR	$-0.5\, e^y = -0.25(x \times x) - 0.5 - 0.25x - 0.25x - 0.5x(x \times x) - 0.25(x \times x)$
Neat-5	Truth	$y = 2\sin(x_1)\cos(x_2)$
	GSR	$0.57735\, y = 0.57735\cos(x_2)\sin(x_1) + 0.57735\cos(x_2)\sin(x_1)$
Neat-6	Truth	$y = \sum_{k=1}^{x} \frac{1}{k}$
	GSR	Not exactly recovered
Neat-7	Truth	$y = 2 - 2.1\cos(9.8x_1)\sin(1.3x_2)$
	GSR	Not exactly recovered
Neat-8	Truth	$y = \frac{e^{-(x_1-1)^2}}{1.2+(x_2-2.5)^2}$
	GSR	Not exactly recovered
Neat-9	Truth	$y = \frac{1}{1+x_1^{-4}} + \frac{1}{1+x_2^{-4}}$
	GSR	Not exactly recovered

Table A.16: Typical expressions (with exact symbolic equivalence) recovered by GSR for the Livermore benchmark set. Note that the coefficients in the GSR expressions form a unit vector due to the normalization constraint imposed by the Lasso problem in Eq. (2.6). It is easy to verify (by simplification) that the GSR expressions are symbolically equivalent to the ground truth expressions. Although GSR does not exactly recover Livermore-7 and Livermore-8, it naturally recovers their Taylor approximations, as discussed in Appendix A.1.

Benchmark		**Expression**
Livermore-1	Truth	$y = \frac{1}{3} + x + \sin(x^2)$
	GSR	$0.65079\, y = 0.21693 + 0.65079 \sin(x \times x) + 0.325395(x + x)$
Livermore-2	Truth	$y = \sin(x^2)\cos(x) - 2$
	GSR	$-0.40825\, y = -0.40825 \cos(x) \sin(x \times x) + 0.8165$
Livermore-3	Truth	$y = \sin(x^3)\cos(x^2) - 1$
	GSR	$-0.57735\, y = 0.57735 - 0.57735 \sin(x \times x \times x) \cos(x \times x)$
Livermore-4	Truth	$y = \ln(x + 1) + \ln(x^2 + 1) + \ln(x)$
	GSR	$-0.50998\, e^y = -0.25499(x \times x) - 0.21064x - 0.022175(x + x) - 0.21064x - 0.25499(x \times x)$ $-0.50998(x \times x)(x \times x) - 0.50998x(x \times x) - 0.022175(x + x)$
Livermore-5	Truth	$y = x_1^4 - x_1^3 + x_1^2 - x_2$
	GSR	$0.44721\, y = -0.44721x_2 + 0.44721(x_1 \times x_1) - 0.44721(x_1 \times x_1)x_1 + 0.44721(x_1 \times x_1)(x_1 \times x_1)$
Livermore-6	Truth	$y = 4x^4 + 3x^3 + 2x^2 + x$
	GSR	$-0.15763\, y = -0.26677x - 0.26677x - 0.093216(x + x) - 0.23918(x \times x)$ $-0.03804(x + x)x - 0.63052x(x)(x \times x) - 0.47289(x \times x \times x)$ $-0.093216(x + x) + 0.253886(x + x + x + x) - 0.26677x$
Livermore-7	Truth	$y = \sinh(x)$
	GSR	Not exactly recovered
Livermore-8	Truth	$y = \cosh(x)$
	GSR	Not exactly recovered
Livermore-9	Truth	$y = x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$
	GSR	$-0.30767\, y = -0.30767x(x \times x \times x)(x \times x) - 0.30767(x \times x \times x)x - 0.266671(x \times x)$ $-0.30767(x \times x \times x)(x)(x \times x \times x) - 0.153835x - 0.30767(x \times x \times x)$ $-0.30767(x \times x \times x \times x \times x \times x)(x \times x \times x \times x \times x) - 0.30767(x \times x \times x \times x \times x)x$ $+0.056037(x + x + x + x)(x + x + x + x) - 0.266671(x \times x) - 0.153835x$ $-0.30767(x \times x \times x \times x \times x)(x)(x \times x \times x) - 0.22364x(x + x + x)$
Livermore-10	Truth	$y = 6 \sin(x_1) \cos(x_2)$
	GSR	$0.22942\, y = 0.68826 \cos(x_2) \sin(x_1) + 0.68826 \sin(x_1) \cos(x_2)$
Livermore-11	Truth	$y = \frac{x_1^2 x_1^2}{x_1 + x_2}$
	GSR	$-0.40825\ \ln(y) = -0.8165 \ln(x_1 \times x_1) + 0.40825 \ln(x_2 + x_1)$

Table A.17: Typical expressions (with exact symbolic equivalence) recovered by GSR for the Livermore benchmark set (cont'd). Note that the coefficients in the GSR expressions form a unit vector due to the normalization constraint imposed by the Lasso problem in Eq. (2.6). It is easy to verify (by simplification) that the GSR expressions are symbolically equivalent to the ground truth expressions.

Benchmark		**Expression**
Livermore-12	Truth	$y = \frac{x_1^5}{x_2^3}$
	GSR	$0.16903\ \ln(y) = 0.84515\ln(x_1) - 0.50709\ln(x_2)$
Livermore-13	Truth	$y = x^{\frac{1}{3}}$
	GSR	$0.97332\ \ln(y) = 0.16222\ln(x) + 0.16222\ln(x)$
Livermore-14	Truth	$y = x^3 + x^2 + x + \sin(x) + \sin(x^2)$
	GSR	$0.40825\, y = 0.40825x(x \times x) + 0.40825\sin(x) + 0.40825(x \times x) + 0.40825\sin(x \times x) + 0.40825x$
Livermore-15	Truth	$y = x^{\frac{1}{5}}$
	GSR	$0.99015\ \ln(y) = 0.099015\ln(x) + 0.099015\ln(x)$
Livermore-16	Truth	$y = x^{\frac{2}{5}}$
	GSR	$0.99504\ \ln(y) = 0.099504\ln(x \times x \times x \times x \times x)$
Livermore-17	Truth	$y = 4\sin(x_1)\cos(x_2)$
	GSR	$0.17408\, y = 0.69632\sin(x_2 + x_1) - 0.69632\cos(x_1)\sin(x_2)$
Livermore-18	Truth	$y = \sin(x^2)\cos(x) - 5$
	GSR	$-0.19245\, y = 0.96225 - 0.19245\cos(x)\sin(x \times x)$
Livermore-19	Truth	$y = x^5 + x^4 + x^2 + x$
	GSR	$-0.46202\, y = -0.46202x(x \times x)x - 0.015609(x + x + x) + -0.46202 * (x^1 * x^1) - 0.46202x(x \times x)(x \times x)$ $-0.290313x - 0.15297(x + x) - 0.15297(x + x) + 0.12175(x + x + x + x)$
Livermore-20	Truth	$y = e^{-x^2}$
	GSR	$-0.89442\ \ln(y) = 0.44721(x + x)x$
Livermore-21	Truth	$y = x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$
	GSR	$-0.38914\, y = -0.027357(x + x + x)x - 0.38914(x \times x \times x)(x \times x) - 0.38914x(x \times x \times x \times x \times x)(x \times x \times x)$ $+0.0714425x(x + x)(x + x) - 0.38914x - 0.38914(x \times x \times x \times x \times x) - 0.22497(x \times x \times x)$ $-0.19457(x \times x \times x \times x \times x)(x + x)(x \times x) - 0.22497x(x \times x) - 0.027357x(x + x + x)$ $-0.22497(x \times x)x - 0.224998(x \times x) - 0.097285x(x + x + x + x)(x \times x \times x \times x \times x)$
Livermore-22	Truth	$y = e^{-0.5x^2}$
	GSR	$0.8165\ \ln(y) = -0.40825(x + x)x + 0.40825(x \times x)$

Table A.18: Typical expressions (with exact symbolic equivalence) recovered by GSR for the SymSet benchmark set. Note that the coefficients in the GSR expressions form a unit vector due to the normalization constraint imposed by the Lasso problem in Eq. (2.6). It is easy to verify (by simplification) that the GSR expressions are symbolically equivalent to the ground truth expressions.

Benchmark		**Expression**
SymSet-1	Truth	$y = x\sinh(x) - \frac{4}{5}$
	GSR	$0.70448\,y = 0.35224e^{x}x - 0.17612xe^{-x} - 0.563584e^{x}e^{-x} - 0.17612xe^{-x}$
SymSet-2	Truth	$y = (x^5 - 3x^4 - 2.8x + 5)^{-1}$
	GSR	$-0.11335\,y^{-1} = -0.23188x - 0.11335 + 0.50651(x + x) - 0.11335 - 0.23188x$ $-0.11335 - 0.11335 - 0.23188x - 0.11335(x \times x \times x \times x)x$ $-0.11335 + 0.34005x(x \times x \times x)$
SymSet-3	Truth	$y = (x^4 - 1.2x^2 + 11.5)^{\frac{1}{3}}$
	GSR	$0.1173\,y^3 = +0.26576x^2 + 0.26576x^2 + 0.26576(x \times x) + 0.44965 + 0.44965$ $+0.02346(x + x + x + x + x)x^3 - 0.23451(x + x + x + x)x + 0.44965$
SymSet-4	Truth	$y = 0.8 - \cos(x) + 4.2e^{x}\sin(x^2)$
	GSR	$0.22485\,y = -0.112425\cos(x) + 0.94437\sin(x \times x)e^{x} - 0.112425\cos(x) + 0.17988$
SymSet-5	Truth	$y = 4.5x_1^2 + x_1x_2^3 - 1.7x_2 - 3.1$
	GSR	$-0.16964\,y = -0.16964(x_2 \times x_2 \times x_2 \times x_1) - 0.76338(x_1 \times x_1) + 0.525884 + 0.288388x_2$
SymSet-6	Truth	$y = \frac{5}{3x_1 - x_2^3}$
	GSR	$0.84515\,y^{-1} = -0.16903(x_2 \times x_2)x_2 + 0.50709x_1$
SymSet-7	Truth	$y = \ln(x_1^3 + 4x_1x_2)$
	GSR	$-0.482715\,e^{y} = -0.64362(x_1 + x_1 + x_1)x_2 + 0.21883x_2 - 0.482715(x_1 \times x_1 \times x_1) - 0.21883x_2$
SymSet-8	Truth	$y = \sqrt{5x_1^5 + 14x_1^3x_2^4 - 2x_2 + 7}$
	GSR	$0.060634\,y^2 = 0.30317(x_1^2 \times x_1^3) + 0.848876x_1^2x_2^3(x_1 \times x_2) + 0.424438 - 0.060634(x_2 + x_2)$
SymSet-9	Truth	$y = (2x_1 + x_2)^{-\frac{2}{3}}$
	GSR	$0.83205\,\ln(y) = -0.5547\ln(x_1 + x_2 + x_1)$
SymSet-10	Truth	$y = 1.5\cos(x_1)\ln(x_1x_2) - 2.5$
	GSR	$0.32444\,y = -0.8111 + 0.48666\ln(x_1 \times x_2)\cos(x_1)$
SymSet-11	Truth	$y = \sqrt{2\cos(x_1) + 30e^{x_2}} + 4$
	GSR	$-0.228568\,y + 0.028571\,y^2 = -0.457136 + 0.85713e^{x_2} + 0.057142\cos(x_1)$
SymSet-12	Truth	$y = 0.4x_1^4 + 6.2x_2 - 3.5x_1x_3 - 4.5$
	GSR	$0.16288\,y = -0.18324 + 0.065152x_1(x_1)(x_1 \times x_1) - 0.57008(x_3 \times x_1) + 0.504928x_2 - 0.18324$ $-0.18324 - 0.18324 + 0.504928x_2$
SymSet-13	Truth	$y = \frac{2x_2}{x_1 + x_3}$
	GSR	$0.57735\,\ln(y) = 0.57735\ln(x_2 + x_2) - 0.57735\ln(x_1 + x_3)$
SymSet-14	Truth	$y = \frac{x_1x_2x_3}{x_1 + x_2 + x_3}$
	GSR	$0.57735\,\ln(y) = 0.57735\ln(x_1 \times x_2 \times x_3) - 0.57735\ln(x_2 + x_3 + x_1)$
SymSet-15	Truth	$y = (x_1 + x_2)^{x_3}$
	GSR	$0.70711\,\ln(y) = 0.70711\,x_3\ln(x_1 + x_2)$
SymSet-16	Truth	$y = e^{2.6x_1 - \ln(x_2) + 9.8\cos(x_3)}$
	GSR	$0.098035\,\ln(y) = 0.960743\cos(x_3) - 0.0490175\ln(x_2 \times x_2) + 0.254891x_1$
SymSet-17	Truth	$y = \ln\left(0.2e^{x_1 + x_2} + 0.5\cos(x_3^2)\right)$
	GSR	$0.88045\,e^{y} = 0.17609e^{x_1 + x_2} + 0.440225\cos(x_3 \times x_3)$

A.5 Limitations

Although GSR achieves great results whether by recovering exact expressions or approximations with low errors, it still has several limitations:

Absence of division operations. The primary limiting factor to our GSR method is that it still cannot handle divisions. This is due to the way we define our encoding scheme. In this current version, we only consider a weighted sum of basis functions where the basis functions are a product of transformations; no divisions are involved. We can overcome this issue by modifying the encoding scheme to include divisions within the basis functions (e.g. a negative integer in the first column of the basis functions implies a division by the corresponding transformation, i.e. using Table A.5, a first-column entry of -8 encodes the division $\frac{1}{\ln}$). However, this will significantly increase the total number of possible combinations in which we can form basis matrices. Due to the lack of divisions in its current version, GSR suffers on some benchmarks such as Neat-6, Neat-8, Neat-9, Livermore-7, Livermore-8. It is worth noting that GSR can recover some divisions with the help of the ln or $\bullet^{-1}$ operators (see Livermore-11, Livermore-12, Livermore-20, Livermore-22, SymSet-2, SymSet-6, SymSet-9, SymSet-13, and SymSet-14). This is only possible when the original function consists of only one term (not a sum of terms).

Composition of tranformations. Another limiting factor to the current version of GSR is that it cannot recover expressions containing composite functions, such as $y = e^{\cos(x)} + \ln(x)$. In this example, the basis function $e^{\cos(x)}$ cannot be recovered by GSR due to our encoding scheme. Again, if $\ln(x)$ was not there, that is, if the function contained the first term only, i.e. $y = e^{\cos(x)}$, then GSR can handle the situation by recovering $\ln(y) = \cos(x)$. The benefits of using $g(y) = f(\mathbf{x})$ can be clearly observed on the SymSet benchmark problems (especially SymSet-16, and SymSet-17).

Choice of hyperparameters and search process. Throughout our experiments, we have observed that, for some benchmarks (such as Jin-6 and Neat-7), althought they are expressible by GSR, they were not fully recovered. GSR only recovered approximations for these benchmarks with very low errors. This can be explained by two reasons: i) The choice of hyperparameters affects the search process, ii) Our matrix-based GP search process may not be very effective on these benchmarks, given the complexity of their corresponding basis functions, and thus they may require a huge number of iterations to be recovered. That is, if we keep our code running for a very long time, we may be able to recover these benchmarks. This can be verified by expanding Jin-6 and Neat-7 as follows:

Jin-6:

$$
\begin{aligned}
y &= 1.35x_1x_2 + 5.5\sin\left((x_1-1)(x_2-1)\right) \\
&= 1.35x_1x_2 + 5.5\sin\left(x_1x_2 - x_1 - x_2 + 1\right) \\
&= 1.35x_1x_2 + 5.5\big[\sin\left(-x_1-x_2\right)\cos\left(x_1x_2+1\right) + \cos\left(-x_1-x_2\right)\sin\left(x_1x_2+1\right)\big] \\
&= 1.35x_1x_2 - 5.5\cos(1)\underbrace{\sin(x_1+x_2)\cos(x_1x_2)}_{\phi_1(\mathbf{x})} + 5.5\sin(1)\underbrace{\sin(x_1+x_2)\sin(x_1x_2)}_{\phi_2(\mathbf{x})} \\
&\quad + 5.5\cos(1)\underbrace{\cos(x_1+x_2)\sin(x_1x_2)}_{\phi_3(\mathbf{x})} + 5.5\sin(1)\underbrace{\cos(x_1+x_2)\cos(x_1x_2)}_{\phi_4(\mathbf{x})}
\end{aligned}
$$

Neat-7:

$$
\begin{aligned}
y &= 2 - 2.1\cos(9.8x_1)\sin(1.3x_2) \\
&= 2 - 2.1\big(\cos(9.8)\cos(x_1) - \sin(9.8)\sin(x_1)\big)\big(\sin(1.3)\cos(x_2) + \cos(1.3)\sin(x_2)\big) \\
&= 2 - 2.1\cos(9.8)\sin(1.3)\underbrace{\cos(x_1)\cos(x_2)}_{\phi_1(\mathbf{x})} + 2.1\sin(9.8)\sin(1.3)\underbrace{\sin(x_1)\cos(x_2)}_{\phi_2(\mathbf{x})} \\
&\quad - 2.1\cos(9.8)\cos(1.3)\underbrace{\cos(x_1)\sin(x_2)}_{\phi_3(\mathbf{x})} + 2.1\sin(9.8)\cos(1.3)\underbrace{\sin(x_1)\sin(x_2)}_{\phi_4(\mathbf{x})}
\end{aligned}
$$

As we can see, GSR has to find the four corresponding basis functions simultaneously in order to recover the expressions.

Indeed, there are plenty of expressions that still cannot be fully recovered by our GSR method. This is the case for all the other methods as well.

Appendix B

Supplementary Material for Chapter 3

B.1 Illustrative derivation of the adjoint equations for the considered cases.

Although the proposed method and its algorithm can be and has been computed in an automated fashion, here we show two detailed illustrative examples for 1-dimensional and 2-dimensional cases presented in Section 3.3 for the sake of better understanding the used notation and how the library of candidate terms looks like.

B.1.1 Heat and Burgers' Equations

As mentioned in Sections 3.3.1 and 3.3.2, for these two cases, we consider a system consisting of a single PDE, i.e. $N = \dim(\boldsymbol{f}) = \dim(\boldsymbol{p}) = 1$ where $\boldsymbol{f} = f$ and $\boldsymbol{p} = p$, in a one-dimensional input space, i.e. $n = \dim(\boldsymbol{x}) = \dim(\boldsymbol{d}) = 1$ where $\boldsymbol{x} = x$, and $\boldsymbol{d} = d$. In addition, we consider candidate terms consisting of derivatives with indices $d \in \{1, 2, 3\}$ and polynomials with indices $p \in \{1, 2, 3\}$). In other words, $d_{\max} = 3$ and $p_{\max} = 3$. The resulting forward model in Eq. (3.1) takes the form

$$\begin{aligned}
\mathcal{L}[f] &= \frac{\partial f}{\partial t} + \sum_{d=1}^{3}\sum_{p=1}^{3} \alpha_{d,p} \frac{\partial^d \left(f^p\right)}{\partial x^d} \\
&= \frac{\partial f}{\partial t} + \alpha_{1,1}\frac{\partial f}{\partial x} + \alpha_{1,2}\frac{\partial \left(f^2\right)}{\partial x} + \alpha_{1,3}\frac{\partial \left(f^3\right)}{\partial x} \\
&\quad + \alpha_{2,1}\frac{\partial^2 f}{\partial x^2} + \alpha_{2,2}\frac{\partial^2 \left(f^2\right)}{\partial x^2} + \alpha_{2,3}\frac{\partial^2 \left(f^3\right)}{\partial x^2} \\
&\quad + \alpha_{3,1}\frac{\partial^3 f}{\partial x^3} + \alpha_{3,2}\frac{\partial^3 \left(f^2\right)}{\partial x^3} + \alpha_{3,3}\frac{\partial^3 \left(f^3\right)}{\partial x^3}
\end{aligned} \tag{B.1}$$

where $\alpha_{d,p}$ denotes the parameter corresponding to the term with d-th derivative and p-th polynomial order. As we can observe, we have 9 terms with unknown coefficients $\boldsymbol{\alpha} = [\alpha_{d,p}]_{d\in\{1,2,3\},p\in\{1,2,3\}}$ that we aim to find using the proposed adjoint method.

The cost functional in this case is simply

$$\mathcal{C} = \sum_{j,k} \left(f^*\big(x^{(k)}, t^{(j)}\big) - f\big(x^{(k)}, t^{(j)}\big) \right)^2 + \frac{1}{\Delta x \Delta t} \int \lambda(x,t) \mathcal{L}[f(x,t)] dx dt + \epsilon_0 ||\boldsymbol{\alpha}||_2^2 \ . \quad \text{(B.2)}$$

Letting variational derivatives of $\mathcal{C}$ with respect to f to be zero, and using integration by parts, the corresponding adjoint equation can be obtained as

$$\begin{aligned}
\frac{\partial \lambda}{\partial t} =& \sum_{d=1}^{3} \sum_{p=1}^{3} (-1)^d \alpha_{d,p} \frac{\partial \big(f^p\big)}{\partial f} \frac{\partial^d \lambda}{\partial x^d} \\
=& - \alpha_{1,1} \frac{\partial \lambda}{\partial x} - \alpha_{1,2} \big(2f\big) \frac{\partial \lambda}{\partial x} - \alpha_{1,3} \big(3f^2\big) \frac{\partial \lambda}{\partial x} \\
&+ \alpha_{2,1} \frac{\partial^2 \lambda}{\partial x^2} + \alpha_{2,2} \big(2f\big) \frac{\partial^2 \lambda}{\partial x^2} + \alpha_{2,3} \big(3f^2\big) \frac{\partial^2 \lambda}{\partial x^2} \\
&- \alpha_{3,1} \frac{\partial^3 \lambda}{\partial x^3} - \alpha_{3,2} \big(2f\big) \frac{\partial^3 \lambda}{\partial x^3} - \alpha_{3,3} \big(3f^2\big) \frac{\partial^3 \lambda}{\partial x^3}
\end{aligned} \quad \text{(B.3)}$$

with final condition $\lambda(x^{(k)}, t^{(j+1)}) = 2(f^*(x^{(k)}, t^{(j+1)}) - f(x^{(k)}, t^{(j+1)}))$ for all j, k. The parameters $\boldsymbol{\alpha}$ are then found using the gradient descent method with update rule

$$\alpha_{d,p} \leftarrow \alpha_{d,p} - \eta \frac{\partial \mathcal{C}}{\partial \alpha_{d,p}} \quad \text{(B.4)}$$

$$\text{where} \quad \eta = \beta \min(\Delta x)^{d - d_{\max}} \quad \text{and} \quad \frac{\partial \mathcal{C}}{\partial \alpha_{d,p}} = (-1)^d \frac{1}{\Delta x \Delta t} \int f^p \frac{\partial^d \lambda}{\partial x^d} dx dt + 2\epsilon_0 \alpha_{d,p} \ . \quad \text{(B.5)}$$

This leads to the update rule for each coefficient, for example

$$\begin{aligned}
\alpha_{1,1} &\leftarrow \alpha_{1,1} - \frac{\beta}{\min(\Delta x)^2} \frac{1}{\Delta x \Delta t} \int f \frac{\partial \lambda}{\partial x} dx dt - 2\beta\epsilon_0 \alpha_{1,1} \\
\alpha_{1,2} &\leftarrow \alpha_{1,2} - \frac{\beta}{\min(\Delta x)^2} \frac{1}{\Delta x \Delta t} \int f^2 \frac{\partial \lambda}{\partial x} dx dt - 2\beta\epsilon_0 \alpha_{1,2} \\
\alpha_{1,3} &\leftarrow \alpha_{1,3} - \frac{\beta}{\min(\Delta x)^2} \frac{1}{\Delta x \Delta t} \int f^3 \frac{\partial \lambda}{\partial x} dx dt - 2\beta\epsilon_0 \alpha_{1,3}
\end{aligned}$$

B.1.2 Reaction Diffusion System of Equations

As mentioned in Section 3.3.5, for this case, we consider a system consisting of two PDEs, i.e. $N = \dim(\boldsymbol{f}) = \dim(\boldsymbol{p}) = 2$ where $\boldsymbol{f} = [f_1, f_2]$ and $\boldsymbol{p} = [p_1, p_2]$, in a two-dimensional input space, i.e. $n = \dim(\boldsymbol{x}) = \dim(\boldsymbol{d}) = 2$ where $\boldsymbol{x} = [x_1, x_2]$, and $\boldsymbol{d} = [d_1, d_2]$. In addition, we consider candidate terms with derivatives such that $\boldsymbol{d} \in \mathcal{D}_{\boldsymbol{d}} = \{[0,0], [1,0], [0,1], [2,0], [0,2]\}$ and polynomials such that $\boldsymbol{p} \in \mathcal{D}_{\boldsymbol{p}} = \{[1,0], [0,1], [1,1], [2,0], [0,2], [2,1], [1,2], [3,0], [0,3]\}$. In other words, $d_{\max} = 2$ and $p_{\max} = 3$. The resulting forward model in Eq. (3.1) takes the form

$$\mathcal{L}_i[\boldsymbol{f}] = \partial_t f_i + \sum_{\boldsymbol{d},\boldsymbol{p}} \alpha_{i,\boldsymbol{d},\boldsymbol{p}} \nabla_{\boldsymbol{x}}^{(\boldsymbol{d})} [\boldsymbol{f}^{\boldsymbol{p}}] \quad \text{(B.6)}$$

where $i \in \{1,2\}$, $\boldsymbol{f^p} = f_1^{p_1} f_2^{p_2}$ and $\nabla_{\boldsymbol{x}}^{(\boldsymbol{d})} = \nabla_{x_1}^{(d_1)} \nabla_{x_2}^{(d_2)}$. This is equivalent to

$$\begin{aligned}
\mathcal{L}_i[f_1,f_2] &= \frac{\partial f_i}{\partial t} + \sum_{[d_1,d_2]\in\mathcal{D}_{\boldsymbol{d}}} \sum_{[p_1,p_2]\in\mathcal{D}_{\boldsymbol{p}}} \alpha_{i,[d_1,d_2],[p_1,p_2]} \frac{\partial^{\,d_1+d_2}\big(f_1^{p_1} f_2^{p_2}\big)}{\partial x_1^{d_1}\partial x_2^{d_2}} \\
&= \frac{\partial f_i}{\partial t} + \alpha_{i,[0,0],[1,0]} f_1 + \alpha_{i,[0,0],[0,1]} f_2 + \alpha_{i,[0,0],[1,1]} f_1 f_2 + \ldots + \alpha_{i,[0,0],[0,3]} f_2^3 \\
&\quad + \alpha_{i,[1,0],[1,0]} \frac{\partial f_1}{\partial x_1} + \alpha_{i,[1,0],[0,1]} \frac{\partial f_2}{\partial x_1} + \alpha_{i,[1,0],[1,1]} \frac{\partial\big(f_1 f_2\big)}{\partial x_1} + \ldots + \alpha_{i,[1,0],[0,3]} \frac{\partial\big(f_2^3\big)}{\partial x_1} \\
&\quad + \quad \ldots \\
&\quad + \alpha_{i,[0,2],[1,0]} \frac{\partial^2 f_1}{\partial x_2^2} + \alpha_{i,[0,2],[0,1]} \frac{\partial^2 f_2}{\partial x_2^2} + \alpha_{i,[0,2],[1,1]} \frac{\partial^2\big(f_1 f_2\big)}{\partial x_2^2} + \ldots + \alpha_{i,[0,2],[0,3]} \frac{\partial^2\big(f_2^3\big)}{\partial x_2^2}
\end{aligned} \tag{B.7}$$

where $i \in \{1,2\}$. As we can observe, we have $|\mathcal{D}_{\boldsymbol{d}}| \times |\mathcal{D}_{\boldsymbol{p}}| = 5 \times 9 = 45$ terms with unknown coefficients $\boldsymbol{\alpha}_i = [\alpha_{i,\boldsymbol{d},\boldsymbol{p}}]_{d\in\mathcal{D}_{\boldsymbol{d}},p\in\mathcal{D}_{\boldsymbol{p}}}$ for the i-th PDE, i.e. a total of 90 terms for the considered system, that we aim to find using the proposed adjoint method.
The cost functional in this case is simply

$$\mathcal{C} = \sum_{i=1}^{2} \left(\sum_{j,k} \big(f_i^*(\boldsymbol{x}^{(k)}, t^{(j+1)}) - f_i(\boldsymbol{x}^{(k)}, t^{(j+1)})\big)^2 + \frac{1}{\Delta\boldsymbol{x}\Delta t} \int \lambda_i(\boldsymbol{x},t) \mathcal{L}_i[\boldsymbol{f}(\boldsymbol{x},t)] d\boldsymbol{x} dt \right) + \epsilon_0 ||\boldsymbol{\alpha}||_2^2 \,. \tag{B.8}$$

The corresponding adjoint equation is given by

$$\begin{aligned}
\frac{\partial \lambda_i}{\partial t} &= \sum_{\boldsymbol{d},\boldsymbol{p}} (-1)^{|\boldsymbol{d}|} \alpha_{i,\boldsymbol{d},\boldsymbol{p}} \nabla_{f_i}[\boldsymbol{f^p}] \nabla_{\boldsymbol{x}}^{(\boldsymbol{d})}[\lambda_i] \\
&= \sum_{[d_1,d_2]\in\mathcal{D}_{\boldsymbol{d}}} \sum_{[p_1,p_2]\in\mathcal{D}_{\boldsymbol{p}}} (-1)^{d_1+d_2} \alpha_{i,[d_1,d_2],[p_1,p_2]} \frac{\partial\big(f_1^{p_1} f_2^{p_2}\big)}{\partial f_i} \frac{\partial^{\,d_1+d_2}\lambda_i}{\partial x_1^{d_1}\partial x_2^{d_2}}
\end{aligned} \tag{B.9}$$

and $\lambda_i(\boldsymbol{x}^{(k)}, t^{(j+1)}) = 2(f_i^*(\boldsymbol{x}^{(k)}, t^{(j+1)}) - f_i(\boldsymbol{x}^{(k)}, t^{(j+1)}))$ for all j,k and where $i \in \{1,2\}$. Assume, without loss of generality, that $i = 1$. Then, we can write

$$\begin{aligned}
\frac{\partial \lambda_1}{\partial t} &= + \alpha_{1,[0,0],[1,0]} \lambda_1 + \alpha_{1,[0,0],[1,1]} f_2 \lambda_1 + \alpha_{1,[0,0],[2,0]} \big(2f_1\big) \lambda_1 + \ldots + \alpha_{1,[0,0],[3,0]} \big(3f_1^2\big) \lambda_1 \\
&\quad - \alpha_{1,[1,0],[1,0]} \frac{\partial \lambda_1}{\partial x_1} - \alpha_{1,[1,0],[1,1]} f_2 \frac{\partial \lambda_1}{\partial x_1} - \alpha_{1,[1,0],[2,0]} \big(2f_1\big) \frac{\partial \lambda_1}{\partial x_1} - \ldots - \alpha_{1,[1,0],[3,0]} \big(3f_1^2\big) \frac{\partial \lambda_1}{\partial x_1} \\
&\quad + \quad \ldots \\
&\quad + \alpha_{1,[0,2],[1,0]} \frac{\partial^2 \lambda_1}{\partial x_2^2} + \alpha_{1,[0,2],[1,1]} f_2 \frac{\partial^2 \lambda_1}{\partial x_2^2} + \alpha_{1,[0,2],[2,0]} \big(2f_1\big) \frac{\partial^2 \lambda_1}{\partial x_2^2} + \ldots + \alpha_{1,[0,2],[3,0]} \big(3f_1^2\big) \frac{\partial^2 \lambda_1}{\partial x_2^2}
\end{aligned} \tag{B.10}$$

and $\lambda_1(\boldsymbol{x}^{(k)}, t^{(j+1)}) = 2(f_1^*(\boldsymbol{x}^{(k)}, t^{(j+1)}) - f_1(\boldsymbol{x}^{(k)}, t^{(j+1)}))$ for all j, k. We can follow the same procedure for $i = 2$. The parameters $\boldsymbol{\alpha}_i$ are then found using the gradient descent method with update rule

$$\alpha_{i,\boldsymbol{d},\boldsymbol{p}} \leftarrow \alpha_{i,\boldsymbol{d},\boldsymbol{p}} - \eta \frac{\partial \mathcal{C}}{\partial \alpha_{i,\boldsymbol{d},\boldsymbol{p}}} \tag{B.11}$$

where

$$\eta = \beta \min(\Delta \boldsymbol{x})^{|\boldsymbol{d}|-d_{\max}} \quad \text{and} \quad \frac{\partial \mathcal{C}}{\partial \alpha_{i,\boldsymbol{d},\boldsymbol{p}}} = (-1)^{|\boldsymbol{d}|} \frac{1}{\Delta \boldsymbol{x} \Delta t} \int \boldsymbol{f}^{\boldsymbol{p}} \nabla_{\boldsymbol{x}}^{(\boldsymbol{d})} [\lambda_i] d\boldsymbol{x} dt + 2\epsilon_0 \alpha_{i,\boldsymbol{d},\boldsymbol{p}} \tag{B.12}$$

with $\Delta \boldsymbol{x} = \Delta x_1 \Delta x_2$, leading to the update rule for each coefficient, for example

$$\alpha_{i,[0,0],[1,0]} \leftarrow \alpha_{i,[0,0],[1,0]} - \frac{\beta}{\min(\Delta \boldsymbol{x})^2} \frac{1}{\Delta \boldsymbol{x} \Delta t} \int f_1 \lambda_i d\boldsymbol{x} dt - 2\beta\epsilon_0 \alpha_{i,[0,0],[1,0]}$$

$$\alpha_{i,[1,0],[1,0]} \leftarrow \alpha_{i,[1,0],[1,0]} - \frac{\beta}{\min(\Delta \boldsymbol{x})} \frac{1}{\Delta \boldsymbol{x} \Delta t} \int f_1 \frac{\partial \lambda_i}{\partial x_1} d\boldsymbol{x} dt - 2\beta\epsilon_0 \alpha_{i,[1,0],[1,0]}$$

Appendix C

Supplementary Material for Chapter 4

C.1 Maximum entropy distribution function

The maximum entropy distribution (MED) function finds the least-biased closure for the moment problem. Given N_b realizable moments $\boldsymbol{\mu} \in \mathbb{R}^{N_b}$ associated with polynomial basis functions $\boldsymbol{H}$ of the unknown distribution function, MED is obtained by minimizing the Shannon entropy with constraint moments using the method of Lagrange multipliers as [114]

$$C[\mathcal{F}(\boldsymbol{x})] := \int \mathcal{F}(\boldsymbol{x})(\log\left(\mathcal{F}(\boldsymbol{x})\right) - 1)d\boldsymbol{x} - \sum_{i=1}^{N_b} \lambda_i \left(\int H_i(\boldsymbol{x})\mathcal{F}(\boldsymbol{x})d\boldsymbol{x} - \mu_i(\boldsymbol{x}) \right) . \tag{C.1}$$

By taking the variational derivative of functional in Eq. (C.1), the extremum is found as

$$\mathcal{F}(\boldsymbol{x}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{N_b} \lambda_i H_i(\boldsymbol{x}) \right), \qquad \text{where} \quad Z = \int \exp\left(\sum_{i=1}^{N_b} \lambda_i H_i(\boldsymbol{x}) \right) d\boldsymbol{x}, \tag{C.2}$$

which is referred to as the maximum entropy distribution function. The Lagrange multipliers $\boldsymbol{\lambda}$ appearing in Eq. (C.2) can be found using the Newton-Raphson approach. As formulated in [216], the unconstrained dual formulation $D(\boldsymbol{\lambda})$ provides us with the gradient $\boldsymbol{g} = \nabla D(\boldsymbol{\lambda})$ and Hessian $\boldsymbol{L}(\boldsymbol{\lambda}) = \nabla^2 D(\boldsymbol{\lambda})$ as

$$g_i = \mu_i - \frac{1}{Z} \int H_i \exp\left(\sum_{k=1}^{N_b} \lambda_k H_k \right) d\boldsymbol{x} \quad \text{for } i = 1, ..., N_b \tag{C.3}$$

$$\text{and} \quad L_{i,j} = -\frac{1}{Z} \int H_i H_j \exp\left(\sum_{k=1}^{N_b} \lambda_k H_k \right) d\boldsymbol{x} \quad \text{for } i, j = 1, ..., N_b \ . \tag{C.4}$$

Once the gradient and Hessian are computed, the Lagrange multipliers $\boldsymbol{\lambda}$ can be updated via

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \boldsymbol{L}^{-1}(\boldsymbol{\lambda})\boldsymbol{g}(\boldsymbol{\lambda}) \tag{C.5}$$

as detailed in Algorithm 8.

Algorithm 8: Newton's method for finding Lagrange multipliers of MED given moments $\boldsymbol{\mu}$ for a given tolerance ϵ.

Input: $\boldsymbol{\mu}$, $\boldsymbol{\lambda}_0$
Initialize $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}_0$;
Compute gradient $\boldsymbol{g}$ and Hessian $\boldsymbol{L}$, i.e. Eqs. (C.3)-(C.4);
while $||\boldsymbol{g}|| > \epsilon$ **do**
    Update $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \boldsymbol{L}^{-1}\boldsymbol{g}$;
    Update gradient $\boldsymbol{g}$ and Hessian $\boldsymbol{L}$ with the new $\boldsymbol{\lambda}$ via numerical integration of Eqs. (C.3)-(C.4);
end
Return $\boldsymbol{\lambda}$;

C.2 Maximum cross-entropy distribution function

The maximum cross-entropy distribution function (MxED) finds the least-biased closure for the moment problem given N_b realizable moments $\boldsymbol{\mu} \in \mathbb{R}^{N_b}$ associated with polynomial basis functions $\boldsymbol{H}$ of the unknown distribution function along with the prior $\mathcal{F}^{\text{Prior}}$ as the input. In other words, in addition to the moments of the target distribution, in the MxED method we also have access to a prior distribution $\mathcal{F}^{\text{Prior}}$ as well as N samples of the former, i.e. $\{\boldsymbol{X}_j^{\text{prior}}\}_{j=1}^N \sim \mathcal{F}^{\text{Prior}}$. MxED is obtained by minimizing the Shannon cross-entropy from the prior with constraint on moments using the method of Lagrange multipliers via the functional

$$C[\mathcal{F}(\boldsymbol{x})] := \int \mathcal{F}(\boldsymbol{x}) \log\left(\frac{\mathcal{F}(\boldsymbol{x})}{\mathcal{F}^{\text{Prior}}(\boldsymbol{x})}\right) d\boldsymbol{x} + \sum_{i=1}^{N_b} \lambda_i \left(\int H_i(\boldsymbol{x})\mathcal{F}(\boldsymbol{x})d\boldsymbol{x} - \mu_i(\boldsymbol{x})\right) . \tag{C.6}$$

By taking the variational derivative of functional in Eq. (C.6), the extremum is found to be

$$\mathcal{F}(\boldsymbol{x}) = \frac{1}{Z}\mathcal{F}^{\text{Prior}}(\boldsymbol{x}) \exp\left(\sum_{i=1}^{N_b} \lambda_i H_i(\boldsymbol{x})\right), \qquad \text{where } Z = \int \mathcal{F}^{\text{Prior}}(\boldsymbol{x}) \exp\left(\sum_{i=1}^{N_b} \lambda_i H_i(\boldsymbol{x})\right) d\boldsymbol{x}. \tag{C.7}$$

Similar to the maximum entropy distribution function, the Lagrange multipliers $\boldsymbol{\lambda}$ appearing in Eq. (C.7) can be found by following the Newton-Raphson approach. As formulated in [216], the unconstrained dual formulation $D(\boldsymbol{\lambda})$ provides us with the gradient $\boldsymbol{g} = \nabla D(\boldsymbol{\lambda})$ and Hessian $\boldsymbol{L}(\boldsymbol{\lambda}) = \nabla^2 D(\boldsymbol{\lambda})$ as

$$g_i = \mu_i - \frac{1}{Z}\int \mathcal{F}^{\text{Prior}} H_i \exp\left(\sum_{k=1}^{N_b} \lambda_k H_k\right) d\boldsymbol{x} \quad \text{for } i = 1, ..., N_b \tag{C.8}$$

$$\text{and} \quad L_{i,j} = -\frac{1}{Z}\int \mathcal{F}^{\text{Prior}} H_i H_j \exp\left(\sum_{k=1}^{N_b} \lambda_k H_k\right) d\boldsymbol{x} \quad \text{for } i, j = 1, ..., N_b \ . \tag{C.9}$$

Once the gradient and Hessian are computed, the Lagrange multipliers $\boldsymbol{\lambda}$ can be updated via

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \boldsymbol{L}^{-1}(\boldsymbol{\lambda})\boldsymbol{g}(\boldsymbol{\lambda}) \ . \tag{C.10}$$

Since we have access to the samples of $\boldsymbol{X}^{\text{Prior}} \sim \mathcal{F}^{\text{Prior}}$, we use the given samples to compute the gradient and Hessian, i.e.

$$g_i \approx \mu_i - \left\langle H_i\big(\boldsymbol{X}^{\text{Prior}}\big) \right\rangle_W \quad \text{for } i = 1, ..., N_b \tag{C.11}$$

$$L_{i,j} \approx -\left\langle H_i\big(\boldsymbol{X}^{\text{Prior}}\big) H_j\big(\boldsymbol{X}^{\text{Prior}}\big) \right\rangle_W \quad \text{for } i, j = 1, ..., N_b \ , \tag{C.12}$$

where $W(\boldsymbol{X}^{\text{Prior}}) = \exp\left(\sum_{k=1}^{N_b} \lambda_k H_k\big(\boldsymbol{X}^{\text{Prior}}\big)\right)$ denotes weights for calculating moments using importance sampling, i.e. $\langle \phi(\boldsymbol{X}) \rangle_W := \sum_{j=1}^N \phi(\boldsymbol{X}_j) W(\boldsymbol{X}_j) / \sum_{j=1}^N W(\boldsymbol{X}_j)$. More details can be found below (Algorithm 9).

Algorithm 9: Newton's method for finding Lagrange multipliers of MxED given moments $\boldsymbol{\mu}$ and samples of prior $\boldsymbol{X}^{\text{Prior}} \sim \mathcal{F}^{\text{Prior}}$ for a given tolerance ϵ.

Input: $\boldsymbol{\mu}$, $\boldsymbol{X}^{\text{Prior}} \sim \mathcal{F}^{\text{Prior}}$
Initialize $\boldsymbol{\lambda} \leftarrow \mathbf{0}$;
Compute gradient $\boldsymbol{g}$ and Hessian $\boldsymbol{L}$, i.e. Eqs. (C.11)-(C.12);
while $||\boldsymbol{g}|| > \epsilon$ **do**
    Update $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \boldsymbol{L}^{-1}\boldsymbol{g}$;
    Update gradient $\boldsymbol{g}$ and Hessian $\boldsymbol{L}$ with the new $\boldsymbol{\lambda}$ using samples, i.e. Eq. (C.11)-(C.12);
end
Return $\boldsymbol{\lambda}$;

C.3 Ablation studies

In this section, we report on the results of an ablation study that shows the effects of each component in the proposed MESSY solution. The four components of MESSY Estimation are: i) the symbolic component, ii) the multi-level component, iii) the orthogonalization, and iv) the cross-entropy step. As we have already compared MESSY-S and MESSY-P (MESSY without the symbolic regression component) throughout Chapter 4, we conduct an ablation study on the remaining components, i.e. the multi-level, the orthogonalization, and the cross-entropy step.

C.3.1 Multi-level component of MESSY

First, in order to show the effect of the multi-level part of the MESSY algorithm, let us consider the bi-modal density near the limit of realizability, see Section 4.8.2. In particular, let us consider the MESSY-P estimate with 4th order polynomials as basis functions. In Fig. C.1, we show that by disabling the multi-level step, the estimated density covers only one of the peaks and entirely misses the other one. We believe that this is due to the high conditionality of the matrix $\boldsymbol{L}^{\text{ME}}$. However, MESSY-P with a multi-level step accurately recovers both peaks of the underlying density.

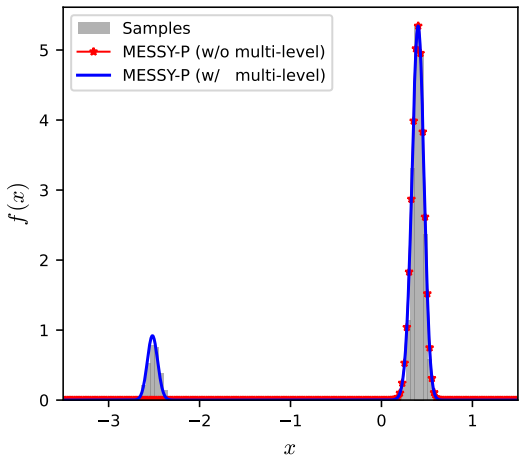


Figure C.1: Ablation study on the multi-level component of MESSY-P estimation for the case of target distribution near the limit of realizability.

C.3.2 Orthogonalization

Another important element of MESSY process is the orthogonalization of basis functions given samples, i.e. the modified Gram-Schmidt Algorithm 4. As an example, here we consider the MESSY-P estimate of target bimodal distribution function presented in Section 4.8.1 as a test case and report the condition number of the matrix $\boldsymbol{L}^{\text{ME}}$ that needs to be inverted to compute the Lagrange multipliers in Eq. (4.18). As shown in Fig. C.2, the condition number of $\boldsymbol{L}^{\text{ME}}$ increases with the polynomial order, making the linear system in

Eq. (4.18) stiff and difficult to solve. However, by orthogonalizing the basis functions, we can maintain a low condition number for the outcome $\boldsymbol{L}^{\mathrm{ME}}$ which makes the resulting linear system tractable.

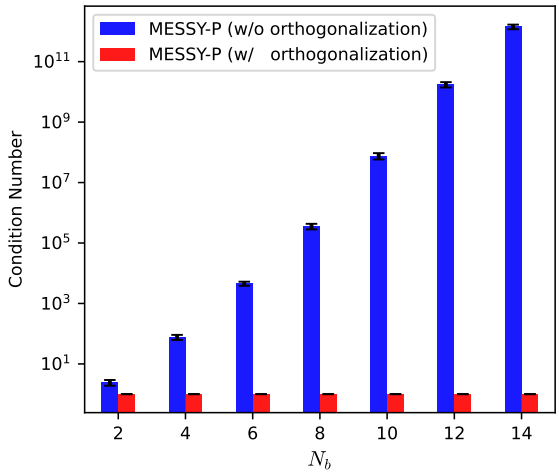


Figure C.2: Ablation study on the orthogonalization component of MESSY-P algorithm for the case of bimodal distribution. Relation between condition number of matrix $\boldsymbol{L}^{\mathrm{ME}}$ and the order of considered polynomial with and without the orthogonalization step.

C.3.3 Cross-entropy step

The final step in the MESSY-P procedure is the cross-entropy. The goal of this step is to reduce the bias that may have been introduced by the multi-level component of the algorithm. In the MESSY Algorithm 6, we take the outcome of the multi-level step as the prior, and correct Lagrange multipliers to match the target moments. In Fig. C.3, we compare the MESSY-P density estimate of the discontinuous density from the test case presented in Section 4.8.3 with and without cross-entropy step. As expected, we observe that the cross-entropy step is essential in recovering distributions with discontinuities. This is due to the fact that the solution obtained via gradient flow assumes a smooth target density. Hence, it fails to find the Lagrange multipliers for the target discontinuous pdf. However, the cross entropy step only assumes that the target distribution is integrable to the extent that the given moments exist. This weaker condition is essential to recovering discontinuous distributions, e.g. the one showed in Fig. C.3.

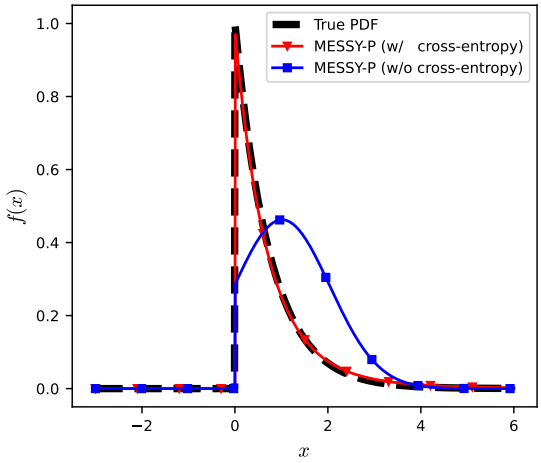
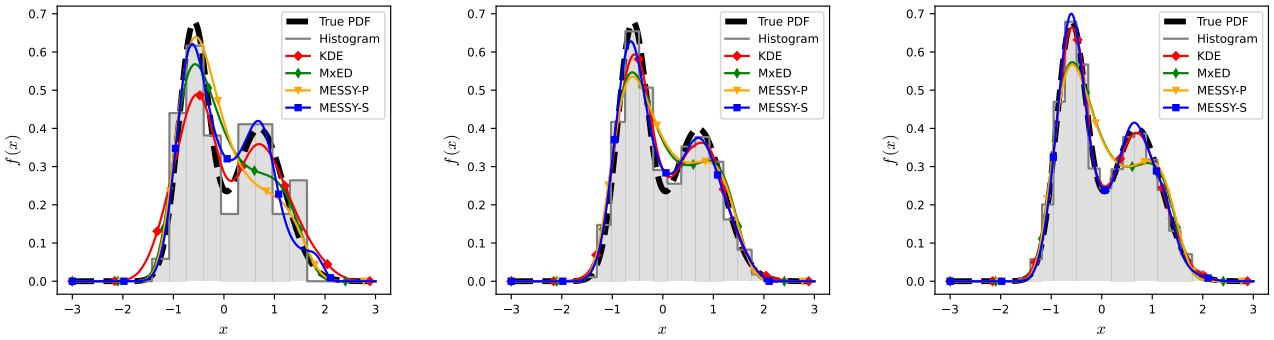


Figure C.3: Ablation study on the cross-entropy step of MESSY-P estimation of a discontinuous density.

C.4 Histogram estimate to the bi-modal distribution function

Figure C.4 provides a comparison between a histogram, KDE, MESSY-P and MESSY-S estimates of the underlying bi-modal distribution function considered in Section 4.8.1 given $N = 100, 1000, 10000$ samples. Clearly, MED estimates provide a better solution than non-parametric ones, i.e. histogram and KDE, when not many samples are available. However, as more samples are considered, the non-parametric estimators become more accurate than MESSY (or any parametric estimator) with fixed number of basis functions.



(a) 100 samples (b) 1,000 samples (c) 10,000 samples

Figure C.4: Density estimation using KDE, MxED, MESSY-P, MESSY-S and histogram given (a) 100, (b) 1,000, and (c) 10,000 samples.

C.5 Solution found by MESSY for the considered test cases

Table C.1: Density expressions recovered by our MESSY estimation method for several distributions.

Example		**Expression**
Bimodal 1D	MESSY-P	$\hat{f}(x) = 0.288e^{-0.017x^{10}+0.106x^9-0.084x^8-0.659x^7+1.209x^6+1.179x^5-3.722x^4+0.075x^3+2.693x^2-0.612x}$
	MESSY-S	$\hat{f}(x) = 0.993e^{-1.85x^2-1.162x\cos(1.5x)+0.232x-0.652\cos(x)-0.424\cos(2x)-0.591\cos(3.5x)+0.47\cos(\cos(3.5x))}$
Limit of Realizability	MESSY-P	$\hat{f}(x) = 1.591 \cdot 10^{-6}e^{-12.876x^4-56.46x^3-38.072x^2+62.617x} + 5.282 \cdot 10^{-27}e^{-7.969x^4-28.862x^3-4.342x^2+20.938x}$
	MESSY-S	$\hat{f}(x) = 4.134 \cdot 10^{81}e^{-21.893x^2\sin(x)+0.025x^2+117.267x\cos(x)+0.861x+395.584\sin^2(x)-57.393\sin(x)+200.421\cos(x)-744.874\cos(\cos(x))}$
Discontinuous	MESSY-P	$\hat{f}(x) = \begin{cases} 1.096\, e^{0.086x^2-1.298x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
	MESSY-S	$\hat{f}(x) = \begin{cases} 0.293\, e^{-0.145x^2+0.018x+0.251\cos(x)\cos(1.5x)+0.713\cos(x)+0.09\cos(1.5x)\cos(3x)+0.076\cos(3.5x)} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
Gaussian 2D	MESSY-P	$\hat{f}(x_1, x_2) = 0.18e^{-0.606x_1^2+0.572x_1x_2+0.029x_1-0.663x_2^2-0.075x_2}$
	MESSY-S	$\hat{f}(x_1, x_2) = 0.182e^{-0.648x_1^2+0.598x_1x_2-0.018x_1-0.644x_2^2-0.044x_2}$
Gamma-exp	MESSY-P	$\hat{f}(x_1, x_2) = \begin{cases} 0.477e^{0.225x_1^2-0.04x_1x_2-2.264x_1-0.376x_2^2+0.913x_2} & x_1 \geq 0,\, x_2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$
	MESSY-S	$\hat{f}(x_1, x_2) = \begin{cases} 0.017e^{-0.13x_1^2+0.016x_1x_2+0.15x_1+0.017x_2^2\cos(2.5\cos(2x_2))-0.37x_2^2+0.858x_2+2.654\cos(x_1)+0.334\cos(3.5x_1)-0.437\cos(2.5x_2)} & x_1 \geq 0,\, x_2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$

Appendix D

Supplementary Material for Chapter 5

D.1 Invertible symbolic expressions recovered by ISR for the considered distributions

Table D.1: Invertible symbolic expressions recovered by our ISR method for density estimation of several distributions (see Figure 5.4 in Section 5.4.1). Here, MoGs denotes "Mixture of Gaussians."

Example	Expression
Gaussian	$\mathbf{u} = \mathbf{x}$, i.e. $u_1 = x_1$, $u_2 = x_2$ $s_1(u_2) = 1.16$ $t_1(u_2) = 0$ $v_1 = u_1 \cdot \exp\left(s_1(u_2)\right) + t_1(u_2)$ $v_2 = u_2$ $s_2(v_1) = 1.14$ $t_2(v_1) = -9.39$ $o_1 = v_1$ $o_2 = v_2 \cdot \exp\left(s_2(v_1)\right) + t_2(v_1)$ $\mathbf{z} = \mathbf{o}$, i.e. $z_1 = o_1$, $z_2 = o_2$
Banana	$\mathbf{u} = \mathbf{x}$, i.e. $u_1 = x_1$, $u_2 = x_2$ $s_1(u_2) = 0.52 \sin\left(1.86\, u_2\right) + 0.084 \sin\left(5.5 \sin\left(1.86\, u_2\right) + 2.55\right) + 1.7$ $t_1(u_2) = 0.74 - 0.12 \sin\left(3.65 \sin\left(2.45\, u_2\right) - 0.82\right)$ $v_1 = u_1 \cdot \exp\left(s_1(u_2)\right) + t_1(u_2)$ $v_2 = u_2$ $s_2(v_1) = 1.72\left(0.025 - 0.47 \sin\left(0.33\, v_1\right)\right)\left(0.23 \sin\left(0.33\, v_1\right) - 0.29\right) + 2.24$ $t_2(v_1) = -3.74\left(0.022 \sin\left(0.62\, v_1\right) + 0.035 \sin\left(0.63\, v_1\right) - 0.76\right)\left(0.45 \sin\left(0.62\, v_1\right) + 0.7 \sin\left(0.63\, v_1\right) + 0.38\right) + 0.027$ $o_1 = v_1$ $o_2 = v_2 \cdot \exp\left(s_2(v_1)\right) + t_2(v_1)$ $\mathbf{z} = \mathbf{o}$, i.e. $z_1 = o_1$, $z_2 = o_2$
Ring	$\mathbf{u} = \mathbf{x}$, i.e. $u_1 = x_1$, $u_2 = x_2$ $s_1(u_2) = -3.14\left(-0.15\, u_2^2 - 0.26 \sin\left(1.26\, u_2\right) + 0.094 \sin\left(3.25\, u_2\right) - 0.3\right)\left(0.098\, u_2^2 + 0.39 \sin\left(1.26\, u_2\right) + 0.014 \sin\left(3.25\, u_2\right) + 0.16\right)$ $\quad + 0.09 \sin\left(0.17\, u_2^2 + 0.69 \sin\left(1.26\, u_2\right) + 0.76 \sin\left(3.25\, u_2\right) + 0.25\right) - 0.30 \sin\left(0.94\, u_2^2 + 2.35 \sin\left(1.26\, u_2\right) - 1.31 \sin\left(3.25\, u_2\right) + 1.57\right) + 0.053$ $t_1(u_2) = -0.012$ $v_1 = u_1 \cdot \exp\left(s_1(u_2)\right) + t_1(u_2)$ $v_2 = u_2$ $s_2(v_1) = 0.037 \sin\left(0.22 \sin\left(0.22\, v_1\right) + 0.22 \sin\left(0.22\, v_1\right) - 0.27\right) + 0.052 \sin\left(0.23 \sin\left(0.22\, v_1\right) + 0.23 \sin\left(0.22\, v_1\right) - 0.39\right) - 0.11$ $t_2(v_1) = 0.13 \sin\left(0.13 \sin\left(0.37\, v_1\right) + 0.13 \sin\left(0.59\, v_1\right) - 1.16\right) + 0.65 \sin\left(0.021\, v_1 + 1.34 \sin\left(0.37\, v_1\right) + 2.29 \sin\left(0.59\, v_1\right) + 1.44\right) + 0.33$ $o_1 = v_1$ $o_2 = v_2 \cdot \exp\left(s_2(v_1)\right) + t_2(v_1)$ $\mathbf{u} = \mathbf{o}$, i.e. $u_1 = o_1$, $u_2 = o_2$ $s_1(u_2) = -3.65\left(-0.47 \sin\left(1.38\, u_2\right) - 0.026 \sin\left(1.93\, u_2\right) - 0.1\right)\left(-0.014 \sin\left(1.38\, u_2\right) - 0.39 \sin\left(1.93\, u_2\right) - 0.35\right) - 0.44 \sin\left(1.74 \sin\left(1.38\, u_2\right) + 1.71 \sin\left(1.93\, u_2\right) + 5.55\right)$ $\quad - 0.46 \sin\left(3.31 \sin\left(1.38\, u_2\right) + 0.44 \sin\left(1.93\, u_2\right) - 0.7\right) + 0.38$ $t_1(u_2) = 0.11 \sin\left(0.85 \sin\left(1.38\, u_2\right) + 0.86 \sin\left(1.38\, u_2\right)\right) + 0.12 \sin\left(0.87 \sin\left(1.38\, u_2\right) + 0.87 \sin\left(1.38\, u_2\right)\right) + 0.053$ $v_1 = u_1 \cdot \exp\left(s_1(u_2)\right) + t_1(u_2)$ $v_2 = u_2$ $s_2(v_1) = 0.25\, v_1^2 - 0.0071 \sin\left(1.67\, v_1\right) - 0.1 \sin\left(5.29\, v_1\right) - 0.63 \sin\left(-1.075\, v_1^2 + 0.55 \sin\left(1.67\, v_1\right) + 0.77 \sin\left(5.29\, v_1\right)\right)$ $\quad + 0.46 \sin\left(1.89\, v_1^2 - 0.27 \sin\left(1.67\, v_1\right) - 1.69 \sin\left(5.29\, v_1\right) + 0.9\right) + 1.22$ $t_2(v_1) = 0.62 \sin\left(1.56 \sin\left(0.43\, v_1\right) + 1.57 \sin\left(0.43\, v_1\right) - 1.68\right) + 0.61 \sin\left(1.56 \sin\left(0.43\, v_1\right) + 1.57 \sin\left(0.43\, v_1\right) - 1.68\right) - 0.48$ $o_1 = v_1$ $o_2 = v_2 \cdot \exp\left(s_2(v_1)\right) + t_2(v_1)$ $\mathbf{z} = \mathbf{o}$, i.e. $z_1 = o_1$, $z_2 = o_2$
MoG	$\mathbf{u} = \mathbf{x}$, i.e. $u_1 = x_1$, $u_2 = x_2$ $s_1(u_2) = -0.039\left(-0.032 \sin\left(1.44\, u_2\right) - \sin\left(1.48\, u_2\right) + 0.94\right)^2 - 3.03\left(0.18 \sin\left(1.44\, u_2\right) + 0.14 \sin\left(1.48\, u_2\right) - 0.63\right)\left(0.26 \sin\left(1.44\, u_2\right) + 0.28 \sin\left(1.48\, u_2\right) - 0.31\right)$ $\quad + 0.047 \sin\left(1.44\, u_2\right) + 0.13 \sin\left(1.48\, u_2\right) - 0.029 \sin\left(0.15 \sin\left(1.44\, u_2\right) + 1.4 \sin\left(1.48\, u_2\right) - 3.47\right) - 0.11 \sin\left(0.45 \sin\left(1.44\, u_2\right) + 1.46 \sin\left(1.48\, u_2\right) + 2.65\right) - 0.17$ $t_1(u_2) = 0.052 - 0.12 \sin\left(1.13 \sin\left(3.14\, u_2\right) + 1.64\right)$ $v_1 = u_1 \cdot \exp\left(s_1(u_2)\right) + t_1(u_2)$ $v_2 = u_2$ $s_2(v_1) = 0.34\left(0.15 \sin\left(1.024\, v_1\right) + 0.13 \sin\left(2.022\, v_1\right)\right) \sin\left(2.022\, v_1\right) + 0.014 \sin^2\left(2.022\, v_1\right) + 0.15 \sin\left(0.98\, v_1 \sin\left(1.024\, v_1\right) + 1.9 \sin\left(2.022\, v_1\right) - 1.41\right)$ $\quad - 0.22 \sin\left(3.38 \sin\left(1.024\, v_1\right) + 1.83 \sin\left(2.022\, v_1\right) + 1.5\right) + 0.39$ $t_2(v_1) = -0.46 \sin\left(0.89\, v_1 + 0.21 \sin\left(1.38\, v_1\right) - 1.56\right) + 0.33 \sin\left(1.23\, v_1 + 1.69 \sin\left(1.38\, v_1\right) - 0.44 \sin\left(1.58\, v_1\right) + 1.75\, v_1\right) + 0.094$ $o_1 = v_1$ $o_2 = v_2 \cdot \exp\left(s_2(v_1)\right) + t_2(v_1)$ $\mathbf{u} = \mathbf{o}$, i.e. $u_1 = o_1$, $u_2 = o_2$ $s_1(u_2) = 3.36\left(-0.36 \sin\left(3.1\, u_2\right) - 0.29\right)\left(0.26 \sin\left(3.1\, u_2\right) + 0.19\right) - 0.45 \sin\left(3.88 \sin\left(3.1\, u_2\right) - 1.83\right) - 0.38$ $t_1(u_2) = 0$ $v_1 = u_1 \cdot \exp\left(s_1(u_2)\right) + t_1(u_2)$ $v_2 = u_2$ $s_2(v_1) = 0.0035\, v_1^2 - 2.88\left(-0.079\, v_1^2 - 0.33\right)\left(0.042\, v_1^2 + 0.4\right) + 2.6\left(-0.34 \sin\left(1.7\, v_1\right) - 0.088 \sin\left(1.71\, v_1\right)\right)\left(-0.053 \sin\left(1.7\, v_1\right) - 0.38 \sin\left(1.71\, v_1\right)\right) + 1.61$ $t_2(v_1) = -0.14$ $o_1 = v_1$ $o_2 = v_2 \cdot \exp\left(s_2(v_1)\right) + t_2(v_1)$ $\mathbf{z} = \mathbf{o}$, i.e. $z_1 = o_1$, $z_2 = o_2$

D.2 Details of network architectures

In our experiments, we train all models using Adam optimizer with a dynamic learning rate decaying from 10^{-2} to 10^{-4}. In addition, for INN, all (hidden) neurons of the subnetworks are followed by Leaky ReLU activations, while for ISR, each neuron in the hidden layers is followed by an activation function from the following library:

$$\left\{1, \mathrm{id}, \bullet^2(\times 4), \sin(2\pi\bullet), , \bullet_1 \times \bullet_2\right\}$$

where 1 represents the constant function, "id" is the identity operator, denotes the sigmoid function, and $\bullet$ denotes placeholder operands, e.g. $\bullet^2$ corresponds to the square operator. Also, $\bullet_1 \times \bullet_2$ denotes the multiplication operator, and each activation function may be duplicated within each layer.

The architecture details are provided below.

Density estimation via normalizing flow. In Section 5.4.1, we train INN and ISR using a batch size of 64. For the "Gaussian" and "Banana" distributions, we adopt 1 affine coupling block with 2 fully connected (hidden) layers per subnetwork. For the "Ring" and "Mixture of Gaussians (MoGs)" distributions, we use 2 invertible blocks with 2 fully connected layers for each subnetwork.

Inverse Kinematics. In Section 5.4.2, we train all models using a batch size of 100. For all models, we adopt 6 reversible blocks with 3 fully connected layers per subnetwork.

Geoacoustic Inversion. In Section 5.4.3, we train all models using a batch size of 200. For all models, we adopt 5 invertible blocks with 4 fully connected layers for each subnetwork.

D.3 Quantitative Evaluation – Inverse Kinematics

Here, we quantitatively evaluate the quality of the estimated posteriors by the different models considered in the inverse kinematics experiment. To ensure a fair comparison among all methods, we use the same training data and train all models for the same number of epochs, and using identical batches and architectures (as provided in the previous section).

As suggested in [196], we evaluate the correctness of the estimated posteriors using two metrics (see Table D.2). First, we use the Maximum Mean Discrepancy (MMD) introduced by [200], which computes the *posterior mismatch* between the distribution $\hat{p}(\mathbf{x} \mid \mathbf{y}^*)$ produced by a model and a ground truth estimate $p_{\text{gt}}(\mathbf{x} \mid \mathbf{y}^*)$, which in this case is obtained via rejection sampling (see Section 5.4.2), i.e.

$$\text{Err}_{\text{post}} = \text{MMD}\big(\hat{p}(\mathbf{x} \mid \mathbf{y}^*), p_{\text{gt}}(\mathbf{x} \mid \mathbf{y}^*)\big) \tag{D.1}$$

Second, we measure the *re-simulation error*, which applies the true forward process f in Eq. (5.20) to the generated samples $\mathbf{x}$ and computes the mean squared distance to the target $\mathbf{y}^*$, i.e.

$$\text{Err}_{\text{resim}} = \mathbb{E}_{\mathbf{x} \sim \hat{p}(\mathbf{x} \mid \mathbf{y}^*)} \left[||f(\mathbf{x}) - \mathbf{y}^*||_2^2 \right] \tag{D.2}$$

Table D.2: Quantitative results for the inverse kinematics benchmark experiment.

Method	Err_{post}	$\text{Err}_{\text{resim}}$
INN	0.0259	0.0163
cINN	0.0162	0.0087
ISR	0.0286	0.0196
cISR	0.0221	0.0134

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[4] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[5] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *International conference on machine learning*, Pmlr, 2021, pp. 8821–8831.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[7] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science advances*, vol. 3, no. 4, e1602614, 2017.

[8] T. Tohme, D. Liu, and K. Youcef-Toumi, "GSR: A generalized symbolic regression approach," *Transactions on Machine Learning Research*, 2023, ISSN: 2835-8856. [Online]. Available: https://openreview.net/forum?id=lheUXtDNvP.

[9] M. Sadr, T. Tohme, and K. Youcef-Toumi, "Data-driven discovery of PDEs via the adjoint method," *arXiv preprint arXiv:2401.17177*, 2024.

[10] T. Tohme, M. Sadr, K. Youcef-Toumi, and N. Hadjiconstantinou, "MESSY Estimation: Maximum-entropy based stochastic and symbolic density estimation," *Transactions on Machine Learning Research*, 2024, ISSN: 2835-8856. [Online]. Available: https://openreview.net/forum?id=Y2ru0LuQeS.

[11] T. Tohme, M. J. Khojasteh, M. Sadr, F. Meyer, and K. Youcef-Toumi, "ISR: Invertible symbolic regression," *arXiv preprint arXiv:2405.06848*, 2024.

[12] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *science*, vol. 324, no. 5923, pp. 81–85, 2009.

[13] M. Quade, M. Abel, K. Shafi, R. K. Niven, and B. R. Noack, "Prediction of dynamical systems by symbolic regression," *Physical Review E*, vol. 94, no. 1, p. 012 214, 2016.

[14] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[15] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Inferring biological networks by sparse identification of nonlinear dynamics," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 2, no. 1, pp. 52–63, 2016.

[16] K. Kaheman, J. N. Kutz, and S. L. Brunton, "Sindy-pi: A robust algorithm for parallel implicit sparse identification of nonlinear dynamics," *Proceedings of the Royal Society A*, vol. 476, no. 2242, p. 20 200 279, 2020.

[17] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, "Discovering symbolic models from deep learning with inductive biases," *arXiv preprint arXiv:2006.11287*, 2020.

[18] C. Wild and G. Seber, *Nonlinear regression*. New York: Wiley, 1989.

[19] A. W. F. Edwards, *Likelihood*. CUP Archive, 1984.

[20] Y. Pawitan, *In all likelihood: statistical modelling and inference using likelihood*. Oxford University Press, 2001.

[21] T. Tohme, K. Vanslette, and K. Youcef-Toumi, "Reliable neural networks for regression uncertainty estimation," *Reliability Engineering & System Safety*, vol. 229, p. 108 811, 2023.

[22] P. M. Lee, *Bayesian statistics*. Arnold Publication, 1997.

[23] T. Leonard and J. S. Hsu, *Bayesian methods: an analysis for statisticians and interdisciplinary researchers*. Cambridge University Press, 2001, vol. 5.

[24] T. Tohme, "The Bayesian validation metric: A framework for probabilistic model calibration and validation," Ph.D. dissertation, Massachusetts Institute of Technology, 2020.

[25] T. Tohme, K. Vanslette, and K. Youcef-Toumi, "A generalized Bayesian approach to model calibration," *Reliability Engineering & System Safety*, vol. 204, p. 107 141, 2020.

[26] K. Vanslette, T. Tohme, and K. Youcef-Toumi, "A general model validation and testing tool," *Reliability Engineering & System Safety*, vol. 195, p. 106 684, 2020.

[27] P. Orzechowski, W. La Cava, and J. H. Moore, "Where are we now? a large benchmark study of recent symbolic regression methods," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 1183–1190.

[28] W. La Cava, P. Orzechowski, B. Burlacu, F. O. de França, M. Virgolin, Y. Jin, M. Kommenda, and J. H. Moore, "Contemporary symbolic regression methods and their relative performance," *arXiv preprint arXiv:2107.14351*, 2021.

[29] Q. Lu, J. Ren, and Z. Wang, "Using genetic programming with prior formula knowledge to solve symbolic regression problem," *Computational intelligence and neuroscience*, vol. 2016, 2016.

[30] S.-M. Udrescu and M. Tegmark, "Ai feynman: A physics-inspired method for symbolic regression," *Science Advances*, vol. 6, no. 16, eaay2631, 2020.

[31] B. K. Petersen, M. L. Larma, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim, "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients," in *International Conference on Learning Representations*, 2021.

[32] M. Virgolin and S. P. Pissis, "Symbolic regression is np-hard," *arXiv preprint arXiv:2207.01018*, 2022.

[33] F. O. de França, "A greedy search tree heuristic for symbolic regression," *Information Sciences*, vol. 442, pp. 18–32, 2018.

[34] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[35] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[36] M. Virgolin, T. Alderliesten, and P. A. Bosman, "Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression," in *Proceedings of the genetic and evolutionary computation conference*, 2019, pp. 1084–1092.

[37] Y. Jin, W. Fu, J. Kang, J. Guo, and J. Guo, "Bayesian symbolic regression," *arXiv preprint arXiv:1910.08892*, 2019.

[38] C. Luo, C. Chen, and Z. Jiang, "A divide and conquer method for symbolic regression," *arXiv preprint arXiv:1705.08061*, 2017.

[39] C. Chen, C. Luo, and Z. Jiang, "Fast modeling methods for complex system with separable features," in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, IEEE, vol. 1, 2017, pp. 201–204.

[40] C. Chen, C. Luo, and Z. Jiang, "Block building programming for symbolic regression," *Neurocomputing*, vol. 275, pp. 1973–1980, 2018.

[41] C. Chen, C. Luo, and Z. Jiang, "A multilevel block building algorithm for fast modeling generalized separable systems," *Expert Systems with Applications*, vol. 109, pp. 25–34, 2018.

[42] G. Martius and C. H. Lampert, "Extrapolation and learning equations," *arXiv preprint arXiv:1610.02995*, 2016.

[43] T. N. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. M. Faissol, and B. K. Petersen, "Symbolic regression via neural-guided genetic programming population seeding," *arXiv preprint arXiv:2111.00053*, 2021.

[44] M. Valipour, B. You, M. Panju, and A. Ghodsi, "Symbolicgpt: A generative transformer model for symbolic regression," *arXiv preprint arXiv:2106.14131*, 2021.

[45] S. Razavi and E. R. Gamazon, "Neural-network-directed genetic programmer for discovery of governing equations," *arXiv preprint arXiv:2203.08808*, 2022.

[46] H. Zhang, A. Zhou, H. Qian, and H. Zhang, "Ps-tree: A piecewise symbolic regression tree," *Swarm and Evolutionary Computation*, vol. 71, p. 101 061, 2022.

[47] S. d'Ascoli, P.-A. Kamienny, G. Lample, and F. Charton, "Deep symbolic regression for recurrence prediction," in *International Conference on Machine Learning*, PMLR, 2022, pp. 4520–4536.

[48] P.-A. Kamienny, S. d'Ascoli, G. Lample, and F. Charton, "End-to-end symbolic regression with transformers," *arXiv preprint arXiv:2204.10532*, 2022.

[49] M. Zhang, S. Kim, P. Y. Lu, and M. Soljačić, "Deep learning and symbolic regression for discovering parametric equations," *arXiv preprint arXiv:2207.00529*, 2022.

[50] S. Sahoo, C. Lampert, and G. Martius, "Learning equations for extrapolation and control," in *International Conference on Machine Learning*, PMLR, 2018, pp. 4442–4450.

[51] S.-M. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark, "Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity," *arXiv preprint arXiv:2006.10782*, 2020.

[52] S. Ahn, J. Kim, H. Lee, and J. Shin, "Guiding deep molecular optimization with genetic exploration," *Advances in neural information processing systems*, vol. 33, pp. 12 008–12 021, 2020.

[53] A. R. Al-Roomi and M. E. El-Hawary, "Universal functions originator," *Applied Soft Computing*, vol. 94, p. 106 417, 2020.

[54] S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, and M. Soljačić, "Integration of neural network-based symbolic regression in deep learning for scientific discovery," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[55] M. Kommenda, B. Burlacu, G. Kronberger, and M. Affenzeller, "Parameter identification for symbolic regression using nonlinear least squares," *Genetic Programming and Evolvable Machines*, vol. 21, no. 3, pp. 471–501, 2020.

[56] B. Burlacu, G. Kronberger, and M. Kommenda, "Operon c++: An efficient genetic programming framework for symbolic regression," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '20, Cancún, Mexico: Association for Computing Machinery, 2020, pp. 1562–1570, ISBN: 9781450371278. DOI: 10.1145/3377929.3398099.

[57] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo, "Neural symbolic regression that scales," in *International Conference on Machine Learning*, 2021.

[58] J. A. Nelder and R. W. Wedderburn, "Generalized linear models," *Journal of the Royal Statistical Society: Series A (General)*, vol. 135, no. 3, pp. 370–384, 1972.

[59] T. McConaghy, "Ffx: Fast, scalable, deterministic symbolic regression technology," in *Genetic Programming Theory and Practice IX*, Springer, 2011, pp. 235–260.

[60] C. Chen, C. Luo, and Z. Jiang, "Elite bases regression: A real-time algorithm for symbolic regression," in *2017 13th International conference on natural computation, fuzzy systems and knowledge discovery (ICNC-FSKD)*, IEEE, 2017, pp. 529–535.

[61] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[62] C. Luo and S.-L. Zhang, "Parse-matrix evolution for symbolic regression," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 6, pp. 1182–1193, 2012.

[63] F. O. de Franca and G. S. I. Aldeia, "Interaction-transformation evolutionary algorithm for symbolic regression," *Evolutionary Computation*, pp. 1–25, 2020.

[64] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[65] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[66] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. Galván-López, "Semantically-based crossover in genetic programming: Application to real-valued symbolic regression," *Genetic Programming and Evolvable Machines*, vol. 12, no. 2, pp. 91–119, 2011.

[67] L. Trujillo, L. Muñoz, E. Galván-López, and S. Silva, "Neat genetic programming: Controlling bloat naturally," *Information Sciences*, vol. 333, pp. 21–43, 2016.

[68] M. Cranmer, *Pysr: Fast & parallelized symbolic regression in python/julia*, Sep. 2020. DOI: 10.5281/zenodo.4041459.

[69] A. Meurer, C. P. Smith, M. Paprocki, O. Čertık, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, *et al.*, "Sympy: Symbolic computing in python," *PeerJ Computer Science*, vol. 3, e103, 2017.

[70] M. L. Larma, B. K. Petersen, S. K. Kim, C. P. Santiago, R. Glatt, T. N. Mundhenk, J. F. Pettit, and D. M. Faissol, "Improving exploration in policy gradient search: Application to symbolic optimization," *arXiv preprint arXiv:2107.09158*, 2021.

[71] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[72] S. Eidnes and K. O. Lye, "Pseudo-hamiltonian neural networks for learning partial differential equations," *Journal of Computational Physics*, p. 112 738, 2024.

[73] T. Matsubara, A. Ishikawa, and T. Yaguchi, "Deep energy-based modeling of discrete-time physics," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 100–13 111, 2020.

[74] N. Sawant, B. Kramer, and B. Peherstorfer, "Physics-informed regularization and structure preservation for learning stable reduced models from data with operator inference," *Computer Methods in Applied Mechanics and Engineering*, vol. 404, p. 115 836, 2023.

[75] J. Duan and J. S. Hesthaven, "Non-intrusive data-driven reduced-order modeling for time-dependent parametrized problems," *Journal of Computational Physics*, vol. 497, p. 112 621, 2024.

[76] I. G. Kevrekidis, C. W. Gear, J. M. Hyman, P. G. Kevrekidis, O. Runborg, C. Theodoropoulos, *et al.*, "Equation-free, coarse-grained multiscale computation: Enabling microscopic simulators to perform system-level analysis," *Commun. Math. Sci*, vol. 1, no. 4, pp. 715–762, 2003.

[77] R. González-Garcıa, R. Rico-Martınez, and I. G. Kevrekidis, "Identification of distributed parameter systems: A neural net based approach," *Computers & chemical engineering*, vol. 22, S965–S968, 1998.

[78] H. U. Voss, P. Kolodner, M. Abel, and J. Kurths, "Amplitude equations from spatiotemporal binary-fluid convection data," *Physical review letters*, vol. 83, no. 17, p. 3422, 1999.

[79] G. Sugihara, R. May, H. Ye, C.-h. Hsieh, E. Deyle, M. Fogarty, and S. Munch, "Detecting causality in complex ecosystems," *science*, vol. 338, no. 6106, pp. 496–500, 2012.

[80] H. Ye, R. J. Beamish, S. M. Glaser, S. C. Grant, C.-h. Hsieh, L. J. Richards, J. T. Schnute, and G. Sugihara, "Equation-free mechanistic ecosystem forecasting using empirical dynamic modeling," *Proceedings of the National Academy of Sciences*, vol. 112, no. 13, E1569–E1576, 2015.

[81] A. J. Roberts, *Model emergent dynamics in complex systems*. SIAM, 2014, vol. 20.

[82] M. D. Schmidt, R. R. Vallabhajosyula, J. W. Jenkins, J. E. Hood, A. S. Soni, J. P. Wikswo, and H. Lipson, "Automated refinement and inference of analytical models for metabolic networks," *Physical biology*, vol. 8, no. 5, p. 055 011, 2011.

[83] B. C. Daniels and I. Nemenman, "Automated adaptive inference of phenomenological dynamical models," *Nature communications*, vol. 6, no. 1, p. 8133, 2015.

[84] B. C. Daniels and I. Nemenman, "Efficient inference of parsimonious phenomenological models of cellular dynamics using s-systems and alternating regression," *PloS one*, vol. 10, no. 3, e0119821, 2015.

[85] A. J. Majda, C. Franzke, and D. Crommelin, "Normal forms for reduced stochastic climate models," *Proceedings of the National Academy of Sciences*, vol. 106, no. 10, pp. 3649–3653, 2009.

[86] D. Giannakis and A. J. Majda, "Nonlinear laplacian spectral analysis for time series with intermittency and low-frequency variability," *Proceedings of the National Academy of Sciences*, vol. 109, no. 7, pp. 2222–2227, 2012.

[87] I. Mezić, "Analysis of fluid flows via spectral properties of the koopman operator," *Annual review of fluid mechanics*, vol. 45, pp. 357–378, 2013.

[88] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 104, no. 24, pp. 9943–9948, 2007.

[89] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 45, pp. 22 445–22 451, 2019.

[90] Z. Long, Y. Lu, X. Ma, and B. Dong, "Pde-net: Learning pdes from data," in *International conference on machine learning*, PMLR, 2018, pp. 3208–3216.

[91] Z. Long, Y. Lu, and B. Dong, "Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network," *Journal of Computational Physics*, vol. 399, p. 108 925, 2019.

[92] M. Raissi and G. E. Karniadakis, "Hidden physics models: Machine learning of nonlinear partial differential equations," *Journal of Computational Physics*, vol. 357, pp. 125–141, 2018.

[93] A. Kothari, T. Tohme, X. Zhang, and K. Youcef-Toumi, "Enhanced human-robot collaboration using constrained probabilistic human-motion prediction," *arXiv preprint arXiv:2310.03314*, 2023.

[94] Z. Chen, Y. Liu, and H. Sun, "Physics-informed learning of governing equations from scarce data," *Nature communications*, vol. 12, no. 1, p. 6136, 2021.

[95] H. P. Flath, L. C. Wilcox, V. Akçelik, J. Hill, B. van Bloemen Waanders, and O. Ghattas, "Fast algorithms for bayesian uncertainty quantification in large-scale linear inverse problems based on low-rank partial hessian approximations," *SIAM Journal on Scientific Computing*, vol. 33, no. 1, pp. 407–432, 2011.

[96] A. Jameson, "Aerodynamic shape optimization using the adjoint method," *Lectures at the Von Karman Institute, Brussels*, 2003.

[97] R. Caflisch, D. Silantyev, and Y. Yang, "Adjoint dsmc for nonlinear boltzmann equation constrained optimization," *Journal of Computational Physics*, vol. 439, p. 110 404, 2021.

[98] T. Antonsen, E. J. Paul, and M. Landreman, "Adjoint approach to calculating shape gradients for three-dimensional magnetic confinement equilibria," *Journal of Plasma Physics*, vol. 85, no. 2, p. 905 850 207, 2019.

[99] A. Geraldini, M. Landreman, and E. Paul, "An adjoint method for determining the sensitivity of island size to magnetic field variations," *Journal of Plasma Physics*, vol. 87, no. 3, p. 905 870 302, 2021.

[100] C. Patrignani, K. Agashe, G. Aielli, C. Amsler, M. Antonelli, D. Asner, H. Baer, S. Banerjee, R. Barnett, T. Basaglia, *et al.*, "Review of particle physics," *CHINESE PHYSICS C*, 2016.

[101] A. Frezzotti, L. Gibelli, and S. Lorenzani, "Mean field kinetic theory description of evaporation of a fluid into vacuum," *Physics of Fluids*, vol. 17, no. 1, p. 012 102, 2005. DOI: 10.1063/1.1824111.

[102] M. Kon, K. Kobayashi, and M. Watanabe, "Method of determining kinetic boundary conditions in net evaporation/condensation," *Physics of Fluids*, vol. 26, no. 7, p. 072 003, 2014.

[103] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," *arXiv preprint arXiv:2011.13456*, 2020.

[104] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *The annals of mathematical statistics*, pp. 832–837, 1956.

[105] M. C. Jones, J. S. Marron, and S. J. Sheather, "A brief survey of bandwidth selection for density estimation," *Journal of the American statistical association*, vol. 91, no. 433, pp. 401–407, 1996.

[106] S. J. Sheather, "Density estimation," *Statistical science*, pp. 588–597, 2004.

[107] M. Hermite, *Sur un nouveau développement en série des fonctions*. Imprimerie de Gauthier-Villars, 1864.

[108] H. Grad, "Note on N-dimensional Hermite polynomials," *Communications on Pure and Applied Mathematics*, vol. 2, no. 4, pp. 325–330, 1949.

[109] Z. Cai, Y. Fan, and R. Li, "A framework on moment model reduction for kinetic equation," *SIAM Journal on Applied Mathematics*, vol. 75, no. 5, pp. 2001–2023, 2015.

[110] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard, "Density estimation by wavelet thresholding," *The Annals of statistics*, pp. 508–539, 1996.

[111] J. N. Kapur, *Maximum-entropy models in science and engineering*. John Wiley & Sons, 1989.

[112] A. Tagliani, "Hausdorff moment problem and maximum entropy: A unified approach," *Applied Mathematics and Computation*, vol. 105, no. 2-3, pp. 291–305, 1999.

[113] A. Y. Khinchin, *Mathematical foundations of information theory*. Courier Corporation, 2013.

[114] C. D. Hauck, C. D. Levermore, and A. L. Tits, "Convex duality and entropy-based moment closures: Characterizing degenerate densities," *SIAM Journal on Control and Optimization*, vol. 47, no. 4, pp. 1977–2015, 2008.

[115] W. Dreyer, "Maximisation of the entropy in non-equilibrium," *Journal of Physics A: Mathematical and General*, vol. 20, no. 18, p. 6505, 1987.

[116] C. D. Levermore, "Moment closure hierarchies for kinetic theories," *Journal of statistical Physics*, vol. 83, no. 5-6, pp. 1021–1065, 1996.

[117] R. V. Abramov, "An improved algorithm for the multidimensional moment-constrained maximum entropy problem," *Journal of Computational Physics*, vol. 226, no. 1, pp. 621–644, 2007.

[118] R. V. Abramov, "The multidimensional moment-constrained maximum entropy problem: A BFGS algorithm with constraint scaling," *Journal of Computational Physics*, vol. 228, no. 1, pp. 96–108, 2009.

[119] M. Sadr, Q. Wang, and M. H. Gorji, "Coupling kinetic and continuum using data-driven maximum entropy distribution," *Journal of Computational Physics*, vol. 444, p. 110 542, 2021.

[120] W. A. Porteous, M. P. Laiu, and C. D. Hauck, "Data-driven, structure-preserving approximations to entropy-based moment closures for kinetic equations," *arXiv preprint arXiv:2106.08973*, 2021.

[121] S. Schotthöfer, T. Xiao, M. Frank, and C. D. Hauck, "Neural network-based, structure-preserving entropy closures for the Boltzmann moment system," *arXiv preprint arXiv:2201.10364*, 2022.

[122] M. Sadr, M. Torrilhon, and M. H. Gorji, "Gaussian process regression for maximum entropy distribution," *Journal of Computational Physics*, vol. 418, p. 109 644, 2020.

[123] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International conference on machine learning*, PMLR, 2015, pp. 1530–1538.

[124] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," *arXiv preprint arXiv:1605.08803*, 2016.

[125] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in neural information processing systems*, vol. 31, 2018.

[126] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural spline flows," *Advances in neural information processing systems*, vol. 32, 2019.

[127] B. Tzen and M. Raginsky, "Neural stochastic differential equations: Deep latent Gaussian models in the diffusion limit," *arXiv preprint arXiv:1905.09883*, 2019.

[128] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 3964–3979, 2020.

[129] S. Wang and Y. Marzouk, "On minimax density estimation via measure transport," *arXiv preprint arXiv:2207.10231*, 2022.

[130] F. Noé, S. Olsson, J. Köhler, and H. Wu, "Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning," *Science*, vol. 365, no. 6457, eaaw1147, 2019.

[131] S.-M. Udrescu and M. Tegmark, "AI Feynman: A physics-inspired method for symbolic regression," *Science Advances*, vol. 6, no. 16, eaay2631, 2020.

[132] S.-M. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark, "AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4860–4871, 2020.

[133] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker, "Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps," *Proceedings of the national academy of sciences*, vol. 102, no. 21, pp. 7426–7431, 2005.

[134] R. R. Coifman and S. Lafon, "Diffusion maps," *Applied and computational harmonic analysis*, vol. 21, no. 1, pp. 5–30, 2006.

[135] F. Li and Y. Marzouk, "Diffusion map particle systems for generative modeling," *arXiv preprint arXiv:2304.00200*, 2023.

[136] S. Chewi, T. Le Gouic, C. Lu, T. Maunu, and P. Rigollet, "SVGD as a kernelized Wasserstein gradient flow of the chi-squared divergence," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2098–2109, 2020.

[137] C. Villani, *Optimal transport: old and new*. Springer, 2009, vol. 338.

[138] Y. Song and S. Ermon, "Improved techniques for training score-based generative models," *Advances in neural information processing systems*, vol. 33, pp. 12 438–12 448, 2020.

[139] Q. Liu, "Stein variational gradient descent as gradient flow," *Advances in neural information processing systems*, vol. 30, 2017.

[140] A. Garbuno-Inigo, F. Hoffmann, W. Li, and A. M. Stuart, "Interacting Langevin diffusions: Gradient structure and ensemble Kalman sampler," *SIAM Journal on Applied Dynamical Systems*, vol. 19, no. 1, pp. 412–441, 2020.

[141] A. Garbuno-Inigo, N. Nusken, and S. Reich, "Affine invariant interacting Langevin dynamics for Bayesian inference," *SIAM Journal on Applied Dynamical Systems*, vol. 19, no. 3, pp. 1633–1658, 2020.

[142] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose bayesian inference algorithm," *Advances in neural information processing systems*, vol. 29, 2016.

[143] Q. Liu and D. Wang, "Stein variational gradient descent as moment matching," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[144] S. Mallat, *A wavelet tour of signal processing*. Elsevier, 1999.

[145] T. Marchand, M. Ozawa, G. Biroli, and S. Mallat, "Wavelet conditional renormalization group," *arXiv preprint arXiv:2207.04941*, 2022.

[146] Z. Kadkhodaie, F. Guth, S. Mallat, and E. P. Simoncelli, "Learning multi-scale local conditional probability models of images," *arXiv preprint arXiv:2303.02984*, 2023.

[147] Z. Botev, "Nonparametric density estimation via diffusion mixing," 2007.

[148] Z. I. Botev, J. F. Grotowski, and D. P. Kroese, "Kernel density estimation via diffusion," *The Annals of Statistics*, vol. 38, no. 5, 2010.

[149] E. Platen and N. Bruti-Liberati, *Numerical solution of stochastic differential equations with jumps in finance*. Springer Science & Business Media, 2010, vol. 64.

[150] R. Abramov, "A practical computational framework for the multidimensional moment-constrained maximum entropy principle," *Journal of Computational Physics*, vol. 211, no. 1, pp. 198–209, 2006.

[151] E. K. Chong and S. H. Zak, *An introduction to optimization*. John Wiley & Sons, 2013, vol. 75.

[152] R. V. Abramov, "The multidimensional maximum entropy moment problem: A review of numerical methods," *Communications in Mathematical Sciences*, vol. 8, no. 2, pp. 377–392, 2010.

[153] G. W. Alldredge, C. D. Hauck, D. P. O'Leary, and A. L. Tits, "Adaptive change of basis in entropy-based moment closures for linear kinetic equations," *Journal of Computational Physics*, vol. 258, pp. 489–508, 2014.

[154] L. Giraud, J. Langou, and M. Rozloznık, "On the round-off error analysis of the gram-schmidt algorithm with reorthogonalization," Technical Report TR/PA/02/33, CERFACS, Toulouse, France, Tech. Rep., 2002.

[155] J. McDonald and M. Torrilhon, "Affordable robust moment closures for CFD based on the maximum-entropy hierarchy," *Journal of Computational Physics*, vol. 251, pp. 500–523, 2013.

[156] N. I. Akhiezer and N. Kemmer, *The classical moment problem: and some related questions in analysis*. Oliver & Boyd Edinburgh, 1965, vol. 5.

[157] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003, vol. 82.

[158] M. Junk, "Domain of definition of levermore's five-moment system," *Journal of Statistical Physics*, vol. 93, pp. 1143–1167, 1998.

[159] M. Sadr, N. G. Hadjiconstantinou, and M. H. Gorji, "Wasserstein-penalized entropy closure: A use case for stochastic particle methods," *arXiv preprint arXiv:2308.02607*, 2023.

[160] H. Risken, *The Fokker-Planck Equation: Methods of solution and applications*. Springer, 1989.

[161] B. Oksendal, *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[162] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, *Handbook of Markov chain Monte Carlo*. CRC press, 2011.

[163] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An introduction to MCMC for machine learning," *Machine learning*, vol. 50, pp. 5–43, 2003.

[164] A. Doucet and X. Wang, "Monte Carlo methods for signal processing: A review in the statistical signal processing context," *IEEE Signal Processing Magazine*, vol. 22, no. 6, pp. 152–170, 2005.

[165] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[166] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press, 2016, vol. 1.

[167] A. Korattikara, Y. Chen, and M. Welling, "Austerity in MCMC land: Cutting the metropolis-hastings budget," in *International conference on machine learning*, PMLR, 2014, pp. 181–189.

[168] Y. F. Atchadé and J. S. Rosenthal, "On adaptive Markov chain Monte Carlo algorithms," *Bernoulli*, vol. 11, no. 5, pp. 815–828, 2005.

[169] V. Kungurtsev, A. Cobb, T. Javidi, and B. Jalaian, "Decentralized Bayesian learning with Metropolis-adjusted Hamiltonian Monte Carlo," *Machine Learning*, pp. 1–29, 2023.

[170] P. G. Constantine, C. Kent, and T. Bui-Thanh, "Accelerating Markov chain Monte Carlo with active subspaces," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, A2779–A2805, 2016.

[171] P. R. Conrad, Y. M. Marzouk, N. S. Pillai, and A. Smith, "Accelerating asymptotically exact MCMC for computationally intensive models via local approximations," *Journal of the American Statistical Association*, vol. 111, no. 516, pp. 1591–1607, 2016.

[172] S. E. Dosso and J. Dettmer, "Bayesian matched-field geoacoustic inversion," *Inverse Problems*, vol. 27, no. 5, p. 055 009, 2011.

[173] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[174] K. Csilléry, M. G. Blum, O. E. Gaggiotti, and O. François, "Approximate Bayesian computation (abc) in practice," *Trends in ecology & evolution*, vol. 25, no. 7, pp. 410–418, 2010.

[175] K. Cranmer, J. Brehmer, and G. Louppe, "The frontier of simulation-based inference," *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30 055–30 062, 2020.

[176] G. Papamakarios, D. Sterratt, and I. Murray, "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows," in *The 22nd International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, pp. 837–848.

[177] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[178] T. Salimans, D. Kingma, and M. Welling, "Markov chain Monte Carlo and variational inference: Bridging the gap," in *International conference on machine learning*, PMLR, 2015, pp. 1218–1226.

[179] A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernández-Lobato, and A. L. Gaunt, "Deterministic variational inference for robust bayesian neural networks," in *International Conference on Learning Representations*, 2018.

[180] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, IEEE, 2018, pp. 80–89.

[181] Z. Liu and M. Tegmark, "Machine learning conservation laws from trajectories," *Physical Review Letters*, vol. 126, no. 18, p. 180 604, 2021.

[182] L. S. Keren, A. Liberzon, and T. Lazebnik, "A computational framework for physics-informed symbolic regression with straightforward integration of domain knowledge," *Scientific Reports*, vol. 13, no. 1, p. 1249, 2023.

[183] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," *arXiv preprint arXiv:1410.8516*, 2014.

[184] S. T. Radev, F. Graw, S. Chen, N. T. Mutters, V. M. Eichel, T. Bärnighausen, and U. Köthe, "Outbreakflow: Model-based Bayesian inference of disease outbreak dynamics with invertible neural networks and its application to the covid-19 pandemics in germany," *PLoS computational biology*, vol. 17, no. 10, e1009472, 2021.

[185] L. Ardizzone, J. Kruse, C. Rother, and U. Köthe, "Analyzing inverse problems with invertible neural networks," in *International Conference on Learning Representations*, 2019.

[186] A. Luce, A. Mahdavi, H. Wankerl, and F. Marquardt, "Investigation of inverse design of multilayer thin-films with conditional invertible neural networks," *Machine Learning: Science and Technology*, vol. 4, no. 1, p. 015 014, Feb. 2023. DOI: 10.1088/2632-2153/acb48d.

[187] X. Zhang and A. Curtis, "Bayesian geophysical inversion using invertible neural networks," *Journal of Geophysical Research: Solid Earth*, vol. 126, no. 7, e2021JB022320, 2021.

[188] S. Wu, Q. Huang, and L. Zhao, "Fast Bayesian inversion of airborne electromagnetic data based on the invertible neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 61, pp. 1–11, 2023.

[189] G. A. Padmanabha and N. Zabaras, "Solving inverse problems using conditional invertible neural networks," *Journal of Computational Physics*, vol. 433, p. 110 194, 2021.

[190] N. R. Chapman and E. C. Shang, "Review of geoacoustic inversion in underwater acoustics," *Journal of Theoretical and Computational Acoustics*, vol. 29, no. 03, p. 2 130 004, 2021.

[191] Z. Xu, H. Zhang, Y. Wang, X. Chang, and Y. Liang, "L 1/2 regularization," *Science China Information Sciences*, vol. 53, pp. 1159–1169, 2010.

[192] W. Wu, Q. Fan, J. M. Zurada, J. Wang, D. Yang, and Y. Liu, "Batch gradient method with smoothing l1/2 regularization for training of feedforward neural networks," *Neural Networks*, vol. 50, pp. 72–78, 2014.

[193] Q. Fan, J. M. Zurada, and W. Wu, "Convergence of online gradient method for feedforward neural networks with smoothing l1/2 regularization penalty," *Neurocomputing*, vol. 131, pp. 208–216, 2014.

[194] L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, "Guided image generation with conditional invertible neural networks," *arXiv preprint arXiv:1907.02392*, 2019.

[195] L. Ardizzone, J. Kruse, C. Lüth, N. Bracher, C. Rother, and U. Köthe, "Conditional invertible neural networks for diverse image-to-image translation," in *Pattern Recognition*, Z. Akata, A. Geiger, and T. Sattler, Eds., Cham: Springer International Publishing, 2021, pp. 373–387, ISBN: 978-3-030-71278-5.

[196] J. Kruse, L. Ardizzone, C. Rother, and U. Köthe, "Benchmarking invertible architectures on inverse problems," *arXiv preprint arXiv:2101.10763*, 2021.

[197] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, "Kan: Kolmogorov-arnold networks," *arXiv preprint arXiv:2404.19756*, 2024.

[198] A. Grover, M. Dhar, and S. Ermon, "Flow-GAN: Combining maximum likelihood and adversarial learning in generative models," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[199] S. Ren, W. Padilla, and J. Malof, "Benchmarking deep inverse models over time, and the neural-adjoint method," *Advances in Neural Information Processing Systems*, vol. 33, pp. 38–48, 2020.

[200] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 723–773, 2012.

[201] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *Journal of Machine Learning Research*, vol. 22, no. 57, pp. 1–64, 2021.

[202] F. B. Jensen, W. A. Kuperman, M. B. Porter, H. Schmidt, and A. Tolstoy, *Computational ocean acoustics*. Springer, 2011, vol. 2011.

[203] W. H. Ali, A. Charous, C. Mirabito, P. J. Haley, and P. F. Lermusiaux, "MSEAS-ParEq for coupled ocean-acoustic modeling around the globe," in *OCEANS 2023-MTS/IEEE US Gulf Coast*, IEEE, 2023, pp. 1–10.

[204] C.-F. Huang, P. Gerstoft, and W. S. Hodgkiss, "Uncertainty analysis in matched-field geoacoustic inversions," *The Journal of the Acoustical Society of America*, vol. 119, no. 1, pp. 197–207, 2006.

[205] M. J. Bianco, P. Gerstoft, J. Traer, E. Ozanich, M. A. Roch, S. Gannot, and C.-A. Deledalle, "Machine learning in acoustics: Theory and applications," *The Journal of the Acoustical Society of America*, vol. 146, no. 5, pp. 3590–3628, 2019.

[206] C. W. Holland, J. Dettmer, and S. E. Dosso, "Remote sensing of sediment density and velocity gradients in the transition layer," *The Journal of the Acoustical Society of America*, vol. 118, no. 1, pp. 163–177, 2005.

[207] J. Benson, N. R. Chapman, and A. Antoniou, "Geoacoustic model inversion using artificial neural networks," *Inverse Problems*, vol. 16, no. 6, p. 1627, 2000.

[208] P. Jaini, K. A. Selby, and Y. Yu, "Sum-of-squares polynomial flow," in *International Conference on Machine Learning*, PMLR, 2019, pp. 3009–3018.

[209] L. Wenliang, D. J. Sutherland, H. Strathmann, and A. Gretton, "Learning deep kernels for exponential family densities," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6737–6746.

[210] C. Yardim, P. Gerstoft, and W. S. Hodgkiss, "Geoacoustic and source tracking using particle filtering: Experimental results," *The Journal of the Acoustical Society of America*, vol. 128, no. 1, pp. 75–87, 2010.

[211] F. Meyer and K. L. Gemba, "Probabilistic focalization for shallow water localization," *J. Acoust. Soc. Am.*, vol. 150, no. 2, pp. 1057–1066, Aug. 2021.

[212] M. B. Porter, "The kraken normal mode program," *Naval Research Laboratory, Washington DC*, 1992.

[213] A. M. Alaa and M. van der Schaar, "Demystifying black-box models with symbolic metamodels," *Advances in neural information processing systems*, vol. 32, 2019.

[214] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," in *Doklady Akademii Nauk*, Russian Academy of Sciences, vol. 114, 1957, pp. 953–956.

[215] R. Beals and J. Szmigielski, "Meijer g-functions: A gentle introduction," *Notices of the AMS*, vol. 60, no. 7, pp. 866–872, 2013.

[216] K. Debrabant, G. Samaey, and P. Zielinski, "A micro-macro acceleration method for the Monte Carlo simulation of stochastic differential equations," *SIAM Journal on Numerical Analysis*, vol. 55, no. 6, pp. 2745–2786, 2017.