PARAMETRIC MATRIX MODELS FOR EMULATION IN NUCLEAR AND MANY-BODY PHYSICS

By

Patrick Cook

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Physics—Doctor of Philosophy
Computational Mathematics, Science, and Engineering—Dual Major

2026

ABSTRACT

Progress in nuclear and many-body physics today is predicated on the ability to solve large-scale, strongly correlated quantum many-body problems. As the theoretical models become more sophisticated, they also become more computationally complex. Simultaneously, quantifying uncertainty in model predictions and fitting free parameters to experimental observations requires repeated evaluation of these expensive models. The desire for progress in the theoretical understanding of such systems will always outpace the rate at which computational resources become available, and so improvements in the efficiency of our high-fidelity methods alone are insufficient. Instead, surrogate models—known as emulators—provide the means of accomplishing these goals.

This thesis provides an introduction into the current state of emulation in nuclear and many-body physics. The motivations, goals, and origins of currently popular emulation methods are discussed along with selected examples. We see how many methods are closely mathematically related and how trade-offs are made to optimize specific properties or applications.

The central work in this thesis is the method of parametric matrix models (PMMs), an emulation and general machine learning framework which combines aspects of traditional reduced basis method with modern parametric machine learning. PMMs are able to retain as much or as little physical information about the underlying system as desired, yielding not only excellent performance but also nearly unparalleled adaptability, interpretability, and trustworthiness as an emulation method.

A formal mathematical framework for PMMs is developed and accompanied by practical step-by-step procedures for the application of the method. Extensions and current research into applying the method to a variety of problems are presented, culminating in the to-date most comprehensive PMM-emulator which is applied to the problem of emulating in-medium similarity renormalization group calculations of nuclear matter.

As part of this thesis, the open-source PYPMM package was developed [1]. This pack-

age enables any researcher to construct, train, share, and deploy PMM-based emulators with modular, extendable, and graphics processing unit (GPU)-optimized code. All PMM examples in this thesis were created using this package.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

χEFT chiral effective field theory

AD automatic differentiation

AI artificial intelligence

ANN artificial neural network

DEIM discrete empirical interpolation method

DFT density functional theory

DMD dynamic mode decomposition

EC eigenvector continuation

EMN Entem–Machleidt–Nosyk

EOS equation of state

eRBM explicit reduced basis method

GP Gaussian process

GPR Gaussian process regression

GPU graphics processing unit

HF Hartree–Fock

IMSRG in-medium similarity renormalization group

iRBM implicit reduced basis method

LEC low-energy constant

LMG Lipkin–Meshkov–Glick

MAD median absolute deviation

ML machine learning

MLP multi-layer perceptron

PCA principal component analysis

PINN physics-informed neural network

PMM parametric matrix model

PNM pure neutron matter

POD proper orthogonal decomposition

RBM reduced basis method

SINDy sparse identification of nonlinear dynamics

SNM symmetric nuclear matter

SVD singular value decomposition

TDDFT time-dependent density functional theory

UQ uncertainty quantification

wMSE weighted mean squared error

wVAR variance of the weighted absolute error

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Foreword

The fields of nuclear theory, many-body theory, computational physics, artificial intelligence (AI), and machine learning (ML) are all rapidly evolving. This dissertation captures only a partial snapshot of these fields and how this work forced itself into them. Many of the statements made in this dissertation—especially in regard to AI and ML—will likely be outdated at the time of reading.[1] Moreover, much of the work itself is still evolving. The reader is encouraged to look up the latest research regarding the topics discussed herein.

It is unfortunate that deliberately deceptive and unquantifiably harmful modern trends in the commercialization of a very specific subfield of AI and ML have brought such a negative connotation to the entire field. Almost nothing in this thesis is relevant to the application of large language models (LLMs) and generative AI. The extent to which this work is related to LLMs begins and ends at the fact that both are in the subfield of ML. The research, engineering, algorithms, mathematics, and science behind LLMs are all exceedingly interesting and impressive and deserve to be appreciated in a vacuum separate from their marketed products.

Declaration on the use of Generative AI Tools

Despite the use of em-dashes[2]—a punctuation I have always made extensive use of—absolutely no text or material content in this thesis was created with the use of generative AI tools. Spelling, grammar, and other typographical errors in drafts of this thesis were detected in part with generative AI tools. Some LaTeX code for the formatting of this document was created with assistance from generative AI tools.

[1] I'm sure they'll either age like wine or milk, nothing in between.

[2] A punctuation that some today associate with AI-generated text.

1.2 Organization

This dissertation is organized as follows: Chapter 1 is this. Chapter 2 introduces emulation and establishes a list of properties, goals, and considerations for emulators in computational physics and concludes with a summary of commonly used methods with selected examples. Chapter 3 introduces parametric matrix models (PMMs), a new class of emulators and machine learning models that use trainable versions of the known or supposed governing equations of a system to learn from data. We discuss the origins of PMMs, formalize the mathematical framework for the method, detail the practical aspects of constructing and training PMMs, and walk through varied example applications. Two uncertainty quantification methods for emulation are discussed in Chapter 4, with particular attention to conformal predictions with PMMs. Finally, Chapter 5 summarizes the work and outlines future directions.

1.3 Reading Guide

This thesis is written for readers with strong backgrounds in quantum mechanics, linear algebra, and numerical methods, passing familiarity with ML, and little to no background in emulation or PMMs. A linear reading should provide a comprehensive understanding of the theory and practice of PMMs. The minimum required reading for just the theory of PMMs is Sections 2.11.1, 2.11.1.1, and 3.1. Those interested in using PMMs should additionally read Sections 3.4, 3.5, and 3.6. For the latest advances in PMMs at time of writing, see Sections 3.7.1, 3.8, 3.10, 3.11, and 3.12.

CHAPTER 2

EMULATION IN NUCLEAR AND MANY-BODY PHYSICS

Our current best models in nuclear physics—both *ab initio* and phenomenological—have free parameters that must be fit to experimental data. These parameters often come in the form of low-energy constants (LECs) arising from the truncation of effective field theories, but also appear in phenomenological models such as energy density functionals [2–5]. The process of fitting these parameters, and just as importantly quantifying the uncertainties in these parameters, requires searching high-dimensional parameter spaces. Each trial point in parameter space requires a full model evaluation. It is not uncommon for a single model evaluation to take hours or days on high-performance computing clusters [6–14]. With hundreds of thousands to millions of trial evaluations required to adequately sample the parameter space, this process quickly becomes intractable.

One may be tempted to replace the full—also called *high-fidelity* or *full-order*—models with relatively simple and fast regression models such as multi-layer perceptrons (MLPs) or Gaussian processes (GPs) fit directly to available experimental data. After all, the high-fidelity models are also fit to experimental data. There are many issues with this approach. Depending on your persuasion, you may focus first on the philosophical issues regarding what are we actually learning about the universe from such models or more on the practical issues in training such models to sufficient accuracy. Our goal is ultimately to understand the physics which leads to the properties of nuclei, and even an artificial neural network (ANN) that perfectly reproduces experimental data cannot provide this understanding.[1] Such a perfect ANN is likely unobtainable anyway, due to the limited amount of experimental data available and the complexity of nuclear systems. Consider an overly optimistic scenario where we have access to experimental data of the energies, decay modes, quadrupole deformation, and any ten other observables imaginable for several low-lying states of all known nuclei. A quick Fermi estimate shows that this would be about 10^5 data points. As a comparison,

[1] Unless it turns out that the universe is just a big neural network.

one of the most widely used and now-solved datasets is the ImageNet Large Scale Visual Recognition Challenge which contains more than 10^6 full-color images [15]. Assuming that just 0.1% of the raw values of that dataset are actual useful information, it would contain about 10^8 data points. This is three orders of magnitude larger than our exceedingly optimistic estimate for all the nuclear data we could ever hope to have. Even if we had enough data, the complexity of nuclear systems—strongly-coupled quantum many-body systems—is substantially greater than any computer vision task. All this is to say that our high-fidelity models play irreplaceable roles both in the theoretical understanding of nuclear systems and in the practical prediction of nuclear properties.

High-fidelity models are irreplaceable but simultaneously too expensive to fit directly to data. The solution to this dilemma comes in the form of *emulators* or *reduced-order models*. The core idea is to use the high-fidelity model to generate a relatively large amount of synthetic data, then use that synthetic data to train a fast surrogate model that approximates the high-fidelity model. Traditionally, if this surrogate model was a low-dimensional (usually projected) approximation of the high-fidelity model it would be called a reduced-order model, and conversely if the surrogate model was a trained "black-box" machine learning model it would be known as an emulator. Recent advancements have blurred the line between reduced-order models and emulators in the traditional sense, and as such we will refer to all surrogate models collectively as emulators and classify them via a series of other properties [16].

While slow, generating data with the high-fidelity model is still far cheaper than obtaining experimental data. Once trained, the emulator can be used in place of the high-fidelity model for expensive tasks such as parameter fitting and uncertainty quantification. If accurate and trustworthy enough, the emulator may even replace the high-fidelity model for most evaluations as is the case in traditional model order reduction and the concept of digital twins. Beyond raw accuracy, there are several important properties that a good emulator should have which are discussed in the following sections.

2.1 Parametricity

Physical models are *parametric*. That is, the behavior, dynamics, or physics of the system depend on some parameters. For instance, the mass of a body, the interaction strength between bodies, or even the number of particles are all parameters of a system. These parameters may be physical, like the preceding examples, or numerical, such as the resolution and tolerance of the numerical representation. One might describe the parametric equations governing such a system as actually defining a family of systems, where each system corresponds to a specific realization of those parameters. For example, consider the family of quadratic polynomials $f(x; b, c) = x^2 + bx + c$. A specific quadratic in this family is determined by the parameters b and c.

There are several distinct definitions for "parametricity" and "parametric" in the context of machine learning, physics, and emulation. The most common definition from machine learning describes whether or not a machine learning model's size depends on the amount of data provided to it. We will distinguish between the mathematical or physical parametricity and the machine learning or numerical parametricity as follows.

Definition (Mathematically Parametric Emulator)**.** *An emulator which is able to model an entire family of systems as a function of some parameters is a mathematically parametric emulator. The opposite is a mathematically non-parametric emulator.*

Definition (Numerically Parametric Emulator)**.** *An emulator or machine learning model with a size that is independent of the amount of data available to it is called a numerically parametric emulator. The opposite is a numerically non-parametric emulator. Equivalently, a numerically parametric emulator is one whose parameters are not explicitly determined from data.*

Numerical parametricity is a particularly useful property in a machine learning model since it often allows for progressive training. A numerically parametric model trained on a set of data can later update its numerical parameters when new data become available—

without needing to consult the original data. This means that as new training examples become available, one can adapt the model with comparatively small changes rather than needing to completely refit the model.

There is an additional overloading of the term "parameter" in machine learning. Namely, the trainable parameters of a machine learning model are often referred to simply as "parameters". It is likely for this reason that specific models will use more specialized terms for trainable parameters, like "weights" and "biases" in the case of ANNs. Most of the time however, it is clear from the context of the problem which parameters are the physical or mathematical ones and which are the trainable ones.

2.2 Linearity

Linear systems are by far the most well-studied types of problems. Most of our mathematical and numerical techniques are maximally effective only for linear systems. Many real-world problems of interest are also linear—or at least well-approximated by linear problems. Modern quantum mechanics, many-body theory, and *ab initio* nuclear theory make heavy use of linear operators. It is perhaps unsurprising then that emulators themselves, especially more traditional methods, are often linear. There are advantages to linear emulators, stemming from the aforementioned effectiveness of tools for linear systems. However, nuclear and many-body physics contain plenty of nonlinear processes. Furthermore as the fields have improved both understanding and measurement, they have increasingly introduced beyond-linear theories.

It is possible to approximate a nonlinear process by a linear one. This is known as *linearization*, and several emulator methods rely on it. However, in order to properly capture the nonlinear effects of a given model, it is best if the emulator can preserve the exact form of the nonlinearity from the high-fidelity model. At minimum, we want the emulator to be able to capture some nonlinear effects.

2.3 Computational Efficiency

The justification for emulation is computational efficiency. While words like "fast" and "cheap" are often used to describe emulators, computational efficiency comes in many forms and is evaluated at different stages of the emulation process. There are scenarios in which a slower algorithm that requires less memory is preferable to a faster one that requires more memory. The cost of an algorithm is often measured in terms of *complexity*, specifically time and space complexity. Complexity measures how the resources required by an algorithm scale with certain parameters, often the size of the input data or the size of the algorithm itself. Complexity theory and analysis is a deep and rich field of study [17], but for our purposes we will use only a loose interpretation of complexity through asymptotic notation, or more commonly *big-O notation*.

Suppose an algorithm $\mathcal{F}$ requires $f(n)$ steps[2] to complete, where n is some parameter regarding the size of the computation such as the size of the input. We say that the time complexity of $\mathcal{F}$ is $\mathcal{O}(g(n))$ if there exist c, n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$. That is to say that past a certain size the runtime of $\mathcal{F}$ does not grow faster than some constant multiple of $g(n)$. Big-O notation is particularly useful since the determination of the exact number of steps required by an algorithm is often difficult, and the leading order behavior is often sufficient to compare algorithms.[3] Table 2.1 lists the time complexities of some common operations and algorithms used in practice in scientific computing as an example and reference.

Similar to time complexity, space complexity measures how the memory requirements of an algorithm scale with certain size parameters. This can include the size of the inputs (total space complexity) or only the size of any intermediate and output data structures (auxiliary space complexity). Auxiliary space complexity is often more relevant in practice, and is

[2]The precise definition of "steps" here is usually related to Turing machines. However, in scientific computing we often take a "step" to mean a single floating point operation, since these can be done in constant time.

[3]I am glossing over the issue of computability and halting. We will assume that all algorithms here terminate unless otherwise stated.

Algorithm	Input Size	Output Size	Time Complexity	Space Complexity
Floating Point $+/\times$	1	1	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Vector Addition/Scaling	n, n or 1	n	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Matrix-Vector Multiplication	(n, n), n	n	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Matrix-Matrix Multiplication [18]	(n, n), (n, n)	(n, n)	$\mathcal{O}\left(n^{\log_2 7}\right)$	$\mathcal{O}(n^2)$
Full Eigendecomposition	(n, n)	n, (n, n)	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
Partial Eigendecomposition	(n, n)	k, (n, k)	$\mathcal{O}(kn^2)$	$\mathcal{O}(kn)$

Table 2.1 In-practice complexities of common algorithms.

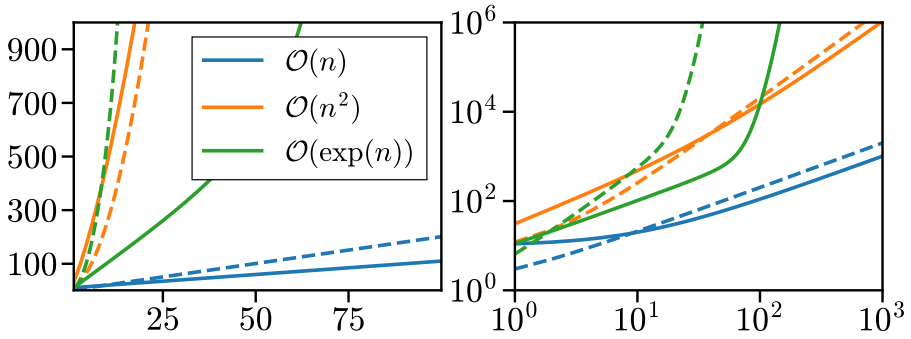

Figure 2.1 Examples of various functions with different big-O complexities. Note that since big-O notation is an asymptotic measure, constant factors and sub-leading-order terms mean that functions in the same big-O class may behave very differently for small n, and some functions in a "worse" big-O class may be smaller than functions in a "better" big-O class for small n.

the convention used here. Table 2.1 also lists the space complexities of the same common operations and algorithms.

In broader computational contexts, we usually say an algorithm is "efficient" if it has polynomial time and space complexity, i.e. $\mathcal{O}\left(n^k\right)$ for some constant k. However, in the context of emulation we are instead interested in outperforming the high-fidelity model—often substantially. So while the complexity of an emulator itself is often still of interest, the constant factors hidden by big-O notation may become important since the size of emulators are often significantly smaller than their high-fidelity counterparts. For example, a high-fidelity model may be efficient in the sense that it requires $\mathcal{O}(N)$ time and $\mathcal{O}(N^2)$ space, but an algorithmically inefficient emulator that operates in $\mathcal{O}(\exp(n))$ time and $\mathcal{O}(n^3)$ space may still be preferable if n is always much smaller than N.

2.3.1 Online and Offline Costs

A complete definition of computational efficiency in the context of an emulator requires consideration of both "online" and "offline" costs. The online cost of an emulator encompasses the resources required to use the deployed emulator. That is, once the emulator is constructed, trained, fit, or otherwise prepared and sent to an end-user, the online cost is the cost of using the emulator for inference. In the context of this work, the offline cost of an emulator encompasses the resources required to prepare it for deployment, excluding the cost of acquiring any training data. The offline cost includes the cost of constructing the emulator, training it, and any other necessary steps to prepare it for deployment. Note that some sources will include the collection of training data—and therefore the cost of evaluating the high-fidelity model the requisite number of times—in the offline costs. Including data collection in the offline costs emphasizes the importance of minimizing the amount of data required, which we will do separately in the following section. The offline cost is a one-time cost, while the online cost is incurred every time the emulator is used. It can then be said that the offline cost is amortized over the course of the online phase. Some examples of online and offline costs include:

- **Online**
  - Storing the emulator
  - Performing inference with the emulator, including any necessary preprocessing of the input data
  - Postprocessing the output of the emulator, including any necessary rescaling or dimensionality expansion
- **Offline**
  - Storing the training data

- Rescaling and preprocessing the training data, including dimensionality reduction techniques such as principal component analysis (PCA)
- Training via gradient descent or other optimization algorithms
- Pruning and other post-training optimizations

This gives us the tools to define computational efficiency in the context of emulation.

Definition (Computationally Efficient Emulator)**.** *An emulator of size n for a high-fidelity model of size N is computationally efficient if $n \ll N$, its offline costs are at most $\mathcal{O}\left(N^2 n^k\right)$, and its online costs are at most $\mathcal{O}\left(n^k\right)$ for some constant k, unless the output of the emulator is of size $\mathcal{O}(N)$, in which case allowance is made only for the costs of projecting the final reduced solution back to the full space and so the online costs must be at most $\mathcal{O}\left(Nn + n^k\right)$ for some constant k. That is, a computationally efficient emulator must be much smaller than its high-fidelity counterpart, scale at most quadratically with the size of the high-fidelity model and polynomially with its own size during the offline phase, and—unless the output size scales with the high-fidelity model—scale independently of the high-fidelity model and at most polynomially with its own size during the online phase.*

2.4 Data Efficiency

A very similar concept to computational efficiency is data efficiency. However, while the number of floating point operations at our disposal is at our discretion to an extent (if we need more, we can just run our code for longer), this is often not the case for data. Especially in the context of nuclear physics emulation, data is at a premium. It is often the case that acquiring more data is exceedingly expensive monetarily or temporally, if possible at all.

Definition (Data Efficient Emulator)**.** *An emulator is data efficient if the error of the emulator converges with the number of training examples, m, at a rate $\mathcal{O}\left(m^{-k}\right)$ for some constant $k > 1$ when not limited by emulator size. That is, a data efficient emulator can be made arbitrarily accurate by increasing the amount of training data at a power-law rate. More efficient emulators have larger values of k.*

While this definition is quite similar to that of computational efficiency, particular effort is focused on improving or optimizing the structure of an emulator in order to improve its rate of convergence in this context. Even the constant factors hidden by big-O notation may be the deciding factor as to whether or not a particular emulator is successful.

Recent interest in large language models (LLMs) highlighted the importance of data efficiency and the exact values of the constant prefactor and exponent of the convergence rate. Such attention was given to this that it became known as "neural scaling laws" in the study of LLMs [19]. Scaling laws like these are not limited to machine learning, emulation, or even physics as they appear in a seemingly universal assortment of systems that exhibit phase transitions [20, 21]. Broadly, they are known as critical phenomena or universal phenomena.

2.5 Hyperparameter Efficiency

The final type of efficiency is the efficiency of the emulator's usage of *hyperparameters*—parameters that are not determined by fitting or training the emulator. Common hyperparameters include the emulator size and the choice of loss function. Hyperparameter tuning can be exceedingly difficult and expensive while also introducing opportunities for human bias from bad intuition about what values might be best. Hyperparameters are an unavoidable part of machine learning, but we generally prefer emulators with fewer possible hyperparameter values to choose from. Additionally, emulators with clear procedures and guidance on choosing the optimal hyperparameters (or at least significantly narrowing down the search space) are generally more desirable.

2.6 Systematic Improvability

Rounding out the scaling laws is the notion of systematic improvability. In general machine learning, the idea of "universal approximators" most closely resembles this concept. While universal approximation theorems for ANNs provide only a proof of existence, in emulation we seek not only the existence but also a description of how the emulator could reach convergence. This is possible because of the existence and knowledge of the high-fidelity model.

When considering systematic improvability, we ignore the issues of data availability and emulator fitting. Instead, focus is placed squarely on whether or not such an emulator could be sufficiently expressive that it would be equivalent to the high-fidelity model given the right parameters.

Definition (Systematically Improvable Emulator)**.** *An emulator of size n for a high-fidelity model of size N is systematically improvable if for $n \to \mathcal{O}(N)$ it converges to the high-fidelity model. That is, a systematically improvable emulator can be made arbitrarily accurate by increasing its size—up to a constant multiple of the high-fidelity model size.*

2.7 Interpretability

Interpretability is somewhat subjective, but has nevertheless become a sought-after property in the machine learning (ML) community. An interpretable emulator, or machine learning model in general, is one whose constituent parts and parameters can be ascribed meaning in the context of the system being modeled. This meaning must be beyond the direct structure of the machine learning model itself, and identifying such properties often requires an expert in both the system being modeled and the machine learning model. For example, the layers of a feedforward ANN fit to a dataset of nuclear binding energies have little to no inherent meaning in the context of nuclear physics: the parameters of the model do not correspond to physical constants or encode physical processes. On the other hand, a convolutional ANN fit to a dataset of images often learns several convolutional filters in the first layer that correspond to edge detectors, which have a clear meaning in the context of image processing.

Interpretability is desirable for several philosophical and practical reasons. In scientific contexts, our goals center around understanding the universe and the laws that govern it. A model that is not interpretable may be able to fit some data without providing any insight into the underlying processes. In this context, we've simply replaced one unknown with another, possibly more complicated, unknown. Moreover, while other avenues like

uncertainty quantification (UQ) can guide the trustworthiness of a model, an interpretable model can be more easily interrogated and have its failure modes identified and understood. Perhaps the most practical concern is adoption. People of all backgrounds are less likely to trust, adopt, and invest in a model that they cannot understand.

2.8 Physical Consistency

More specific to physics emulation, we want our emulators to be physically consistent. One might think that if an emulator is accurate enough, it will automatically respect the underlying conservation and symmetry laws of the system. This is seldom the case however, as the same data may be generated from many different underlying processes with different conservation and symmetry properties.

It is often impossible to guarantee exact adherence to physical laws with an emulator. Instead, we want our emulators to have quantifiable bounds on the degree to which they violate said laws and for these bounds to systematically improve with the size of the emulator and the amount of training data. The exact form and scaling of these bounds will depend on the specifics of the emulator and system being modeled, but the key point is that a physically consistent emulator should have a well-defined procedure for quantifying these bounds. Furthermore, the emulator should be able to be modified to accommodate additional physical constraints if desired.

2.9 Intrusiveness

Emulators have been broadly grouped into two classes: *intrusive* (or *model-driven*) and *non-intrusive* (or *data-driven*) [22, 23]. The fact that each class has two commonly used names reflects the convergent history from multiple fields independently researching methods which were only later realized to be equivalent.

Definition (Intrusive or Model-Driven Emulator)**.** *An emulator which uses complete knowledge of the associated high-fidelity model is an intrusive or model-driven emulator.*

Definition (Non-Intrusive or Data-Driven Emulator)**.** *An emulator which makes no assump-*

tions about the associated high-fidelity model is a non-intrusive or data-driven emulator.

Intrusiveness describes the structure, or form, of the emulator as well as the data required to fit or train it. Emulators with a static form that does not change when the form of the high-fidelity model changes (i.e. when applying the same emulator to a completely distinct problem) are non-intrusive. For example, ANNs are nearly maximally non-intrusive as their structure always mimics biological neural networks regardless of the data they aim to fit. Although the specific layers and activation functions of the ANN may be changed when applying the method to different problems, those choices are guided purely by the properties of the data and not of the high-fidelity model.

Recent research developments have introduced and increasingly focused on *hybrid* models, which use limited knowledge or assumptions about the high-fidelity model to combine aspects of both data- and model-driven emulators [24]. Perhaps the most well-known hybrid emulators are physics-informed neural networks (PINNs), which combine a data-driven ANN with a "physics-informed" loss function which penalizes violations of equalities required by the high-fidelity model.

Definition (Hybrid Emulator)**.** *In the context of the intrusiveness of an emulator, one which uses limited or selected information about the high-fidelity model and combines aspects of both data-driven and model-driven emulators is called a hybrid emulator.*

All of the three classes presented so far describe emulators which dictate what information (or lack thereof) they require. One might wonder if a class of emulators exists that, rather than specifying precise information and data requirements, instead can be adapted to any amount and type of information. In other words, is there a class of emulators that are always applicable (like data-driven emulators) and can make use of varying amounts of information provided about the high-fidelity model, all the way from limited constraints (like hybrid emulators) to complete access to the high-fidelity model (like intrusive emulators)? Whether

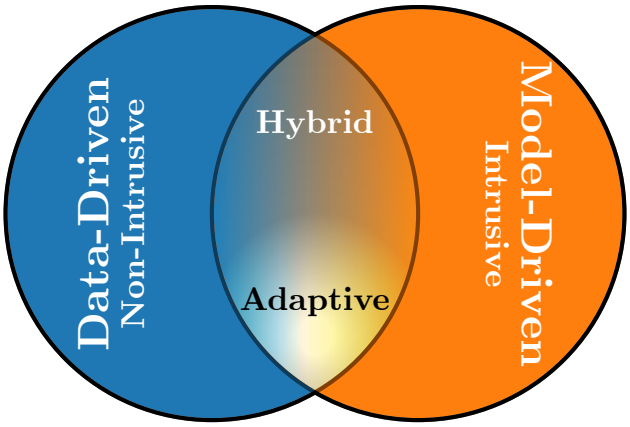

Figure 2.2 Illustration of various classes of emulation techniques. The left circle contains data-driven (non-intrusive) methods, the right circle encompasses model-driven (intrusive) methods, the overlapping region includes hybrid methods with varying degrees of partially data-driven and partially model-driven techniques, and finally a subset of this middle region—highlighted at the bottom—shows adaptive methods which are not fixed to a single point in the diagram but instead have the ability to adapt to be as data-driven or model-driven as required. Inspired by a similar figure in Ref. [22].

or not this class is empty will be left as an open question for now, but we will label such a class as *adaptive* emulators.

Definition (Adaptive Emulator)**.** *In the context of the intrusiveness of an emulator, one which does not have any requirements as to the type or extent of high-fidelity model information necessary to apply the emulator while at the same time being able to effectively use any high-fidelity model information is called an adaptive emulator.*

Figure 2.2 summarizes the relations between the four described classes of emulators, emphasizing the mixing of intrusive and non-intrusive concepts in hybrid emulators and the close relationship between hybrid and adaptive emulators.

Within the context of a given problem, an emulator which makes use of as much information as possible is generally the best one. Remember that the motivation for emulation is the relative lack of data, so any emulator that ignores or forgoes information is at an additional disadvantage. This is why so much focus has been placed on hybrid emulators, as full knowledge of the high-fidelity model is often impossible or impractical but access to limited knowledge or assumptions about it is practically universal.

2.10 Quantifiable Uncertainties

The ability of any machine learning method to communicate its uncertainty is crucial. This once again touches on the idea of trustworthiness. Beyond this, emulators trained on physical data should account for the uncertainty in the data itself and be able to propagate this uncertainty through to the predictions. A phrase often repeated in machine learning and UQ circles is some variation of "a bad model isn't necessarily so if it tells you it's bad." This strikes at the key point: a successful emulator may make mistakes sometimes, but it should be able to give some indication that it is doing so.

It can be difficult to find a single quantitative definition of uncertainty. Many different entirely distinct things can be reported as uncertainties, such as standard deviations, confidence intervals, and relative errors. The best definition of uncertainty comes from the field of metrology, which treats all measurements as coming from some probability distribution around the true value and the uncertainty as some metric for the dispersion of this distribution. With this definition, it is clear that even unusual things like the full width half maximum (FWHM) or the median absolute deviation (MAD) can be the basis of an uncertainty measure.

With such an open-ended definition for uncertainty, it may seem impossible to identify what makes some more useful or more precise than others. Generally, it is better if fewer unproven assumptions about the data and model are made. This is where the ubiquitous standard deviation may break down, as it relies on the assumption that the data and the model predictions come from normal distributions. Uncertainty measurements that correspond directly to a notion of "confidence" are popular for good reason. Again calling back to interpretability and trustworthiness, an uncertainty measure that communicates "X% of the time the value will be within Y of the prediction" is clear and understandable. This is so understandable that it is used for weather forecasts presented to the public, e.g. "20% chance of rain on Saturday."

Altogether, we desire in our emulators the ability—either inherent or as an additional

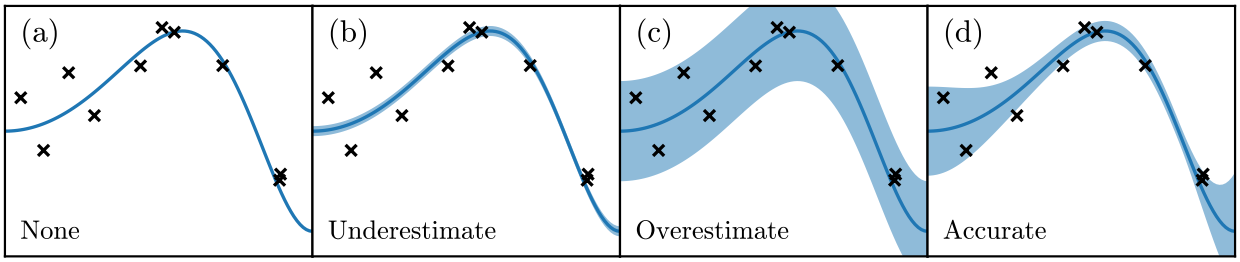

Figure 2.3 Illustration of varying amounts of UQ for a model which is inaccurate on the left half of the data and reasonably accurate on the right half of the data. (a) A model with no UQ. (b) A model with underestimated (or overly optimistic) UQ. (c) A model with overestimated (or overly pessimistic) UQ. (d) A model with accurate (or well-calibrated) UQ that adapts to show the model's different degrees of confidence over the domain.

step—to accurately convey the uncertainty associated with any and all predictions. This uncertainty must be clearly presented and should be easily understandable. Figure 2.3 shows an illustration of different levels of UQ for the same model and data.

2.11 Existing Methods

This section provides a brief overview of some existing methods for emulation in nuclear and many-body physics. Rather than a complete corpus of all methods, only those which are the most widely used (with one exception in Section 2.11.1.3) are included. Each method has its own strengths and weaknesses, and certainly all are useful in specific contexts. However, the topic of this thesis necessarily draws focus to the downsides of each of these methods so that in Chapter 3 they may be addressed by the titular method.

2.11.1 Explicit Reduced Basis Methods (Galerkin's Method)

Referred to in literature simply as reduced basis methods (RBMs) (the cause for the "explicit" modifier will be made clear in Chapter 3), nearly all explicit reduced basis methods (eRBMs) can trace their origins back to what became known in the West as Galerkin's Method or The Galerkin Method [25, 26]. The basic statement of Galerkin's Method is that one can project a high-dimensional (potentially infinite-dimensional) linear problem onto a lower-dimensional subspace and solve the resulting projected problem [27, 28]. In familiar matrix-vector notation, the original problem is that given a vector space $\mathcal{V}$, a linear operator

$A : \mathcal{V} \mapsto \mathcal{V}$ and vector $b \in \mathcal{V}$, find $x \in \mathcal{V}$ such that

$$Ax = b. \tag{2.1}$$

Galerkin realized that one could choose a set of n linearly-independent vectors $\{u_i\}$ in $\mathcal{V}$ representing a basis of a subspace $\mathcal{V}_n \subseteq \mathcal{V}$ and project onto it while retaining the linear structure. We call $\mathcal{V}_n$ the *reduced space* and $\{u_i\}$ the *reduced basis*. Let $\tilde{x} \in \mathcal{V}$ be the projection of x into this subspace, given by

$$\tilde{x} = \sum_i c_i u_i \tag{2.2}$$

where c_i are to-be-determined scalar coefficients. The residual of $\tilde{x}$ is

$$\varepsilon = b - A\tilde{x} = Ax - A\tilde{x} = A(x - \tilde{x}). \tag{2.3}$$

Finally, we impose the Galerkin orthogonality condition on $\tilde{x}$ which states that this residual must be orthogonal to $\mathcal{V}_n$ (i.e. $v_i^\dagger \varepsilon = 0 \, \forall v \in \mathcal{V}_n$).[4] This implies

$$\sum_i c_i u_j^\dagger A u_i = u_j^\dagger b, \tag{2.4}$$

which is an equation with only as many unknowns, c_i, as vectors in our chosen subspace. These relatively few unknowns can be used via Eq. (2.2) to construct an approximate solution to the original problem. Of course, A need not be a matrix and b need not be a vector. Everything above applies identically for any well-posed linear operator acting on abstract objects in a vector space with a consistent inner product. A linear operator A is well-posed if for all $x, y \in \mathcal{V}$ there exist positive constants $\alpha(A)$ and $\beta(A)$ such that

$$\begin{aligned} \left|x^\dagger A y\right| &\leq \alpha(A)\|x\|\|y\| \\ \left|x^\dagger A x\right| &\geq \beta(A)\|x\|^2. \end{aligned} \tag{2.5}$$

If A is a Hermitian matrix, then $\alpha(A)$ is the absolute value of the eigenvalue with largest magnitude and $\beta(A)$ is the absolute value of the eigenvalue with smallest magnitude. Crucially for many of the original applications of this method, derivatives are well-posed linear

[4] Alternatively in Petrov-Galerkin methods, a separate subspace is introduced that is orthogonal to the residual, $\mathcal{W}_n$ [29, 30]. This subspace is sometimes referred to as the test space, while $\mathcal{V}_n$ is referred to as the trial space.

operators over functions and functions live in a vector space with a consistent inner product. Equation (2.1) may represent a differential equation of a continuous function just as much as it may represent a matrix inversion problem.

In the context of this thesis, there is a much more suggestive form of Eq. (2.4). One can rewrite the equation in vector form as

$$\underline{A}\underline{x} = \underline{b} \tag{2.6}$$

where $\underline{x} \equiv [c_1, c_2, \ldots] \in \mathbb{C}^n$ is the vector of to-be-determined coefficients,[5] $\underline{A} \in \mathbb{C}^{n\times n}$ is given by $\underline{A}_{ij} = u_i^\dagger A u_j$, and similarly $\underline{b} \in \mathbb{C}^n$ is given by $\underline{b}_j = u_j^\dagger b$. Regardless of what sort of mathematical objects A, x, and b were originally, their low-dimensional counterparts $\underline{A}$, $\underline{x}$, and $\underline{b}$ are a matrix and two vectors. This is guaranteed simply by the fact that the inner product defined in the original space must return a scalar value and the elements of these low-dimensional representations are defined purely by those inner products. Equation (2.6) is often referred to as the RBM of the original problem in Eq. (2.1).

Now consider parameterizing the projection $\mathbb{P} : \mathcal{V} \mapsto \mathcal{V}_n$ by two operators Q and R,

$$\mathbb{P} \equiv Q\big(R^\dagger Q\big)^{-1} R^\dagger, \tag{2.7}$$

such that

$$\mathbb{P}x = Q\big(R^\dagger Q\big)^{-1} R^\dagger x = \tilde{x}, \tag{2.8}$$

and

$$R^\dagger x = \underline{x}. \tag{2.9}$$

Finally, it is useful to identify and define the *projector*[6] ${}^\downarrow P : \mathcal{V} \mapsto \mathbb{C}^n \equiv R^\dagger$ and its "inverse"[7] $P^\uparrow : \mathbb{C}^n \mapsto \mathcal{V}_n \equiv Q\big(R^\dagger Q\big)^{-1}$. This notation emphasizes the role of these operators: ${}^\downarrow P$

[5]The notation of underlining to denote the Galerkin-projected objects is not arbitrary, but instead inspired by the notation used in Ref. [27] where an overline was used to denote the projected objects. To avoid confusion with the common overline notation used to denote complex conjugation, we use an underline instead.

[6]In broader mathematical contexts, a projector is something that—among other properties—is idempotent ($PP = I$). This is not the case here as the product ${}^\downarrow P {}^\downarrow P$ is not even defined. In general machine learning this object is often referred to as a "linear encoder".

[7]Likewise, $P^\uparrow$ is not a true inverse since $P^\uparrow {}^\downarrow P \neq I$ (but ${}^\downarrow P P^\uparrow = I$ is always true by definition). As the subspace approaches the full space ($\mathcal{V}_n \to \mathcal{V}$) it behaves more and more like an inverse ($P^\uparrow {}^\downarrow P \to I$). In general machine learning this object is often referred to as the "linear decoder" associated with ${}^\downarrow P$.

projects objects "down" into the reduced space and $P^{\uparrow}$ projects back "up" into the original space. ${}^{\downarrow}P$ may be said as "P-down" and $P^{\uparrow}$ may be said as "P-up". With these definitions we are able to write Eqs. (2.4) and (2.6) as

$$ {}^{\downarrow}PAP^{\uparrow}{}^{\downarrow}Px = {}^{\downarrow}Pb $$

or,

$$ \underline{A}\underline{x} = \underline{b} $$

where

$$ \begin{aligned} \underline{A} &\equiv {}^{\downarrow}PAP^{\uparrow} \\ \underline{x} &\equiv {}^{\downarrow}Px \\ \underline{b} &\equiv {}^{\downarrow}Pb \end{aligned} \tag{2.10} $$

and

$$ x \approx \tilde{x} = P^{\uparrow}\underline{x} $$

For Galerkin methods, R and Q are equal.[8] In the case that the $\{u_i\}$ are vectors and the inner product is the usual vector dot product, then both R and Q are the matrices whose columns are the $\{u_i\}$. This makes ${}^{\downarrow}P$ the matrix whose rows are $\left\{u_i^{\dagger}\right\}$ and $P^{\uparrow}$ the matrix whose columns are $\{u_i\}$ normalized by their inner products, $P^{\uparrow} = R\left(R^{\dagger}R\right)^{-1} = {}^{\downarrow}P^{\dagger}\left({}^{\downarrow}P{}^{\downarrow}P^{\dagger}\right)^{-1}$.

Owing to the relative simplicity of Galerkin's Method, one can derive error bounds on the error between the approximate solution $\tilde{x}$ and the true solution x

$$ \|x - \tilde{x}\| \leq \kappa(A) \inf_{v \in \mathcal{V}_n} \|x - v\| \tag{2.11} $$

for positive constant $\kappa(A) = \alpha(A)/\beta(A)$ which is the condition number of A [31]. That is, the approximate solution from Galerkin's Method is at least as good as any vector in the space spanned by the chosen $\{u_i\}$, up to a constant. Additionally, this proves that Galerkin's Method is systematically improvable, as successively increasing the dimensionality of $\mathcal{V}_n$ by adding new u_i that have at least some component which is linearly independent of all previous

[8] Petrov-Galerkin methods have $R \neq Q$ [29, 30].

$\{u_i\}$, thereby driving $\mathcal{V}_n \to \mathcal{V}$, the error of the approximation goes to 0, reaching 0 when $\mathcal{V}_n = \mathcal{V}$. The rate of this convergence depends on both the specific A and chosen $\{u_i\}$.

We can additionally quantify the computational complexity of Galerkin's Method. We denote the dimensionality of the original space $\mathcal{V}$ by N and assume that Ax and $x^\dagger y$ can be computed in $\mathcal{O}(N^2)$ and $\mathcal{O}(N)$ time respectively, for all $x, y \in \mathcal{V}$, regardless of what type of mathematical objects x, y, and A represent. Then the offline cost of Galerkin's Method is comprised only of the $\mathcal{O}(N^2 n)$ and $\mathcal{O}(N^2)$ time of computing $\underline{A}$ and $\underline{b}$ respectively. A precise complexity of the online phase would depend on the method used to solve the linear problem, but even naive approaches would be at most $\mathcal{O}(n^3)$. The online phase would also include transforming $\underline{x}$ to $\tilde{x}$, which requires $\mathcal{O}(Nn)$ time. The vectors $\{u_i\}$ would require $\mathcal{O}(Nn)$ space both in the offline and online phases. So, assuming $n \ll N$, the offline costs are $\mathcal{O}(N^2 n)$ time and $\mathcal{O}(Nn)$ space and the online costs are $\mathcal{O}(Nn)$ time and $\mathcal{O}(Nn)$ space.

Galerkin's Method allows us to take a high- or infinite-dimensional linear problem and approximate it by a smaller linear problem with a dimensionality of our choosing. Importantly, this applies equally to *parametric* linear problems—those where the problem itself is parameterized by some list of parameters, $\mu \equiv [\mu_1, \mu_2, \ldots]$. The most general form of such a problem is

$$A(\mu)x(\mu) = b(\mu). \tag{2.12}$$

Consider the simplest case where μ is a scalar and A is an affine combination of two linear operators A_0 and A_1,

$$A(\mu)x(\mu) = (A_0 + \mu A_1)x(\mu) = b. \tag{2.13}$$

The application of Eq. (2.10) is unchanged from before,

$$\begin{aligned} & {}^{\downarrow}P A(\mu) P^{\uparrow} {}^{\downarrow}P x(\mu) = {}^{\downarrow}P b \\ \iff & \left({}^{\downarrow}P A_0 P^{\uparrow} + \mu {}^{\downarrow}P A_1 P^{\uparrow}\right) {}^{\downarrow}P x(\mu) = {}^{\downarrow}P b \\ \iff & \left(\underline{A_0} + \mu \underline{A_1}\right) \underline{x(\mu)} = \underline{b}. \end{aligned} \tag{2.14}$$

This generalizes to significantly more complex parameterizations where μ can represent an entire array of parameters as well as parameterizations which are nonlinear in μ, so long as

$A(\mu)$ is a linear operator for all μ and $A(\mu)$ and $b(\mu)$ are expressible linearly in terms of any functions of μ.

For

$$A(\mu)x(\mu) = b(\mu)$$

with

$$A(\mu) \equiv \sum_i f_i(\mu) A_i$$

$$b(\mu) \equiv \sum_j g_j(\mu) b_j$$

Galerkin's Method yields

$$\begin{aligned} & \underline{A(\mu)}\,\underline{x(\mu)} = \underline{b(\mu)} \\ \iff & \left(\sum_i f_i(\mu)\underline{A_i}\right)\underline{x(\mu)} = \sum_j g_j(\mu)\underline{b_j} \end{aligned} \tag{2.15}$$

where

$$\underline{A_i} \equiv {}^{\downarrow}P A_i P^{\uparrow}$$

$$\underline{b_j} \equiv {}^{\downarrow}P b_j$$

and

$$x(\mu) \approx \tilde{x}(\mu) = P^{\uparrow}\underline{x(\mu)}$$

Note that no assumption on the parametric dependence of x on μ was made. Although the problem itself is a linear one for fixed μ, the dependence of the problem may be highly nonlinear in μ, and thus the solution $x(\mu)$ may be arbitrarily nonlinear in μ. The distinction between the linearity of the problem and the linearity of the parameterization is an important one that will appear several times throughout this thesis.

One can see that Galerkin's Method does not change the form or structure of the original problem. This allows for the transformed equation to be interpretable in much the same way as the original problem. It also allows for secondary operators or equations, such as those corresponding to conservation or symmetry laws, to be projected in the same way using the

same projector in order for the degree to which they are conserved to be analyzed just as the approximate solution $\tilde{x}$ was.

Table 2.2 contains a brief summary of each of the relevant emulator properties as they relate to Galerkin's Method.

What Galerkin's Method stops short of is providing guidance on how to find or choose the basis vectors that define the projection, how to proceed when the linear operator, A, or inhomogeneous part of the problem, b, are not fully known, how the method generalizes to nonlinear problems, if at all, or how to apply the method when the problem setup is not exactly like that in Eq. (2.1). The methods discussed below can fundamentally be equated to some formulation of Galerkin's Method but are distinguished by their approaches to each of these unanswered questions.

Emulator Summary for Galerkin's Method	
Mathematically Parametric	Yes, so long as the form of the parametric dependence is known.
Numerically Parametric	No. Reduced objects have dimensions corresponding to the subspace $\mathcal{V}_n$.
Nonlinear-Capable	Strictly speaking, no. Some extensions to POD-Galerkin exist (see Section 2.11.1.1) for some nonlinear systems.
Computationally Efficient	Dependent on the cost of the high-fidelity linear operator and inner product. If those time complexities are $\mathcal{O}(A(N))$ and $\mathcal{O}(\text{dot}(N))$ respectively and have space complexity $\mathcal{O}(N)$ (typical of linear operators and inner products), then the method has $\mathcal{O}(A(N)\text{dot}(N)n)$ time and $\mathcal{O}(N+n^2)$ space offline complexity and $\mathcal{O}(n^3)$ time and $\mathcal{O}(n)$ space online complexity when using standard iterative solvers. An additional $\mathcal{O}(Nn)$ online time cost is incurred when projecting the Galerkin solution back to the original space $\mathcal{V}$.
Data Efficient	Heavily dependent on the choice of the subspace $\mathcal{V}_n$.
Hyperparameter Efficient	The only potential hyperparameter is the process one chooses to form $\mathcal{V}_n$ from available data.
Systematically Improvable	Yes. Convergence rate heavily dependent on the specific choice of successive $\mathcal{V}_n$.
Interpretable	Yes. The projected system has a form identical to the original.
Physically Consistent	Tied directly to the accuracy of the reduced equations, which depends heavily on $\mathcal{V}_n$.
Intrusiveness	Maximally intrusive. Requires computing ${}^{\downarrow}PAP^{\uparrow}$ and ${}^{\downarrow}Pb$, which requires full knowledge of the high-fidelity model.
Quantified Uncertainties	Not inherent. Can be added on top of the method. Error bounds and knowledge of the high-fidelity model in conjunction with $\mathcal{V}_n$ may greatly improve the accuracy of uncertainty quantification.

Table 2.2 Summary of properties for emulators based on Galerkin's Method.

2.11.1.1 Proper Orthogonal Decomposition

Originally introduced in the field of computational fluid dynamics, proper orthogonal decomposition (POD) guides the selection of the reduced space $\mathcal{V}_n$ which is then used via Galerkin's Method to create a reduced-order model [32, 33]. This method is often referred to as the POD-Galerkin method. The context for POD is a vector-valued parametric linear problem of the form in Eq. (2.12)—often, the operator is constant and the only parameter is time—with some vector data that are solution *snapshots*, that is, a set of solutions for some set of parameter realizations. Let $\mu^{(i)}$ denote a specific realization for the parameters of the problem, then $x\big(\mu^{(i)}\big)$ which satisfies Eq. (2.12) with $\mu = \mu^{(i)}$ is the snapshot of the system for parameters $\mu^{(i)}$. Then the data available to the POD method are the m snapshots $\big\{x\big(\mu^{(1)}\big), x\big(\mu^{(2)}\big), \ldots, x\big(\mu^{(m)}\big)\big\}$ and associated parameter realizations $\big\{\mu^{(1)}, \mu^{(2)}, \ldots, \mu^{(m)}\big\}$. In the simplest implementation, one forms the $N \times m$ matrix X whose columns are the snapshots and computes its singular value decomposition (SVD)

$$X = U\Sigma V^\dagger. \tag{2.16}$$

By convention the singular values and singular vectors are ordered by decreasing singular value, so $\Sigma = \operatorname{diag}(\sigma_1, \sigma_2, \ldots)$ with $\sigma_1 \geq \sigma_2 \geq \cdots$. The matrix whose columns are the first n columns of U—often denoted as $U[:, :n]$, $U[:, 1:n]$, or $U(:, 1:n)$—is then taken to be $P^\uparrow$. That is, the n orthonormal left singular vectors of X corresponding to the n largest singular values of X are the $\{u_i\}$ that form a reduced basis via Galerkin's Method. Since the columns of $P^\uparrow$ are orthonormal, $\big(P^\uparrow\big)^\dagger P^\uparrow = I_n$ and so ${}^\downarrow P = \big(P^\uparrow\big)^\dagger$. For notational simplicity, it is common to use the notation $P \equiv P^\uparrow$ and work in terms of P and $P^\dagger$. Computing P can be nontrivial or even intractable via the exact SVD if the size or number of snapshots is too great. Several approximate or iterative methods exist to find P in such cases. POD has close ties to PCA; the columns of P are exactly the principal components of the snapshots. This allows the number of components n to be automatically determined from a specified desired cumulative explained variance—that is, the proportion of the variance in the snapshots that is retained under the projection $PP^\dagger$. In fact, the Eckart-Young-Mirsky theorem proves that

the best possible rank-n approximation to the matrix of snapshots is the one given by PCA, $X_n = PP^\dagger X$ [34, 35].

Orthogonal projections have several properties that make them particularly attractive for the basis of an RBM. Perhaps the most important is the preservation of inner products. For any two vectors u and v, we have $u^\dagger v \approx \tilde{u}^\dagger \tilde{v} = (P\underline{u})^\dagger P\underline{v} = \underline{u}^\dagger \big(P^\dagger P\big)\underline{v} = \underline{u}^\dagger \underline{v}$. What makes this property so important is that it renders projecting back to the full space unnecessary if the quantity of interest is a scalar which is the result of only inner products—all because $\tilde{u}^\dagger \tilde{v} = \underline{u}^\dagger \underline{v}$. This includes eigenvalues, singular values, ℓ_2 norms, expectation values, and much more. For operators, Hermiticity (or symmetry in the case of real-valued operators and projectors) is exactly preserved purely from the semi-unitarity ($P^\dagger P = I$) of the projector. For Hermitian operators, the Poincaré separation theorem[9] guarantees that if $\lambda_i(A)$ is the i^{th} eigenvalue of $A \in \mathbb{H}(N)$ and $\lambda_j(\underline{A})$ is the j^{th} eigenvalue of $\underline{A} \in \mathbb{H}(n)$ with both sets of eigenvalues in ascending order ($\lambda_1 \leq \lambda_2 \leq \cdots$) then

$$\lambda_j(A) \leq \lambda_j(\underline{A}) \leq \lambda_{N-n+j}(A) \tag{2.17}$$

for $j = 1, 2, \ldots, n$ [36–38]. This result can be generalized to the singular values of any matrix[10]

$$\sigma_j(A) \geq \sigma_j(\underline{A}) \geq \sigma_{2(N-n)+j}(A) \tag{2.18}$$

where the singular values are in the standard descending order $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$ and $j = 1, 2, \ldots, n$ [37, 39]. Equations (2.17) and (2.18) guarantee too many exact and approximate conservations between the full and reduced models to exhaustively list here. Some of the most important properties are listed below.

- The condition number, $\kappa(\cdot)$, of $\underline{A}$ is at most the condition number of A—that is, $1 \leq \kappa(\underline{A}) \leq \kappa(A)$. This guarantees that the reduced model will be either numerically easier or no harder than the original.

[9] Also referred to as the Cauchy interlacing theorem due to the result of the special case where $n = N-1$.

[10] It can also be generalized to transformations of the form ${}^\downarrow P A P^\uparrow$ where ${}^\downarrow P$ and $P^\uparrow$ are both semi-unitary, possibly with differing dimensions.

- The definiteness of A and $\underline{A}$ are identical—unless A is indefinite in which case $\underline{A}$ may have any definiteness.

- For any unitarily-invariant norm $\|\cdot\|$, the norm of $\underline{A}$ is bounded by that of A. That is, $\|A\| \geq \|\underline{A}\|$. This includes the spectral norm,[11] Frobenius norm, nuclear norm,[12] and all Schatten p-norms [40].

The POD-Galerkin method can be naively applied to nonlinear systems in the following way. Consider a general parametric nonlinear problem

$$A(\mu)x(\mu) + F(x(\mu);\mu) = b(\mu), \tag{2.19}$$

where $F(\,\cdot\,;\mu) : \mathcal{V} \to \mathcal{V}$ is the nonlinear part. Applying POD-Galerkin to this problem yields the partially-reduced model

$$\begin{aligned} P^\dagger A(\mu) P P^\dagger x(\mu) + P^\dagger F\big(P P^\dagger x(\mu);\mu\big) &= P^\dagger b(\mu) \\ \iff \underline{A(\mu)}\,\underline{x(\mu)} + P^\dagger F\Big(P\underline{x(\mu)};\mu\Big) &= \underline{b(\mu)}. \end{aligned} \tag{2.20}$$

This is "partially" reduced because the evaluation of the nonlinear term cannot directly be reduced without more specific knowledge about it. Evaluating this term requires projecting back to the full space, evaluating the full-dimensional nonlinearity, and then projecting back down—a process which alone is at least $\mathcal{O}(C_F(N) + Nn^2)$, where $C_F(N)$ is the complexity of evaluating F on a size-N input. In practice, even the simplest nonlinearities have complexities no better than $\mathcal{O}(N)$. Combined with the cost of solving Eq. (2.20) with an iterative solver, this means that an emulator formed in this fashion would have an online time complexity exceeding $\mathcal{O}(C_F(N)n + Nn^3)$. When applied this way, a POD-Galerkin emulator for a nonlinear problem can actually be just as, if not more, expensive than the full high-fidelity model [41–43]. The usual workaround for this problem is an approximation of $F(\,\cdot\,;\mu)$ on top of the existing reduced-basis approximation. This additional approximation

[11] Also known as the induced ℓ_2-norm, $\|\cdot\|_2$. For Hermitian matrices this is equal to the maximum singular value.

[12] Also called the trace norm.

is referred to as a "hyper-reduction" and the most popular technique is the discrete empirical interpolation method (DEIM) [41].

DEIM assumes that the nonlinear term is evaluated *elementwise*. That is, if x is the vector $[x_1, x_2, \ldots, x_N]$ then $F(x) = [F(x_1), F(x_2), \ldots, F(x_N)]$. While seemingly a limiting assumption, many useful and realistic nonlinear terms behave this way. However, DEIM also assumes that x and $F(x)$ are component-wise smooth—which can be a very limiting assumption. The method avoids scaling with the size of the full space, N, by approximating F by utilizing not just snapshots of $x(\mu)$ but also of $F(x(\mu); \mu)$ (which usually incur no additional cost as they are required for the high-fidelity model to produce the snapshots of x). During the online phase, x is "downsampled" by selecting a fixed set of only $m \ll N$ indices to evaluate F at and DEIM then attempts to linearly interpolate the rest of the indices based on the set of example nonlinear snapshots. This is where the component-wise smoothness assumption becomes necessary. Written out explicitly, denote the $N \times m$ matrix whose columns are the standard one-hot basis vectors which encode the chosen m indices by

$$B = \begin{bmatrix} | & | & & | \\ e_{d_1} & e_{d_2} & \cdots & e_{d_m} \\ | & | & & | \end{bmatrix}, \tag{2.21}$$

where d_i are the chosen m indices, and e_j is the j^{th} standard basis vector. B selects the encoded indices of $x(\mu)$ from $\underline{x(\mu)}$ simply by

$$B^\dagger P \underline{x(\mu)}. \tag{2.22}$$

Note that $B^\dagger P$ can be precomputed in the offline phase by taking the d_i columns of P and is only $m \times n$. The elementwise nonlinearity is then evaluated over the m components of this vector and interpolated using the basis spanned by the m nonlinear snapshots. Finally, the result is projected back down to the original size-n POD basis. Denote the $m \times N$ Galerkin projector for the nonlinear snapshots—that is the matrix whose columns are the nonlinear

snapshots—by Q, then DEIM approximates the nonlinear term in Eq. (2.20) by

$$P^\dagger F\Big(P\underline{x(\mu)};\mu\Big) \approx P^\dagger \underbrace{Q\big(B^\dagger Q\big)^{-1}}_{\text{Empirical interpolation}} F\Big(\underbrace{B^\dagger}_{\text{Select } m \text{ indices}} \underbrace{P\underline{x(\mu)}}_{\text{Project to full space}};\mu\Big). \tag{2.23}$$

Project back down to POD-Galerkin basis

Again note that $P^\dagger Q\big(B^\dagger Q\big)^{-1}$ is an $n \times m$ matrix which can be precomputed during the offline phase, just like $B^\dagger P$. Utilizing these precomputations, the cost of evaluating Eq. (2.23) is only $\mathcal{O}(C_F(m) + nm)$, now completely independent of the size of the high-fidelity model. Thus from the extra assumptions that F is an elementwise nonlinearity and x and $F(x)$ are component-wise smooth and the use of additional data that is usually already available, DEIM is able to efficiently extend the POD-Galerkin method to a wide array of nonlinear problems. Further details and analysis of DEIM can be found in Ref. [41].

DEIM is not without its downsides. For strongly nonlinear problems—those where the contribution of the nonlinear terms is comparable or potentially even greater than that of the linear terms—it can be the case that the required number of nonlinear snapshots, m, approaches the size of the high-fidelity model, N [43, 44]. Additionally, introducing a second independent approximation scheme on top of POD-Galerkin complicates convergence, stability, and improvability analyses.

To handle nonlinear terms without additional approximations, the concept of lifting transformations was proposed in Ref. [45]. Lifting transformations introduce auxiliary variables to transform the original form of the problem to one which is more amenable to POD-Galerkin. This is particularly powerful in emulators for systems of nonlinear differential equations [43, 45]. First, we note that polynomial nonlinearities can be handled directly by the standard POD-Galerkin method. Consider the quartic function

$$F(x) = Ax + Bx^2 + Cx^3 + Dx^4 \tag{2.24}$$

where A, B, C, D are linear operators and $x^k \equiv x \odot \overset{k}{\cdots} \odot x$ denotes the elementwise k^{th} power

of x. We can directly apply POD-Galerkin to this nonlinear function as in Eq. (2.20),

$$\begin{aligned}
P^\dagger F(x) &\approx P^\dagger F(P\underline{x}) \\
&= P^\dagger A P\underline{x} + P^\dagger B[(P\underline{x}) \odot (P\underline{x})] \\
&\qquad + P^\dagger C[(P\underline{x}) \odot (P\underline{x}) \odot (P\underline{x})] \\
&\qquad + P^\dagger D[(P\underline{x}) \odot (P\underline{x}) \odot (P\underline{x}) \odot (P\underline{x})] \\
\iff \left(P^\dagger F(x)\right)_i &\approx \underline{A}_{i\mu}\underline{x}_\mu + \underbrace{\left[P^\dagger{}_{i\nu} B_{\nu\mu} P_{\mu\sigma} P_{\mu\omega}\right]}_{\underline{B}} \underbrace{\underline{x}_\sigma \underline{x}_\omega}_{\underline{x}\otimes\underline{x}} \\
&\qquad + \underbrace{\left[P^\dagger{}_{i\nu} C_{\nu\mu} P_{\mu\sigma} P_{\mu\omega} P_{\mu\alpha}\right]}_{\underline{C}} \underbrace{\underline{x}_\sigma \underline{x}_\omega \underline{x}_\alpha}_{\underline{x}\otimes\underline{x}\otimes\underline{x}} \\
&\qquad + \underbrace{\left[P^\dagger{}_{i\nu} D_{\nu\mu} P_{\mu\sigma} P_{\mu\omega} P_{\mu\alpha} P_{\mu\beta}\right]}_{\underline{D}} \underbrace{\underline{x}_\sigma \underline{x}_\omega \underline{x}_\alpha \underline{x}_\beta}_{\underline{x}\otimes\underline{x}\otimes\underline{x}\otimes\underline{x}} \\
\iff P^\dagger F(x) &\approx \underline{A}\underline{x} + \underline{B}(\underline{x}\otimes\underline{x}) + \underline{C}(\underline{x}\otimes\underline{x}\otimes\underline{x}) + \underline{D}(\underline{x}\otimes\underline{x}\otimes\underline{x}\otimes\underline{x}),
\end{aligned} \tag{2.25}$$

where $\otimes$ denotes the Kronecker product and the third step is written in Einstein summation notation for clarity, with all indices denoted by Greek letters being summed over. $\underline{B}$, $\underline{C}$, $\underline{D}$ are rank $3, 4, 5$ tensors respectively and are contracted over the Greek indices with the corresponding rank $2, 3, 4$ ($\underline{x}\otimes\cdots\otimes\underline{x}$). It is not uncommon to reshape the tensors to $n \times n^k$ matrices and the ($\underline{x}\otimes\cdots\otimes\underline{x}$) to n^k vectors, where $k = 2$, 3, 4, so that all operations in the last line of Eq. (2.25) are matrix-vector multiplications. All the operators, $\underline{A}$, $\underline{B}$, $\underline{C}$, and $\underline{D}$ can be precomputed during the offline phase at costs of $\mathcal{O}(N^2n^2)$, $\mathcal{O}(N^2n^3)$, $\mathcal{O}(N^2n^4)$, and $\mathcal{O}(N^2n^5)$ respectively and the online cost of evaluating this reduced form of the nonlinearity is $\mathcal{O}(n^5)$. In principle, any polynomial nonlinearity can be handled this way, but in practice it is often more beneficial to apply lifting transformations to limit the order of the polynomial.

To illustrate how auxiliary variables and lifting transformations can make an arbitrary nonlinearity into a polynomial one, consider the following toy problem,

$$\frac{dx}{dt} = Ax + B\sin(x). \tag{2.26}$$

Using the auxiliary variables $y = \sin(x)$ and $z = \cos(x)$, we are able to lift the problem from

a nonlinear one of size N to a quadratic one of size $3N$,

$$\begin{cases} \dfrac{dx}{dt} = Ax + By \\ \dfrac{dy}{dt} = \cos(x) \odot \dfrac{dx}{dt} = z \odot (Ax + By) \\ \dfrac{dz}{dt} = -\sin(x) \odot \dfrac{dx}{dt} = -y \odot (Ax + By) \end{cases} \tag{2.27}$$

which in matrix form is

$$\begin{aligned} \frac{d}{dt}\underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{v} &= \underbrace{\begin{bmatrix} A & B & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\mathcal{A}}\begin{bmatrix} x \\ y \\ z \end{bmatrix} + \left(\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}}_{\mathcal{B}}\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) \odot \left(\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ A & B & 0 \\ A & B & 0 \end{bmatrix}}_{\mathcal{C}}\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) \\ \iff \frac{dv}{dt} &= \mathcal{A}v + (\mathcal{B}v) \odot (\mathcal{C}v), \end{aligned} \tag{2.28}$$

which can then be reduced via POD-Galerkin for polynomial nonlinearities in a few different ways. The first is to construct the projector from stacked snapshots of the original and auxiliary variables $v = [x\, y\, z]^T$ and project Eq. (2.28) as

$$\begin{aligned} \frac{d\underline{v}}{dt} &= \underline{\mathcal{A}}\underline{v} + P^\dagger([P\underline{\mathcal{B}}\underline{v}] \odot [P\underline{\mathcal{C}}\underline{v}]) \\ \iff \left(\frac{d\underline{v}}{dt}\right)_i &= \underline{\mathcal{A}}_{i\mu}\underline{v}_\mu + {P^\dagger}_{i\mu}P_{\mu\nu}\underline{\mathcal{B}}_{\nu\omega}\underline{v}_\omega P_{\mu\sigma}\underline{\mathcal{C}}_{\sigma\lambda}\underline{v}_\lambda \\ &= \underline{\mathcal{A}}_{i\mu}\underline{v}_\mu + \underbrace{{P^\dagger}_{i\mu}P_{\mu\nu}P_{\mu\sigma}}_{\underline{G}^{(2)}}\underbrace{\underline{\mathcal{B}}_{\nu\omega}\underline{v}_\omega\underline{\mathcal{C}}_{\sigma\lambda}\underline{v}_\lambda}_{(\underline{\mathcal{B}}\underline{v})\otimes(\underline{\mathcal{C}}\underline{v})} \\ \iff \frac{d\underline{v}}{dt} &= \underline{\mathcal{A}}\underline{v} + \underline{G}^{(2)}[(\underline{\mathcal{B}}\underline{v}) \otimes (\underline{\mathcal{C}}\underline{v})]. \end{aligned} \tag{2.29}$$

The second way treats snapshots of x, y, and z on equal footing, considering them essentially as independent snapshots of the same variable, and constructing the POD projector from

this data and reducing Eq. (2.27) directly

$$
\begin{aligned}
&\begin{cases}
\dfrac{d\underline{x}}{dt} = \underline{A}\underline{x} + \underline{B}\underline{y} \\
\dfrac{d\underline{y}}{dt} = P^{\dagger}\big([P\underline{z}] \odot \big[P\big(\underline{A}\underline{x} + \underline{B}\underline{y}\big)\big]\big) \\
\dfrac{d\underline{z}}{dt} = -P^{\dagger}\big(\big[P\underline{y}\big] \odot \big[P\big(\underline{A}\underline{x} + \underline{B}\underline{y}\big)\big]\big)
\end{cases} \\
\iff &\begin{cases}
\dfrac{d\underline{x}}{dt} = \underline{A}\underline{x} + \underline{B}\underline{y} \\
\left(\dfrac{d\underline{y}}{dt}\right)_i = \underbrace{P^{\dagger}{}_{i\mu} P_{\mu\nu} P_{\mu\omega}}_{\underline{G}^{(2)}} \underbrace{\underline{z}_{\nu}\big(\underline{A}\underline{x} + \underline{B}\underline{y}\big)_{\omega}}_{\underline{z}\otimes\left(\underline{A}\underline{x}+\underline{B}\underline{y}\right)} \\
\left(\dfrac{d\underline{z}}{dt}\right)_i = -\underbrace{P^{\dagger}{}_{i\mu} P_{\mu\nu} P_{\mu\omega}}_{\underline{G}^{(2)}} \underbrace{\underline{y}_{\nu}\big(\underline{A}\underline{x} + \underline{B}\underline{y}\big)_{\omega}}_{\underline{y}\otimes\left(\underline{A}\underline{x}+\underline{B}\underline{y}\right)}
\end{cases} \\
\iff &\begin{cases}
\dfrac{d\underline{x}}{dt} = \underline{A}\underline{x} + \underline{B}\underline{y} \\
\dfrac{d\underline{y}}{dt} = \underline{G}^{(2)}\big[\underline{z} \otimes \big(\underline{A}\underline{x} + \underline{B}\underline{y}\big)\big] \\
\dfrac{d\underline{z}}{dt} = -\underline{G}^{(2)}\big[\underline{y} \otimes \big(\underline{A}\underline{x} + \underline{B}\underline{y}\big)\big].
\end{cases}
\end{aligned}
\tag{2.30}
$$

There is a third option involving multiple projectors, one for each of the original and auxiliary variables (i.e. P_x, P_y, P_z). For an example of this technique see the Appendix of Ref. [46]. Which of the three methods to use comes down to problem specifics and the choices of auxiliary variables. Often, multiple options are tried and the one with the best performance on some held-out data is selected.

This example is somewhat contrived, and it may seem unlikely that such a small number of simple variable substitutions could lift an arbitrary problem to polynomial form. Nevertheless, it is known that the number of auxiliary variables necessary to transform any nonlinear problem to a polynomially nonlinear one is proportional to the number of elementary nonlinear functions in the problem [45]. Thus the size of the lifted problem is $\mathcal{O}(n_f N)$ where n_f is the number of elementary nonlinear functions. This linear factor is manageable by the POD-Galerkin reduction that follows.

Error and convergence analysis of POD-Galerkin comes down to the convergence of PCA. We first derive the error between X (the $N \times m$ matrix of training snapshots from

Eq. (2.16)) and the associated rank-n PCA approximation $X_n = PP^\dagger X$. By the definition of the 2-norm, it is straightforward to verify that the error in this approximation is exactly $\left\|X - PP^\dagger X\right\|_2^2 = \sigma_{n+1}^2$. Any vector $v \in \mathcal{V}$, can be written as a sum of the part that is in the column space of X and the part perpendicular to that space: $v = v_X + v_\perp$ where $v_X = Xw$ for some w. The error from the PCA projection will serve as a rough upper bound for the error of the POD-Galerkin method,

$$\begin{aligned}\left\|v - PP^\dagger v\right\|_2^2 &= \left\|Xw - PP^\dagger Xw\right\|_2^2 + \left\|v_\perp - PP^\dagger v_\perp\right\|_2^2 \\ &\leq \left\|X - PP^\dagger X\right\|_2^2 \|w\|_2^2 + \|v_\perp\|_2^2 \\ &= \sigma_{n+1}^2 \|w\|_2^2 + \|v_\perp\|_2^2.\end{aligned} \tag{2.31}$$

Thus for vectors that are almost exactly normalized linear combinations of the snapshots (pure linear interpolation), in which case $\|w\|_2^2 \approx 1$ and $\|v_\perp\|_2^2 \approx 0$, the error is approximately bounded by σ_{n+1}^2. However, if we assume that the snapshots were drawn from some distribution and this test vector v is drawn from the same distribution, then we have an expected relative error given by

$$\mathbb{E}\left[\frac{\left\|v - PP^\dagger v\right\|_2^2}{\|v\|_2^2}\right] = \frac{\sum_{i=n+1}^N \sigma_i^2}{\sum_{i=1}^N \sigma_i^2}, \tag{2.32}$$

which is exactly 1 minus the cumulative explained variance of the PCA projector [47]. This is a reasonable assumption that holds well in nearly all practical applications—one essentially just ensures that the snapshots are representative of the space of interest. Another in-practice truth is that the singular values of X decay exponentially or as a power-law. This is not true of an arbitrary X and many pathological examples can be made, such as random matrices, where the singular values do not decay at all. However, it is overwhelmingly true in practice that either

$$\sigma_k \sim \sigma_1 \alpha^{-\beta(k-1)} \quad \text{or} \quad \sigma_k \sim \sigma_1 k^{-\gamma}, \tag{2.33}$$

for some constants α, β, or γ. This fact is why POD-Galerkin emulators can be so accurate with $n \ll N$. Figure 2.4 shows the singular values and cumulative explained variance as a

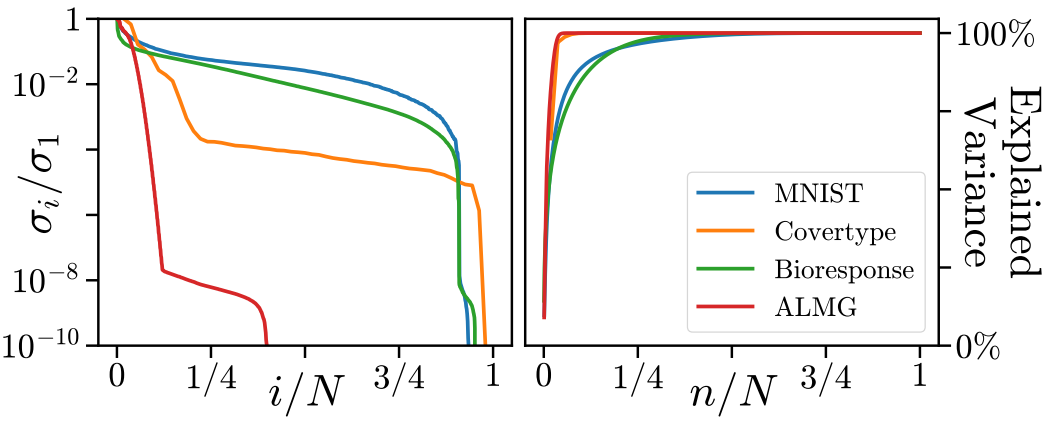

Figure 2.4 Singular values and cumulative explained variance for four real-world datasets: images of handwritten digits (MNIST) [48], cartographic data from the US Geological Survey and US Forest Service (Covertype) [49], molecular descriptors (Bioresponse) [50], and low-lying energy eigenstates of the anharmonic Lipkin–Meshkov–Glick (LMG) model for various coupling strengths (ALMG) [51, 52]. The left plot shows the normalized singular value as a function of the normalized index i/N where N is the dimensionality of the data (the number of features). The right plot shows the cumulative explained variance from the first n principal components as a function of n/N.

function of n/N for various real-world datasets spanning applications in image recognition, forestry, biochemistry, and quantum mechanics.

Example

Consider the problem of diffusion in one dimension, which is governed by the partial differential equation

$$\begin{aligned} \frac{\partial u}{\partial t} &= \alpha \frac{\partial^2 u}{\partial x^2}, \\ u(x, t = 0) &= u_0(x), \end{aligned} \tag{2.34}$$

for $0 < x < L$, $t > 0$, and diffusivity parameter $\alpha > 0$. A typical context is that where the boundaries are held fixed, which imposes Dirichlet boundary conditions,

$$\left.\frac{\partial u}{\partial t}\right|_{x=0} = \left.\frac{\partial u}{\partial t}\right|_{x=L} = 0 \tag{2.35}$$

Typical numerical methods will discretize the system over the spatial dimension, rendering the problem a system of coupled ordinary differential equations in time. Choosing a uniform grid of N parts over x makes u a size-N vector and the spatial derivative an $N \times N$ matrix

representing the finite difference approximation of the derivative with spacing $L/(N-1)$.

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{N-1}{L}\right)^2 \underbrace{\begin{bmatrix} 0 & & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & & \ddots & & & & \\ & & & 1 & -2 & 1 & & \\ & & & & 1 & -2 & 1 & \\ & & & & & & & 0 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{u} \tag{2.36}$$

$$\iff \frac{\partial u}{\partial t} = \alpha D u,$$

where we have used the Dirichlet boundary condition at the boundaries, the central finite difference everywhere else, and the elements of u relate to the original continuous problem by $u_k(t) \equiv u\big(x = \frac{kL}{N-1}, t\big)$, $k = 0, 1, \ldots, N-1$. We can now solve for $u_k(t)$ for any N, t, and $u_0(x)$ via numerical integration.

This process so far is the typical numerical treatment of partial differential equations. The form of the problem is now amenable to POD-Galerkin parameterized by α. Suppose our goal is to find the evolution of a Gaussian $u_0(x)$ for some α_{test}, but due to stability constraints with numerical integration we are only able to run the high-fidelity simulation for $\alpha \leq \alpha_{\text{test}}/2$. Choosing $N = 1000$ for a sufficiently high-resolution high-fidelity solution, we run just five full simulations using a matrix-free implementation[13] at uniformly spaced sample values of α between 0.1 and $\alpha_{\text{test}}/2$ for N_t timesteps from 0 to our desired final time T. From these high-fidelity simulations we now have $5N_t$ snapshots. As is typical in time-dependent POD-Galerkin, we temporally downsample by taking every $f_t = 100$ snapshot to lessen the cost of computing the SVD of the matrix of snapshots.[14] We thus form the $5N_t/f_t \times N$ snapshot matrix X and compute the leading $n = 10$ principal components using a

[13] See Appendix A.

[14] For systems that are smooth in time and simulations with suitably small timesteps, both of which are often true in practice, the snapshots $u(x, t_i)$ and $u(x, t_{i+1})$ contain functionally identical information. This justifies temporal downsampling before SVD without appreciable information loss.

standard SVD algorithm. These n vectors form the columns of the POD-Galerkin projector P. Applying the projector to our discretized equations, we have the reduced n-dimensional system of coupled differential equations

$$\begin{aligned} \frac{\partial}{\partial t}\underline{u} &= \alpha \underline{D}\,\underline{u} \\ \underline{u}(x, t=0) &= \underline{u_0} \end{aligned} \tag{2.37}$$

with $\underline{D} = P^\dagger D P$ and $\underline{u_0} = P^\dagger u_0$. The approximate POD solution to the original problem is $\tilde{u} = P\underline{u} \approx u$. This is a system just 1% the size of the original, and thus can be quickly numerically integrated with a much smaller timestep if necessary.[15] So to approximate $u(x, t; \alpha_{\text{test}})$, we integrate our reduced system with $\alpha = \alpha_{\text{test}}$ and obtain $\tilde{u}(x, t; \alpha_{\text{test}}) = P\underline{u}(x, t; \alpha_{\text{test}})$.

Figure 2.5(a) shows the results of the high-fidelity $u(x, t)$ solution at α_{test} using a smaller step size (twice the number of timesteps) such that the simulation converged. The diffusion of the Gaussian initial state over time is clearly visible. The process described above is used to find the POD solution at α_{test}, which is shown in Fig. 2.5(b) and exhibits strong qualitative agreement. Quantitative errors are shown in Fig. 2.5(c) as the elementwise percent error

$$\% \text{ Error} = \frac{\tilde{u}(x, t; \alpha_{\text{test}}) - u(x, t; \alpha_{\text{test}})}{|u(x, t; \alpha_{\text{test}})|} \cdot 100\%. \tag{2.38}$$

Figure 2.5(c) demonstrates that despite a reduction of 99% in the size of the problem, the POD approximate solution is within 1% of the exact solution.[16] Further comparisons as a function of α are shown in Fig. 2.5(e), where the values of $u(x, t; \alpha)$ and $\tilde{u}(x, t; \alpha)$ are shown as a function of α at six (x, t) points indicated in Fig. 2.5(a). This figure shows the accuracy of the POD approximation on the training data as well as the smoothness of the solution as a function of α. Finally, the decay of the singular values of the data matrix is shown in Fig. 2.5(d). The rapid exponential decay of the singular values is not specific to this example and is what allows for subspaces with $n \ll N$ to accurately approximate the high-fidelity

[15] Recall, however, that the condition number of the reduced model will be at most the condition number of the original. Even if this were not the case, the dimensionality reduction allows us to use more expensive methods or resolutions while maintaining a fast and efficient emulator.

[16] Ignoring regions where $|u(x, t; \alpha_{\text{test}})|$ is so small as to cause the percent error to diverge, in which case the absolute difference is a better metric, but these regions are not of interest in this problem.

model. Recall from Eq. (2.32) that the expected relative error in the approximation[17] is the ratio of the area under the squared curve for $i > n$ to the area under the entire squared curve in Fig. 2.5(d).

[17]Over the norm of the vector, not elementwise as is shown in Fig. 2.5(c).

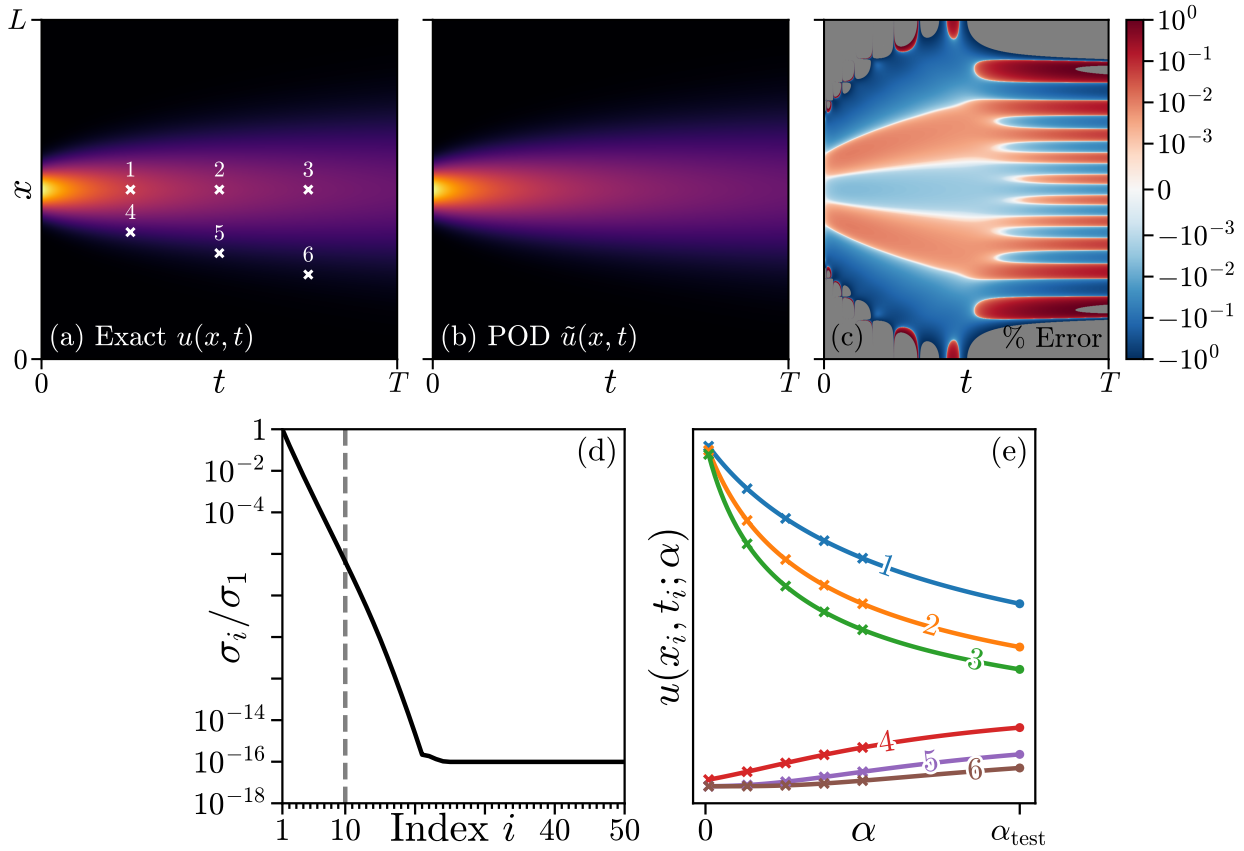

Figure 2.5 Results of applying the POD-Galerkin method to the problem of 1D diffusion. (a) The exact solution to the 1D diffusion equation for diffusivity α_{test} that was not used when training the POD-Galerkin emulator. The six points marked in (a) are the (x_i, t_i) that are sampled as a function of α in (e). The POD-Galerkin approximation to the system for the same diffusivity α_{test} is shown in (b). (c) The percent error between (b) the POD-Galerkin solution and (a) the true solution at α_{test}—errors above 1% where the exact value was very small are shown in grey since a relative error metric is not meaningful for such points. The first 50 singular values of the matrix of snapshots, X, used for training the emulator are shown as a function of their index i in (d), normalized by the largest singular value σ_1. The vertical dashed line in (d) denotes the index of the smallest singular value associated with any singular vector used to construct the POD projector; that is, the vertical line denotes the emulator dimension, n. Finally, (e) shows values of the approximate solution as a function of α at the six points indicated in (a). Crosses in (e) denote the values of the snapshots used to train the emulator at the same locations, showing only five unique values of α were used to compute the projector. The circles in (e) indicate the values of the exact solution at α_{test}, from (a), at the corresponding locations, showing the accuracy of the POD-Galerkin emulator.

Quantity	Symbol	Value
High-fidelity spatial discretization	N	1000
Domain size	L	10
Final time	T	10
Diffusivities used for snapshots	α_{train}	0.1, 1.325, 2.55, 3.775, 5
Timesteps from $t = 0$ to T for snapshot generation	N_t	10 000
Target or testing diffusivity	α_{test}	10
Timesteps used for exact simulation of test diffusivity		20 000
Initial state	$u_0(x)$	$\exp\left\{-\left(\frac{16}{L}\right)^2\left(x - \frac{L}{2}\right)^2\right\}$
Temporal downsampling factor	f_t	100
POD-Galerkin emulator dimension	n	10
Timesteps for each emulator simulation		10 000
Numerical integration method		Explicit (forward) Euler

Table 2.3 Exact values of all quantities used in the 1D diffusion equation POD-Galerkin example.

Emulator Summary for Proper Orthogonal Decomposition (POD-Galerkin)	
Mathematically Parametric	Yes, so long as the form of the parametric dependence is known.
Numerically Parametric	No. Even with the choice of the number of retained principal vectors, n, the addition of new snapshots changes all principal vectors.
Nonlinear-Capable	Can be naively applied to nonlinear systems inefficiently. Efficient approximations to nonlinearities via DEIM exist [41]. Efficient exact treatment of nonlinearities is possible via lifting transformations [43].
Computationally Efficient	Broadly, yes. Nonlinearities must be treated carefully to preserve efficiency.
Data Efficient	Yes. Size of subspace determines the retained explained variance of the snapshots, which in practice converges to 1 either exponentially or polynomially.
Hyperparameter Efficient	Only inherent hyperparameter is the size of the subspace, or equivalently the cumulative explained variance of the snapshots.
Systematically Improvable	Yes. Convergence rate in practice is either exponential or polynomial.
Interpretable	Yes. The projected system has a form identical to the original.
Physically Consistent	Tied directly to the accuracy of the reduced equations. Important properties like normalization and inner products are approximately conserved.
Intrusiveness	Maximally intrusive.
Quantified Uncertainties	Not inherent, but can be added on top of the method. Error bounds and error estimates available.

Table 2.4 Summary of properties for the POD-Galerkin method. Many properties are inherited from Galerkin's Method.

2.11.1.2 Dynamic Mode Decomposition

Another method originally introduced in the field of computational fluid dynamics, dynamic mode decomposition (DMD) in its original form is a purely data-driven technique for sequential data [53, 54]. That is, the data is of the form $\{z_1, z_2, \ldots, z_n\}$ where the true underlying process is assumed to transform z_i to z_{i+1}. This was generalized to "Exact" DMD, where instead of a set sequence, the data are in the familiar input $\{x_1, \ldots, x_n\}$ and output $\{y_1, \ldots, y_n\}$ sets and the underlying process is assumed to connect them via $x_i \to y_i$ [54]. Being a purely data-driven method, DMD requires (and uses) no knowledge of the underlying problem or high-fidelity model. Instead, it imposes an assumption on the underlying process, namely that it is purely linear,

$$Ax_i = y_i, \tag{2.39}$$

for some unknown A. This assumption is equal parts simple and powerful as many systems—even nonlinear ones—can be well-approximated in this way. The goal of DMD will be to find, or at least approximate, A. In practice, one usually obtains and returns only a limited number of "dynamic modes" or DMD modes and DMD eigenvalues, which are the leading few eigenvectors and eigenvalues of this operator. These dynamic modes become the basis of a Galerkin subspace when DMD is used for dimensionality reduction.

There are many methods used in practice to compute A, and the one used often depends on the dimensionality of the data, the amount of data, and desired accuracy versus speed of the approximation of A. Fundamentally, these methods all aim to approximate the exact solution which is

$$A = YX^{+} \tag{2.40}$$

where X is the matrix whose columns are x_i, Y is the matrix whose columns are y_i, and X^{+} denotes the Moore-Penrose pseudoinverse of X. This is simply the least-squares solution, which minimizes $\|AX - Y\|_F$ [54].

Generalizing DMD to parametric problems is nontrivial and a properly parametric version was only introduced relatively recently [55, 56]. As DMD has no knowledge of the underlying

equations, the parametric dependence is unknown and so parametric DMD utilizes data-driven interpolations of the DMD modes at known parameters to approximate the modes at new parameters.

DMD can be generalized beyond linear systems, particularly in the case of dynamical systems. Through deliberate feature engineering and careful regularization, the method known as sparse identification of nonlinear dynamics (SINDy) aims to discover the form of any nonlinearities from data alone [57, 58]. SINDy first constructs a "library" of candidate nonlinearities of the input data. For a single snapshot this is something like

$$\Theta(x) = \begin{bmatrix} 1 \\ x \\ (x \otimes x) \\ (x \otimes x \otimes x) \\ \vdots \\ \sin(x) \\ \vdots \end{bmatrix} \tag{2.41}$$

where $x \otimes x$ represents quadratic polynomials of the elements of the snapshot, i.e. the vector containing all quadratic combinations of the elements of x. The specific candidate nonlinearities are hyperparameters and can vary from problem to problem. We then suppose that the problem is linear in these new features

$$y_i = \Xi^T \Theta(x_i), \tag{2.42}$$

for some unknown matrix Ξ^T. The size of $\Theta(x)$ can far exceed that of x depending on the type and amount of candidate nonlinearities, so SINDy makes the assumption that only a very small number of the candidate nonlinearities will contribute for each element of y_i. This corresponds to the rows of Ξ being very sparse—consisting of mostly 0. Finding Ξ is now just a relatively simple matter of solving the sparse regression problem

$$Y^T = \Theta\big(X^T\big)\Xi \tag{2.43}$$

for Ξ. While this method does not reduce the dimensionality of the problem directly, it can in principle be combined with ANNs such as autoencoders to achieve effective reductions or the resulting linear system may be treated with standard DMD. However, it appears in literature that extensions to SINDy are limited to $\mathcal{O}(10)$-dimensional systems—though this is an area of active research [59–61].

Example

We consider the same example as in the previous section (Section 2.11.1.1), that of diffusion in one dimension. This is a linear system and thus should be well-approximated by DMD. All parameters of the high-fidelity simulations and training data are unchanged, including temporal downsampling. We create the emulator using the implementation of parametric DMD provided in the PyDMD package [56, 62] with a reduced subspace of size $n = 10$ to match the POD-Galerkin example. Significantly more preprocessing and hyperparameter tuning is required to achieve a reasonable emulator result compared to the previous example. We utilize an SVD preprocessor (PCA) to reduce both the dimensionality and the condition number of the training data. Additionally, there are many choices for the interpolation method used to achieve the parametric dependence. The one chosen—radial basis function interpolation with the standard multiquadratic kernel—represented the most accurate for both interpolation and extrapolation in α. It is likely that more careful hyperparameter tuning, preprocessing, and the use of constrained DMD methods could improve the accuracy of the emulator. Figure 2.6 shows the results of the DMD emulator for the same test diffusivity α_{test} as in the previous example. The DMD emulator interpolates in α well, especially given the fact that it has no knowledge of the underlying system since it is a fully data-driven method. However, the emulator struggles with extrapolation, especially at late times, as shown in Fig. 2.6(d). This is a known limitation of DMD and is not specific to this example.

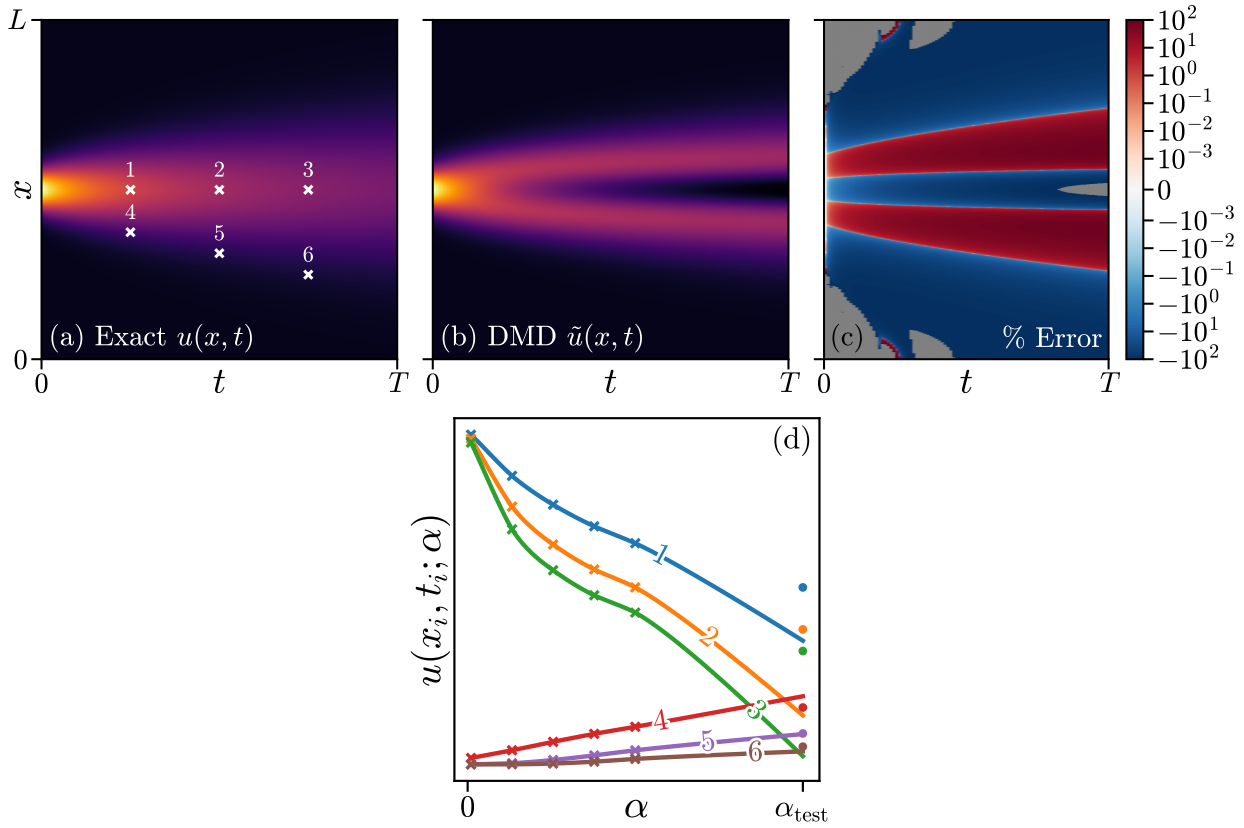

Figure 2.6 Results of applying parametric DMD to the problem of 1D diffusion. (a) The exact solution to the 1D diffusion equation for diffusivity α_{test} that was not used when training the DMD emulator. The six points marked in (a) are the (x_i, t_i) that are sampled as a function of α in (d). The DMD approximation to the system for the same diffusivity α_{test} is shown in (b). (c) The percent error between (b) the DMD solution and (a) the true solution at α_{test}—errors above 100% where the exact value was very small are shown in grey since a relative error metric is not meaningful for such points. Note that this limit is larger than in Fig. 2.5. Finally, (d) shows values of the approximate solution as a function of α at the six points indicated in (a). Crosses in (d) denote the values of the snapshots used to train the emulator at the same locations, showing only five unique values of α were used to fit the emulator. The circles in (d) indicate the values of the exact solution at α_{test}, from (a), at the corresponding locations, showing that the emulator struggles with extrapolating in α.

Emulator Summary for Dynamic Mode Decomposition	
Mathematically Parametric	Recent extensions for parametric problems exist [55, 56].
Numerically Parametric	No. Similar to POD-Galerkin, the addition of new snapshots changes all existing DMD modes.
Nonlinear-Capable	Limited in practice to linear systems, or systems that can be linearized well, for dimensionality reduction and emulation. SINDy can handle nonlinear systems, but not efficiently for high-dimensional systems.
Computationally Efficient	Yes.
Data Efficient	Yes, DMD modes and DMD eigenvalues behave similarly to the principal values and principal components.
Hyperparameter Efficient	Only inherent hyperparameter is the number of DMD modes kept.
Systematically Improvable	Yes for linear systems. Only in the infinite-data limit for nonlinear systems. In practice only for linear systems.
Interpretable	Yes. The projected system is a simple linear map.
Physically Consistent	Not necessarily. The projected system is a simple linear map, which may not preserve physical properties of the original system, especially if the underlying snapshots are produced by a nonlinear map.
Intrusiveness	Purely data-driven.
Quantified Uncertainties	Not inherent, but can be added on top of the method.

Table 2.5 Summary of properties for DMD.

2.11.1.3 Tractable *a priori* Dimensionality Reduction for Quantum Dynamics

This section makes an exception to the stated goal of covering methods which are widely used and contains the relevant parts of the method described in Ref. [63]. A common task in nuclear and many-body theory is the time evolution of a known arbitrary initial state $|\Psi_0\rangle$ under a Hamiltonian H of dimension $N \times N$, as well as the expectation value of any set of observables as a function of time.

$$\left\{ \begin{array}{c} |\Psi_0\rangle \\ H \\ \{O_i\} \end{array} \right\} \quad \longrightarrow \quad \left\{ \begin{array}{r} |\Psi(t)\rangle \\ \{\langle\Psi(t)|O_i|\Psi(t)\rangle\} \end{array} \right\} \tag{2.44}$$

There are two traditional approaches to this problem. The first is the full eigenvalue decomposition (diagonalization) of the Hamiltonian,

$$H|E_i\rangle = E_i|E_i\rangle, \tag{2.45}$$

followed by writing the initial state as a linear combination of the energy eigenstates

$$|\Psi_0\rangle = \sum_{i=1}^{N} c_i|E_i\rangle = \sum_{i=1}^{N}\langle E_i|\Psi_0\rangle|E_i\rangle. \tag{2.46}$$

Finally, time propagation is accomplished by independently evolving each energy eigenstate,

$$|\Psi(t)\rangle = \sum_{i=1}^{N}\langle E_i|\Psi_0\rangle e^{-iE_it}|E_i\rangle. \tag{2.47}$$

The time complexity of this approach is about $\mathcal{O}(N^3)$ in practice owing to the full dense eigenvalue decomposition.

The second way is the direct computation and application of the time evolution operator

$$|\Psi(t)\rangle = \exp\{-iHt\}|\Psi_0\rangle. \tag{2.48}$$

This approach is $\mathcal{O}(N^3)$ in practice as well, but the complexity is much more dependent on the specific structure (or lack thereof) of the Hamiltonian due to the difficulties in numerical matrix exponentiation [64].

All previously discussed Galerkin-like methods in this chapter have operated in the context that some snapshots were provided without any choice in how those snapshots were collected. By contrast, for a desired number of snapshots n we now seek to find a way to collect only the best possible snapshots for the problem of Hamiltonian dynamics. To accomplish this, we seek to find the n vectors such that $|\Psi(t)\rangle$ projected onto these vectors is best conserved.

Consider Eq. (2.46) with a reordering of the coefficients such that $|c_i| \geq |c_{i+1}| \geq 0$, meaning the energy eigenvalues are no longer in the traditional ascending order. Define the approximate wavefunction to order $n \ll N$,

$$\begin{aligned} |\psi\rangle &= \frac{1}{\mathcal{N}} \sum_{i=1}^{n} c_i |E_i\rangle, \\ \mathcal{N}^2 &\equiv \sum_{i=1}^{n} |c_i|^2, \, \mathcal{N} \in \mathbb{R}^+. \end{aligned} \tag{2.49}$$

Here I show that this approximation minimizes the error in the ℓ^2 norm of the dynamics for a given number of contributing basis states n, given that the error is time-independent. Consider the most general form of such an n^{th} order approximation,

$$\begin{aligned} |\phi(t)\rangle &= \frac{1}{\mathcal{M}(t)} \sum_{q=1}^{n} |a_q\rangle\langle a_q|\Psi(t)\rangle, \\ \mathcal{M}(t)^2 &\equiv \sum_{q=1}^{n} |\langle a_q|\Psi(t)\rangle|^2, \, \mathcal{M}(t) \in \mathbb{R}^+, \end{aligned} \tag{2.50}$$

where $\{|a_q\rangle\}$ is an arbitrary set of n orthonormal basis vectors. This can equivalently be written as $|\phi(t)\rangle = PP^\dagger|\Psi(t)\rangle/\mathcal{M}(t)$ where $P \in \mathbb{U}(N, n)$ is the POD-Galerkin-like projector formed by taking $\{|a_q\rangle\}$ as the columns. Note that $P^\dagger P = I_n$. The ℓ^2 error of this approximation is

$$\begin{aligned} \||\Psi(t)\rangle - |\phi(t)\rangle\|_2^2 &= 2 - \frac{2}{\mathcal{M}(t)} \Re\big\{\big\langle \Psi(t)\big| PP^\dagger \big|\Psi(t)\big\rangle\big\} \\ &= 2(1 - \mathcal{M}(t)). \end{aligned} \tag{2.51}$$

For the error to be independent of time we must have

$$\begin{aligned}\mathcal{M}(t)^2 &= \mathcal{M}^2 \\ &= \big\langle \Psi(t)\big| PP^\dagger \big|\Psi(t)\big\rangle \\ &= \sum_{q=1;\, a,b=1}^{n;\, N} c_a^* c_b \langle E_a|a_q\rangle\langle a_q|E_b\rangle e^{-i(E_b-E_a)t} \\ &= \sum_{q=1;\, a=1}^{n;\, N} |c_a|^2 |\langle E_a|a_q\rangle|^2.\end{aligned} \tag{2.52}$$

That is, all the time-dependent terms in $\mathcal{M}(t)$ must cancel. Since $\mathcal{M} \leq 1$, the error is minimized when $\mathcal{M}$, or equivalently $\mathcal{M}^2$, is maximized. So we must solve

$$\arg\max_{\{|a_q\rangle\}} \big\{\mathcal{M}^2\big\}. \tag{2.53}$$

This is equivalent to solving the Rayleigh quotient problem

$$\arg\max_{\{|a_q\rangle\}} \left\{ \sum_{q=1}^{n} \langle a_q|\Xi|a_q\rangle \right\}, \tag{2.54}$$

where $\Xi \equiv \sum_{a=1}^{n} |c_a|^2 |E_a\rangle\langle E_a|$ is the matrix with eigenvalues $\big\{|c_a|^2\big\}$ and eigenvectors $\{|E_a\rangle\}$. The solution to this problem is the n eigenvectors of Ξ with the largest eigenvalues, and so the error is minimized when $\{|a_q\rangle\} = \{|E_1\rangle, |E_2\rangle, \ldots, |E_n\rangle\}$ up to arbitrary phase factors. This proves that Eq. (2.49) is the approximation that minimizes the time-independent approximation error. The resulting error is

$$\big\||\Psi(t)\rangle - |\psi(t)\rangle\big\|_2^2 = 2(1-\mathcal{N}). \tag{2.55}$$

To derive an error bound for any observable, O, let $\langle O\rangle_\theta \equiv \langle\theta(t)|O|\theta(t)\rangle$ and define the state $|\Phi(t)\rangle$ by

$$|\Phi(t)\rangle \equiv |\Psi(t)\rangle - \mathcal{N}|\psi(t)\rangle. \tag{2.56}$$

The norm squared of this state is $\langle\Phi(t)|\Phi(t)\rangle = 1-\mathcal{N}^2$. Using a standard Rayleigh quotient identity,

$$\langle O\rangle_\phi \leq \langle\phi(t)|\phi(t)\rangle \|O\|_2^2, \tag{2.57}$$

one can see that the error in any observable O from the approximation in Eq. (2.49) is bounded by

$$\begin{aligned}\left|\langle O\rangle_{\Psi} - \langle O\rangle_{\psi}\right|^2 &= \left|\left(\mathcal{N}^2 - 1\right)\langle O\rangle_{\psi} + \langle O\rangle_{\Phi}\right|^2 \\ &\leq \left|\left(\mathcal{N}^2 - 1\right)\langle O\rangle_{\psi}\right|^2 + \left|\langle O\rangle_{\Phi}\right|^2 \\ &\quad + 2\left|\left(\mathcal{N}^2 - 1\right)\langle O\rangle_{\psi}\langle O\rangle_{\Phi}\right| \\ &\leq 4\left(1 - \mathcal{N}^2\right)^2 \|O\|_2^4,\end{aligned} \tag{2.58}$$

where $\|O\|_2^2$ is the square of the matrix ℓ^2 norm of O, which for Hermitian operators is equal to the maximum absolute eigenvalue.

So we have shown that the n eigenstates of the Hamiltonian with maximum overlap with the target state are the best possible snapshots and derived error bounds for the Galerkin emulator based on those snapshots. However, it may seem that the only way to obtain these snapshots is by a full diagonalization of the Hamiltonian, rendering this a purely academic exercise. Indeed, most existing implementations of numerical eigensolvers either compute the complete eigendecomposition or a subset of eigenpairs selected by eigenvalue. For example, the ubiquitous ARPACK library provides subroutines `SSEUPD` and `DSEUPD` for the implicitly restarted Arnoldi iteration and allows for targeting a subset of eigenpairs based on eigenvalue [65]. Fortunately, a lesser-used algorithm for the generalized eigenvalue decomposition, the Jacobi-Davidson algorithm, can be modified to target eigenpairs by any property of either the eigenvalue or eigenstate [66–68]. This modification allows the targeting of eigenpairs by $|\langle\Psi_0|E_q\rangle|^2 = |c_q|^2$ to compute only the n eigenpairs with maximum overlap with the target state without the need to over-compute extra eigenpairs. The algorithm can be trivially modified for the case where multiple target states are of interest.

The computation of the n eigenstates of interest requires $\mathcal{O}(nN^2)$ time and $\mathcal{O}(N^2)$ space for dense representations of the Hamiltonian. In practice however, physical operators are nearly universally local and can be implemented in a matrix-free way such that the cost of matrix-vector multiplication is only $\mathcal{O}(N)$ time and space. Leveraging such a matrix-free implementation of the Hamiltonian reduces the cost of computing the n eigenstates to just

$\mathcal{O}(nN)$ time and $\mathcal{O}(N)$ space. Since $n \ll N$ and in practice can be fixed to a small value, the time complexity is roughly $\mathcal{O}(N)$. This is the same time and space complexity of just directly writing down the time-evolved state.

It is fortuitous that the optimal reduced basis states are energy eigenstates, as this renders the Galerkin-projected Hamiltonian diagonal and therefore the time evolution trivial in the reduced space,

$$P^\dagger H P = \underline{H} = \mathrm{diag}(E_1, E_2, \ldots, E_n). \tag{2.59}$$

Which makes the time evolution in the reduced basis just $\mathcal{O}(n)$. Recall that due to the reordering of the energy eigenstates, E_1 is not necessarily the ground state and it is not necessarily the case that the E_q are ordered. Any other operator(s) of interest can be projected down to the reduced space, allowing the dynamics of any expectation value to be calculated in just $\mathcal{O}(n^2)$ time.

Including the cost of the Galerkin projection of the initial state and the observable(s) of interest, as well as the time evolution in the reduced space, the entire method requires $\mathcal{O}(Nn^2)$ time and $\mathcal{O}(N)$ space for matrix-free implementations of the Hamiltonian and observable(s).

Example

Consider the Hamiltonian detailed in Appendix B, a 1D spin chain of L spins with couplings J_{ij} and transverse field B,

$$H = -\frac{1}{\mathcal{J}} \sum_{i<j}^{L} J_{ij} \left(\gamma_x \sigma_i^x \sigma_j^x + \gamma_y \sigma_i^y \sigma_j^y \right) - B \sum_{i}^{L} \sigma_i^z, \tag{2.60}$$

where $\gamma_{x/y}$ are the relative strengths of the x and y couplings, σ_i^v is the Pauli spin operator acting on the i^{th} site in the v axis, and $\mathcal{J}$ is the Kac normalization factor

$$\mathcal{J} \equiv \frac{1}{L-1} \sum_{i \neq j}^{L} J_{ij}. \tag{2.61}$$

For this example we will take the couplings to be a power-law decay with scale J and exponent p,

$$J_{ij} = \frac{J}{|i-j|^p}. \tag{2.62}$$

We choose $J = \gamma_x = B = 1$, $\gamma_y = 0$, the initial state to be the fully polarized-up state

$$|\Psi_0\rangle \equiv \Big|\uparrow\uparrow\overset{L}{\cdots}\uparrow\Big\rangle, \tag{2.63}$$

and the observable of interest to be

$$S_x^2/L \equiv \frac{1}{L}\sum_{i,j}^{L} \sigma_i^x \sigma_j^x. \tag{2.64}$$

We will consider two values for the decay exponent, a long range decay $p = 0.1$ and a shorter-range decay $p = 0.9$. The significance of this Hamiltonian, these specific values, and why they make for an interesting problem are discussed in Appendix B and Refs. [20, 63].

To visualize how the Jacobi-Davidson method and the optimal approximate state work, we consider the case of $L = 12$ spins. The dimension of the Hilbert space, and therefore the Hamiltonian, is $N = 2^L = 4096$. We choose $n = 8$ for the size of the Galerkin subspace. Figure 2.7 shows the squared magnitude of the components of the exact and optimal approximate initial states in the energy basis of the Hamiltonian with $p = 0.1$ and $p = 0.9$. That is, Fig. 2.7 shows $|c_i|^2$ for the c_i in Eq. (2.46) for both values of the decay exponent. It is immediately clear that the eigenstates with maximum overlap with the initial state are not simply the lowest lying states, nor are they grouped by their associated energies. The $n = 8$ energy eigenstates with maximum overlap with Eq. (2.63), which make up the optimal approximate state $|\psi_0\rangle$, are denoted with crosses and span more than a fifth (more than 800 states) of the spectrum. This means that simply approximating the initial state with the lowest lying eigenstates would require more than 100 times as many basis states to achieve roughly the same accuracy. A strategy of randomly sampling eigenstates would not work either, as the proportion of eigenstates with an initial state overlap greater than just 10^{-6} is less than 2% for both Hamiltonians. Note that the magnitude of the components decay

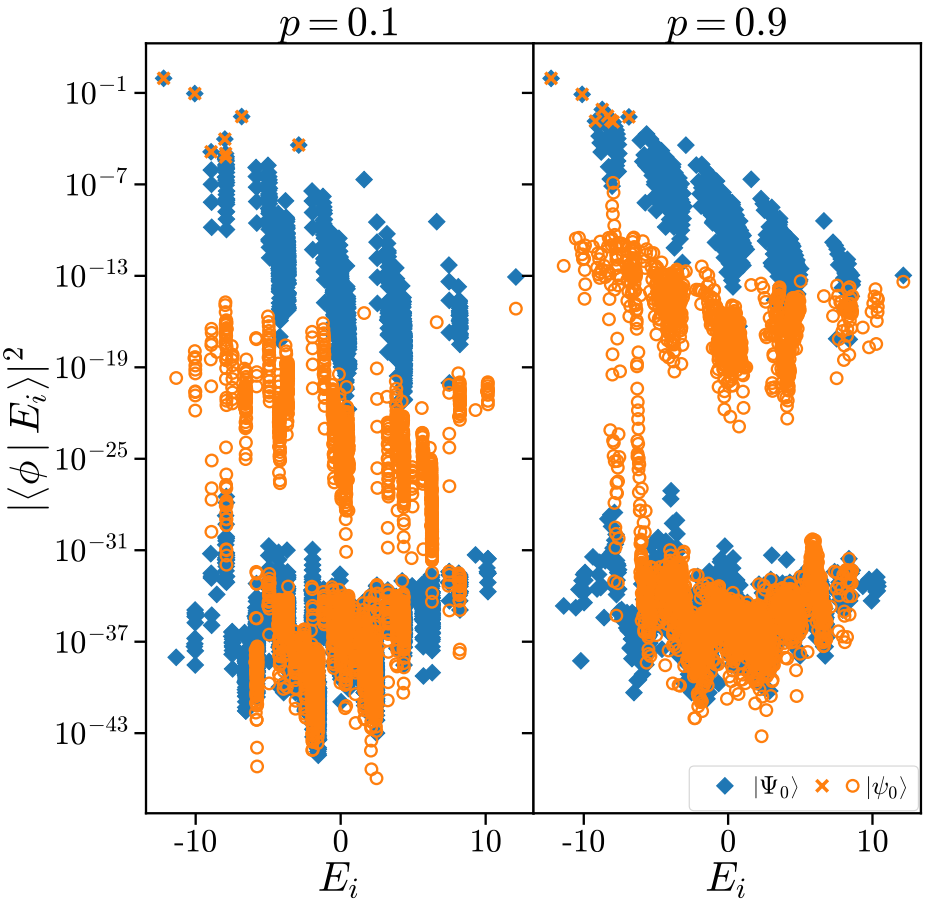

Figure 2.7 Squared magnitude of the components of the $L = 12$ initial state in Eq. (2.63) in the energy basis of the Hamiltonian with $p = 0.1$ (left) and $p = 0.9$ (right) power-law couplings and $[L, J, \gamma_x, \gamma_y, B] = [12, 1, 1, 0, 1]$. The exact state ($|\phi\rangle \equiv |\Psi_0\rangle$) is shown as blue diamonds. The optimal approximate state ($|\phi\rangle \equiv |\psi_0\rangle$) is shown as orange crosses and circles, with crosses denoting the automatically targeted states with maximum overlap. Nonzero solver tolerances result in nonzero overlap with the non-targeted states.

exponentially, with the 8^{th} largest component five orders of magnitude smaller than the first for $p = 0.1$ and three orders of magnitude smaller for $p = 0.9$. This is generally the case in practice and is what allows for $n \ll N$ to still produce an accurate prediction.

The efficacy of the method is demonstrated with a larger $L = 14$ system, corresponding to $N = 16\,384$, with the same $n = 8$ number of Galerkin basis states. The exact evolution of the state under the two Hamiltonians is computed and the expectation value of S_x^2/L is found at each time t. Then for each Hamiltonian, the optimal projector P is found and used to project the initial state, Hamiltonian, and S_x^2/L to the reduced space where time

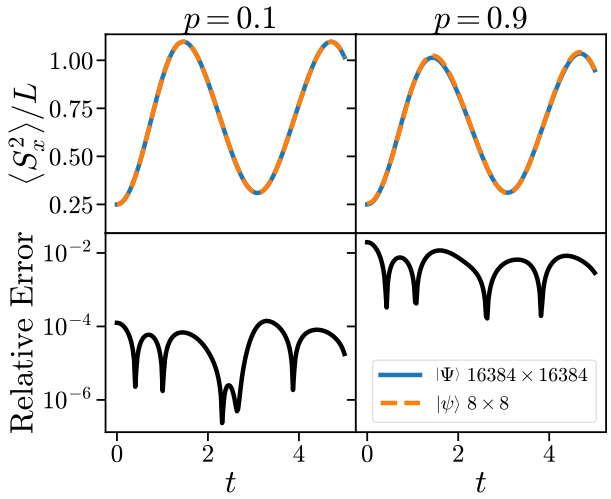

Figure 2.8 Exact (solid blue) and approximate (dashed orange) dynamics of S_x^2/L for the Hamiltonian with $p = 0.1$ (left) and $p = 0.9$ (right). Relative errors in the approximations are shown in the lower plots.

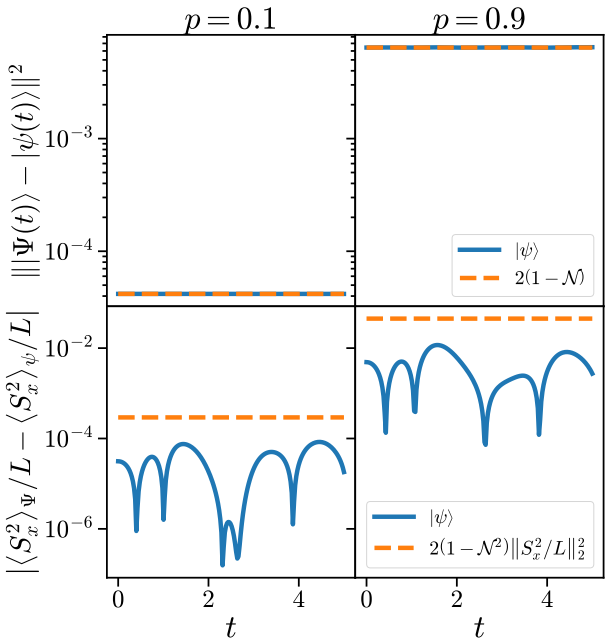

Figure 2.9 Numerically computed (solid blue) and analytically predicted (dashed orange) error for the dynamics of the state (top) and observable (bottom) for the Hamiltonian with $p = 0.1$ (left) and $p = 0.9$ (right).

evolution is performed. Once the state and operators are projected to the reduced space, no computations need to be performed in the full space to carry out the time evolution. Figure 2.8 shows these dynamics along with the relative error in the observable. The relative error is on the order of 0.01% for $p = 0.1$ and 1% for $p = 0.9$ while the size of the Galerkin subspace is less than 0.05% that of the full space. Finally, Fig. 2.9 shows the numerically computed and analytically predicted error (or error bound) in the time evolution of the approximate state, validating Eqs. (2.55) and (2.58).

Emulator Summary for Tractable *a priori* Dimensionality Reduction for Quantum Dynamics	
Mathematically Parametric	No.
Numerically Parametric	No. Follows the same reasons as POD-Galerkin.
Nonlinear-Capable	No. Limited to dynamics of Hermitian systems with known initial states.
Computationally Efficient	Yes.
Data Efficient	Yes, maximally, as the method dictates what data are chosen.
Hyperparameter Efficient	Only inherent hyperparameter is the Galerkin subspace dimension.
Systematically Improvable	Yes, converges absolutely to the exact time dynamics as $n \to N$.
Interpretable	Yes, the projected system is exactly a truncated form of the original system.
Physically Consistent	Yes, retained eigenvalues are equivalent, normalization is preserved, and energy is conserved.
Intrusiveness	Maximally intrusive. Requires partial diagonalization of the exact Hamiltonian.
Quantified Uncertainties	UQ not applicable, but error estimates and bounds are available and practically useful.

Table 2.6 Summary of properties for the method presented in this section.

2.11.1.4 Eigenvector Continuation

Within the field of *ab initio* nuclear theory, eigenvector continuation (EC) has perhaps been the most popular Galerkin-type emulator since its introduction [22, 69–79]. While developed independently of the Galerkin method, it can nevertheless be viewed as a Galerkin projection of the standard parametric Hermitian eigenproblem

$$H(\mu)\Psi_k(\mu) = E_k(\mu)\Psi_k(\mu) \tag{2.65}$$

where $H(\mu)$ is some parametric Hamiltonian with parameters μ and $\Psi_k(\mu)$ is the energy eigenstate associated with the energy eigenvalue $E_k(\mu)$. EC utilizes snapshots, or "training vectors", of the eigenstates of interest at various parameter realizations. Originally this consisted of only the ground state, but later extensions showed that excited states could be included without modification [69, 77]. The EC method proceeds as follows. Let $H(\mu) \in \mathbb{H}(N)$ be the parametric Hamiltonian of interest and $v_j\big(\mu^{(i)}\big) \in \mathbb{C}^N$ with $\big\|v_j\big(\mu^{(i)}\big)\big\| = 1$ be the snapshot eigenvector for parameter realization $\mu^{(i)}$ associated with the j^{th} eigenvalue. For n snapshots, we form the Galerkin projectors ${}^{\downarrow}P$ and $P^{\downarrow}$ as before in Eq. (2.10), where ${}^{\downarrow}P$ is the matrix whose rows are $\left\{v_j\big(\mu^{(i)}\big)^\dagger\right\}$ and $P^{\downarrow} = {}^{\downarrow}P^\dagger\Big({}^{\downarrow}P{}^{\downarrow}P^\dagger\Big)^{-1}$. Following the conventional EC notation, let T_V be the matrix whose columns are the training vectors, so ${}^{\downarrow}P = T_V^\dagger$, and $\mathcal{N} = T_V^\dagger T_V = {}^{\downarrow}P{}^{\downarrow}P^\dagger$ be the *norm matrix*. The elements of the norm matrix are simply all the inner products of the training vectors. The relation to the Galerkin projectors from Eq. (2.10) is then ${}^{\downarrow}P = T_V^\dagger$ and $P^{\downarrow} = T_V\mathcal{N}^{-1}$. Projecting Eq. (2.65) onto the Galerkin

subspace with these projectors yields

$$\begin{aligned}
&\underbrace{\left(\text{⌊}P H(\mu) P^{\text{⌈}}\right)}_{\underline{H(\mu)}}\underbrace{\left(\text{⌊}P\Psi_k(\mu)\right)}_{\underline{\Psi_k(\mu)}} = E_k(\mu)\underbrace{\left(\text{⌊}P\Psi_k(\mu)\right)}_{\underline{\Psi_k(\mu)}} \\
\iff &\Big(\underbrace{T_V^\dagger H(\mu) T_V}_{\tilde{H}(\mu)}\mathcal{N}^{-1}\Big)\Big(T_V^\dagger\Psi_k(\mu)\Big) = E_k(\mu)\Big(T_V^\dagger\Psi_k(\mu)\Big) \\
\iff &\tilde{H}(\mu)\Big(\underbrace{\mathcal{N}^{-1}T_V^\dagger\Psi_k(\mu)}_{\psi_k(\mu)}\Big) = E_k(\mu)\overbrace{\mathcal{N}\Big(\mathcal{N}^{-1}}^{I_n}T_V^\dagger\Psi_k(\mu)\Big) \\
\iff &\tilde{H}(\mu)\psi_k(\mu) = E_k(\mu)\mathcal{N}\psi_k(\mu).
\end{aligned} \tag{2.66}$$

Notice that this is simply the generalized eigenvalue problem[18] for $n \times n$ matrices $\tilde{H}(\mu)$ and $\mathcal{N}$. One can reconstruct the full-space solution with $\Psi_k(\mu) \approx \Psi_k^{\text{EC}}(\mu) = T_V\psi_k(\mu)$.

There are many recent extensions, applications, and analyses of EC throughout the literature [22, 69–85]. Perhaps the interpretation that led to its rapid adoption is that of its close ties to perturbation theory and its success either as a faster-converging alternative to perturbation theory or in drastically improving the convergence rate of perturbation theory calculations [70, 72]. This same interpretation—that of a series expansion of the eigenstate—gives rise to rigorous bounds on the convergence rate of EC [22]. See Ref. [22] for an in-depth discussion and derivation of the error bounds as well as an excellent review of EC. The impressive result is that EC converges exponentially with the number of snapshots. For d parameters, i.e. $\mu = [\mu_1, \mu_2, \ldots, \mu_d]$, the norm of the residual vector between the true and EC-approximate eigenstate is

$$\left\|\Psi_k(\mu) - \Psi_k^{\text{EC}}(\mu)\right\|_2 \sim \alpha^{n^{1/d}}, \tag{2.67}$$

for some $0 < \alpha < 1$, where n is the number of snapshots. This bound is valid for large n and converges faster than any polynomial for sufficiently large n. Since n is both the amount of

[18] Also notice that this would have been the standard eigenvalue problem if the method used the POD-Galerkin projector and stopped after the first line of Eq. (2.66). In practice, the difficulty of the generalized eigenvalue problem leads to implementations of EC orthonormalizing the matrix of training vectors, essentially recovering the POD-Galerkin method for the eigenvalue problem [22].

data and the size of an EC emulator, this shows that EC is both computationally and data efficient as well as systematically improvable.

Example

Consider the same Hamiltonian from the example in Section 2.11.1.3 shown in Eq. (2.60) and detailed in Appendix B. As in the previous example, we choose $L = 14$. We suppose that we want to find the ground state and the expectation value of the observable S_x^2/L in the ground state as a function of B while all other Hamiltonian parameters remain fixed ($\{J, p, \gamma_x, \gamma_y\} = \{1, 0.9, 1, 0\}$). This allows us to frame the problem as an affine family of Hamiltonians

$$H(B) = H_0 + BH_1 \tag{2.68}$$

where

$$\begin{aligned} H_0 &\equiv -\frac{1}{\mathcal{J}} \sum_{i<j}^{L} J_{ij}\big(\gamma_x \sigma_i^x \sigma_j^x + \gamma_y \sigma_i^y \sigma_j^y\big), \\ H_1 &\equiv -\sum_{i}^{L} \sigma_i^z. \end{aligned} \tag{2.69}$$

We sample $n = 5$ values of B between 0 and 0.75 to solve the exact eigenproblem at and use the resulting eigenstates to form the $N \times n$ matrix of snapshots,

$$T_V = \begin{bmatrix} | & | & | & | & | \\ \Psi_0(B_1) & \Psi_0(B_2) & \Psi_0(B_3) & \Psi_0(B_4) & \Psi_0(B_5) \\ | & | & | & | & | \end{bmatrix} \tag{2.70}$$

and orthonormalize the columns via the QR decomposition or other orthogonalization algorithm. This orthonormalization ensures that $\mathcal{N} = I$, greatly simplifying the projected problem. To make a prediction for the ground state energy using the EC emulator for this system at a new target B_t, we solve the projected eigenproblem

$$\begin{aligned} &\Big(T_V^\dagger H(B_t) T_V\Big)\Big(T_V^\dagger \Psi_0(B_t)\Big) = E_0^{\mathrm{EC}}\Big(T_V^\dagger \Psi_0(B_t)\Big) \\ \iff &\Big(T_V^\dagger H_0 T_V + B_t T_V^\dagger H_1 T_V\Big)\Big(T_V^\dagger \Psi_0(B_t)\Big) = E_0^{\mathrm{EC}}\Big(T_V^\dagger \Psi_0(B_t)\Big) \\ \iff &\Big(\tilde{H}_0 + B_t \tilde{H}_1\Big)\psi_0(B_t) = E_0^{\mathrm{EC}}\psi_0(B_t). \end{aligned} \tag{2.71}$$

Leveraging knowledge of the structure of $H(B)$ to project each of its constituent operators (H_0 and H_1) during the offline phase greatly decreases the online cost of the EC emulator since the construction and subsequent projection of $H(B)$ can be exceedingly expensive and scales with N. Similarly, instead of reconstructing the N-dimensional EC approximation of the eigenstate in order to make a prediction for the expectation value of S_x^2/L, it is much more efficient to simply project S_x^2/L during the offline phase,

$$\tilde{S}_x^2/L = T_V^\dagger S_x^2 T_V/L, \tag{2.72}$$

and use the reduced-space eigenvector to compute expectation values with this reduced operator. It is straightforward to show that this is equivalent to the more expensive alternative,

$$\begin{aligned}
\left\langle \Psi_0(B)\middle|S_x^2\middle|\Psi_0(B)\right\rangle &\approx \left\langle \Psi_0^{\mathrm{EC}}(B)\middle|S_x^2\middle|\Psi_0^{\mathrm{EC}}(B)\right\rangle \\
&= \left\langle T_V\psi_0(B)\middle|S_x^2\middle|T_V\psi_0(B)\right\rangle \\
&= \left\langle \psi_0(B)\middle|T_V^\dagger S_x^2 T_V\middle|\psi_0(B)\right\rangle \\
&= \left\langle \psi_0(B)\middle|\tilde{S}_x^2\middle|\psi_0(B)\right\rangle.
\end{aligned} \tag{2.73}$$

This process results in an EC emulator for the $N = 2^L = 16384$-dimensional Hamiltonian which for any target value B_t requires only the solution of a 5×5 standard eigenproblem and the expectation value with a 5×5 operator. The results for this example are shown in Fig. 2.10, demonstrating the predictive power of EC for parametric Hamiltonian problems such as this one.

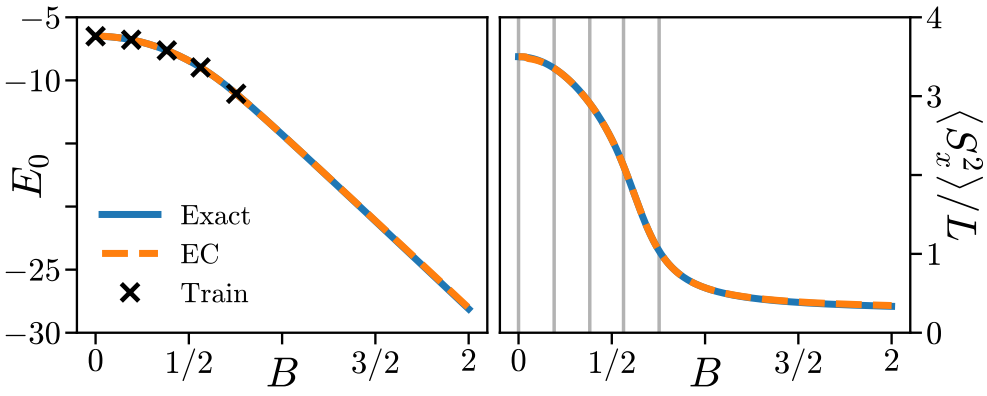

Figure 2.10 Exact and EC-predicted results for the ground state energy (left) and expectation value of S_x^2/L in the ground state (right) for the Hamiltonian detailed in Appendix B as a function of B. All other Hamiltonian parameters are fixed $\{L, \gamma_x, \gamma_y\} = \{14, 0, 1\}$ with a power-law coupling defined by $J = 0$ and $p = 0.9$. The locations of the training eigenvectors are shown as crosses (left) and vertical lines (right).

Emulator Summary for Eigenvector Continuation	
Mathematically Parametric	Yes.
Numerically Parametric	No. Size of the emulator is the number of snapshots, n.
Nonlinear-Capable	No. Limited to standard eigenvalue problems.
Computationally Efficient	Only if the form of $H(\mu)$ is known and the parameter dependence is separable, allowing for the projection(s) to be done during the offline phase.
Data Efficient	Yes. Exponential convergence.
Hyperparameter Efficient	Only inherent hyperparameter is the number of snapshots.
Systematically Improvable	Yes.
Interpretable	Yes. The projected system is a generalized eigenvalue problem as opposed to the original standard eigenvalue problem, but the interpretation of all operators are largely unchanged. The projected system can be a standard eigenproblem of identical form to the original if the matrix of snapshots is orthonormalized first.
Physically Consistent	Yes, eigenvalues represented in the snapshots are equivalent, normalization is preserved, and energy is conserved.
Intrusiveness	Maximally intrusive. Requires evaluation of the high-fidelity $H(\mu)$ at all parameter realizations of interest. If the form of $H(\mu)$ is known and the parameter dependence separable, much of this cost can be moved to the offline phase.
Quantified Uncertainties	UQ not applicable, but error bounds are available and practically useful. Can be combined with UQ methods [85].

Table 2.7 Summary of properties for EC.

2.11.2 Gaussian Processes

In data-driven machine learning and emulation, the usual paradigm is to prescribe a functional form for the model with some unknown parameters—and then fit those parameters to data. There can be several downsides to this typical approach, of which Gaussian process (GP)—or more specifically in the context of emulation, Gaussian process regression (GPR)—addresses two: interpretability and trustworthiness or UQ. GPs instead frame the problem not as determining a single function but instead as determining a probability distribution of functions themselves. A single GP represents infinitely many functions, called realizations, each with some associated probability. Importantly, GPR sidesteps explicitly enumerating these functions and instead works only in correlations between the data samples—a method ubiquitous in machine learning known as the *kernel trick* [86, 87]. The point-wise prediction from GPR often takes the form of a mean value—the value of the *mean function* of the GP—and confidence intervals which describe the distribution over the pointwise prediction. For a comprehensive introductory reference on GPs and GPR, see Refs. [88, 89].

GPR has seen enormous success in varied applications in nuclear physics emulation and model UQ [90–94]. It can be applied as the emulator directly, as part of an emulation framework, or as a UQ model fit to the underlying emulator. Despite its numerical non-parametricity, an excellent and unique use-case for GPR in scientific contexts is in informing where in feature space data or snapshots should be collected [95]. By guiding either experimental measurements or high-fidelity models to regions where an associated GPR-based emulator is most uncertain, one can effectively maximize the informativeness of each additional sample. In applications where every data point is expensive, maximizing the usefulness of each sample is critical.

In Ref. [96], the authors show that the convergence rate of the mean function as the number of training samples, m, approaches infinity is $\mathcal{O}\left(m^{-k}\right)$ with $1/2 < k < 1$. The exact value of k is determined by the dimensionality of the data, among other things, and generally decreases as the dimensionality increases. This can limit the effectiveness of GPR

Emulator Summary for Gaussian Process Regression	
Mathematically Parametric	Yes.
Numerically Parametric	No.
Nonlinear-Capable	Yes.
Computationally Efficient	No, though factorized and approximate methods exist which improve efficiency [97, 98].
Data Efficient	No. Converges as $\mathcal{O}\left(m^{-k}\right)$ with $1/2 < k < 1$.
Hyperparameter Efficient	Determined by the choice of kernel function. Usually only one or two relevant hyperparameters that each have clear interpretable meaning.
Systematically Improvable	No. Given infinite data, GPR will converge to the true function, but there is no guarantee of convergence with finite data.
Interpretable	Interpretable only in "data-space," as predictions are purely combinations of training examples. No guaranteed interpretability in the context of the original problem or high-fidelity model.
Physically Consistent	No.
Intrusiveness	Purely data-driven.
Quantified Uncertainties	All predictions are inherently realizations from probability distributions. The output of GPR can be the probability distribution itself, leading to built-in uncertainty estimates.

Table 2.8 Summary of properties for GPR-based emulators.

in high-dimensional regression.

2.11.3 Artificial Neural Networks

Artificial neural networks (ANNs) are a now-ubiquitous class of machine learning models that need little introduction. See Refs. [99, 100] for an introduction to ANNs. Perhaps

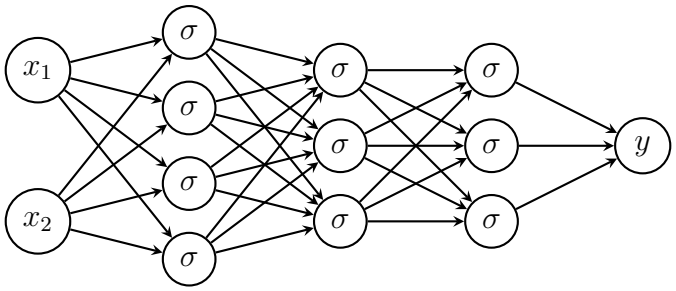

Figure 2.11 Schematic illustration of a multi-layer perceptron with two inputs (x_1, x_2), three hidden layers, and one output y. The hidden layers consist of four, three, and three neurons, respectively. Each edge represents a scalar multiplication by a *weight*. When multiple edges converge on a single neuron, the sum of the weighted inputs is passed through a nonlinear *activation function* σ, and then a scalar *bias* is added. For example, the output of the top node in the first hidden layer would be $\sigma(w_{11}x_1 + w_{12}x_2) + b_1$, where w_{11} and w_{12} are the weights of the edges connecting x_1 and x_2 to the node, and b_1 is the bias.

the simplest useful and most common ANN is the multi-layer perceptron (MLP), a type of feedforward neural network where each layer is fully connected to the next. Figure 2.11 shows a schematic of an MLP with two inputs, three hidden layers, and one output. MLPs can be used for nonlinear regression and classification, and therefore can be directly applied to emulation tasks. However, a more sophisticated approach which aims to incorporate the underlying physics has gained significant traction relatively recently. This method is called PINNs and combines both the versatility and variety of ANNs with available physics knowledge in order to produce models and emulators that are more accurate than just directly fitting the ANN as a black-box regression model.

For a review of PINN methods see Ref. [101], and for a practical introduction see Ref. [102]. The core principle of PINNs is the modification of the typical data-driven loss function with the addition of the residual of the underlying physics equations. This makes PINNs a hybrid emulation technique—being part data-driven and part intrusive. Uninformed ANNs are fit directly to data, such as snapshots of the system as in Galerkin's Method, such that the difference between the ANN's predictions and the data is minimized. In contrast, PINNs are fit such that the sum of this "data loss" and the residuals of the physics equations is minimized. For example, consider a PINN for a 2D parametric partial differential equation

of the form

$$\frac{\partial^2}{\partial x^2}u(x,y;c) + c\frac{\partial}{\partial y}u(x,y;c) = b(x,y), \tag{2.74}$$

for some function $b(x,y)$. We represent $u(x,y;c)$ as an ANN with three inputs x, y, and c, and one output $u(x,y;c)$. Given m_{data} snapshots of solutions of Eq. (2.74) for various values of x, y, and c, which we denote by $\{(x_i, y_i, c_i, u_i)\}$ with $i = 1, 2, \ldots, m$, the "data loss" would be

$$\mathcal{L}_{\text{data}} = \frac{1}{m_{\text{data}}}\sum_i |u(x_i, y_i, c_i) - u_i|^2, \tag{2.75}$$

which is known as the mean squared error.[19] The "physics loss" is given by the average squared residual of Eq. (2.74) over some set of m_{physics} points (x_j, y_j, c_j) which need not be the same points as in the snapshots,

$$\mathcal{L}_{\text{physics}} = \frac{1}{m_{\text{physics}}}\sum_j \left|\frac{\partial^2}{\partial x^2}u(x_j,y_j;c_j) + c_j\frac{\partial}{\partial y}u(x_j,y_j;c_j) - b(x_j,y_j)\right|^2, \tag{2.76}$$

where the derivatives of the output of the ANN, $\frac{\partial^2}{\partial x^2}u(x,y;c)$ and $\frac{\partial}{\partial y}u(x,y;c)$, are computed via automatic differentiation (AD) (see Section 3.4.1). The ANN is then fit, or trained, to minimize a weighted sum of these two losses,

$$u = \arg\min_u [w_{\text{data}}\mathcal{L}_{\text{data}} + w_{\text{physics}}\mathcal{L}_{\text{physics}}]. \tag{2.77}$$

Setting $w_{\text{physics}} = 0$ is equivalent to the traditional uninformed ANN approach.

This illustrates only the core idea of PINNs. Additional terms penalizing violations of desired physical properties can be devised and added to the loss function to improve the accuracy of the trained model. Significant research effort has been put in to devising specific ANN architectures that have certain symmetries or other properties that each physical system should have.

Both the offline and online efficiency of PINNs depends heavily on the specific ANN architecture. The offline cost is also strongly influenced by the cost of evaluating the residual of the underlying equations. Compared to RBMs, some sacrifices are made, particularly regarding

[19] Many different loss functions are equally valid here.

data efficiency, hyperparameter efficiency, and interpretability. Highly over-parameterized ANNs—the usual regime—can require significant amounts of data to train. Additionally, the performance of a PINN is highly dependent on a suitable set of hyperparameters and different target problems may require very different hyperparameters even for the same ANN architecture [103].

However, more recent works have indicated that many PINN results are overoptimistic in both accuracy and computational advantage [104, 105]. Traditional numerical methods—not emulators—still outperform PINNs in many cases [104].

Emulator Summary for Physics-Informed Neural Networks	
Mathematically Parametric	Yes.
Numerically Parametric	Yes.
Nonlinear-Capable	Yes.
Computationally Efficient	Highly dependent on the specific ANN architecture and evaluation of the residual of the informing equations.
Data Efficient	Usually only with sophisticated ANN architectures tailored to the target problem.
Hyperparameter Efficient	No. A vast continuum of ANN architectures exist to choose from. Even within a single ANN architecture there can be several hyper-parameters.
Systematically Improvable	In theory, the universal approximation theorem guarantees that a large enough ANN will converge absolutely to the data. However, the rate at which the size of the ANN must increase will, in practice, outpace the ability to train the model.
Interpretable	No. In general, the ANN architecture has little to do with the true physical system.
Physically Consistent	Approximately physically consistent, to the degree that the physics loss represents all relevant physical properties and is sufficiently minimized.
Intrusiveness	Hybrid.
Quantified Uncertainties	Not inherent, but can be added on top of the method.

Table 2.9 Summary of properties for PINN-based emulators.

2.12 Summary

This chapter introduced the concept of emulators, established the definitions for various properties of emulators, and presented several commonly used methods. Many of these methods can either trace their origins to, or be reinterpreted as, variations on Galerkin's Method—which here we call the explicit reduced basis method (eRBM) (Section 2.11.1). It is still useful to differentiate these methods in name from Galerkin's Method, not only for conciseness[20] but also for analyses which are only valid under the assumptions of specific methods.[21] Gaussian process regression (GPR) and Physics-informed neural networks (PINNs) are outliers as they cannot be cast as Galerkin-like methods and instead are formulated from probability distributions and artificial neural networks (ANNs) respectively (Sections 2.11.2 and 2.11.3). Each emulator has its own strengths, weaknesses, use-cases, variations, and ongoing research. Table 2.10 provides a summary of the properties of each emulator as discussed in this chapter.

[20] For instance, it is much simpler to say "proper orthogonal decomposition (POD)" instead of "Galerkin's Method with snapshots chosen as the first n principal components of the full matrix of snapshots" while conveying the same information.

[21] Such as the exponential convergence of eigenvector continuation (EC) which is not true in general for Galerkin's Method.

Method	Legend	Mathematically Parametric	Numerically Parametric	Nonlinear-Capable	Computationally Efficient	Data Efficient	Hyperparameter Efficient	Systematically Improvable	Interpretable	Physically Consistent	Intrusiveness	Quantified Uncertainties
	✓ Yes ✗ No ∼ Partial or conditional **I** Intrusive **N** Non-intrusive **H** Hybrid **A** Adaptive **E** Error bounds or estimates											
Galerkin												
Proper orthogonal decomposition		✓	✗	∼	✓	✓	✓	✓	✓	✓	**I**	**E**
Dynamic mode decomposition		∼	✗	∼	✓	✓	✓	∼	✓	✗	**N**	✗
Method in Section 2.11.1.3		✗	✗	✗	✓	✓	✓	✓	✓	✓	**I**	**E**
Eigenvector continuation		✓	✗	✗	✓	✓	✓	✓	✓	✓	**I**	**E**
Gaussian process regression		✓	✗	✓	✗	✗	∼	✗	∼	✗	**N**	✓
Physics-informed neural networks		✓	✓	✓	∼	∼	✗	∼	✗	∼	**H**	✗

Table 2.10 Summary of the properties of emulator methods discussed in this chapter. There are a few noteworthy trends: the methods that can be equated to some variation of Galerkin's Method often share many properties. Of the methods here, only the PINN is numerically parametric—an often highly desirable property—though this is at the expense of many other properties.

CHAPTER 3

PARAMETRIC MATRIX MODELS

Parametric matrix models (PMMs) were first introduced in Ref. [52] by me, Dr. Danny Jammooa, Dr. Morten Hjorth-Jensen, Dr. Daniel D. Lee, and Dr. Dean Lee. Originally developed simply as the next-generation eigenvector continuation (EC)-like emulator, the scope of PMMs reach far beyond standard eigenproblem emulation to nonlinear dynamics and general machine learning.

3.1 Parametric Matrix Models as *Implicit* Reduced Basis Methods

In the previous chapter, Galerkin-like methods were referred to as explicit reduced basis methods (eRBMs), despite literature referring to them simply as reduced basis methods (RBMs). In each of the Galerkin-like methods, both the *reduced space* $\mathcal{V}_n$ and the *reduced basis* $\{u_i\}$ were explicitly specified or chosen. However, note that the form of the Galerkin-projected problem in Eq. (2.10) is entirely independent of both the reduced space and reduced basis. Beyond just the form, there are cases where the predictions themselves are independent of the reduced basis or reduced space.

To illustrate this, suppose only the eigenvalues from a standard Hermitian eigenproblem are to be emulated. Using a proper orthogonal decomposition (POD)-Galerkin approach, following the same notation from Chapter 2 where P is the POD projector and $\underline{X} = P^\dagger X P$ and $\underline{x} = P^\dagger x$ denote reduced operators and vectors respectively, we have

$$H\Psi_k = E_k\Psi_k \quad \Longrightarrow \quad \big(P^\dagger H P\big)\big(P^\dagger\Psi_k\big) = E_k\big(P^\dagger\Psi_k\big) \quad \Longrightarrow \quad \underline{H}\,\underline{\Psi}_k = \underline{E}_k\underline{\Psi}_k. \tag{3.1}$$

Recall the most important property of the POD-Galerkin method: scalars, inner products, and norms are approximately preserved. This property guarantees that the projected problem is still a standard eigenproblem—that is, the $\underline{\Psi}_k$ are orthonormal—and $\underline{E}_k \approx E_k$. Now, consider all the transformations of the original problem that leave E_k unchanged as well as all the transformations of the projected problem that leave $\underline{E}_k$ unchanged. These are the set of all unitary transformations on the original space and reduced space respectively. Denote

a particular pair of these transformations by U and $\underline{U}$, then we know

$$\begin{aligned}
H\Psi_k &= E_k\Psi_k \\
\iff UHU^\dagger U\Psi_k &= E_k U\Psi_k \\
\implies \underbrace{\left(P^\dagger UHU^\dagger P\right)}_{\hat{\underline{H}}}\underbrace{\left(P^\dagger U\Psi_k\right)}_{\hat{\underline{\Psi}}_k} &= \underline{E}_k\left(P^\dagger U\Psi_k\right) \\
\iff \hat{\underline{H}}\hat{\underline{\Psi}}_k &= \underline{E}_k\hat{\underline{\Psi}}_k \\
\iff \underbrace{\underline{U}\hat{\underline{H}}\underline{U}^\dagger}_{\check{\underline{H}}}\underbrace{\underline{U}\hat{\underline{\Psi}}_k}_{\check{\underline{\Psi}}_k} &= \underline{E}_k\underline{U}\hat{\underline{\Psi}}_k \\
\iff \check{\underline{H}}\check{\underline{\Psi}}_k &= \underline{E}_k\check{\underline{\Psi}}_k
\end{aligned} \tag{3.2}$$

for the same $E_k \approx \underline{E}_k$ as before. That is, there are as many reduced models of the original problem with identical form that produce identical results as there are possible combinations of U and $\underline{U}$ (infinitely many). Instead of applying these transformations to the reduced model, we can apply them to the POD projector, $\hat{P} = U^\dagger P\underline{U}$, thus showing that for this problem there are infinitely many reduced bases—all corresponding to orthonormal bases of the same reduced space—that yield an emulator with identical predictions. Thus there exist many more emulators with identical forms and predictions than POD-Galerkin, or any Galerkin method—any *explicit* reduced basis method—can produce. Although illustrated with a particular example, this is true in general for any system with *symmetries* that leave the target values unchanged.

We have thus far neglected any parametric dependence of the system. Despite these transformations leaving the predictions of the emulator invariant at specific parameter realizations, the predictions may be significantly affected at other values of the parameters. Consider expanding upon the previous example by introducing the simplest affine parameter dependence,

$$H(\mu) = H_0 + \mu H_1. \tag{3.3}$$

It is again true that unitary transformations of $H(\mu)$ leave the eigenvalues $E_k(\mu)$ unchanged for all μ. However, any mixed unitary transformations of H_0 and H_1 trivially leave $E_k(\mu = 0)$

invariant while potentially altering $E_k(\mu \neq 0)$. Once more, this is not a specific quality of this example but true in general for any parametric system whose constituent components exhibit independent symmetries.

So we now see that any Galerkin emulator is actually just one element from a vast family of emulators with identical forms and equivalent predictions at some countable set of parameter realizations—usually the parameter realizations associated with training data. This observation leads us to two fundamental questions:

Q.1 Do there exist emulators in this family that are more useful than the Galerkin ones?

And if so,

Q.2 Can we describe an emulation method that is able to take advantage of these non-Galerkin emulators?

There are two complimentary ways to answer Q.1 in the affirmative. The first is the recognition that there are many real-world problems where the information required to construct any eRBM is unavailable either due to practical or fundamental reasons—despite the known existence of the reduced model. Multitudes of many-body methods for computing eigenenergies do not compute the full associated eigenstates—which would become the snapshots an eRBM uses—often because they are impractically large.[1] However, the complete state vector of a system—especially a quantum one—is rarely the quantity of interest. Instead, relatively few scalar quantities like energies, expectation values, and transition amplitudes are the more common goal. Even problems which require the full "microstate" solution are often solved without direct access to the full space operators and instead use ensemble or Monte Carlo methods, such is the case in fluid dynamics with the lattice Boltzmann method [106]. In either of these cases, the form of the full model is still known, and given unlimited computational power we usually could attain the information necessary to construct an eRBM, so we

[1] Just one eigenvector from a $9 \times 9 \times 9$ lattice chiral effective field theory calculation of ^{6}Li would require more than 1000 TB to store.

know that reduced-space models exist. Having a method that allows us to shortcut directly to the reduced model without the need to explicitly construct a basis and project the full model onto it would allow RBM emulation in these ubiquitous contexts.

The second approach to answering Q.1 is to show there exists at least one case where the eRBM approach fails to find a suitable reduced model despite the existence of one. Following the example laid out in Ref. [52, Methods], we again consider a standard parametric Hermitian eigenproblem. Consider a system of S non-interacting spin-1/2 particles with the parametric Hamiltonian

$$H(c) = \frac{1}{2S}\sum_{i}^{S}(\sigma_i^z + c\sigma_i^x) = \frac{1}{2S}(Z + cX), \tag{3.4}$$

where c is the real-valued parameter, σ_i^u is the standard u Pauli matrix for spin i, and Z, X are the total z-spin and total x-spin operators respectively. We seek an emulator for the ground state energy of this system as a function of c for a given system size S based on data from $n = 5$ samples for $c < 0$. Constructing the 5×5 EC—or equivalently POD—emulator $\underline{H}(c) = \frac{1}{2S}(\underline{Z} + c\underline{X})$ for this problem shows significant deviations in the prediction from the true energy for $c > 0$ when S becomes large, as shown in Fig. 3.1. However, there are infinitely many solutions of the same 5×5 form as this EC emulator which reproduce the true ground state energies exactly. One such family of emulators is given by the ground state energy of the block-diagonal parametric matrix

$$M(c) = \frac{1}{2}\left(\begin{bmatrix} \sigma^z & 0 \\ 0 & h \end{bmatrix} + c\begin{bmatrix} \sigma^x & 0 \\ 0 & 0 \end{bmatrix}\right), \tag{3.5}$$

where σ^z and σ^x are the standard 2×2 z and x Pauli matrices and h is any 3×3 Hermitian matrix with all eigenvalues strictly greater than -1. The predictions from this emulator are shown in Fig. 3.1 as well and match the true energies exactly.

[2]In Ref. [52] the total z-spin basis (basis vectors $|Z\rangle \in \{|-S/2\rangle, |-S/2+1\rangle, \ldots, |S/2-1\rangle, |S/2\rangle\}$) was used for evaluations of the Hamiltonian in Eq. (3.4) which allowed for system sizes as large as $S = 256$ while only requiring exact eigenvectors of size $N = 257$. By contrast, here we use the individual z-spin basis (basis vectors $|\sigma_1^z\sigma_2^z\cdots\sigma_S^z\rangle \in \{|\downarrow\downarrow\cdots\downarrow\downarrow\rangle, |\downarrow\downarrow\cdots\downarrow\uparrow\rangle, \ldots, |\uparrow\uparrow\cdots\uparrow\downarrow\rangle, |\uparrow\uparrow\cdots\uparrow\uparrow\rangle\}$) which require exact eigenvectors of size $N = 2^S$. Thus, Fig. 3.1 represents a dimensionality reduction from at most $N = 2^{12} = 4096$ to $n = 5$ while the equivalent figure in Ref. [52] represents only a reduction from at most $N = S + 1 = 257$ to $n = 5$. The conclusions are unchanged in either case.

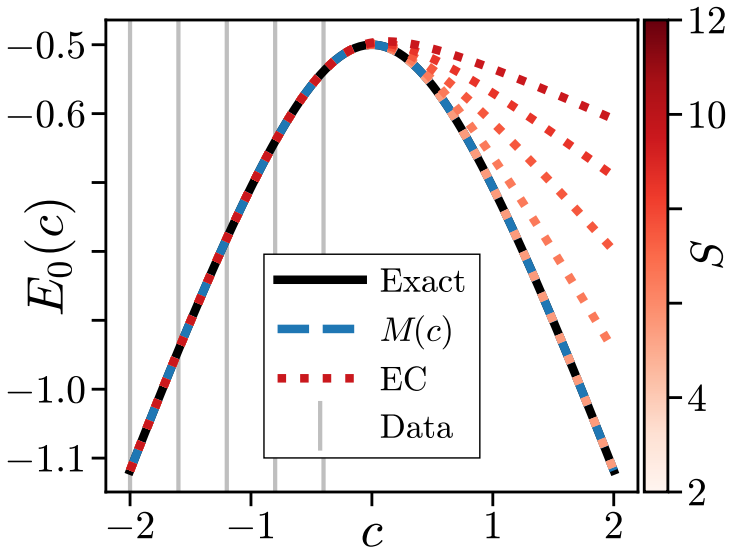

Figure 3.1 Ground state energy as a function of c for the parametric Hamiltonian in Eq. (3.4) (solid black) as well as the predictions from an eRBM (specifically EC, dotted red) and from one realization from the family of emulators described by Eq. (3.5) (dashed blue) for different system sizes S. The location of training data for the emulators is indicated by vertical gray lines. Based on a similar figure from Ref. [52] with permission.[2,3]

The answer to Q.2 is the central topic of this thesis, parametric matrix models (PMMs), which can be viewed as a class of implicit reduced basis methods (iRBMs). Instead of specifying a subspace $\mathcal{V}_n$ and projection operators $P^{\uparrow}$ and ${}^{\downarrow}P$, a PMM emulator specifies only the absolute minimum to describe the form of the reduced model: the size of the reduced space n and properties of the projection operators (e.g. orthogonality). This mathematical form is then concretized by solving an optimization problem to fit the reduced model to available data—known as *training* the model. In this sense the basis of the resulting reduced model is *implicit*: the information for the reduced basis is contained within and inseparable from the operators of the reduced model. Following the previous example, we saw that a reduced model of the ground state energy of Eq. (3.4) for an $n = 5$ reduced space arising from an orthogonal projector had the form of the ground state energy of

$$\underline{H}(c) = \frac{1}{2S}(\underline{Z} + c\underline{X}). \tag{3.6}$$

In the PMM approach, it is not the case that $\underline{A} = P^{\dagger}AP$ since P is not known and may

[3] I am the author of the original figure in Ref. [52] and gave permission.

vary between constituent components of the reduced model. Instead, the reduced operators are the solution to the minimization of some loss function relating the predictions of the emulator to available data. For this example,

$$\underline{Z}, \underline{X} = \underset{\underline{Z}, \underline{X} \in \mathbb{S}(n)}{\arg\min} \mathcal{L}\big(\{E_0(c_i)\}, \{\underline{E}_0(c_i)\}\big) \tag{3.7}$$

where $\underline{E}_0(c_i)$ is the ground state energy of Eq. (3.6) for $c = c_i$, $\{E_0(c_i)\}$ is the set of data to fit the reduced model to, $\{\underline{E}_0(c_i)\}$ are the predictions from the reduced model at the same values of c as the data, $\mathcal{L}$ is some *loss function* which attains its minimum when $\{E_0(c_i)\} = \{\underline{E}_0(c_i)\}$, and $\mathbb{S}(n)$ is the set of all $n \times n$ symmetric matrices. In this example, we know $\underline{Z}$ and $\underline{X}$ are symmetric because we specified that they are the result of an orthogonal projection on Z and X which are also symmetric. There is no unique solution to Eq. (3.7), and by construction the less-desirable eRBM is a solution. Guiding the minimization to more robust solutions is an exercise in machine learning solved through various techniques such as regularization and validation which will be discussed later.

In the broadest possible terms, the steps of using the PMM method to form an emulator are as follows:

(0.) Provide the *form* of the full model[4] that produces target values as well as a set of training examples of those target values.

1. Select the desired reduced-space dimension n and properties of an unknown (implicit) projection from the full space to a reduced space of this size.

2. Follow the process of deriving the form of the eRBM that would result from projecting the full model with the chosen projection. This includes identifying any known properties (such as matrix symmetry and vector normalization) of the objects in the reduced model.

3. Replace all unknown projected objects with unknown numerically parameterized ones that respect the identified known properties.

[4]It is also possible to prescribe or suppose a form for the underlying model if it is not wholly known.

4. Identify a suitable *loss function* which encodes the dissimilarity between predictions from the parameterized reduced model and the training examples.[5]

5. Find the location of the minimum of the loss function over all possible values of the parameterized objects.

6. The values of the parameterized objects at the minimum are the *trained* objects and the reduced model with these parameters is the trained PMM emulator.

A projection with many of the properties of an orthogonal projection is nearly universally chosen in step 1 due to the numerous properties that are preserved under such transformations—see Section 2.11.1.1 in Chapter 2. Unless stated otherwise, in this thesis we will assume all PMMs choose a projection that exactly preserves Hermiticity and approximately preserves some eigenvalues (if applicable) and inner products[6]—which is similar to but not necessarily exactly an orthogonal projection. As a consequence of the PMM following the same form as the associated eRBM, all the parameterized objects in the reduced model will be scalars, vectors, matrices, or tensors.[7] Just like in many other machine learning tasks, the loss function chosen in step 4 may include additional penalties; however, this is usually unnecessary for PMMs. The potential drawbacks of the PMM method lie in steps 2, 3, and 5. Compared to purely data-driven methods and eRBMs, PMMs require work on the part of the implementer to carefully derive the form for any given problem. Laying out a PMM for a new problem requires intimate understanding of both the problem itself and the theory behind RBMs. Through Chapter 2 and the numerous examples in this chapter, the reader should

[5]The problem statement here is in the context of *supervised learning*—that is, the context in which the target predictions for the training data are known. However, all discussions in this chapter generalize to *unsupervised* contexts—those where the exact target output values are not known and instead it is often properties of the distribution of output values like clustering that are desired—only by changing to a suitable unsupervised loss function.

[6]There is research that suggests oblique (non-orthogonal) projections can perform just as well or better than orthogonal ones for certain problems [107]. However, orthogonal-like projections are still currently preferred in practice due to the numerous conserved properties of operators under orthogonal projections as well as the added difficulty in training an oblique PMM.

[7]Of course, these are all tensors: order-0, order-1, order-2, and above. Equivalently, it can be said that the reduced objects are always multidimensional arrays.

be well-prepared and find this potential drawback a non-issue. The optimization problem in step 5 is exceedingly nontrivial for all but the smallest and simplest PMMs. Fortunately, decades of research in training ever-larger artificial neural networks (ANNs) have produced both the knowledge and the software necessary to obtain satisfactory approximate minima in a controlled and efficient manner.

3.2 Variations

The typical PMM application, as described above, is one where scalar values resulting from a process with a known form is emulated. However, PMMs which produce vector outputs like the associated eRBM as well as fully- and partially-data-driven PMMs are possible and useful. These variations are not mutually exclusive; a vector-valued PMM can also be partially data-driven. There are more variations than discussed in this section.

3.2.1 Vector-Valued Parametric Matrix Models

In order for a PMM to produce an output vector in the full space, there is little choice but to make use of the eRBM projector[8] to take predictions from the reduced space back to the full space. Recall from the statement of Galerkin's Method in Eq. (2.15) that the reduced (linear) problem $\underline{A(\mu)}\underline{x(\mu)} = \underline{b(\mu)}$ is solved independently of the full space and only afterwards projected back to the full space via $x(\mu) \approx P^{\uparrow}\underline{x(\mu)}$. For a vector-valued PMM, the operators and various objects in the reduced space can still be learned directly from data as opposed to constructed like in the eRBM approach, and this learning can happen entirely in the reduced space. But some of the expressiveness granted by the implicit reduced basis approach is lost. Still only the form of the full model is necessary, which does grant these PMMs more flexibility compared to their associated eRBMs. The exact modifications necessary to the steps outlined in Section 3.1 are as follows. Two additional steps between steps 1 and 2 are added which state,

+1.1. Compute the chosen projectors $P^{\uparrow}$ and ${}^{\downarrow}P$ (almost always the POD projectors $P^{\uparrow} = P$

[8] Or have the dimensionality of the PMM be the same as the full space, defeating the point of emulation in nearly all cases.

and ${}^{\downarrow}P = P^{\dagger}$) for the chosen reduced-space dimension n using the available training snapshots.

+1.2. Project the training snapshots to the reduced space using the computed projector ${}^{\downarrow}P$.

Then the loss function in step 4 is modified to encode the difference between the projected training snapshots from step 1.2 and the PMM predictions still in the reduced space ($\underline{x(\mu)}$ following the linear example discussed before). Finally, after step 6 we make it clear that the PMM predictions must be projected back to the full space,

+6.1. Denoting the predictions of the PMM in the reduced space as $\underline{x}$, the predictions in the full space are $P^{\uparrow}\underline{x}$.

3.2.2 Fully- and Partially-Data-Driven Parametric Matrix Models

There are many applications where even just the form of the full space model is not wholly known. On one end of the spectrum, problems like purely-data-driven regression have no information about the equations of the underlying system—if one even exists. Instead, machine learning methods for these problems must assume suitably expressive—or *universal*—forms that can be tuned to fit almost any data. This is what techniques like least-squared regression or multi-layer perceptrons do by supposing a model of linear combinations of the data in the former and a feedforward ANN in the latter. PMMs for general non-sequential data-driven regression and classification can take many forms, but one proven option is that which is detailed in Ref. [52] where the process from inputs to outputs is described by a set of observables in a quantum system whose Hamiltonian is parameterized by the inputs. This is discussed in detail in Section 3.8. Near the opposite end of the spectrum, there are countless problems where most of the equations are known but a single or few important terms are not. In this case, missing terms can be replaced by data-driven machine learning methods while the known terms remain intact. This technique of a partially-data-driven PMM is discussed in Section 3.12.

3.3 Properties

For now we will put aside the task of training and consider the properties of trained PMMs. By construction, a PMM inherits almost all of the properties of the associated eRBM. This is why the choice of an implicit orthogonal projection is so common; as discussed in Section 2.11.1.1 orthogonal projections preserve many properties and operations. However, a key advantage of PMMs is that they can strategically depart from the associated eRBM construction to avoid downsides or deficiencies. We have already seen this to some extent, but will see more when discussing nonlinear PMMs in this section. The central takeaway is that PMMs can be constructed to inherit all of the advantages of the associated eRBM with little or none of the downsides.

To avoid unnecessary repetition, only properties of PMMs which differ or warrant clarification compared to the associated eRBM are included in this section.

3.3.1 Parametricity

PMMs inherit mathematical parametricity from the associated eRBM. Just as discussed in Section 2.11.1, the parametric dependence on any control parameters can be arbitrarily nonlinear and complicated without affecting the complexity of the emulator. On the other hand, PMMs are numerically parametric by design. No method in Chapter 2, besides the physics-informed neural network (PINN), was numerically parametric. In emulation and broader machine learning, numerical parametricity is a highly desirable property both for computational efficiency and model flexibility.

The dimensionality of a PMM is a completely free hyperparameter independent of the data, and the introduction of additional data consistent with the original does not necessitate re-training the parameters of the PMM—similar to ANNs. For a general order-d linear operator[9] acting on elements of an N-dimensional vector space, the number of parameters for the PMM representation of the operator is n^d where n is the chosen dimensionality of the PMM. This is the same for eRBMs, but now the chosen n and emulator parameters are

[9]For example, a matrix is an order-2 linear operator.

independent of the data or snapshots.

3.3.2 Linearity

Following the treatment of nonlinear operations in Section 2.11.1.1, PMMs can also efficiently handle polynomially nonlinear equations directly and arbitrary elementwise nonlinearities via lifting transformations.

Additionally, a more efficient approach with close ties to the original eigenvalue emulator interpretation of PMMs exists. Strictly speaking, this approach would be available to the POD-Galerkin eRBM as well, but it does not appear in literature.[10,11] This alternative approach relies on the canonical matrix-valued version of the nonlinearity to transform the full-space problem before constructing the reduced basis form. Consider the problem of computing

$$F(v) = y(v) \odot v \tag{3.8}$$

in the full space. The only requirement of y is that it is elementwise and analytic; it does not need to be elementwise smooth like in the discrete empirical interpolation method (DEIM). This form of nonlinearity—a vector-valued nonlinear term multiplied elementwise with the vector itself—is ubiquitous throughout scientific computing. In nuclear and quantum physics, several high-profile nonlinear systems exhibit this form including the exchange operator in the Hartree–Fock method, the mean-field term in the energy functional in density functional theory, and the mean-field term in the Gross–Pitaevskii equation [108–110]. The first step is to replace the elementwise product in Eq. (3.8) with a different nonlinearity, the $\mathrm{diag}(\cdot)$ operator,

$$F(v) = y(v) \odot v = \mathrm{diag}(y(v))v, \tag{3.9}$$

where $\mathrm{diag}(x)$ is a diagonal matrix whose elements are the elements of x. The product between $\mathrm{diag}(y(v))$ and v is the standard inner product. We now introduce the canonical matrix-valued version of y. For any elementwise and analytic y, we define Y to be the

[10] To the best of my knowledge at the time of writing.

[11] I include it in this chapter rather than the previous since it was developed in the context of PMMs and has only been verified with PMMs.

matrix-valued version as

$$Y(A) \equiv V_A \text{diag}(y(\lambda_A)) V_A^{-1} \quad \text{for} \quad A = V_A \text{diag}(\lambda_A) V_A^{-1}, \tag{3.10}$$

where λ_A is the vector of eigenvalues of A. $Y(A)$ is only defined in this way for diagonalizable A with eigenvalues that y is analytic over. Note that by definition $Y(A)$ commutes with similarity transformations of A—that is, $Y(QAQ^{-1}) = QY(A)Q^{-1}$. This is not a new concept; the matrix exponential, logarithm, power, inverse, trigonometric functions, and many more are often expressed and implemented this way. Making explicit use of this allows us to factor the nonlinear function y out of the $\text{diag}(\cdot)$ operation in Eq. (3.9),

$$F(v) = \text{diag}(y(v))v = Y(\text{diag}(v))v. \tag{3.11}$$

While these transformations may initially seem to have made the problem more complicated, we can now derive the form of the reduced model assuming an orthogonal projector, P. We follow the naive handling[12] of the nonlinearity from Section 2.11.1.1 and project any remaining objects down to the reduced space. Denote the reduced-space representation of $\text{diag}(v)$ by $\underline{D_v}$. Then,

$$\begin{aligned} Y(\text{diag}(v)) \to P^\dagger Y\big(P\underline{D_v}P^\dagger\big)P &\approx P^\dagger P Y\big(\underline{D_v}\big)P^\dagger P \\ &= Y\big(\underline{D_v}\big), \end{aligned} \tag{3.12}$$

where the approximation is valid from the fact that Y commutes with similarity transforms, which translates to an approximate commutation with the compression. We find $\underline{D_v}$ similarly,[13]

$$\begin{aligned} &\text{diag}(v) \to \underline{D_v} = P^\dagger \text{diag}(P\underline{v})P \\ \iff &\underline{D_v}_{ij} = \underbrace{P^\dagger{}_{i\nu} P_{\nu\mu} P_{\nu j}}_{\underline{G}^{(2)}} \underline{v}_\mu \\ \iff &\underline{D_v} = \underline{G}^{(2)} \underline{v}, \end{aligned} \tag{3.13}$$

[12] Since the nonlinear term is operator-valued (matrix-valued) here instead of vector-valued as in Section 2.11.1.1, we have additional right multiplications on both the argument and the result: $Y(X) \to P^\dagger Y(P\underline{X}P^\dagger)P$.

[13] Here, we only have an additional right multiplication by P on the result of the nonlinear operation $\text{diag}(\cdot)$ since its argument is a vector.

where $\underline{G}^{(2)}$ is the same order-3 tensor of the same name from Section 2.11.1.1 which enabled nonlinearities of the form $a \odot b$—the difference here is that only one index is contracted with the reduced vector $\underline{v}$ instead of two. Combining Eqs. (3.12) and (3.13) to form the complete reduced nonlinearity which multiplies $\underline{v}$ we have

$$F(v) = y(v) \odot v \to Y\left(\underline{G}^{(2)}\underline{v}\right)\underline{v}, \tag{3.14}$$

which only requires operations in the reduced space. Note that despite the fact that we have made no assumptions on the content of y other than that it is elementwise and analytic over all relevant v, the largest tensor is the order-3 tensor $\underline{G}^{(2)}$. Previously in Section 2.11.1.1 we saw that $\underline{G}^{(2)}$ was the reduced form of the elementwise product nonlinear operation and higher-order tensors were required for higher powers. Now we see that we can instead use $y(x) = x^p$ for any $p \in \mathbb{C}$ and treat any exponential nonlinearity with only this single order-3 tensor. Moreover, nonlinear operations that may have required numerous lifting transformations can instead be treated with the same computational overhead as the simplest polynomial nonlinearity. Importantly, the nonlinear operation itself has been preserved. This method requires no linearization, additional data, or hyper-reduction and converges to the exact nonlinearity as the dimension of the reduced space approaches the dimension of the full space.

3.3.3 Computational Efficiency

The complexity of making a prediction with a PMM is the same as the cost of the associated eRBM. Previously in Section 2.11.1.1 we saw that predictions from POD-Galerkin emulators scaled polynomially in the size of the emulator so long as all nonlinearities were handled efficiently.

Prediction complexity only reflects the online costs of the model. Since PMMs do not necessarily construct the projectors, the only offline costs are those of fitting the unknown operators. Just as for the method of PINNs in Section 2.11.3, if we assume an iterative fitting scheme then the offline time complexity of training the model is merely the online time complexity times the number of training examples, and the space complexity is unchanged

from the online phase. The desired accuracy sets the prefactor of this scaling. Therefore both the online and offline complexities are independent of the full space dimension and scale polynomially with the reduced-space dimension.

As will be seen later this chapter, PMMs that target vector-valued outputs the size of the full space will make use of the associated eRBM projector and thus for POD-Galerkin-based PMMs will require an additional $\mathcal{O}(N^2 n)$ offline and $\mathcal{O}(Nn)$ online time complexity—where N is the dimension of the full space and n is the dimension of the reduced space. Importantly, many loss functions used during training can be computed directly in the reduced space. This bypasses the additional $\mathcal{O}(Nn)$ prediction cost during training.

In the case of purely data-driven PMMs, the forms presented in Ref. [52] are shown to have online costs which scale polynomially in n.

3.3.4 Data Efficiency

The convergence rate of a PMM is highly dependent on the underpinning eRBM. By construction, the resulting PMM—if trained properly—will perform at least as well as the associated eRBM. For general POD-Galerkin-based PMMs, following the discussion surrounding Eqs. (2.31) and (2.32), approximate error bounds depend on the singular values of the matrix of training vectors. As emphasized before, there may not be any training vectors when using a PMM and therefore seemingly no concept of singular values of the training data. However, just as the projectors are guaranteed to exist even when they are not obtainable, all scalar "snapshots" have some unknown (possibly unknowable) associated vector which together constitute this hidden matrix of snapshot vectors. The associated singular values, while unknown, come from this data. If this data is typical real-world data, then just as discussed previously around Eq. (2.33) the singular values will approximately decay either polynomially or exponentially. This allows us to say that the approximate upper bound on the relative error for scalar values is either α^{-n} or $n^{-\beta}$ for a PMM of dimension n and some positive constants α and β. In either case, this shows that the typical upper bound on the error is $\mathcal{O}\big(n^{-\beta}\big)$. At first glance, this appears to be independent of the number of "snapshots"

and to depend only on the size of the PMM. This is due to the simplification we made in ignoring the training process. As expected by intuition, larger PMMs (larger values of n) require more data to accurately fit the unknown parameters. A simple upper bound for the error in terms of the number of training examples is found by taking the minimum of the PMM dimension n and the number of training examples m, since the assumed underlying eRBM for $n > m$ must have a rank-deficient projector. Thus we can state the final approximate error scaling for POD-Galerkin-based PMMs to be $\mathcal{O}\Big(\min(m,n)^{-\beta}\Big)$ for some problem-dependent $\beta > 0$.

3.3.5 Hyperparameter Efficiency

The only hyperparameter inherent in the PMM method is the size of the reduced space n. Purely data-driven PMM forms in Ref. [52] introduce just one additional hyperparameter with a very restricted set of possible values. Improvements on these data-driven forms, discussed in Section 3.8 introduce one more hyperparameter.

Partially data-driven PMMs can have many more hyperparameters due to the freedom in choosing universal functions. Whether or not the hyperparameters introduced are few and interpretable is problem- and architecture-specific.

3.3.6 Intrusiveness

As alluded to in Chapter 2, PMMs do not fit cleanly into one of the three existing categories of emulator intrusiveness. Without leaving the PMM framework, intrusive, data-driven, and hybrid emulators can be created. If the training process leverages the same information and techniques as the underlying eRBM then the PMM is functionally identical to the maximally-intrusive eRBM. Conversely, if absolutely no information of the full system is known or used and instead a universal PMM form is substituted, then the resulting emulator will be a fully data-driven PMM. The spectrum of hybrid PMM emulators follows from relaxing the training process and potentially substituting parts of the PMM emulator for data-driven components.

This qualifies PMMs as hybrid emulators. What sets PMMs apart from existing hybrid

emulators is the idea that the "hybrid-ness" of the PMM method is not fixed and instead the extent to which a particular PMM is intrusive or data-driven is determined by the context of the problem and available data. Existing hybrid emulation techniques, like PINNs, have limits to how intrusive or data-driven they can be. For example, a PINN can be purely data-driven—in which case it is equivalent to a feedforward ANN regression model. But a PINN cannot be fully intrusive, as by design it will always represent the reduced-space system as an ANN.

It is for this reason we introduce a fourth class of emulators called *adaptive* emulators, defined previously in Section 2.9. Emulators in this class adapt to the types of data available and are able to use all relevant data to inform the reduced-space model. Adaptive emulation methods represent large frameworks that are always applicable[14] regardless of the specific problem statement.

3.3.7 Quantifiable Uncertainties

The useful error bounds and estimates of the underlying eRBM are still applicable for PMMs. However, one important distinction is that—without more careful parameterization to explicitly restrict the possible values for the reduced operators—the condition number of the reduced operators are not necessarily smaller than their full-space counterparts. This is a consequence of choosing a projector with only some of the properties of an orthogonal projector and a central example of the freedom and importance in choosing a "good" parameterization for the reduced objects.

In theory, uncertainty estimates can be obtained directly from the Hessian of the loss function [111]. This arises through a quadratic expansion of the loss around the solution to the minimization problem. However, this is almost universally prohibitively expensive due to the potentially large number of fit parameters.

Compared to traditional hybrid emulation methods that utilize ANNs, PMMs are particularly amenable to uncertainty quantification via conformal prediction—covered in Sec-

[14]Note that this is not the same as saying they are always effective or the best method.

tion 4.2.

3.4 Training

The process of determining the values of the numerically parameterized objects in a PMM, or any machine learning model, in order to minimize some loss function—step 5 in Section 3.1—is called *optimizing*,[15] *fitting*, or *training*. Since there are as many different loss functions and PMMs as there are RBMs and as many RBMs as possible problems and data, we seek an optimization method that is highly versatile and minimally intrusive.[16] To accommodate diverse PMMs from just a few trainable parameters through to millions of trainable parameters[17] and datasets of equally variable size, we adopt the deep learning approach of gradient-based optimization, usually called *gradient descent*.

Consider the loss function for a particular model as a function of the trainable parameters, $\mathcal{L}(\theta)$. The loss function must be bounded from below—it must have a minimum value. Gradient descent methods are iterative methods that update the parameters based on the gradient of the loss function, $\frac{\partial \mathcal{L}}{\partial \theta}$, and potentially higher-order derivatives. The loss function is a real-valued function of which we want to find the location of the minimum. The simplest gradient descent method is the method of steepest descent,[18] which proposes the iterative update

$$\theta \leftarrow \theta - \eta \left.\frac{\partial \mathcal{L}}{\partial \theta}\right|_{\theta} \tag{3.15}$$

where $\eta > 0$ is the *learning rate*, the step size for each iteration. Figure 3.2 shows an illustrative example for this update in 1D. The figure additionally illustrates a fundamental challenge of gradient descent methods: local minima and saddle points. At these critical points, the gradient of the loss function is zero ($\frac{\partial \mathcal{L}}{\partial \theta} = 0$) but the function has yet to achieve its

[15] "Optimizing" in this context is distinct from and should not be confused with the optimization of the architecture itself or the implementation (code).

[16] Not to be confused with emulator intrusiveness.

[17] PMMs with trainable parameter counts beyond several million would also be trainable by these methods, but no work to date has done this.

[18] This method was originally just called "gradient descent," but this term now refers to a family of methods. The original gradient descent method is sometimes called "vanilla gradient descent" or "method of steepest descent" as used here.

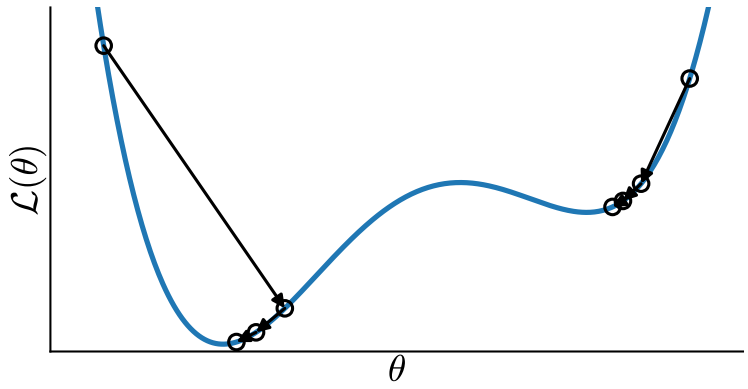

Figure 3.2 Illustrative example of three steps of the steepest descent method in 1D. Two initial points are shown. The initial point on the left converges to the global minimum while the initial point on the right converges to a local minimum.

global minimum. When optimizing any model, we are interested in the global minimum—the best possible model given the model architecture and loss function. Higher-order derivatives of the loss function can help overcome this challenge, but are significantly more expensive and quickly become computationally impractical. It is for this reason that first-order methods—those which only use the first derivative of the loss function—will often include momentum-like terms or adaptive learning rates to encourage the iterations to pass through or by any critical points that are not the global minimum.

A local minimum with a loss value close to the global minimum is also an acceptable result from the training process, since it would yield a model with nearly identical performance. Luckily, as the number of trainable parameters increases, the likelihood that any critical point is a "bad" local minimum—that is, one with a loss value significantly larger than the global minimum value—is vanishingly small [112]. Instead, almost all critical points are either saddle points or nearly equal to the global minimum. A sketch of the proof for this starts with the identification that, at a critical point, any one direction in parameter space has a non-zero, non-unit, probability $0 < p < 1$ to have positive curvature (and therefore would be a local minimum along this direction only). This probability is around 0.5 initially and approaches 1 as the loss function nears its global minimum value—independent of the location θ. Away from the global minimum value where p is not yet 1, the probability that

a particular critical point is a local minimum is p^N with N being the number of trainable parameters—which approaches 0 as N increases. Near the global minimum value however, $p \approx 1$, and nearly every critical point is a local minimum. This is acceptable however, since their values are very near the global minimum value.

To support PMMs with complex parameters, we treat the real and imaginary components of each parameter as independent real variables. Let the vector of complex parameters be decomposed as $\theta = a + ib$ with a and b purely real vectors. We can then update a and b independently using Eq. (3.15),

$$\begin{aligned} a &\leftarrow a - \eta \left.\frac{\partial \mathcal{L}}{\partial a}\right|_a, \\ b &\leftarrow b - \eta \left.\frac{\partial \mathcal{L}}{\partial b}\right|_b. \end{aligned} \tag{3.16}$$

We now switch to using Wirtinger derivatives for notational compactness, but this will also be practically impactful shortly. In Wirtinger calculus, a complex variable z and its complex conjugate z^* are treated as independent variables. Suppose $z = x + iy$ with x and y real numbers, then the Wirtinger derivatives are defined as [113]

$$\begin{aligned} \frac{\partial}{\partial z} &\equiv \frac{1}{2}\left(\frac{\partial}{\partial x} - i\frac{\partial}{\partial y}\right), \\ \frac{\partial}{\partial z^*} &\equiv \frac{1}{2}\left(\frac{\partial}{\partial x} + i\frac{\partial}{\partial y}\right). \end{aligned} \tag{3.17}$$

This readily generalizes to vector derivatives with complex vectors. Using these operators, we can simplify Eq. (3.16) to

$$\theta \leftarrow \theta - \eta \left.\frac{\partial \mathcal{L}}{\partial \theta^*}\right|_\theta, \tag{3.18}$$

where the factor of $1/2$ in the definition of the Wirtinger derivative has been absorbed into the learning rate η [114]. Note that the derivative is now with respect to the conjugate of the trainable parameters, θ^*. Since the loss function is a purely real-valued function,[19] we have

$$\left(\frac{\partial \mathcal{L}}{\partial \theta}\right)^* = \frac{\partial \mathcal{L}^*}{\partial \theta^*} = \frac{\partial \mathcal{L}}{\partial \theta^*}. \tag{3.19}$$

[19]Since complex numbers cannot be ordered, the concept of "minimizing" a complex-valued loss function is undefined.

And thus the complex-valued update can be equivalently written as

$$\theta \leftarrow \theta - \eta \left(\left. \frac{\partial \mathcal{L}}{\partial \theta} \right|_{\theta} \right)^{*}, \tag{3.20}$$

where the derivative is the Wirtinger derivative [115]. Beyond just a notational convenience, major libraries including JAX, PyTorch, and TensorFlow all implement automatic differentiation of real functions[20] of complex variables via Wirtinger derivatives [116–118].[21]

Significantly more advanced and successful gradient descent methods have been developed, and the training of machine learning models remains a vast and evolving topic. For an excellent introduction and overview of gradient descent techniques, see Refs. [100, 119].[22] For the training of PMMs we utilize a modification of the Adam[23] method of mini-batch stochastic gradient descent [120]—modified to support complex-valued parameters. Adam uses two auxiliary vectors, m and v, of the same shape as the vector of trainable parameters to track a running average of the gradients and second moment of the gradients respectively for each trainable parameter. These running averages additionally "forget" the history exponentially—with rates set by two independent hyperparameters, β_1 and β_2. This allows Adam to adjust the learning rate independently for each parameter based on the trajectory of the training process. "Mini-batch stochastic" gradient descent refers to using a subset of the training data (a mini-batch) in the loss function at each update.[24] In practice, training is divided into *epochs*. During each epoch, the whole of the training data is randomly shuffled and divided into these mini-batches, and the model parameters are updated using the update rule for each mini-batch sequentially until the model has seen all of the training samples. Denoting the epoch number by t (starting at epoch 1) and taking the initial values of m and

[20] And complex-valued holomorphic functions, but the details of the implementations here are only in the context of real-valued functions.

[21] It is worth noting some inconsistencies between the three libraries mentioned here at the time of writing: JAX omits the factor of 1/2 from the definitions of the derivatives in Eq. (3.17) and the provided `grad` method implements $\partial/\partial z$, while the equivalent methods in PyTorch and TensorFlow include the 1/2 factor but implement the conjugate gradient $\partial/\partial z^*$.

[22] Specifically, Chapters 4 and 8 in Ref. [119] and Chapter 12 in Ref. [100].

[23] From "adaptive moments."

[24] Pure stochastic gradient descent uses just a single training example per update.

v to be the zero vector, the update rule for Adam with complex parameters is

$$\begin{aligned}
g &\equiv \left(\frac{\partial \mathcal{L}}{\partial \theta}\bigg|_{\theta} \right)^{*} && \text{(conjugate of the gradient of the loss)} \\
m &\leftarrow \beta_1 m + (1-\beta_1) g && \text{(update gradient running average)} \\
v &\leftarrow \beta_2 v + (1-\beta_2)|g|^2 && \text{(update second moment running average)} \\
\hat{m} &\equiv \frac{m}{1-\beta_1^t} && \text{(bias correction)} \\
\hat{v} &\equiv \frac{v}{1-\beta_2^t} && \text{(bias correction)} \\
\theta &\leftarrow \theta - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \varepsilon} && \text{(update parameters)}
\end{aligned} \tag{3.21}$$

where $|g|^2$ denotes the elementwise absolute value squared ($\left(|g|^2\right)_i \equiv |g_i|^2 = g_i^* g_i$), the two bias correction steps account for the bias introduced by initializing the running averages to zero, η is the learning rate, and $0 < \varepsilon \ll 1$ is a small positive value to prevent division by zero. Despite the introduction of three new hyperparameters (β_1, β_2, and ε), it is rarely the case that these need to be changed from their commonly[25] used values of $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$. This leaves only the learning rate, η, as the important hyperparameter for the training algorithm. Common values for the learning rate range from 10^{-2} to 10^{-4}, but values outside this range are not unreasonable. Additionally, the learning rate may be epoch-dependent, $\eta(t)$, which is known as *learning rate scheduling* and is outside the scope of this work [100].

3.4.1 Automatic Differentiation

The update rules in gradient descent algorithms require the derivative of the loss function with respect to the trainable parameters of the model, which by the chain rule requires the gradient of the model itself. Luckily, instead of requiring that each implementation of a model comes with an implementation of its gradient, we can use *automatic differentiation* (AD) to automatically create this gradient implementation from the model's implementation.

[25]These values are common default values across many different libraries and publications, not just for PMMs. The default value of ε is slightly less consistent across literature, but present only for numerical stability and otherwise does not significantly change the behavior of the algorithm.

Another vast topic far beyond the scope of this work, AD has been a central tool in machine learning for several decades [121]. Here, we cover only a sketch of the core principle of AD. See Ref. [122] for a review of AD and Ref. [100] for a practical introduction to AD.

AD encapsulates a large family of techniques and is distinct from both symbolic differentiation and numerical differentiation. In contrast, AD is significantly more computationally efficient than symbolic differentiation and more accurate than numerical differentiation—in fact, it is exact to numerical precision. Perhaps most importantly for the use in gradient descent, it is computationally inexpensive even for functions with a large number of independent variables. In the broadest possible terms, AD exploits the chain rule of differentiation and the fact that all operations can be rewritten as function compositions. Regardless of the complexity of any function, numerically it must be written as a sequence of basic operations. So long as the derivatives of these basic operations are known, then the gradient of the entire function can be found via the chain rule. As an illustrative example, consider the function

$$f(x, y) = x \sin(y). \tag{3.22}$$

This is actually a composition of the multiplication function, $m(u, v) = uv$, with the sine function,

$$f(x, y) = m(x, \sin(y)). \tag{3.23}$$

Now that f is written purely as compositions of basic operations with known derivatives, we can apply the chain rule during the evaluation,

$$\begin{aligned} u &= x & \partial u &= \partial x \\ v &= \sin(y) & \partial v &= \cos(y)\partial y \\ m(u, v) &= uv & \partial m(u, v) &= u\partial v + (\partial u)v \\ \implies f(x, y) &= x \sin(y) & \partial f(x, y) &= x \cos(y)\partial y + (\partial x)\sin(y). \end{aligned} \tag{3.24}$$

The same basic principle allows libraries like JAX, PyTorch, and TensorFlow to automatically compute the gradients of loss functions for significantly more complex models than in this example [116–118].

3.4.1.1 Vanishing and Exploding Gradients

There are many difficulties in training deep and recurrent ANNs and perhaps the most well-known are the problems of the vanishing or exploding gradient [100, 123, 124]. These occur due to the long chain of multiplications in the derivative of the loss function which arise from the chain rule. If the gradients at each step are routinely less than 1, then the gradient approaches 0 exponentially, yielding no or very slow training progress. If the gradients at each step are routinely greater than 1, then the gradient diverges exponentially, yielding unstable training, overflows, and other numerical errors. These are problems which worsen with larger and deeper ANNs. However, most PMMs do not involve significant levels of nested function evaluations—since most physical problems do not. Even operations that are implemented via iterative methods often have non-iterative gradient implementations or implementations that otherwise avoid long chain rule applications. For instance, the k^{th} eigenvalue, E_k, of a Hermitian matrix, H, is computed iteratively—possibly with many iterations—which would lead to vanishing gradients if AD was performed through the iterative algorithm itself. Instead, AD treats the eigenvalue decomposition as a basic operation and implements its gradients directly. The Hellmann–Feynman theorem—first published in [125, Eq. (11)]—provides the gradient as

$$\frac{\partial E_k}{\partial z} = v_k^\dagger \left(\frac{\partial H}{\partial z} \right) v_k, \tag{3.25}$$

where z is some independent variable that H (and therefore E_k) depends on and v_k is the eigenvector of H associated with the k^{th} eigenvalue of H, E_k. Expressions for the gradients for the eigenvectors themselves also exist [114]. These kinds of gradient implementations, which are already present in major AD libraries [116–118], and the relatively shallow "depth" of most PMMs solve the problem of vanishing and exploding gradients when training PMMs by avoiding the long chain of gradient multiplications.

3.4.2 Parameter Initialization

In the field of deep learning, significant effort is spent on initializing the trainable parameters of models in such a way that the subsequent iterative training algorithm performs

as well as possible [119]. There are many different heuristics in this regard with varying levels of success and formalization. At the moment, none of this has been brought to PMMs. It is unclear what knowledge from ANN parameter initialization is applicable to PMMs in general. However, two trivial heuristics have proven successful.

- The magnitudes of the initial values for trainable parameters should be small. For a complex parameter $z = a + ib$, typical initialization scales for a and b are 10^{-1} or 10^{-2}.
- The parameter initialization must break any model symmetries. That is, if two independent parameters contribute equally to the model output, they must be initialized with distinct values. Otherwise, the training process will update these parameters identically and not break the symmetry automatically.

Currently, PMM initialization is usually done by drawing the real and imaginary components of all trainable parameters from independent and identical distributions—either a zero-mean normal distribution or a uniform distribution centered around zero.

3.4.3 Data Preprocessing

To vastly improve both the numerical precision, stability, and convergence of the training algorithm, both the features (inputs) and targets (expected outputs) of the training data are often rescaled. This is common practice in general machine learning [99, 100]. However, when using PMMs—or any RBM—to emulate physical systems, extra care must be taken to ensure that the link between the original equations of the high-fidelity model and the reduced equations of the emulator is not broken. That link is of course the linear projection which is implicit for PMMs but otherwise must remain linear. In practice, this means that any preprocessing applied to projected objects (e.g. vectors and operators) is restricted to linear rescaling. Due to scalar invariance under the assumed orthogonal projection for PMMs as discussed previously, both input and output scalars can be transformed in any consistent way without the linear restriction.

Common affine rescaling transformations include *standardization*, *min-max scaling*, and *robust scaling*. Standardization, or standard scaling, is a preprocessing method that shifts and scales the data so that each feature has mean 0 and standard deviation 1. For a set of training data T, the standard-scaled data would be [99, 126]

$$T_{\text{standard}} \equiv \left\{ \frac{t - \text{mean}(T)}{\text{std}(T)} \text{ for } t \in T \right\}. \tag{3.26}$$

Min-max scaling transforms the data so that the minimum and maximum of the rescaled data correspond to some chosen a and b with $a < b$,

$$T_{\text{min-max}} \equiv \left\{ a + \frac{t - \min(T)}{\max(T) - \min(T)}(b - a) \text{ for } t \in T \right\}. \tag{3.27}$$

Popular choices for $[a, b]$ are $[0, 1]$ and $[-1, 1]$ [99, 126]. Finally, robust scaling transforms the data such that the median of the transformed data is 0 and the interquartile range (the distance between the 75^{th} and 25^{th} percentiles of the data) is 1. This transformation is designed to be less sensitive to outliers.

$$T_{\text{robust}} \equiv \left\{ \frac{t - P_{50}(T)}{P_{75}(T) - P_{25}(T)} \text{ for } t \in T \right\}, \tag{3.28}$$

where $P_p(T)$ denotes the p^{th} percentile of T [126]. While these three transformations represent some of the most common for continuous variables, there are in principle an unlimited number of different useful preprocessing methods depending on the specific data and context. See Ref. [126] for a large compendium of different data preprocessing methods. In practice, the training of PMMs has performed best with min-max scaling on the range $[-1, 1]$ for both inputs and outputs—but this is far from a rule and just like parameter initialization warrants further investigation.

When rescaling the targets (output values) in the training data, the model will be trained to produce predictions in this rescaled regime. This necessitates inverse scaling to produce predictions that correspond to the original unscaled data. The three transformations described above are trivially invertible. However, a vital point which segues nicely into the

next section is that the parameters of the transformation—the values which arose from statistics on the training data such as mean(T), max(T), and $P_{50}(T)$—are always determined only by the original training data, even when rescaling predictions from the model for previously unseen data.

3.4.4 Generalization, Calibration, and Evaluation

We have thus far only discussed the training set, T. However, if a model is trained on all of the available data and performs perfectly on it, then how can we know it will generalize well to new, unseen data? A model that performs perfectly on the training data but poorly on new data is called an *overfit* model and is highly undesirable. Instead, we want to ensure that our emulators generalize well to unseen data. Similarly, how can we transform pointwise predictions to probabilistic ones, or ensure that already probabilistic predictions match the true probabilities? Finally, how can we accurately and honestly evaluate the performance of a trained model? An answer to all of these questions is data splitting—the process of dividing the whole of the available data into **disjoint** subsets. In general, there are four subsets that data can be split into.

1. Testing[26] Data, E

    - The subset of data which the model's performance is evaluated with to obtain unbiased performance metrics
    - Ideally, is representative of data the model is expected to encounter when deployed
    - Must be handled with the most care to ensure no information from any other subsets contaminates it or vice versa, and for that reason should be set aside first before any further splitting, preprocessing, training, or other work

[26] Also referred to as "test", "evaluation", "unseen", or "holdout" data.

2. Training Data, T

    - The only subset that can be used to determine any constants involved in data preprocessing and to fit the model's trainable parameters
    - Usually the largest subset

3. Validation Data, V

    - The subset used to evaluate the performance of the model during training and to make decisions about the training process such as when to stop training, which hyperparameters to use, or even which model architecture to use
    - Functions similarly to the test set, but only used for evaluating models during training and model selection and not for final performance metrics
    - Should be representative of the test set, and therefore of the data the model will encounter when deployed

4. Calibration Data, C

    - The subset used to calculate any stratified bias corrections or calibrate the model's probabilistic predictions, for example via conformal prediction [127, 128]
    - Least common, often entirely absent when only pointwise predictions are required
    - Should be statistically similar to the test set, and therefore to the data the model will encounter when deployed

We have already seen how the training set, T, is used. The validation set, V, is used as a proxy for the testing set, E, when designing and training an emulator. The relative performance on the validation set is used to select a specific architecture or hyperparameters as well as for stopping criteria during training. On the latter point, iterative training

algorithms will track the loss on the validation set,[27] called the validation loss, and stop iterating and return the parameters with the lowest validation loss when the validation loss fails to improve for a set number of iterations. By evaluating the model on a dataset disjoint from the training data and ideally representative of the test data, this process drastically improves the generalization performance of the trained model. The validation set is also used for more manual failure-case exploration, which assists in debugging and model improvement. The calibration set is usually only present when the model's predictions are probabilistic, when a pointwise-predictive model is to be calibrated to produce probabilistic outputs, or for stratified bias correction. Sometimes in machine learning, the validation set is used for these tasks, but this borders on data leakage and can quickly lead to overfitting and overly optimistic model evaluations. More advanced schemes like cross-validation—which are popular and powerful in data-scarce contexts—will treat the training, validation, and calibration datasets as a single large dataset from which to train multiple models (usually the same model multiple times) with different choices for the specific training, validation, and calibration sets from this larger set [99, 100]. The best performing model in this ensemble of models can often generalize much better than using a fixed split. Alternatively, the models can be combined using averaging or another statistical aggregation to produce a single model with strong generalization performance.

A complementary technique for improving generalization performance is *regularization* [119]. The simplest and most common form of regularization involves constraining the trainable parameters of the model during training—either by hard algorithmic constraints like using the same parameter for multiple parts of the model or by penalty terms in the loss function like adding the ℓ^2 norm of the vector of trainable parameters to the loss function. Currently, these techniques have not shown to improve the performance of any PMMs. When approaching PMMs as just machine learning models this may seem surprising, but

[27] It is not necessarily the case that the loss function here is the same as the one being minimized by the training process. This is particularly useful when the desired loss metric is not differentiable. In this case, a proxy loss function which is differentiable is minimized during training and the original non-differentiable loss function is tracked with the validation data. Many classification tasks follow this exact scheme.

when taking the iRBM approach this becomes almost obvious. A form of regularization for PMMs comes in the form of effective parameterizations that purposely conserve specific properties of the projected objects and operators, rather than of the vector of all trainable parameters. This is discussed further in Section 3.5.

The test set is used only once and only after all data processing, model fitting, and model calibration steps are done and locked in. Adjusting the model or data pipeline in any way based on results from the test set constitutes data leakage and will result in overly optimistic performance metrics at best [99, 100]. This last point is worth restating: at no point should any amount of information from or about the test data play any role in the design, fitting, calibration, or selection of any model.

3.5 Effective Parameterization

A vital consideration when building the reduced equations for a PMM which goes beyond simply following the eRBM procedure is choosing how the reduced objects will be parameterized by the trainable parameters. For instance, consider a single square operator A in a full space of dimension N and its size-n reduced-space representation, the $n \times n$ matrix $\underline{A}$. There are two extremes for the parameterization of $\underline{A}$, the first is to treat every element as a completely independent trainable complex value—yielding the maximum flexibility for $\underline{A}$ to adjust during the training process in order to accommodate the training data. The opposite extreme, which is not always available, is to determine all of the elements of $\underline{A}$ directly from the eRBM projection of A, that is $\underline{A} = \lfloor PAP \rceil$—leaving no trainable parameters whatsoever but conforming exactly to the eRBM prescription. Note that the first parameterization discards all properties of both A and the chosen implicit projector while the second preserves all properties of A that the corresponding eRBM would. Striking a balance between these two extremes is perhaps the only part of the PMM method where model "design" exists beyond simple hyperparameter tuning. It is here we must take care to preserve—or add—any important properties of the high-fidelity model without over-constraining our parameterizations and sacrificing the flexibility of our emulator. Intimate knowledge and familiarity with the

high-fidelity model is necessary here, as is knowing how to impose constraints in an exact and computationally efficient way.

Consider a trivial non-parametric example: the PMM for the lowest lying eigenvalue of a Hermitian matrix, H. Suppose we have chosen to base the PMM for this on the POD eRBM. We start by enumerating some of the properties of H that would be preserved under that projection: Hermiticity and definiteness as well as the consequences of the Poincaré separation theorem (see Section 2.11.1.1). There are many more, but we restrict our effort to properties that would meaningfully change the predictive ability of the emulator. Consider the consequences of *not* incorporating each property: would an emulator for a Hermitian problem that has the ability to produce complex eigenvalues be useful or sensible? Often not, and for this example we will say no, so we should incorporate Hermiticity into our parameterization. Suppose we do not know the definiteness of the high-fidelity H nor the value of any of the eigenvalues beyond the lowest. In that case we have no reason or ability to impose the other two listed constraints. With the properties—or in this example property—that we want to impose, we want to parameterize $\underline{H}$ such that they are exactly adhered to. One way to do this is simply labeling the $n(n-1)/2$ complex values of the strictly upper triangular portion of $\underline{H}$ along with the n real values of the diagonal of $\underline{H}$ as the trainable parameters. Identifying the trainable parameter vector as $\theta = [\theta_1, \theta_2, \ldots]$ with $\theta_i \in \mathbb{R}$ for $i \neq n$ and $\theta_i \in \mathbb{C}$ for $i > n$, we could then construct $\underline{H}$ from these parameters as

$$\underline{H} \equiv \begin{bmatrix} \theta_1 & \theta_{n+1} & \theta_{2n} & \theta_{3n-2} & \cdots \\ \theta_{n+1}^* & \theta_2 & \theta_{n+2} & \theta_{2n+1} & \ddots \\ \theta_{2n}^* & \theta_{n+2}^* & \theta_3 & \theta_{n+3} & \ddots \\ \theta_{3n-2}^* & \theta_{2n+1}^* & \theta_{n+3}^* & \theta_4 & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}, \tag{3.29}$$

which is guaranteed to be Hermitian for any values of the trainable parameters. This method requires the minimum number of trainable real values, but manually constructing $\underline{H}$ element-

by-element is cumbersome[28] and can be computationally inefficient. An alternative way that is less parameter efficient but significantly simpler and often more performant is to represent $\underline{H}$ as the result of some matrix-valued function of an unconstrained trainable complex matrix $\Theta \in \mathbb{C}^{n\times n}$. This function should be a surjective function from the space of all complex $n \times n$ matrices to the space of all $n \times n$ Hermitian matrices—that is, it should be defined for any $n \times n$ complex matrix input and be able to produce as output any $n \times n$ Hermitian matrix. For effective training with gradient-descent-type methods, it must also be differentiable and well-behaved. This function is simply $h(X) \equiv \left(X + X^\dagger\right)/2$. Then for $\Theta \in \mathbb{C}^{n\times n}$ being the trainable matrix, we construct $\underline{H}$ as

$$\underline{H} \equiv h(\Theta) \equiv \frac{\Theta + \Theta^\dagger}{2}. \tag{3.30}$$

Although this requires about twice as many trainable real values as the elementwise parameterization, the number of independent real values is exactly the same and thus the dimensionality of the parameter space which the training algorithm must explore is unchanged. Constructing the constrained reduced-space objects from closed-form expressions of unconstrained objects is often simpler, more understandable, and more performant. This example illustrates the core principles of effective parameterizations, but is by no means comprehensive. The following sections cover some of the most common constraints and ways to parameterize objects subject to them.

3.5.1 Hermiticity and Symmetry

As discussed in example above, an effective way to parameterize Hermitian or symmetric operators is to represent them by suitably shaped unconstrained trainable matrices and then "Hermitianize" or "symmetrize" them. For a Hermitian matrix $\underline{H} \in \mathbb{H}(n)$, we parameterize it by the unconstrained complex matrix $\Theta \in \mathbb{C}^{n\times n}$ as

$$\underline{H} \equiv h(\Theta) \equiv \frac{\Theta + \Theta^\dagger}{2}, \qquad \Theta \in \mathbb{C}^{n\times n}. \tag{3.31}$$

[28] This method especially complicates debugging and portability.

Similarly for a symmetric matrix $\underline{S}$, we parameterize it by the unconstrained real matrix $\Theta \in \mathbb{R}^{n\times n}$ as

$$\underline{S} \equiv s(\Theta) \equiv \frac{\Theta + \Theta^T}{2}, \qquad \Theta \in \mathbb{R}^{n\times n}. \tag{3.32}$$

Note that both of these parameterizations are idempotent—that is, $h(h(\Theta)) = h(\Theta) = \underline{H}$ and $s(s(\Theta)) = s(\Theta) = \underline{S}$. Anti-Hermitian and anti-symmetric matrices are treated similarly by simply subtracting the Hermitian conjugate or transpose instead of adding.

This concept generalizes to symmetries in arbitrary indices of higher-order tensors, and can be composed with additional Hermitianization or symmetrization expressions to handle multiple symmetries simultaneously. For example, consider an order-4 tensor $\underline{T}_{ijkl}$ which should be symmetric in the first two indices, symmetric in the last two indices, and Hermitian in $ij \leftrightarrow kl$. That is,

$$\begin{aligned} \underline{T}_{ijkl} &= \underline{T}_{jikl} \qquad &&\text{(symmetric in } i \leftrightarrow j\text{)}, \\ \underline{T}_{ijkl} &= \underline{T}_{ijlk} \qquad &&\text{(symmetric in } k \leftrightarrow l\text{)}, \\ \underline{T}_{ijkl} &= \underline{T}^*_{klij} \qquad &&\text{(Hermitian in } ij \leftrightarrow kl\text{)}. \end{aligned} \tag{3.33}$$

We can enforce these symmetries by applying them sequentially to an unconstrained trainable order-4 tensor,

$$\begin{aligned} \Theta &\in \mathbb{C}^{n\times n\times n\times n} \qquad &&\text{(unconstrained trainable tensor)}, \\ \underline{T}^{(1)}_{ijkl} &\equiv \frac{\Theta_{ijkl} + \Theta_{jikl}}{2} \qquad &&\text{(symmetric in } i \leftrightarrow j\text{)}, \\ \underline{T}^{(2)}_{ijkl} &\equiv \frac{\underline{T}^{(1)}_{ijkl} + \underline{T}^{(1)}_{ijlk}}{2} \qquad &&\text{(symmetric in } k \leftrightarrow l\text{)}, \\ \underline{T}_{ijkl} &\equiv \frac{\underline{T}^{(2)}_{ijkl} + \underline{T}^{(2)*}_{klij}}{2} \qquad &&\text{(Hermitian in } ij \leftrightarrow kl\text{)}. \end{aligned} \tag{3.34}$$

3.5.2 Unitarity and Matrix Orthogonality

One may be tempted to parameterize unitary or orthogonal matrices by the result of some Gram-Schmidt process or QR decomposition on an unconstrained matrix. However, the resulting matrix from this approach can have abrupt changes for relatively small changes in the unconstrained matrix—in other words, this approach would not be well-behaved in the

way necessary for gradient descent. In fact, the group of all orthogonal matrices is actually two disconnected sets (the set with determinant $+1$ and the set with determinant -1) and thus there is no well-behaved expression mapping unconstrained matrices to the entire set of orthogonal matrices. The solution to this problem is to leverage the knowledge of the high-fidelity model to restrict the matrix to one of the two types of orthogonal matrix first. Any real orthogonal matrix $\underline{R}$ with $\det(\underline{R}) = +1$ can be parameterized as

$$\underline{R} \equiv \exp(A), \qquad A \equiv \frac{\Theta - \Theta^T}{2}, \tag{3.35}$$

where $\exp(\cdot)$ is the matrix exponential and $\Theta \in \mathbb{R}^{n \times n}$ is the unconstrained trainable matrix as before. Similarly, a real orthogonal matrix $\underline{O}$ with $\det(\underline{O}) = -1$ can be parameterized as

$$\underline{O} \equiv \mathrm{diag}([-1, 1, 1, \ldots, 1]) \exp(A), \qquad A \equiv \frac{\Theta - \Theta^T}{2}, \tag{3.36}$$

where $\mathrm{diag}([-1, 1, 1, \ldots, 1])$ is the diagonal matrix with all $+1$ on the diagonal except for the first element, which is -1.

In contrast, unitary matrices are a continuous group that can be parameterized from an unconstrained complex matrix $\Theta \in \mathbb{C}^{n \times n}$ by

$$\underline{U} \equiv \exp(iH), \qquad H \equiv \frac{\Theta + \Theta^\dagger}{2}, \tag{3.37}$$

where i is the imaginary unit.[29]

For semi-unitary and semi-orthogonal matrices, one simply takes the leading few columns or rows of the respective parameterized unitary or orthogonal matrix.[30] This is often denoted as $X(:, 1:k)$ for the first k columns of X and $X(1:k, :)$ for the first k rows of X.

The parameterizations described here have been used to apply orthogonality constrains to the weight matrices of ANNs without any appreciable computational overhead [129].

[29] Of course, iH could be replaced by the parameterization of an anti-Hermitian matrix. However, this form is more familiar to those with a background in quantum mechanics.

[30] Choosing $\underline{R}$ from Eq. (3.35) as the base orthogonal matrix is sufficient to parameterize all possible semi-orthogonal matrices.

3.5.3 Vector Orthogonality and Normalization

Parameterizing vectors with a fixed norm can be done straightforwardly by directly normalizing an unconstrained vector $\theta \in \mathbb{C}^n$,

$$\underline{v} \equiv \mathcal{N} \frac{\theta}{\|\theta\|}, \tag{3.38}$$

where $\mathcal{N}$ is the desired value for the norm of the vector and $\|\cdot\|$ is any well-defined vector norm.

When parameterizing a set of orthonormal vectors, one follows the parameterization for parameterizing semi-unitary (for complex vectors) or semi-orthogonal (for real vectors) matrices in the previous section, taking the columns (or rows) of the parameterized semi-unitary/semi-orthogonal matrix as the parameterized orthonormal vectors. For non-normalized orthogonal vectors, one introduces an additional trainable real-valued vector for the norms of the vectors and uses these elements to rescale the parameterized orthonormal vectors.

3.5.4 Eigenvalue and Singular Value Constraints

The most general and straightforward way to impose constraints directly on the eigenvalues or singular values of an operator is to first parameterize the operator following its other constraints, such as Hermiticity,[31] while ignoring the eigenvalue and singular value constraints. Then by computing the corresponding decomposition of the operator, constraints can be applied directly to the eigenvalues or singular values before reconstructing the now-fully-constrained matrix. For example, a Hermitian, positive semi-definite matrix $\underline{H}^{(+)}$ would be parameterized from an unconstrained complex matrix $\Theta \in \mathbb{C}^{n \times n}$ by

$$\begin{aligned} H &\equiv \frac{\Theta + \Theta^\dagger}{2} && \text{(Hermitian matrix)}, \\ H &= V \mathrm{diag}(\lambda_1, \lambda_2, \ldots) V^H && \text{(eigenvalue decomposition)}, \\ \underline{H}^{(+)} &\equiv V \mathrm{diag}\big(\lambda_1^2, \lambda_2^2, \ldots\big) V^H && \text{(positive semi-definite matrix)}. \end{aligned} \tag{3.39}$$

[31] Note that AD for non-Hermitian eigenvalue decomposition is not implemented in all AD libraries. At the time of writing, support was only recently added to JAX in version v0.10.1.

This parameterization of $\underline{H}^{(+)}$ has the same eigenvectors as H, but with non-negative eigenvalues.[32] Note that an equivalent, but more performant, parameterization of a Hermitian positive semi-definite matrix would be

$$\begin{aligned} H &\equiv \frac{\Theta + \Theta^\dagger}{2} \qquad \text{(Hermitian matrix)}, \\ \underline{H}^{(+)} &\equiv HH^\dagger \qquad \text{(positive semi-definite matrix)}. \end{aligned} \tag{3.40}$$

A powerful and important singular value constraint is limiting the condition number of an operator, the ratio of the maximum and minimum singular values $\kappa = \sigma_{\max}/\sigma_{\min}$. As discussed in Section 2.11.1.1, the Poincaré separation theorem guarantees that POD-projected operators must have a condition number no greater than that of their full-space counterparts. Imposing an inequality constraint is less straightforward than the previous constraints, and a common tool for this—and a staple in machine learning—is the *logistic* or *sigmoid* function defined by

$$s(x) \equiv \frac{1}{1 + e^{-x}}. \tag{3.41}$$

Defined on the domain $x \in (-\infty, \infty)$, this function is monotonically increasing and smoothly varies from 0 to 1, asymptotically approaching each value at the respective infinity. We can use this to limit the ratio of minimum and maximum values of any list of numbers, such as the singular values of an operator. Consider the parameterized operator X which we wish to constrain such that the constrained operator, $\underline{X}$, satisfies $\kappa(\underline{X}) \leq \kappa'$ for some chosen κ'. We start by taking the singular value decomposition of X,

$$X = U\text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r, 0, \ldots, 0)V^\dagger \tag{3.42}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_r > 0$ are the singular values of X and r is the rank of X. The condition number of X is σ_1/σ_r. Notice that normalizing the singular values by the smallest (nonzero) singular value, σ_r, does not change the condition number and ensures that all the singular values are greater than or equal to 1. By applying a shifted and rescaled logistic

[32] For positive definite matrices (strictly positive eigenvalues), one would simply add the smallest positive machine-representable number to each of the squared eigenvalues. For single-precision floats this is 2^{-149} and for double-precision floats this is 2^{-1074}.

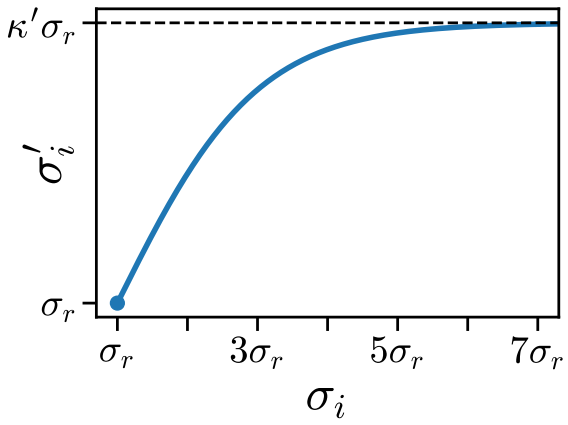

Figure 3.3 Illustration of the logistic function in Eq. (3.43) applied to the problem of constraining the condition number of a parameterized operator. The horizontal axis shows the unconstrained value of an arbitrary singular value σ_i while the vertical axis shows the resulting singular value σ_i'. The ticks on both axes show the scale relative to the smallest nonzero singular value, σ_r. This function ensures that the new largest singular value, σ_1' over the new smallest nonzero singular value, which is unchanged from the original σ_r, is less than or equal to some desired maximum condition number κ'.

function to each of the normalized singular values then un-normalizing them, the minimum singular value remains unchanged while large singular values are smoothly reduced in order to keep the condition number less than or equal to κ',

$$\sigma_i' \equiv \sigma_r \left[2(\kappa' - 1) s\left(\frac{\sigma_i}{\sigma_r} - 1 \right) - \kappa' + 2 \right]. \tag{3.43}$$

This is shown in Fig. 3.3. These rescaled singular values can then be taken to be the singular values of $\underline{X}$ with the same left and right singular vectors as X,

$$\underline{X} \equiv U \mathrm{diag}(\sigma_1', \sigma_2', \ldots, \sigma_r', 0, \ldots, 0) V^\dagger. \tag{3.44}$$

3.5.5 Rank

One way to enforce a particular rank for a parameterized operator would be to follow the method of constraining singular values and zero out all singular values beyond the desired rank. However, there is a far more parameter and computationally efficient method based on rank-1 matrices.

For a general rank-r operator of shape $n \times m$, $\underline{X}$, we parameterize it by two sets of r

unconstrained trainable complex (or real) vectors $\theta_i \in \mathbb{C}^n$ and $\phi_i \in \mathbb{C}^m$ for $i = 1, 2, \ldots, r$,

$$\underline{X} \equiv \sum_i^r \theta_i \phi_i^\dagger. \tag{3.45}$$

That is, the sum of the outer products of each pair of θ_i, ϕ_i. For an $n \times n$ Hermitian (or symmetric) rank-r operator, $\underline{Y}$, we need only a single set of r unconstrained trainable complex (or real) vectors $\theta_i \in \mathbb{C}^n$ and r unconstrained trainable real values $\varphi \in \mathbb{R}^r$,

$$\underline{Y} \equiv \sum_i^r \varphi_i \theta_i \theta_i^\dagger. \tag{3.46}$$

Anti-Hermitian rank-deficient operators are handled similarly, where each term in the sum is multiplied by i. Antisymmetric rank-deficient operators are a special case, since all antisymmetric real matrices must have even rank. Thus, rank-deficient real antisymmetric matrices, $\underline{Z}$, must be parameterized as the sum of rank-2 matrices,

$$\underline{Z} \equiv \sum_i^{r/2} \theta_i \phi_i^T - \phi_i \theta_i^T, \qquad \text{for } r \text{ even.} \tag{3.47}$$

3.5.6 Tensor Hypernetworks

While not a common object in high-fidelity models directly, tensor hypernetworks[33] appear in the reduced-space representation of nearly all high-fidelity models with nonlinear terms. See Sections 2.11.1.1 and 3.3.2 for discussions of nonlinear terms RBMs and PMMs respectively. The resulting tensor for nonlinear operations can be prohibitively large—the smallest and simplest, $\underline{G}^{(2)}$, which encodes the elementwise product of two vectors in the full space, is an order-3 tensor which would require n^3 trainable parameters if parameterized naively. Instead however, we can use the expression for these tensors which arises from the eRBM formulation of the problem. For $\underline{G}^{(2)}$ this is

$$\underline{G}^{(2)}_{ijk} \equiv \left(P^\dagger\right)_{i\mu} P_{\mu j} P_{\mu k}, \tag{3.48}$$

[33]Tensor hypernetworks are expressions of tensor contractions where any contracted index appears more than twice. For an example using standard Einstein summation notation, $A_{ij}B_{jk}C_{klm}D_m$ is a standard tensor network since each index appears at most twice, but $A_{ij}B_{jk}C_{jl}D_j$ and $A_{jj}B_{jj}C_{jj}D_j$ are tensor hypernetworks since the index j appears three or more times.

using Einstein summation notation, where $P \in \mathbb{U}(N, n)$ is the $N \times n$ unitary POD projector from the N-dimensional full-space to the n-dimensional reduced space. We can approximate this tensor in the reduced space by a method similar to "trimming" in tensor networks [130]. We replace each P in the expression with an unconstrained trainable $m \times n$ complex (or real for purely real-valued data) matrix $\Theta \in \mathbb{C}^{m \times n}$ where $n \leq m \ll N$ and take the approximation of $\underline{G}^{(2)}$ to be

$$\underline{G}^{(2)}_{ijk} \approx \left(\Theta^\dagger\right)_{i\mu} \Theta_{\mu j} \Theta_{\mu k}, \tag{3.49}$$

which requires only nm trainable complex parameters. Higher-order tensors for nonlinearities in the reduced space follow similar forms and can be efficiently approximated in the same manner. See Section 3.10 for a worked example.

3.6 Potential Pitfalls

There are several mistakes one can make when parameterizing objects with constraints or interpreting the post-training values of parameterized objects. The most common is to ascribe some meaning to the individual elements of the reduced-space objects. A usual manifestation of this is assuming that a diagonal operator in the full space will also be a diagonal operator in the reduced space, which is not true in general.

$$D = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_N \end{bmatrix}_{N \times N} \quad \not\longrightarrow \quad \underline{D} = \begin{bmatrix} \underline{d}_1 & & \\ & \ddots & \\ & & \underline{d}_n \end{bmatrix}_{n \times n} \tag{3.50}$$

Instead, these kinds of operators should be parameterized as general symmetric operators—potentially with definiteness constraints.

The core issue is over-constraining parameterized objects. Unless the constraints represent real physical properties or other properties guaranteed by the full-space model, parameterizations of objects in the reduced space should be as general as possible. An example of over-constraining the parameterized objects which has appeared many times comes in the form of breaking the unitary freedom of eigenvalue problems. Recall from Section 3.1 that PMMs and iRBMs rely on symmetries in both the full and reduced spaces for their flexibility.

In a parametric affine eigenproblem, $H(c) = H_0 + \sum_{i>0} c_i H_i$, it is true that the eigenvalues are invariant under any unitary transformation and thus one could choose to always work in the basis where H_0 is diagonal. However, restricting H_0 or its reduced-space representation, $\underline{H}_0$, to be diagonal unnecessarily constrains not only $\underline{H}_0$, but all the $\underline{H}_i$, and can severely hinder training performance. Closely related is the mistake of replacing complex Hermitian operators with real symmetric ones. This most often appears in data-driven PMMs, discussed in Sections 3.2.2 and 3.8. The consequences of these unnecessary constraints often become more detrimental for more complex problems and emulators, potentially preventing successful emulation altogether.

One may be tempted to impose constraints on the reduced-space objects by adding penalty terms to the loss function, as is common practice in machine learning. Except for in very limited applications,[34] this is not recommended for physical constraints or properties which usually can be more efficiently and accurately imposed via the methods discussed in Section 3.5.

When constructing an emulator[35] for vector-valued predictions, the projector must represent a suitable subspace for all target objects. For example, if the emulator is to produce predictions for some vector-valued functions $f(\cdot)$ and $g(\cdot)$, the subspace that the projector represents must approximately span snapshots of both $f(\cdot)$ and $g(\cdot)$. In this example, if the subspace only approximately spanned snapshots of $f(\cdot)$ then there would be no guarantee or expectation that the emulator would be able to produce reasonable predictions for $g(\cdot)$. Similarly, if the projector came from snapshots of an entirely distinct third function $h(\cdot)$, then in general the emulator would not produce accurate predictions for either $f(\cdot)$ or $g(\cdot)$. This is due to the nature of the linear transformation that the projector represents: the predictions from the emulator can be only linear combinations of the snapshots used to construct the projector. See Section 3.11 for an example of the consequences of this and how PMMs can adapt when the ideal snapshots are not available.

[34] Such as penalizing, but not constraining, the condition number of an operator.

[35] This pitfall applies to all RBMs, not just the PMM method.

Finally, one of the most challenging parts of effective PMM emulation is deriving the form of the associated eRBM when nonlinear terms are present in the high-fidelity model. Whether following the POD method of handling polynomially nonlinear terms (see Section 2.11.1.1) or the matrix-valued-function method described in Section 3.3.2, it can help to replace the nonlinear operations with a black-box function F—which operates only on objects in the full space and returns an object in the full space—and perform as much of the eRBM derivation first before substituting the known form for F and switching to Einstein summation notation. For example, there are three common nonlinear function *signatures*: vector-input vector-output, vector-input matrix-output, and matrix-input matrix-output. Denoting these by $F^{(\mathrm{vv})}$, $F^{(\mathrm{vM})}$, and $F^{(\mathrm{MM})}$ respectively and the full- and reduced-space vectors and matrices as v, $\underline{v}$, M, and $\underline{M}$, the eRBM derivation for each of the three nonlinearities would proceed as

$$\begin{aligned} F^{(\mathrm{vv})}(v) &\to P^\dagger F^{(\mathrm{vv})}(P\underline{v}), \\ F^{(\mathrm{vM})}(v) &\to P^\dagger F^{(\mathrm{vM})}(P\underline{v})P, \\ F^{(\mathrm{MM})}(M) &\to P^\dagger F^{(\mathrm{MM})}\big(P\underline{M}P^\dagger\big)P. \end{aligned} \tag{3.51}$$

As we have seen, the form of the PMM is not merely a guess or ansatz based on the high-fidelity model form, nor is it just the same form as the high-fidelity equations. It is a concrete result of the careful derivation of an associated eRBM.

3.7 Affine Eigenproblems

The original context in which PMMs were developed, affine eigenproblems are problems which seek the eigenvalues and some sesquilinear forms[36] of the associated eigenvectors of a (usually Hermitian) matrix which is affine in some control parameters. That is, for control

[36]The sesquilinear form represented by a matrix A on two complex vectors u and v is $v^\dagger Au$ [131]. This is sometimes confused with bilinear forms, which are represented as $v^T Au$—note the lack of complex conjugation on v. We can see the distinction between bilinear forms and sesquilinear forms if we take $f(u,v) \equiv v^\dagger Au$ and a to be some complex scalar, then $f(au,v) = af(u,v)$ and $f(u,av) = a^* f(u,v)$. That is, a sesquilinear form f is linear in the first argument and anti-linear in the second—compared to a bilinear form which is linear in both arguments. The word "sesquilinear" means one and one half times linear, just as "bilinear" means twice linear. For completeness, Hermitian forms are represented as $u^\dagger Au$ and quadratic forms by $u^T Au$.

parameters $c \equiv [c_1, c_2, \ldots]$ the matrix is of the form

$$H(c) = H_0 + \sum_{i>0} c_i H_i \tag{3.52}$$

and the quantities of interest are a subset of the eigenvalues $\{E_j\}$ (usually a few of the lowest lying) and the values of some set of particular sesquilinear forms $v_a^\dagger O_k v_b$ where v_j is the j^{th} eigenvector of $H(c)$ and O_k are matrices.

The PMM for these problems has a form identical to that of the original problem. The emulated eigenvalues are the eigenvalues of the reduced-space parametric matrix

$$\underline{H(c)} = \underline{H_0} + \sum_{i>0} c_i \underline{H_i} \tag{3.53}$$

and the emulated sesquilinear forms are the values of the sesquilinear forms of the eigenvectors of $\underline{H(c)}$ with the reduced-space matrices $\underline{O_k}$.

Example

Consider the same example problem as in Section 2.11.1.4: a Hamiltonian describing a 1D spin chain of $L = 14$ spins with couplings J_{ij} in the influence of a transverse field with strength B,

$$H = -\frac{1}{\mathcal{J}} \sum_{i<j}^{L} J_{ij} \left(\gamma_x \sigma_i^x \sigma_j^x + \gamma_y \sigma_i^y \sigma_j^y \right) - B \sum_{i}^{L} \sigma_i^z, \tag{3.54}$$

where $\gamma_{x/y}$ are the relative strengths of the x and y couplings, σ_i^u is the Pauli spin operator acting on the i^{th} spin in the u axis, and $\mathcal{J}$ is the Kac normalization factor

$$\mathcal{J} \equiv \frac{1}{L-1} \sum_{i \neq j}^{L} J_{ij}. \tag{3.55}$$

This Hamiltonian is detailed further in Appendix B. We are interested in emulating the ground state eigenvalue as a function of B as well as the expectation value of S_x^2/L in the ground state, where

$$S_x^2/L \equiv \frac{1}{L} \sum_{i,j}^{L} \sigma_i^x \sigma_j^x. \tag{3.56}$$

For this example we take the couplings to be a power-law decay with scale $J = 1$ and exponent $p = 0.9$, so $J_{ij} = J/|i-j|^p$, with $\gamma_x = 1$ and $\gamma_y = 0$. We now follow the steps to

emulate this problem with a PMM as described in Section 3.1. For step 0, we note that the form of the problem in terms of the varying parameter B is simply

$$\begin{aligned} H(B) &= H_0 + BH_1, \\ H(B)\Psi_0(B) &= E_0(B)\Psi_0(B), \\ \left\langle S_x^2/L \right\rangle(B) &= \frac{1}{L}\Psi_0(B)^\dagger S_x^2 \Psi_0(B), \end{aligned} \tag{3.57}$$

and we suppose we have a known set of B values $\{B_i\}$ and data for the ground state of energy of Eq. (3.54) at those B values, $\{E_0(B_i)\}$, as well as data for the expectation value of S_x^2/L in the ground state for the same set of B values, $\{\langle S_x^2/L\rangle(B_i)\}$. Note that we do not require access to any of the operators in the full problem. For step 1 we choose a reduced-space size of $n = 5$ and an orthogonal implicit projection. In step 2 we note that the projected problem will be

$$\begin{aligned} \underline{H(B)} &= \underline{H_0} + B\underline{H_1}, \\ \underline{H(B)}\,\underline{\Psi_0(B)} &= \underline{E_0(B)}\,\underline{\Psi_0(B)}, \\ \underline{\left\langle S_x^2/L \right\rangle(B)} &= \underline{\Psi_0(B)}^\dagger \underline{\left(S_x^2/L\right)}\,\underline{\Psi_0(B)}, \end{aligned} \tag{3.58}$$

where we have absorbed the factor of $1/L$ into the operator for the expectation value. Due to the properties of orthogonal projections discussed before, we know $E_0(B) \approx \underline{E_0(B)}$, $\langle S_x^2/L\rangle(B) \approx \underline{\langle S_x^2/L\rangle(B)}$, and that $\underline{H_0}$ and $\underline{H_1}$ must be Hermitian and $\underline{S_x^2/L}$ must be positive semi-definite since H_0 and H_1, are Hermitian and S_x^2/L is positive semi-definite. We parameterize these reduced-space operators in step 3 by representing them as unknown general square complex matrices X, Y, $Z \in \mathbb{C}^{n\times n}$ and enforcing Hermiticity and positive semi-definiteness via

$$\begin{aligned} \underline{H_0} &= \left(X + X^\dagger\right)/2 \\ \underline{H_1} &= \left(Y + Y^\dagger\right)/2 \\ \underline{S_x^2/L} &= Z^\dagger Z, \end{aligned} \tag{3.59}$$

which allows us to perform unconstrained optimization over X, Y, and Z. We identify the mean squared error between the training data and the PMM predictions as the loss function

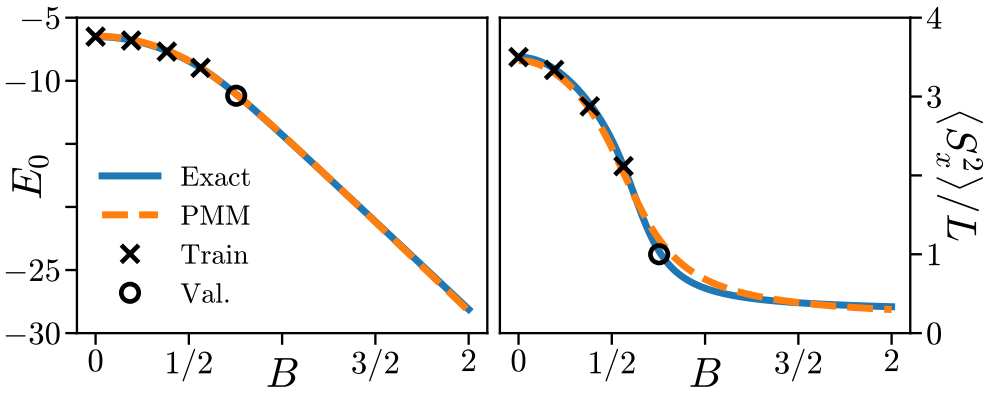

Figure 3.4 Exact and PMM-predicted results for the ground state energy (left) and expectation value of S_x^2/L in the ground state (right) for the Hamiltonian detailed in Appendix B as a function of B. All other Hamiltonian parameters are fixed $\{L, \gamma_x, \gamma_y\} = \{14, 0, 1\}$ with a power-law coupling defined by $J = 0$ and $p = 0.9$. The locations of the training and validation values are shown as crosses and open circles respectively.

for step 4,

$$\mathcal{L}(X, Y, Z) = \frac{1}{m} \sum_{i}^{m} \left[\left(E_0(B_i) - \underline{E_0(B_i)} \right)^2 + \left(\langle S_x^2/L \rangle (B_i) - \underline{\langle S_x^2/L \rangle (B_i)} \right)^2 \right], \tag{3.60}$$

where m is the number of training examples. We use complex-valued gradient descent to find a suitable approximation to the location of the minimum of this loss function in step 5. We can then use the optimized values for X, Y, and Z to compute the reduced-space operators and make predictions with this trained PMM. Figure 3.4 shows the results of this PMM, demonstrating strong interpolation and extrapolation performance. The predictions are less accurate than the EC emulator for this same problem (shown in Section 2.11.1.4), but in contrast the PMM emulator requires significantly less information, about 0.01% as much as for EC: it required only 15 real values ($\{B_i, E_0(B_i), \langle S_x^2/L \rangle (B_i)\}$) compared to $2 \times 2^{15} + 5 \times 2^{14} + 5 = 147\,461$ complex values ($\{H_0, H_1, V_T, B_i\}$).

3.7.1 Smoothing via Level Repulsion

A particularly useful demonstration of the PMM method's ability to intentionally deviate from the eRBM involves the re-parameterization of the bias matrix ($\underline{H_0}$) in an affine Hermitian eigenproblem to encourage smooth eigenvalue and eigenvector functions. Depending on the problem context, this re-parameterization can instead be interpreted as encoding more

information from the full-dimensional model rather than a deviation from it. As a function of the control parameters c, consider the eigenvalues $\lambda_k(c)$ and arbitrary Hermitian forms[37] of the associated eigenvectors $x_k \equiv v_k(c)^\dagger X v_k(c)$ of Eq. (3.52) where X is any arbitrary Hermitian matrix of compatible shape. As a function of c, $\lambda_k(c)$ for a fixed k will exhibit sharp behavior (large second derivative magnitude) when the eigenvalues of $H(c)$ are degenerate or nearly degenerate. This behavior is potentially more pronounced for any Hermitian forms of the associated eigenvector. These are known as "avoided crossings" and have great significance in quantum mechanics [132]. The sharpness of these avoided crossings are determined by the coupling of the associated eigenvector to adjacent levels. Denoting the location of the avoided crossing by $c = c'$, then the magnitude of

$$v_k(c')^\dagger H(c') v_{k'}(c') \quad \text{and} \quad v_{k'}(c')^\dagger H(c') v_k(c') \tag{3.61}$$

determines the sharpness of the avoided crossing, where $k' = k \pm 1$, with smaller magnitudes indicating sharper behavior. When c represents only a single control parameter, then only when the magnitude of both parts of Eq. (3.61) for a fixed k' is 0 can the eigenvalue be degenerate—that is $\lambda_k(c') = \lambda_{k'}(c')$. In quantum mechanics, this corresponds to the respective eigenstates having different symmetries.

There are various reasons why one might want to deliberately encourage more smooth behavior near an avoided crossing in a PMM for the affine eigenproblem. From a machine learning perspective, encouraging smoother output is akin to regularization and can help prevent overfitting. The straightforward and physically motivated way to do this is with the commutator between each of the H_i. A true non-avoided crossing (the sharpest case) occurs only when a pair of H_i commute, and the sharpness of the avoided crossing decreases as the magnitude[38] of the commutator increases. To promote smoothness then, we want to control the magnitude of all the commutators between pairs of H_i. For a given pair of operators H_k

[37] This smoothing method equally applies to quadratic $(v_k(c)^T X v_k(c))$, sesquilinear $(v_k(c)^\dagger X v_{k'}(c))$, and bilinear $(v_k(c)^T X v_{k'}(c))$ forms as well.

[38] Here we use "magnitude" of an operator to mean any consistent norm.

and H_l with $k \neq l$ then, we add

$$i[H_k, H_l] = i(H_k H_l - H_l H_k) \tag{3.62}$$

to the affine Hermitian form of $H(c)$. The imaginary unit appears since the commutator of two Hermitian operators is anti-Hermitian. We can scale this term by a tunable hyperparameter s to give us finer control over the smoothing, where larger values of s correspond with more aggressive encouragement of smooth eigenpair behavior. This term can be interpreted as increasing the "level repulsion" between the eigenvalue levels k and l by deliberately introducing a gap (energy difference). We can efficiently compute this term for all pairs of H_i by exploiting the linearity of the commutator, giving a revised form for the PMM,

$$\begin{aligned} \underline{H(c)} &= \underline{H_0} + sC + \sum_{i>0} c_i \underline{H_i} \\ C &\equiv i \sum_{k>0} \left[\underline{H_k}, \sum_{l<k} \underline{H_l} \right]. \end{aligned} \tag{3.63}$$

Some important remarks about this revised PMM form,

- The additional smoothing term is independent of the control parameters c, and thus can be thought of purely as a modification to the constant $\underline{H_0}$ matrix.
- All operations are still linear and hence the form of the PMM and the supposed full-space model are still equivalent.
- Regardless of the value of s, this term does not guarantee smoothness. If any pair of $\underline{H_k}$ and $\underline{H_l}$ commute, then the associated contribution to this term will be 0. This does not represent a shortcoming but instead a desirable property of this approach. We have not restricted the space of eigenproblems the PMM can represent. Instead, we have lessened the proportion of those representable eigenproblems which exhibit non-avoided crossings or sharp avoided crossings. This term's effects are felt during the training phase of the emulator, as it makes it much more difficult for the smooth training process to converge to a solution with sharp behavior. It is still possible and will happen if the specific training data cannot be fit any other way.

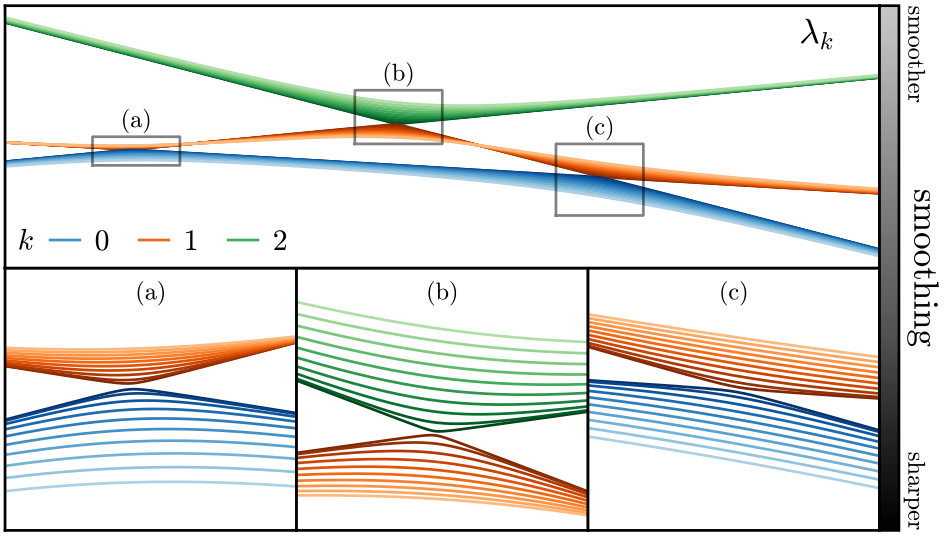

Figure 3.5 Demonstration of the smoothing term in Eq. (3.63) on the eigenvalues of a parametric affine Hermitian eigenproblem. The top panel (λ_k) shows the three eigenvalues as a function of the control parameter for values of the smoothing parameter starting from no smoothing (darkest) to strong smoothing (lightest). The insets provide a zoomed-in view of each avoided crossing.

Figures 3.5 and 3.6 illustrate the sharp behavior that can occur in a parametric affine eigensystem and demonstrate the effect this smoothing term has on the eigensystem for increasing values of the smoothing parameter s. The figures use a one-parameter 3×3 parametric affine Hermitian eigenproblem with random operators where H_0 and H_1 nearly commute. Figure 3.5 shows the behavior of the three eigenvalues as a function of c and s. The insets in Fig. 3.5 make clear how sharp the level repulsion can be when no smoothing is present. Figure 3.6 shows the value of a Hermitian form of the eigenvectors associated with a Hermitian matrix X, equivalent to the expectation value of X in each of the three eigenstates. We see that despite having no knowledge of X the additional term in Eq. (3.63) is able to smooth not only the eigenvalues in Fig. 3.5 but also the Hermitian form defined by X in Fig. 3.6. Both figures additionally show the fine-grain and continuous control the smoothing parameter s has over the induced smoothing.

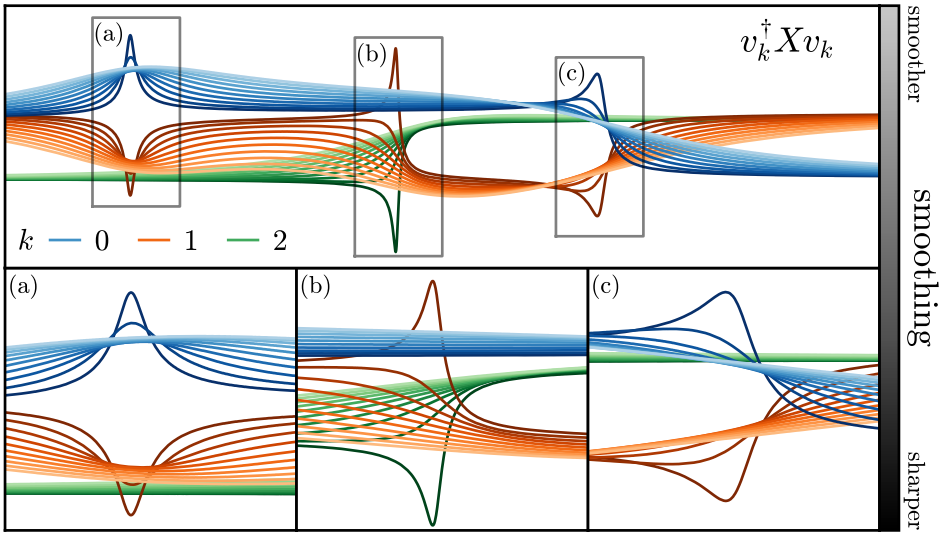

Figure 3.6 Demonstration of the smoothing term in Eq. (3.63) on a Hermitian form of the eigenvectors of the same parametric affine Hermitian eigenproblem as in Fig. 3.5. The top panel ($v_k^\dagger X v_k$) shows the value of a Hermitian form for each of the eigenvectors v_k as a function of the control parameter for values of the smoothing parameter starting from no smoothing (darkest) to strong smoothing (lightest). The insets provide a zoomed-in view of the location of each avoided crossing.

3.8 Data-Driven Regression

Despite its foundations as an RBM, the PMM method can be used as a purely data-driven machine learning model simply by choosing a suitable form. In Ref. [52] it is shown that the eigenvalues of a parametric affine Hermitian matrix can approximate any other function. However, more efficient forms involving sesquilinear forms of the eigenvectors of the same parametric matrix are used in practice. Here we cover the most recent architecture used for data-driven PMMs which is a refinement of the forms used in Ref. [52].

To form a data-driven model for p real-valued inputs $c \in \mathbb{R}^p$ that produces q real-valued outputs $z \in \mathbb{R}^q$, we start by choosing the PMM dimension n and form the parametric affine Hermitian matrix $\underline{H(c)}$ with an optional smoothing term controlled by the smoothing parameter s as in Eq. (3.63). Denote the eigenvalues and associated eigenvectors of this matrix by $\lambda_k(c)$ and $v_k(c)$. We choose $r < n$ of these eigenvectors and for each output compute the sum of the squared magnitudes of a series of sesquilinear forms between every pair of eigenvectors defined by l Hermitian matrices. Each output has an independent set of

l Hermitian matrices, so we have

$$y_j = \sum_{\omega=1}^{l} \sum_{\mu,\nu=1}^{r} \left| v_\mu^\dagger \underline{D_{j\omega}} v_\nu \right|^2, \text{ for } j = 1, \ldots, q, \tag{3.64}$$

where $\underline{D_{j\omega}}$ is the Hermitian matrix defining the ω^{th} sesquilinear form for the j^{th} output. Altogether, there are $q \times l$ independent sesquilinear forms and $q \times l \times r^2$ terms in the sum. The specific subset of r eigenvectors is usually the r eigenvectors with algebraically smallest or largest associated eigenvalue. Note that choosing the subset based on the magnitude of the associated eigenvalue can result in extremely sharp and unstable output since the order of the magnitudes of the eigenvalues is not stable as a function of c and smoothing via level repulsion does not alleviate this. The intermediate result of y_j in Eq. (3.64) is a non-negative real number. To allow for negative outputs, we introduce a centering term and trainable biases to give the final form of the data-driven PMM,

$$z_j = \underline{b_j} + y_j - \frac{1}{2} \sum_{\omega=1}^{l} \left\| \underline{D_{j\omega}} \right\|_2^2, \text{ for } j = 1, \ldots, q, \tag{3.65}$$

where $\underline{b_j} \in \mathbb{R}$ is the trainable bias for the j^{th} output and $\left\| \underline{D_{j\omega}} \right\|_2^2$ is the squared spectral norm of the matrix $\underline{D_{j\omega}}$. This centering term is roughly motivated by the average contribution of each sesquilinear form to the output[39] and has been found to improve training performance.

The number of trainable real values for this form of data-driven PMM is

$$\underbrace{(p+1)n^2}_{\underline{H_0}, \underline{H_i}} + \underbrace{qln^2}_{\underline{D_{j\omega}}} + \underbrace{q}_{\underline{b_j}}, \tag{3.66}$$

and inference can be performed with a time complexity of $\mathcal{O}(qr^2n^2)$ when using an optimized partial eigensolver to compute the r eigenpairs of $\underline{H(c)}$ or $\mathcal{O}(qr^2n^2 + n^3)$ when using a full eigensolver. In Ref. [52] a less efficient variant of this form is shown to perform well on a variety of regression tasks and consistently outperforms existing standard machine

[39] The actual mean value of the terms in Eq. (3.64) is not known since the distribution of the eigenvectors is difficult to characterize for a generic parametric affine Hermitian matrix. This centering term is a partially heuristic choice.

learning regression models, including feedforward ANNs, random forests, and support vector regression.

Data-driven PMMs in emulation are particularly useful for identifying hidden features that become the control parameters for physically-grounded PMM emulators. This application is central to solidifying the method as an adaptive emulation method and enabling the emulation of significantly more complex systems. Section 3.12.1 provides an example of this application in the context of the nuclear equation of state.

3.8.1 Data-Driven Classification

A model for classification can be formed by taking the data-driven regression form of Eq. (3.65) and applying the *softmax* function, defined as

$$\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{q} e^{z_k}}, \text{ for } j = 1, \ldots, q, \tag{3.67}$$

to the output to give a probability distribution over the classes. The resulting model is a data-driven PMM classifier. The number of trainable parameters and inference time complexity are the same as for the regression form, with the additional cost of the softmax function, which has a time complexity of $\mathcal{O}(q)$ [119]. The loss functions used for training, validation, and testing of classification models are much different than those for regression and involve applications of information theory—see Refs. [99, 100] for and introduction to training classification models. Reference [52] contains additional details on the architecture modifications necessary to accept images as input and demonstrates strong classification performance on a variety of standard small image classification datasets.

3.9 Differential Equations and Chaotic Systems

Emulating systems described by differential equations (usually in time) with PMMs can be done in one of two complimentary ways. The first is by direct fitting of samples of the state vector and the second is by fitting to the derivative vector(s). To make the distinction clear, consider a system described by the state vector v and an arbitrary explicit first-order

ordinary differential equation in time,

$$\frac{dv}{dt} = F(t, v), \tag{3.68}$$

with initial state v_0. Given snapshots of the state vector $\{t_i, v(t_i)\}$, in the first approach the PMM encapsulates the entirety of Eq. (3.68) and the initial state and produces as an output the emulated state vector at time t, $\tilde{v}(t)$. In this case, computing the loss for a single snapshot $\mathcal{L}(v(t), \tilde{v}(t))$ requires integrating the PMM in time. Training the PMM in this approach can become both inefficient due to the full integration for each snapshot and difficult since snapshots at large t provide very little meaningful information for gradient-descent based algorithms. This is akin to the problem of vanishing gradients in deep learning. The alternative PMM approach avoids this issue by training directly on snapshots for the derivative of the state vector, usually approximated from finite differences of the state vector snapshots. This way, the PMM aims only to implement a single evaluation of the right hand side of Eq. (3.68)—that is, the emulated derivative of the state vector, $\widetilde{\frac{dv}{dt}}(t)$. In this case, the loss for a single sample $\mathcal{L}\left(\frac{dv}{dt}(t), \widetilde{\frac{dv}{dt}}(t)\right)$ is efficient to compute and contributes equally for all t during training. These two approaches are complimentary in that the second can be used to train a PMM which, now producing reasonable predictions at later t, can then be fine-tuned by training with the first approach.

Chaotic systems best demonstrate the need for the second approach. In a chaotic system, the approximate present does not determine the approximate future, which means that gradient descent is ineffective for training a PMM which integrates over time.

3.10 Nonlinear Eigenproblems

Density functional theory (DFT) is a powerful method for computing various properties of nuclear systems and fundamentally relies on a nonlinear eigenproblem [133–135]. See Refs. [133–135] for detailed reviews of DFT in nuclear physics, which will not be covered here.[40] The central nonlinear operation in DFT comes in the form of a density-dependent

[40]This section will greatly over-simplify some points of DFT, but the core elements relevant to emulation in the context of RBMs and specifically PMMs will remain intact.

interaction,[41] of which the lowest few eigenstates[42] and eigenenergies are sought. The density that this Hamiltonian depends on is the sum of the individual eigenstate densities, $\rho = \sum_k \left|\Psi^{(k)}\right|^2$. Computing an orthonormal set of eigenstates for such a Hamiltonian is accomplished by the self-consistent loop [133–135]. A simplified description of the procedure begins with a reasonable initial guess for the eigenstates. Those eigenstates are used to compute the density-dependent parts of the Hamiltonian, yielding a now no-longer density-dependent Hamiltonian,[43] which can be diagonalized to find a new set of eigenstates [134]. This new set is again used to compute the density-dependent terms of the Hamiltonian, and the iteration repeats until some convergence criteria is met.

Example

Here, we consider a highly-simplified, toy, density-dependent Hamiltonian with all the mathematical features that would need to be considered for the effective emulation of a realistic DFT Hamiltonian.

$$H\left(\alpha,\, c;\, \left\{\Psi^{(k)}\right\}\right) = T + \alpha V - c\sum_k \left|\Psi^{(k)}\right|^2 \tag{3.69}$$

This Hamiltonian is parametric with the parameter α scaling the typical non-density-dependent potential V and the parameter c scaling the density-dependent term which is a sum of a set of states $\left\{\Psi^{(k)}\right\}$. In this example, T is the usual kinetic energy operator, V is a harmonic oscillator potential, and we take the number of states to be 2. To build a PMM emulator for this system, we can treat the nonlinear term in either of the two ways described in Section 3.3.2.

Before we can write either form of PMM emulator, we must disambiguate the notation in Eq. (3.69). Specifically, at first glance the nonlinear term appears to be a vector, but the linear terms are unmistakably operators (matrices) which implies an impossible addition between these operators and the nonlinear term. The truth is that $\left|\Psi^{(k)}\right|^2$ is a slight abuse

[41] Specifically, the Kohn-Sham potential or in reality an approximation of it [135].

[42] Usually called Kohn-Sham "orbitals" in DFT literature [134].

[43] This is not dissimilar to evaluating parameter-dependent Hamiltonians at a specific parameter realization, thereby yielding a fixed Hamiltonian without any parameters.

of notation, instead it actually denotes either the diagonal matrix whose elements are $|\Psi_k|^2$ or hides an implicit elementwise product $\odot$ after the term, so that when H acts on a vector, v, the nonlinear term appears as

$$-c\sum_k \left|\Psi^{(k)}\right|^2 \odot v. \tag{3.70}$$

Both more-explicit interpretations of the nonlinear term in Eq. (3.69) are equivalent; however, since the self-consistent iteration relies on an eigenvalue decomposition of H with explicit $\Psi^{(k)}$ realizations, it is almost necessary[44] to take the diagonal-operator interpretation to form the emulator, which we use in this example.

We adopt the usual notation where n is the dimension of the reduced space and N is the dimension of the full space. The first PMM, denoted here as PMM–I, follows the treatment of polynomially nonlinear terms in the way of the POD-Galerkin method (see Section 2.11.1.1),

$$\underline{H}^{\mathrm{I}}\left(\alpha,\, c;\, \left\{\underline{\Psi}^{(k)}\right\}\right) = \underline{T} + \alpha\underline{V} - c\sum_k \underline{G}^{(3*)}\left(\underline{\Psi}^{(k)*}\otimes\underline{\Psi}^{(k)}\right), \tag{3.71}$$

where $\underline{G}^{(3*)} \in \mathbb{C}^{n\times n\times n\times n}$ is an order-4 tensor. To arrive at the nonlinear term in Eq. (3.71), we first consider each nonlinear term in the sum as a black-box function, $F\left(\Psi^{(k)}\right)$, given by

$$F(\Psi^{(k)}) \equiv \left|\Psi^{(k)}\right|^2 \equiv \mathrm{diag}\left(\left|\Psi^{(k)}\right|^2\right), \tag{3.72}$$

which in the reduced space under the POD projector P becomes

$$P^\dagger F\left(P\underline{\Psi}^{(k)}\right)P. \tag{3.73}$$

Now, expanding the known form of $F(\cdot)$ in Einstein summation notation yields

$$\begin{aligned}\left[P^\dagger F\left(P\underline{\Psi}^{(k)}\right)P\right]_{ij} &= \left(P^\dagger\right)_{i\mu}\delta_{\mu\nu}\left(P\underline{\Psi}^{(k)}\right)^*_\mu\left(P\underline{\Psi}^{(k)}\right)_\nu P_{\nu j}\\ &= \underbrace{\left(P^\dagger\right)_{i\mu}P^*_{\mu\omega}P_{\mu\sigma}P_{\mu j}}_{\underline{G}^{(3*)}_{ij\omega\sigma}}\Psi^{(k)*}_\omega\Psi^{(k)}_\sigma\\ \implies \left|\Psi^{(k)}\right|^2 &\to \underline{G}^{(3*)}\left(\underline{\Psi}^{(k)*}\otimes\underline{\Psi}^{(k)}\right).\end{aligned} \tag{3.74}$$

[44] It is possible to implement the Hamiltonian in a matrix-free way, which would enable the elementwise-product interpretation (see Appendix A) but doing so in the reduced space of the emulator is unnecessary complexity and would likely result in lesser performance.

The second PMM, denoted by PMM–II, is

$$\underline{H}^{\mathrm{II}}\Big(\alpha,\, c;\, \Big\{\underline{\Psi}^{(k)}\Big\}\Big) = \underline{T} + \alpha\underline{V} - c\sum_{k}\Big(\underline{G}^{(2)}\underline{\Psi}^{(k)}\Big)^{\dagger}\Big(\underline{G}^{(2)}\underline{\Psi}^{(k)}\Big), \tag{3.75}$$

where $\underline{G}^{(2)} \in \mathbb{C}^{n\times n\times n}$ is an order-3 tensor. To derive this form, we first rewrite the nonlinear term in Eq. (3.69) using the canonical matrix-valued magnitude-squared operation $F_S(A) \equiv A^{\dagger}A = V_A \mathrm{diag}\big(|\lambda_A|^2\big)V_A^{-1}$,

$$\sum_{k}\big|\Psi^{(k)}\big|^2 \equiv \sum_{k} F_S\big(\mathrm{diag}\big(\Psi^{(k)}\big)\big) \equiv \sum_{k}\mathrm{diag}\big(\Psi^{(k)}\big)^{\dagger}\mathrm{diag}\big(\Psi^{(k)}\big). \tag{3.76}$$

This equivalent form involves no nonlinear operations and so can be projected straightforwardly. Let $D^{(k)} \equiv \mathrm{diag}\big(\Psi^{(k)}\big)$, then

$$D^{(k)} \equiv \mathrm{diag}\big(\Psi^{(k)}\big) \to P^{\dagger}\mathrm{diag}\Big(P\underline{\Psi}^{(k)}\Big)P \equiv \underline{D}^{(k)}, \tag{3.77}$$

which expanded in Einstein summation notation is

$$\begin{aligned}
\underline{D}^{(k)}_{ij} = \Big[P^{\dagger}\mathrm{diag}\Big(P\underline{\Psi}^{(k)}\Big)P\Big]_{ij} &= \big(P^{\dagger}\big)_{i\mu}\delta_{\mu\nu}P_{\nu\omega}\underline{\Psi}^{(k)}_{\omega}P_{\nu j} \\
&= \underbrace{\big(P^{\dagger}\big)_{i\mu}P_{\mu\omega}P_{\mu j}}_{\underline{G}^{(2)}_{ij\omega}}\underline{\Psi}^{(k)}_{\omega} \\
\implies \underline{D}^{(k)} &= \underline{G}^{(2)}\underline{\Psi}^{(k)}.
\end{aligned} \tag{3.78}$$

And so,

$$\big|\Psi^{(k)}\big|^2 \to \underline{D}^{(k)\dagger}\underline{D}^{(k)} \equiv \Big(\underline{G}^{(2)}\underline{\Psi}^{(k)}\Big)^{\dagger}\Big(\underline{G}^{(2)}\underline{\Psi}^{(k)}\Big). \tag{3.79}$$

Both PMMs above represent the linear terms, $T + \alpha V$, as $\underline{T} + \alpha\underline{V}$ where $\underline{T}$ and $\underline{V}$ are trainable $n \times n$ Hermitian matrices with no further constraints. PMM–I and PMM–II approach equivalence with one another as the size of the reduced space approaches the size of the full space, and both are valid dimensionality-reduced forms of the full Hamiltonian in Eq. (3.69). However, following the naive parameterization of each, $\underline{G}^{(3*)}$ in PMM–I would require n^4 trainable complex parameters compared to the n^3 trainable complex parameters required by $\underline{G}^{(2)}$ in PMM–II. Following the discussion in Section 3.5.6, both of these

tensors can be approximated in the reduced space by parameterizing their projector-based construction directly. Consider the definition of each tensor with the projector $P \in \mathbb{U}(N, n)$,

$$\begin{aligned} \underline{G}^{(2)}_{ijk} &\equiv \left(P^\dagger\right)_{i\mu} P_{\mu j} P_{\mu k}, \\ \underline{G}^{(3*)}_{ijkl} &\equiv \left(P^\dagger\right)_{i\mu} P_{\mu j} P^*_{\mu k} P_{\mu l}. \end{aligned} \tag{3.80}$$

Instead, we truncate the larger dimension of P to m with $n \leq m \ll N$ and call this truncated approximation of the projector[45] $\underline{Q} \in \mathbb{U}(m, n)$ which we can use to approximate $\underline{G}^{(2)}$ and $\underline{G}^{(3*)}$ by

$$\begin{aligned} \underline{G}^{(2)}_{ijk} &\approx \left(\underline{Q}^\dagger\right)_{i\mu} \underline{Q}_{\mu j} \underline{Q}_{\mu k}, \\ \underline{G}^{(3*)}_{ijkl} &\approx \left(\underline{Q}^\dagger\right)_{i\mu} \underline{Q}_{\mu j} \underline{Q}^*_{\mu k} \underline{Q}_{\mu l}. \end{aligned} \tag{3.81}$$

This allows the number of parameters in each of the two PMMs to be equal, as now the nm complex elements of $\underline{Q}$ are the only trainable parameters for the nonlinear term. However, the cost of computing the approximations in Eq. (3.81) is still more expensive in PMM–I by a factor of n, as is the cost of using the tensor to compute the reduced nonlinear term. This example highlights the efficiency improvements that occur when replacing nonlinear terms by their canonical matrix-valued counterparts.

Both PMMs solve for the eigenstates and eigenenergies of their respective reduced-space state-dependent Hamiltonians via the same self-consistent iteration that would be used in the full space. Since all operations in the self-consistent loop are linear—only matrix-vector products and vector addition—nothing needs to be modified to accurately emulate this method in the reduced space.

In this example, we take $N = 25$ and randomly sample 100 training points uniformly in the domain $(\alpha, c) \in [0, 2] \times [0, 1]$. For a given α and c, the high-fidelity solution is given by the self-consistent iteration that results in $\left[\Psi^{(0)}, \Psi^{(1)}\right]$ and $[E_0, E_1]$ such that the $\Psi^{(k)}$ are orthonormal and

$$\begin{aligned} H\big(\alpha,\, c;\, \left\{\Psi^{(0)}, \Psi^{(1)}\right\}\big)\Psi^{(0)} &= E_0 \Psi^{(0)}, \\ H\big(\alpha,\, c;\, \left\{\Psi^{(0)}, \Psi^{(1)}\right\}\big)\Psi^{(1)} &= E_1 \Psi^{(1)}. \end{aligned} \tag{3.82}$$

[45] This is only an approximation of the projector in the context of computing $\underline{G}^{(2)}$ and $\underline{G}^{(3*)}$. It is not an approximation of the projector in the context of projecting the high-fidelity equations to create an eRBM.

We seek an emulator for the energies and the states, so we follow the steps for vector-valued PMMs in Section 3.2.1. The states from the training data are used to form the POD projector, $P \in \mathbb{U}(N, n)$, for a reduced dimension of size $n = 5$. The states in the training data are projected to the reduced space by $\hat{\Psi}^{(k)} \equiv P^\dagger \Psi^{(k)}$, and the predictions from the emulator are projected from the reduced space to the full space by $\tilde{\Psi}^{(k)} = P\underline{\Psi}^{(k)}$. The loss between the reduced-space predictions from the emulator, $\underline{\Psi}^{(k)}$ and $\underline{E_k}$, and the high-fidelity results projected to the reduced space, $\hat{\Psi}^{(k)}$ and E_k, is taken to be

$$\mathcal{L}\Big(\Big\{\Big(\hat{\Psi}^{(k)}, E_k\Big)\Big\}, \Big\{\Big(\underline{\Psi}^{(k)}, \underline{E}_k\Big)\Big\}\Big) = 1 - \frac{1}{2}\sum_{k=0,1}\left[\left|\hat{\Psi}^{(k)\dagger}\underline{\Psi}^{(k)}\right|^2 - |E_k - \underline{E}_k|^2\right], \tag{3.83}$$

which is the sum of the mean squared error of the energies and the mean "overlap error" of the states. The overlap error between two unit vectors u and v is simply $1 - \left|u^\dagger v\right|^2$ which is bounded between 0 and 1. Note that since the loss involves only inner products of the states, it is equivalent to computing the same loss in the full space due to the properties of the POD projector (see Section 2.11.1.1). In this simple example we are unconcerned with overfitting and so do not withhold any data for validation. After training both PMMs, we evaluate their performance on test data taken uniformly on a 50×50 grid over the same domain that the training data was sampled from. The results are shown in Figs. 3.7 and 3.8. Figure 3.7 shows the true E_0 and E_1 values from the $N = 25$ system and the percent error between both emulators and the true values, demonstrating that both are able to achieve sub-1% errors for both energies across nearly the entire parameter domain. Figure 3.8 shows the overlap error for each emulator's prediction of each state in the full space—that is, $1 - \left|\Psi^{(k)\dagger}\tilde{\Psi}^{(k)}\right|^2$, as opposed to the overlap error in the reduced space $1 - \left|\hat{\Psi}^{(k)\dagger}\underline{\Psi}^{(k)}\right|^2$ although these should be equal due to the properties of P—demonstrating that both are capable of highly accurate predictions except near $\alpha = 0$, where the system develops a continuous degeneracy.

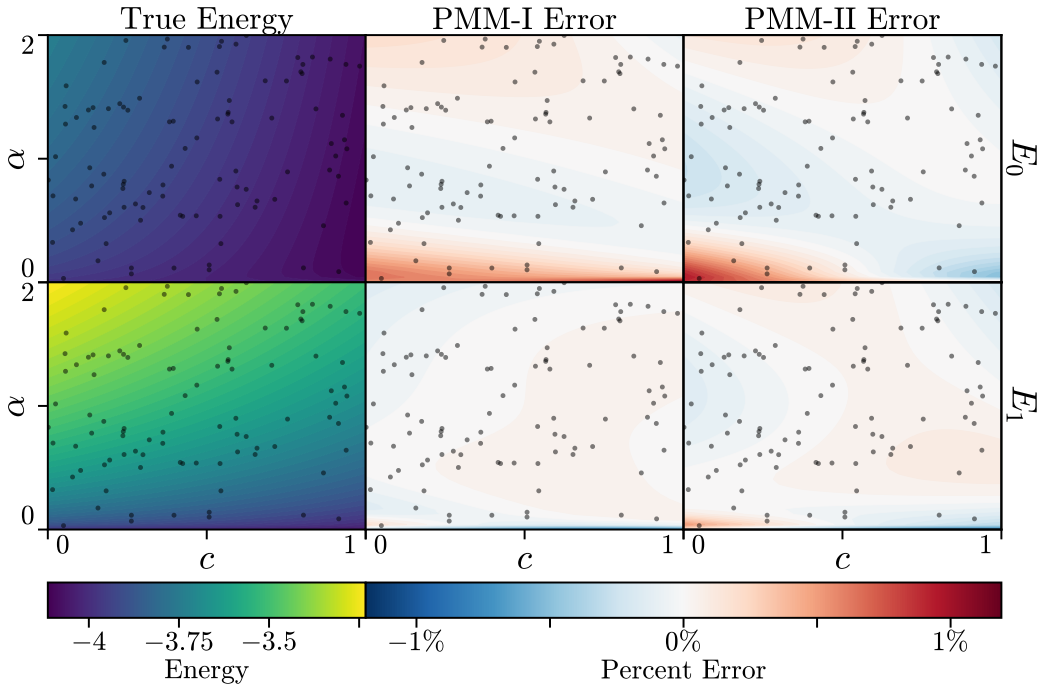

Figure 3.7 (Left column) high-fidelity calculations of the ground state (E_0, top row) and first excited state (E_1, bottom row) energies for the example nonlinear Hamiltonian in Eq. (3.69) throughout the phase space $(\alpha, c) \in [0, 2] \times [0, 1]$. (Middle column and right column) percent errors between the high-fidelity calculations and PMM–I (middle column) and PMM–II (right column) for the energies. The locations of the training data are shown as black points.

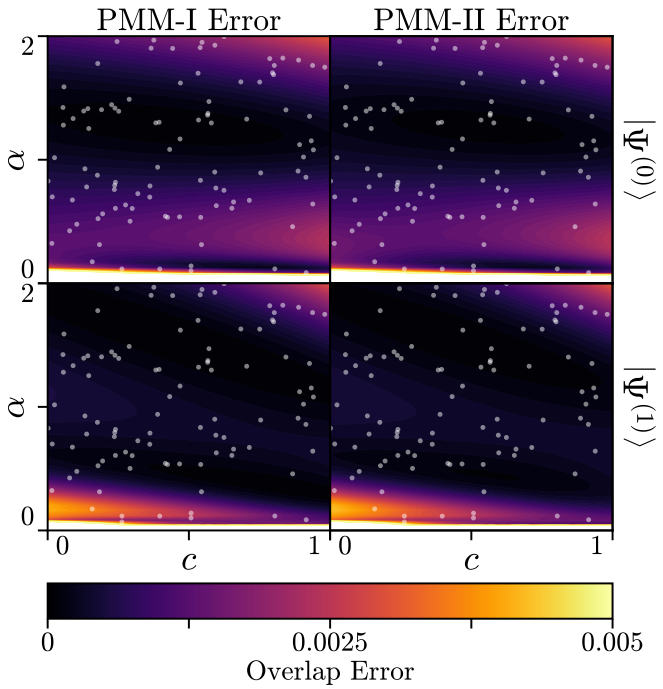

Figure 3.8 Results for the overlap error, $1 - \left| \Psi^{(k)\dagger} \tilde{\Psi}^{(k)} \right|^2$, between the high-fidelity calculations of the ground state (top row) and first excited state (bottom row) eigenvectors of Eq. (3.69) and the corresponding predictions from PMM–I (left column) and PMM–II (right column) across the phase space $(\alpha, c) \in [0, 2] \times [0, 1]$. The locations of the training data are shown as white points.

3.11 Nonlinear Dynamics and Macrostate Data

Nonlinear dynamics combines differential equations and nonlinear terms. This appears in time-dependent density functional theory (TDDFT), in which the time-dependent Schrödinger equation involves a (local) density-dependent parametric interaction [136][46]—that is, in the limit of no pairing (Hartree–Fock (HF) theory),

$$\begin{aligned} i\frac{\partial}{\partial t}\Psi^{(k)}(c,t) &= H\big(c, \{\Psi^{(k)}(c,t)\}\big)\Psi^{(k)}(c,t), \\ \text{where } H\big(c, \{\Psi^{(k)}(c,t)\}\big) &= T + V(c) + U\Big(c, \Big\{\big|\Psi^{(k)}(c,t)\big|^2\Big\}\Big). \end{aligned} \tag{3.84}$$

See Ref. [136] for a review of TDDFT.[47] Direct integration of this equation is exceedingly expensive for sufficiently high-fidelity systems of heavy nuclei, warranting emulation. However, a barrier in the way of emulation is the lack of available data for the wavefunctions. Snapshots of the wavefunction, not only the density, are seemingly required since the dynamics of the system are determined by the wavefunction and not by the density.[48] TDDFT simulations in general do not return snapshots of the single-particle wavefunctions—denoted here by $\Psi^{(k)}$—as both the number and dimension of these wavefunctions are prohibitively large [137].[49] Instead of taking snapshots of all the single-particle states, one might propose using snapshots of the HF many-body wavefunction. However, the many-body wavefunction is a Slater determinant of the single-particle wavefunctions, which would require the computation of $A!$ terms for an A-nucleon system[50]—a completely infeasible calculation. In

[46] In principle, the interaction can be non-local and depend on the full density matrices, $\Psi^{(k)}(c,t)\Psi^{(k)\dagger}(c,t)$, but this greatly complicates the problem, potentially to the point of being prohibitively computationally expensive [136].

[47] Similar to the previous section, this is a greatly oversimplified description of TDDFT that focuses only on the core mathematical elements which are relevant to emulation in the context of RBMs and PMMs. Application-specific complications such as devising the nuclear energy density functional, bases with good quantum numbers, and pairing correlations do not change the core principles discussed here.

[48] It is not possible in general to determine the dynamics of any quantum system purely from the instantaneous local density. Consider the simple 1D harmonic oscillator, where—in the position basis—the momentum of the state is encoded in the complex phase of the wavefunction which is lost in the local density.

[49] In Ref. [137, Section 3.5.3], the authors note that reasonably-sized fission studies with TDDFT require $\sim 10^{11}$ double-precision floating point values per time step and $\gtrsim 10^5$ time steps. This would require more than 10 PB to store all the snapshots.

[50] For example, a TDDFT simulation for ^{240}Pu fission, which appears in Refs. [138, 139], would require $240! > 10^{468}$ terms in the Slater determinant.

contrast, the one-body density is comparatively cheap in terms of both computation time and storage size.

In traditional eRBM-based emulation, the lack of wavefunction snapshots would prohibit the construction of a proper and effective reduced form of Eq. (3.84). However, given the relation between the wavefunctions and the one-body density,

$$\rho = \sum_k \left|\Psi^{(k)}\right|^2, \tag{3.85}$$

we can still form the PMM for this system as if we had snapshots of the wavefunctions. Denote the unknown projector for the wavefunctions by $P^{(\Psi)}$ and the projector for the one-body density by $P^{(\rho)}$; then the eRBM-reduced form for Eq. (3.85) is

$$\begin{aligned}
\underbrace{P^{(\rho)\dagger}\rho}_{\underline{\rho}} &= P^{(\rho)\dagger}\sum_k \Psi^{(k)*}\odot\Psi^{(k)} && \text{(project into } \rho \text{ subspace)}\\
\rightarrow \underline{\rho} &= P^{(\rho)\dagger}\sum_k \left(P^{(\Psi)}\underline{\Psi}^{(k)}\right)^*\odot\left(P^{(\Psi)}\underline{\Psi}^{(k)}\right) && \text{(handle nonlinearity)}\\
\iff \underline{\rho}_i &= \left(P^{(\rho)\dagger}\right)_{i\mu}\sum_k P^{(\Psi)*}_{\mu\nu}\underline{\Psi}^{(k)*}_{\nu}P^{(\Psi)}_{\mu\omega}\underline{\Psi}^{(k)}_{\omega} && \text{(Einstein notation)}\\
&= \sum_k \underbrace{\left(P^{(\rho)\dagger}\right)_{i\mu}P^{(\Psi)*}_{\mu\nu}P^{(\Psi)}_{\mu\omega}}_{\underline{W}^{(\Psi\rightarrow\rho)}_{i\nu\omega}}\underline{\Psi}^{(k)*}_{\nu}\underline{\Psi}^{(k)}_{\omega}\\
\iff \underline{\rho} &= \sum_k \underline{W}^{(\Psi\rightarrow\rho)}\left(\underline{\Psi}^{(k)*}\otimes\underline{\Psi}^{(k)}\right),
\end{aligned} \tag{3.86}$$

where we introduce $\underline{W}^{(\Psi\rightarrow\rho)}$, the order-3 tensor which encodes the nonlinear operation from $\Psi^{(k)}$ to $\left|\Psi^{(k)}\right|^2$ in the reduced basis of ρ. This completes the reduced-basis form only of Eq. (3.85). The eRBM derivation of Eq. (3.84) uses only $P^{(\Psi)}$ since the operators only act on $\Psi^{(k)}$,

$$\begin{aligned}
i\frac{\partial}{\partial t}\underbrace{\left(P^{(\Psi)\dagger}\Psi^{(k)}\right)}_{\underline{\Psi}^{(k)}} &= \left[\begin{array}{l}\underbrace{P^{(\Psi)\dagger}TP^{(\Psi)}}_{\underline{T}}+\underbrace{P^{(\Psi)\dagger}VP^{(\Psi)}}_{\underline{V}}\\ +\,P^{(\Psi)\dagger}U\left(P^{(\Psi)}\underline{G}^{(\Psi,\,2*)}\left[\left(P^{(\Psi)\dagger}\Psi^{(k)*}\right)\otimes\left(P^{(\Psi)\dagger}\Psi^{(k)}\right)\right]\right)P^{(\Psi)}\end{array}\right]\underline{\Psi}^{(k)}\\
\iff i\frac{\partial}{\partial t}\underline{\Psi}^{(k)} &= \left[\underline{T}+\underline{V}+P^{(\Psi)\dagger}U\left(P^{(\Psi)}\underline{G}^{(\Psi,\,2*)}\left(\underline{\Psi}^{(k)*}\otimes\underline{\Psi}^{(k)}\right)\right)P^{(\Psi)}\right]\underline{\Psi}^{(k)},
\end{aligned} \tag{3.87}$$

where the parameter and time dependences, (c, t), have been left implicit for brevity. To complete the reduced form of the model, leaving all objects in the reduced space, the precise form of the functional U would need to be specified. So long as U is either linear in the density or can be handled by the methods for nonlinear terms discussed in Sections 2.11.1.1 and 3.3.2, then the final $P^{(\Psi)\dagger}$ and $P^{(\Psi)}$ in Eq. (3.87) will cancel or otherwise be able to be condensed into a single tensor in the reduced space. Along with Eq. (3.86), this yields a complete RBM form for an HF TDDFT calculation.[51] Computing the reduced-space objects explicitly is not possible due to the aforementioned information gap regarding snapshots of the wavefunctions; however, it is possible to take the implicit approach and train parameterized versions of these operators such that $P^{(\rho)}\underline{\rho}$ fits snapshots of the density.

This concept of relating the relevant state vector of the high-fidelity model to snapshots of some properties of that state vector is the generalization of the typical PMM problem context where only data for a small set of scalar observables is available, prohibiting the application of eRBMs. This more general data, that lies between scalar values and state vectors, is called *macrostate data*.

Example

As an example, we consider the problem of emulating the dynamics of the time-dependent Gross–Pitaevskii equation using only snapshots of the density, $|\Psi|^2$. The equation describes the dynamics of the single-particle wavefunction of a Bose-Einstein condensate and is

$$i\frac{\partial}{\partial t}\Psi(r,t) = \left[-\frac{1}{2m}\nabla^2 + V(r) + g|\Psi(r,t)|^2\right]\Psi(r,t), \tag{3.88}$$

where $V(r)$ is some external trapping potential and we have taken $\hbar = 1$ [140, 141]. Given only snapshots of the density, we can see that this system has all of the components that made TDDFT emulation challenging. For this example we will restrict the problem to one spatial dimension and take the external potential to be the harmonic oscillator potential

[51] The methods discussed here can be generalized to Hartree–Fock–Bogoliubov TDDFT calculations with pairing interactions, as the fundamental nonlinear operations do not change.

with scale parameter a,

$$i\frac{\partial}{\partial t}\Psi(x,t) = \left[-\frac{1}{2m}\frac{\partial^2}{\partial x^2} + ax^2 + g|\Psi(x,t)|^2\right]\Psi(x,t), \tag{3.89}$$

and take the initial state to be a normalized off-center Gaussian

$$\Psi(x,0) \sim \exp\left[-\frac{(x-x_0)^2}{2\sigma^2}\right]. \tag{3.90}$$

For the high-fidelity simulation, we use periodic boundary conditions with $x \in [-10, 10]$ discretized into $N = 1000$ points. Using explicit time integration, we compute the dynamics for 5 values of m taken on a uniform grid in the range $[1, 2]$, 10 values of a taken on a uniform grid in the range $[1/2, 2]$, and 10 values of g taken on a uniform grid in the range $[-40, 0]$. The system is evolved for 500 time steps to $t = 50$. The first 100 snapshots of the density evolution ($0 \le t < 10$) become the training snapshots, the next 100 ($10 \le t < 20$) are reserved for validation, and the remaining 300 ($20 \le t < 50$) are withheld for testing the emulator.

Following the discussion for emulating TDDFT, we derive the form of the PMM emulator for Eq. (3.89) similar to the example in Section 3.10 but instead taking the elementwise-product interpretation of the nonlinear term ($|\Psi|^2\Psi \equiv \Psi^* \odot \Psi \odot \Psi$) rather than the diagonal-operator one.

$$\begin{aligned}
i\frac{\partial}{\partial t}\underbrace{\left(P^{(\Psi)\dagger}\Psi\right)}_{\underline{\Psi}} =& -\frac{1}{m}\underbrace{P^{(\Psi)\dagger}\left(\frac{1}{2}\frac{\partial^2}{\partial x^2}\right)P^{(\Psi)}}_{\underline{T}}\underline{\Psi} + a\underbrace{P^{(\Psi)\dagger}\left(x^2\right)P^{(\Psi)}}_{\underline{V}}\underline{\Psi} \\
&+ gP^{(\Psi)\dagger}\left[\left(P^{(\Psi)}\underline{\Psi}\right)^* \odot \left(P^{(\Psi)}\underline{\Psi}\right) \odot \left(P^{(\Psi)}\underline{\Psi}\right)\right] \\
\iff i\frac{\partial}{\partial t}\underline{\Psi}_i =& -\frac{1}{m}\underline{T}_{i\mu}\underline{\Psi}_\mu + a\underline{V}_{i\mu}\underline{\Psi}_\mu + g\underbrace{\left(P^{(\Psi)\dagger}\right)_{i\mu}P^{(\Psi)*}_{\mu\nu}P^{(\Psi)}_{\mu\sigma}P^{(\Psi)}_{\mu\omega}}_{\underline{G}^{(\Psi,3*)}_{i\nu\sigma\omega}}\underbrace{\underline{\Psi}^*_\nu\underline{\Psi}_\sigma\underline{\Psi}_\omega}_{(\underline{\Psi}^*\otimes\underline{\Psi}\otimes\underline{\Psi})_{\nu\sigma\omega}} \\
\iff i\frac{\partial}{\partial t}\underline{\Psi} =& -\frac{1}{m}\underline{T}\underline{\Psi} + a\underline{V}\underline{\Psi} + g\underline{G}^{(\Psi,3*)}(\underline{\Psi}^* \otimes \underline{\Psi} \otimes \underline{\Psi}),
\end{aligned} \tag{3.91}$$

where the density in the reduced space is given by

$$\underline{\rho} = \underline{W}^{(\Psi\to\rho)}(\underline{\Psi}^* \otimes \underline{\Psi}), \tag{3.92}$$

and predictions for $\rho(t)$ in the full space, denoted by $\tilde{\rho}(t)$, are given by

$$\rho(t) \approx \tilde{\rho}(t) = P^{(\rho)}\underline{\rho}, \tag{3.93}$$

where $P^{(\rho)}$ is the POD projector computed from the training snapshots of the density. We choose a reduced-space dimension of $n = 8$, less than 1% the size of the full space. We parameterize the reduced-space operators following Section 3.5, and—since we are only concerned with a single initial state[52]—we directly parameterize the reduced-space initial state $\underline{\Psi}_0$ as a trainable unit vector. We train this PMM by minimizing the mean squared error between the reduced-space projection of the target density, $P^{(\rho)\dagger}\rho(t)$, and the reduced-space prediction $\underline{\rho}(t)$.

Figure 3.9 shows the results of using this trained emulator to predict the dynamics for a parameter set included in the training data, $[m, a, g] = [1.50, 1.33, -17.78]$, and for a parameter set not present in either the training or validation data, $[m, a, g] = [1.33, 1.75, -20.00]$. Both cases show good agreement with the high-fidelity result—and importantly stability at late times—well beyond the training and validation horizons. Additionally, the aggressive dimensionality reduction provided by the emulator reduces the real-time cost of the dynamics by nearly a factor of 50.[53]

[52] In general, different initial states that, for example, come from the ground state of a separate parametric Hamiltonian could be emulated by including the reduced-space representation of their generating process in the emulator.

[53] On a consumer-grade laptop, a single high-fidelity simulation of the dynamics required 7 minutes 22 seconds, compared to 9 seconds for the emulator. Since these times include data- and model-loading overhead—which would impact the emulator proportionally much more—the 50× figure is a lower bound.

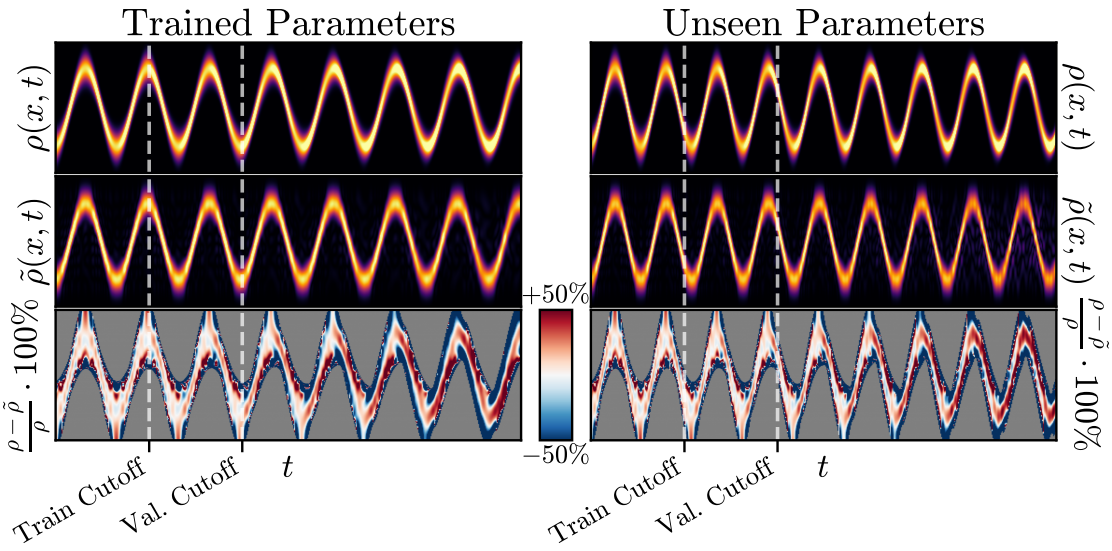

Figure 3.9 High-fidelity and emulator-predicted results for the dynamics of the system in Eq. (3.89). The left column shows the dynamics for a system with parameters that were included in the training data, $[m, a, g] = [1.50, 1.33, -17.78]$. The right column, conversely, shows results for parameters not seen by the emulator during training or validation, $[m, a, g] = [1.33, 1.75, -20.00]$. The top row shows the local density dynamics, $\rho(x, t)$, from the high-fidelity solution with x on the vertical axis and t on the horizontal axis. The middle row shows the emulator-predicted local density dynamics, $\tilde{\rho}(x, t)$. Finally, the bottom row shows the percent error between the high-fidelity dynamics and the emulated dynamics—regions where the high-fidelity density is too small are shown in grey since a relative error metric is not meaningful there. The training cutoff (initial 1/5 of the dynamics) and validation cutoff (initial 2/5 of the dynamics) are indicated by dashed lines. The visible range of x values is $[-5, 5]$ to better show the dynamics of the system.

3.12 Incomplete Underlying Equations

The applications discussed so far have operated with full, explicit knowledge of the form of the equations governing the high-fidelity model or, as in Section 3.8, no knowledge of any underlying equations whatsoever, in which case a universal form was prescribed. Between these two extremes lies problems where much or some, but not all, of the form of the high-fidelity equations are known. In these cases, missing terms can be replaced with physically constrained—or entirely unconstrained—data-driven machine learning models while leaving the known terms unchanged. This essentially amounts to devising an informed guess for a full, explicit form of the high-fidelity model from which the eRBM derivation for the associated PMM emulator can begin.

It is usually the case that significant physical insight is known about the missing terms in the underlying equations. The ability to incorporate this information is what makes the PMM method so powerful and adaptable. Consider the case of emulating a system-size-dependent, parametric, quantum many-body interaction (parameterized by c) which preserves particle number. We know that the Hamiltonian for L bodies is a projection of some infinitely large Hamiltonian—which describes the interaction for any possible system size—onto the states which describe L particles. We can write this as

$$H_L(c) = U^\dagger(L) H_\infty(c) U(L) \tag{3.94}$$

where $H_\infty(c)$ represents the particle-number-independent representation of the interaction parameterized by c and $U(L)$ is the projection onto the space which spans the states of L particles. We may well know the form of $H_\infty(c)$ (for example, affine) but we may not know how the basis projection $U(L)$ depends on L. For now however, we continue with the eRBM derivation, introducing the implicit POD projector P_1 which represents a suitable n_1-dimensional reduced space for $H_L(c)$,

$$\underline{H}_L(c) \equiv P_1^\dagger H_L(c) P_1, \tag{3.95}$$

where $n_1 \leq \dim(H_L(c))$. Substituting the definition of $H_L(c)$ and introducing the implicit POD projector P_2 which represents a suitable n_2-dimensional reduced space for $H_\infty(c)$—with $n_1 < n_2 \leq \dim(H_\infty(c))$—we have

$$\begin{aligned} \underline{H}_L &= \underbrace{P_1^\dagger U^\dagger(L) P_2}_{\underline{U}^\dagger(L)} \underbrace{P_2^\dagger H_\infty(c) P_2}_{\underline{H}_\infty(c)} \underbrace{P_2^\dagger U(L) P_1}_{\underline{U}(L)} \\ &= \underline{U}^\dagger(L) \underline{H}_\infty(c) \underline{U}(L), \end{aligned} \tag{3.96}$$

where $\underline{U}(L) \equiv P_2^\dagger U(L) P_1$ is the reduced-space representation of the finite-particle-number basis projection and $\underline{H}_\infty(c) \equiv P_2^\dagger H_\infty(c) P_2$ is the reduced-space representation of the particle-number-independent representation for the interaction. Note that this would be systematically improvable if the form of $U(L)$ was known, and note that the exact form of $\underline{H}_\infty(c)$

would be known in this example and thus so would the form of $\underline{H}_\infty(c)$. Even without the exact form of $U(L)$, we still know one very important physical constraint on it: it is semi-unitary. Here, we introduce a data-driven, universal form able to approximate any function which maps any number of real inputs $x \in \mathbb{R}^p$ to a semi-unitary operator that represents a basis compression from a space of size N to one of size $n \leq N$,

$$U_{\text{univ}}(x) \equiv \exp\left[i \sum_{j=1}^{l} f_j(x) M_j\right](:, 1:n), \tag{3.97}$$

where M_j are $N \times N$ Hermitian operators, $f_j(x)$ are universal functions (for example, the output of an ANN), and $A(:, 1:n)$ represents taking the first n columns of A. For sufficiently large l—a hyperparameter akin to a "hidden model" size—and expressive f_j, this expression can represent any parameterized semi-unitary operator to arbitrary accuracy. In analogy with traditional data-driven machine learning, we refer to the $f_j(x)$ as the identified *hidden features* of the emulator. We can now approximate $U(L)$ by this universal form, $U(L) \approx U_{\text{univ}}(L)$. From the properties of orthogonal projections, we then know that $\underline{U}(L)$ is approximated by

$$\underline{U}(L) \approx \underline{U}_{\text{univ}}(L) \equiv \exp\left[i \sum_{j=1}^{l} f_j(x) \underline{M}_j\right](: 1:n_1), \tag{3.98}$$

where $\underline{M}_j$ are $n_2 \times n_2$ Hermitian matrices. By substituting this into our expression for $\underline{H}_L(c)$, we see that we now have a complete, partially-data-driven, PMM for $H_L(c)$,

$$\begin{aligned} \underline{H}_L(c) &= \underline{U}^\dagger(L) \underline{H}_\infty(c) \underline{U}(L), \\ \text{where } \underline{U}(L) &= \exp\left[i \sum_{j=1}^{l} f_j(L) \underline{M}_j\right](:, 1:n_1). \end{aligned} \tag{3.99}$$

To use this PMM emulator, one need only to substitute the known form of $\underline{H}_\infty(c)$, choose suitable values of n_1, n_2, and l, and select the type of universal functions to use for f_j.

To concretize this example, we again consider the Hamiltonian described in Appendix B, the same Hamiltonian from the examples in Sections 2.11.1.4 and 3.7, which describes a 1D chain of L spins with couplings J_{ij} in the influence of a transverse field with strength B,

$$H = -\frac{1}{\mathcal{J}} \sum_{i<j}^{L} J_{ij} \left(\gamma_x \sigma_i^x \sigma_j^x + \gamma_y \sigma_i^y \sigma_j^y\right) - B \sum_{i}^{L} \sigma_i^z, \tag{3.100}$$

where $\gamma_{x/y}$ are the relative strengths of the x and y couplings, σ_i^u is the Pauli spin operator acting on the i^{th} spin in the u axis, and $\mathcal{J}$ is the Kac normalization factor

$$\mathcal{J} \equiv \frac{1}{L-1}\sum_{i\neq j}^{L} J_{ij}. \tag{3.101}$$

Here we take J_{ij} to be the power-law interaction, $J_{ij} = 1/|i-j|^{0.5}$, and γ_y to be 0. We take $n_1 = 5$, $n_2 = 7$, $l = 2$, and f_1 and f_2 to be the outputs of a simple multi-layer perceptron (MLP) with 4 hidden layers with 2, 4, 4, and 2 nodes and softplus[54] activation functions. This makes the form of the PMM

$$\begin{aligned} H(\gamma_x;\, L) &= \underline{U}^\dagger(L)\underline{H}_\infty(\gamma_x)\underline{U}(L), \\ \text{where } \underline{H}_\infty(\gamma_x) &= \underline{H}_0 + \gamma_x \underline{H}_1, \\ \text{and } \underline{U}(L) &= \exp\Big[i f_1^{(\text{MLP})}(L)\underline{M}_1 + i f_2^{(\text{MLP})}(L)\underline{M}_2\Big](:, 1:n_1). \end{aligned} \tag{3.102}$$

To emulate the ground state energy as a function of both system size L and x coupling strength γ_x, we train and validate this PMM on high-fidelity results for $E_0(\gamma_x;\, L)$ for $(\gamma_x, L) \in [0,2] \times [3,10]$. This represents high-fidelity Hamiltonian dimensions of $2^3 = 8$ to $2^{10} = 1024$. We test the emulator by extrapolating to $L = 14$, representing a full-space dimensionality 16 times greater than that of any of the data the emulator was trained on. The results for this example are shown in Fig. 3.10. Note that despite the purely-data-driven "hidden" model within the emulator depending only on L, the emulator is able to accurately capture the correlation between L and γ_x in the energy.

Alternatively, it may instead be the case that a term is known but is too complicated or expensive to include in the emulator. Consider the case of a differential equation which is known to be linear and parametrically affine to first order, but the form of the higher-order corrections are unknown

$$\frac{dv(\mu)}{dt} = (A + \mu B)v(\mu) + F(\mu; v(\mu)), \tag{3.103}$$

[54] $\text{softplus}(x) \equiv \ln(1 + \exp(x))$.

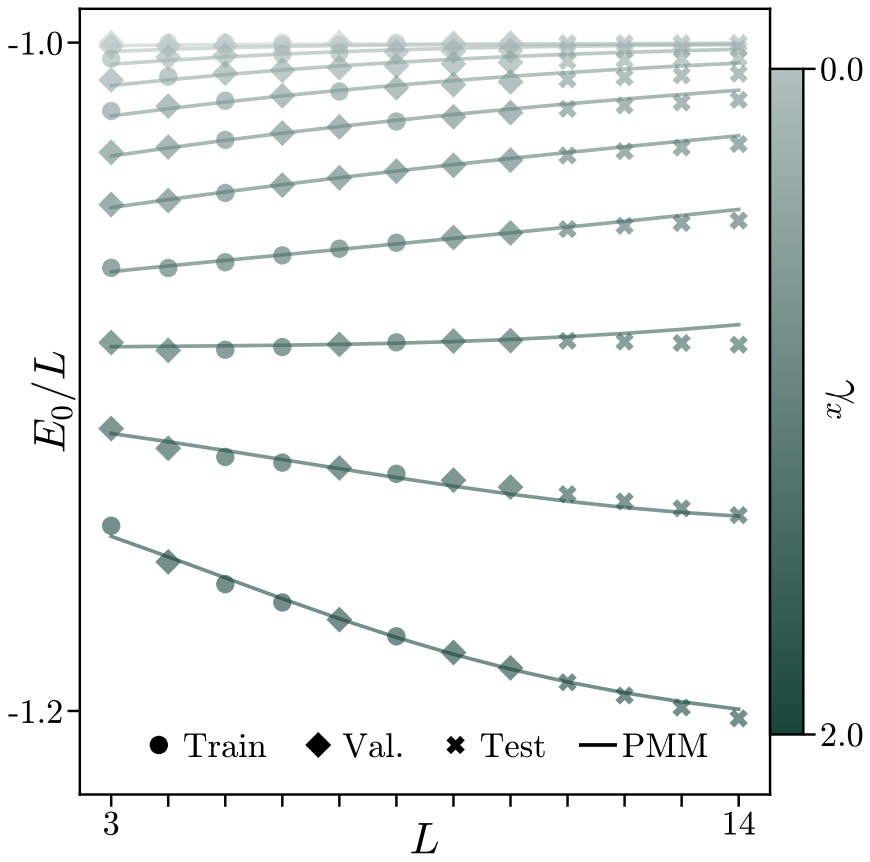

Figure 3.10 High-fidelity (markers) and PMM-emulated (lines) ground state energy per particle of the parametric Hamiltonian in Eq. (3.100) as a function of particle number (L, horizontal axis) and x-coupling (γ_x, color). Training data is denoted by circles, validation data is denoted by square diamonds, and withheld test data is denoted by crosses. Darker shade indicate a stronger x-coupling via larger γ_x.

where $F(\mu; v(\mu))$ denotes the parametric higher-order (potentially nonlinear) terms whose form is unknown. The form of the RBM would then be

$$\frac{d\underline{v(\mu)}}{dt} = (\underline{A} + \mu\underline{B})\underline{v(\mu)} + f\Big(\mu; \underline{v(\mu)}\Big) \tag{3.104}$$

where $f\Big(\mu; \underline{v(\mu)}\Big)$ is some unknown parametric function of μ and $\underline{v(\mu)}$. We again can substitute a numerically parametric universal approximator for f and train its parameters along with the reduced operators. Often, even without knowledge of the form of F (and therefore f), properties are known that greatly constrain the form of the universal function. For example, a common case is that the dependence on μ and $v(\mu)$ is multiplicatively separable,

$$f\Big(\mu, \underline{v(\mu)}\Big) = \sum_i g_i(\mu) h_i\Big(\underline{v(\mu)}\Big), \tag{3.105}$$

for independent universal functions g_i and h_i.

Incorporating as many constraints as possible into the form of any data-driven parts of a partially-data-driven PMM vastly improves the performance of the resulting emulator—especially in low-data regimes and extrapolation. Common choices for universal functions used as parts of a partially-data-driven PMM are polynomials, ANNs, and purely-data-driven PMMs of the form discussed in Section 3.8.

3.12.1 Nuclear Equation of State

This section details the, at the time of writing, most comprehensive and in-depth application of the PMM method to the emulation of a real, non-toy, high-fidelity model of a physical system. The contents of this section largely come from a manuscript written by Kang Yu, myself, Dr. Christian Drischler, and Dr. Scott Bogner that at the time of writing is still in preparation [142].

The (zero-temperature) nuclear equation of state (EOS) relates the energy per nucleon of nuclear matter—an infinite, homogeneous, isotropic system of protons and neutrons—to the densities of protons and neutrons in that matter. This in turn describes many thermodynamic properties of nuclear matter, such as the pressure and compressibility. The EOS is fundamental to understanding the structure of dense astrophysical objects such as neutron stars and connects nuclear physics at the microscopic scale to observables at the stellar scale [74, 143–146]. Therefore, constraining the EOS using microscopic, *ab initio* methods has drawn considerable attention and progress using chiral effective field theory (χEFT) paired with a wide variety of many-body methods has continued. One specific many-body method which we will use here, in-medium similarity renormalization group (IMSRG), was recently extended to nuclear matter and the nuclear EOS [147, 148]. Constraining the nuclear EOS to match experimental observations requires repeated calculations, most often of sampling the various interaction strengths, for rigorous uncertainty quantification (UQ). For sophisticated and accurate many-body methods, like IMSRG, this is computationally intractable as a single evaluation can take significantly more than 24 hours even in a high-

performance computing environment and many tens or hundreds of thousands of evaluations may be necessary. Emulators currently provide the only way of surmounting this computational resource limitation.

In this section we construct and train a comprehensive PMM emulator for IMSRG calculations of the energy of nuclear matter both in the pure neutron matter (PNM) ($Z = 0$) and symmetric nuclear matter (SNM) ($N = Z$) regimes as a function of the nuclear density with fully quantified emulator uncertainties. Additional nuclear matter properties such as the pressure, symmetry energy, slope parameter, and incompressibility can be calculated from this. This section will focus only on the creation, calibration, and evaluation of the emulator. For the application of this emulator to the problem of constraining the coupling constants of the quark-mass-dependent three-nucleon interactions in the Entem–Machleidt–Nosyk (EMN) model, see Ref. [142].

3.12.1.1 The Entem–Machleidt–Nosyk Interaction

We first consider the details of the high-fidelity interaction that we aim to emulate. As in Ref. [149], we use the EMN interaction developed in Ref. [150] with a momentum cutoff $\Lambda = 450$ MeV. The free-space Hamiltonian is affine in parameters known as low-energy constants (LECs) and is a chiral expansion with the chiral order denoted by $\nu = \{0, 2, 3, 4\}$ corresponding to leading order (LO), next-to-leading order (NLO), next-to-next-to-leading order ($\mathrm{N^2LO}$), and next-to-next-to-next-to-leading order ($\mathrm{N^3LO}$) respectively;[55]

$$H_{\text{free}}(c, \nu) = \sum_{j=0}^{\nu} \left[V_0^{(j)} + \sum_{k=1} c_k V_k^{(j)} \right], \tag{3.106}$$

where c is the vector of LECs, $V_k^{(j)}$ are Hermitian operators, and $V_k^{(1)} \equiv 0$ in Weinberg power counting [151]. Here we consider emulating only the $\mathrm{N^2LO}$ three-nucleon LECs

[55] There is no limit to the order of the chiral expansion in general, but here we only consider up to $\mathrm{N^3LO}$.

$c = [c_D, c_E, D_2, F_2]$, and so for our purposes here the free-space Hamiltonian simplifies to

$$\begin{aligned} H_{\text{free}}(c,\nu) = V_0^{(0)} & && \text{(LO)} \\ &+ \left\{ V_0^{(2)} \text{ if } \nu \geq 2 \right\} && \text{(NLO)} \\ &+ \left\{ V_0^{(3)} + \sum_{k=1}^{4} c_k V_k^{(3)} \text{ if } \nu \geq 3 \right\} && \text{(N}^2\text{LO)} \\ &+ \left\{ V_0^{(4)} + \sum_{k=1}^{4} c_k V_k^{(4)} \text{ if } \nu \geq 4 \right\} && \text{(N}^3\text{LO).} \end{aligned} \tag{3.107}$$

Additionally, the interaction does not depend on c_D or c_E for PNM, in which case $V_{c_D}^{(j)} \equiv 0$ and $V_{c_E}^{(j)} \equiv 0$ [152].

3.12.1.2 In-medium similarity renormalization group

We now consider the details of the high-fidelity method used to find the energy of nuclear matter given this interaction and a realization of (c, ν). For a complete review of IMSRG and its application to nuclear matter, see Refs. [147, 148, 153–157]. For our purposes here, the important aspects are the "in-medium" representation of the Hamiltonian and the "flow". IMSRG works directly with the in-medium Hamiltonian as opposed to the free-space Hamiltonian. This is achieved by "normal-ordering" the Hamiltonian with respect to a reference state that is specific to the system under consideration. This can be re-interpreted as a projection of the free-space interaction onto the specific system. In IMSRG calculations of nuclear matter, the basis for the in-medium system is described by the density, n, the maximum allowed number of single-particle states, N_s, or equivalently the number of closed shells, and the proton fraction, Y_p. The density n describes the number of nucleons per unit volume.[56] The number of allowed single-particle states N_s is a numerical parameter describing the size of the model space, which in practice cannot be infinite unlike in the physical system. Calculations with larger model spaces are in general more accurate but also more computationally expensive. Finally, Y_p describes the ratio of the mixture of protons and neutrons in the system with $Y_p = 0$ denoting PNM and $Y_p = 0.5$ denoting SNM. In the

[56] We use n for the density here to be consistent with nuclear matter literature. This is not to be confused with the reduced-space dimension notation used in other sections.

basis-projection interpretation, we can write the in-medium Hamiltonian as a parameterized projection of Eq. (3.107),

$$H_{\mathrm{IM}}(c,\nu;\, N_s, Y_p, n) = U^\dagger(N_s, Y_p, n) H_{\mathrm{free}}(c,\nu) U(N_s, Y_p, n), \tag{3.108}$$

where $U(\cdot)$ is a partial isometry—the semi-unitary operator that induces the projection $U^\dagger(\cdot)U(\cdot)$—parameterized by N_s, Y_p, and n. The IMSRG method aims to construct a continuous unitary transformation $V(s)$ that renders this Hamiltonian block-diagonal as $s \to \infty$,

$$H_{\mathrm{IMSRG}}(s) = V^\dagger(s) H_{\mathrm{IM}} V(s) \equiv H_{\mathrm{d}}(s) + H_{\mathrm{od}}(s), \tag{3.109}$$

where

$$\lim_{s\to\infty} H_{\mathrm{od}}(s) = 0, \tag{3.110}$$

$H_{\mathrm{d}}(s)$ is the block-diagonal part of the Hamiltonian, and $H_{\mathrm{od}}(s)$ is the off-diagonal part. The continuous parameter s is known as the flow parameter and this process of computing $H(s)$ as $s \to \infty$ is referred to as "flowing" the Hamiltonian. In principle, the IMSRG result at finite s is the exact ground state energy of the in-medium Hamiltonian plus errors which depend on the $s = 0$ Hamiltonian (and therefore the control parameters of the Hamiltonian) and decay exponentially in s. Denoting the set of Hamiltonian parameters as $X_H \equiv [c, \nu;\, N_s, Y_p, n]$, then

$$\begin{aligned} E_0^{\mathrm{IMSRG}}(X_H;\, s) = {} & E_0(X_H) && \text{(true ground state energy)} \\ & + \sum_i a_i(X_H) \exp[-b_i(X_H) s] && \text{(finite-}s\text{ errors)}, \end{aligned} \tag{3.111}$$

where $a_i(X_H)$ and $b_i(X_H)$ are smooth, non-negative functions which depend only on the Hamiltonian parameters, and $E_0(X_H)$ is the algebraically lowest eigenvalue of $H_{\mathrm{IM}}(X_H)$. Crucially, the number and exact form of $a_i(X_H)$ and $b_i(X_H)$ are not known.

3.12.1.3 The PMM–IMSRG Emulator

From Eq. (3.107), Eq. (3.111), and the discussion at the beginning of this section (Section 3.12), we can write the form of the PMM emulator for IMSRG predictions of the energy

of nuclear matter as

$$E_0^{\mathrm{PMM}}(X_H; s) = \underline{E}_0(X_H) + \sum_i^{l_1} \alpha_i(X_H) \exp[-\beta_i(X_H)s], \tag{3.112}$$

where $\alpha_i(X_H)$ and $\beta_i(X_H)$ are non-negative universal functions which will be trained to approximate the exponentially decreasing error terms and l_1 is a hyperparameter determining the number of these universal functions to use, and

$$\begin{aligned} &\underline{H}_{\mathrm{IM}}(X_H)\underline{\Psi}_0(X_H) = \underline{E}_0(X_H)\underline{\Psi}_0(X_H) \\ &\quad \text{with } \underline{H}_{\mathrm{IM}}(X_H) = \underline{U}^\dagger(N_s, Y_p, n)\underline{H}_{\mathrm{free}}(c,\nu)\underline{U}(N_s, Y_p, n), \end{aligned} \tag{3.113}$$

$$\underline{U}(N_s, Y_p, n) = \exp\left[i \sum_{j=1}^{l_2} f_j(N_s, Y_p, n)\underline{M}_j\right](:, 1:n_1), \tag{3.114}$$

with $\underline{H}_{\mathrm{free}} \in \mathbb{H}(n_2)$ and $\underline{H}_{\mathrm{IM}} \in \mathbb{H}(n_1)$. $f_j(N_s, Y_p, n)$ are again trainable universal functions of which there are l_2, and

$$\begin{aligned} \underline{H}_{\mathrm{free}}(c,\nu) = \underline{V}_0^{(0)} &+ \left\{\underline{V}_0^{(2)} \text{ if } \nu \geq 2\right\} \\ &+ \left\{\underline{V}_0^{(3)} + \sum_{k=1}^{4} c_k \underline{V}_k^{(3)} \text{ if } \nu \geq 3\right\} \\ &+ \left\{\underline{V}_0^{(4)} + \sum_{k=1}^{4} c_k \underline{V}_k^{(4)} \text{ if } \nu \geq 4\right\}. \end{aligned} \tag{3.115}$$

An additional preprocessing step is required to enforce the physical constraints of the model and its lack of dependence on LECs at specific Y_p and ν,

$$\begin{aligned} c_D &\leftarrow \begin{cases} 0 & \text{if } \nu \leq 2 \text{ or PNM} \\ c_D & \text{otherwise} \end{cases} \qquad c_E \leftarrow \begin{cases} 0 & \text{if } \nu \leq 2 \text{ or PNM} \\ c_E & \text{otherwise} \end{cases} \\ D_2 &\leftarrow \begin{cases} 0 & \text{if } \nu \leq 2 \\ D_2 & \text{otherwise} \end{cases} \qquad F_2 \leftarrow \begin{cases} 0 & \text{if } \nu \leq 2 \\ F_2 & \text{otherwise} \end{cases}. \end{aligned} \tag{3.116}$$

The complete PMM–IMSRG emulator as described above, with additional data rescaling steps, is summarized diagrammatically in Fig. 3.11. All input features, except for N_s and s, are rescaled to the range $[-1, +1]$ via min-max scaling (see Section 3.4.3). N_s is rescaled to the range $[0, +1]$ using min-max scaling. The flow parameter, s, is rescaled so that the minimum value in the training data becomes 0 and the 90^{th} percentile value becomes 1. This is similar to the robust scaling method in Section 3.4.3 but with the 0^{th} and 90^{th} percentiles. To help the model identify the expected exponential convergence with N_s through the data-driven functions f_j, the value of N_s is rescaled in the model by a trainable exponential decay whose coefficients vary between PNM and SNM. The final step of the model is inverse scaling since the model will be trained on data rescaled to $[-1, +1]$ using min-max scaling. All data-driven functions in the emulator were chosen to be the output of smoothed, data-driven PMMs as detailed in Section 3.8 since these were found to perform the best during model selection.

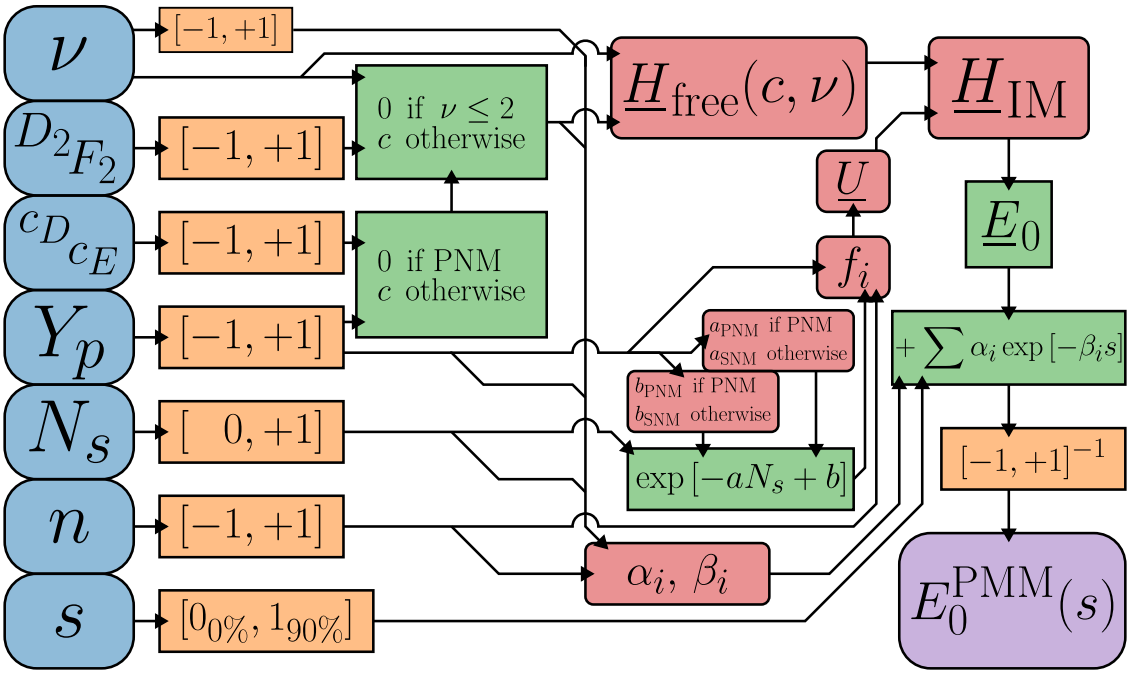

Figure 3.11 Comprehensive diagram of the PMM–IMSRG emulator. Inputs are shown on the left (blue boxes) and the predicted energy on the right (purple box). The rescaled chiral order and LECs feed into the construction of the reduced-space free-space Hamiltonian $\underline{H}_{\text{free}}$, while the learned projection encodes the dependence on the basis size, proton fraction, and density to form the reduced-space in-medium Hamiltonian $\underline{H}_{\text{IM}}$. Red-shaded blocks indicate trainable components, including data-driven PMMs that capture exponential convergence with N_s and finite-s corrections. Diagonalization of $\underline{H}_{\text{IM}}$ produces the $s \to \infty$ ground-state energy, which finite-s corrections are added to and then rescaled to yield the emulator prediction $\underline{E}_0(s) \equiv \underline{E}_0(X_H;\, s) \equiv \underline{E}_0(c, \nu;\, N_s, Y_p, n;\, s)$.

3.12.1.4 Data Collection, Cleaning, and Stratified Partitioning

We collect data for the training, validation, calibration, and testing of the PMM–IMSRG emulator from hundreds of IMSRG calculations. Each calculation specified by the parameters $X_H \equiv [c, \nu;\, N_s, Y_p, n]$ produces an IMSRG prediction for the energy several values of s: $\left\{\left(X_H, s_i;\, E_0^{\text{IMSRG}}(s_i)\right)\right\}$. The Hamiltonian parameters for the calculations uniformly span a broad range of LEC values and densities and cover a wide range of model space sizes, N_s, at all chiral orders from LO to N^{3}LO, for both PNM and SNM. For each IMSRG calculation, the flow continued until either convergence—determined by the criteria $\Delta(s) \equiv |(\Delta E_2(s) + \Delta E_3(s))/E(s)|$ where $E(s)$ is the current IMSRG prediction for the energy at the given s value, $\Delta E_2(s)$ and $\Delta E_3(s)$ are the second- and third-order many-body

perturbation theory energy corrections of the IMSRG-evolved Hamiltonian, respectively, and ε is some tolerance—or divergence, determined by the criteria $\|H_{\mathrm{od}}(s+ds)\| \geq \|H_{\mathrm{od}}(s)\|$. Any IMSRG results which diverged before $s = 5$ are discarded entirely; otherwise only values after the flow diverged are discarded. Any results with $\Delta(s) \geq 0.1$ MeV were discarded as they are considered to not be converged enough to be informative. Finally, de-duplication is performed—retaining only a single IMSRG flow for each unique set of (N_s, Y_p, n) for LO and NLO, since the LECs considered only enter at N^{2}LO and above.

The goal is for the emulator to not only reproduce IMSRG calculations but be able to accurately extrapolate in model size N_s and flow s with fully calibrated uncertainty estimates. To accurately assess real-world model performance, we partition the IMSRG data such that the test set is disproportionally from more-converged (generally larger s) results with larger N_s. Similarly, to encourage the training process to prefer models with more robust extrapolation, we partition the remaining data such that the validation set favors more-converged results with larger N_s. To satisfy the exchangeability requirement of our chosen UQ method (see Section 4.2), we ensure that the calibration set is partitioned equivalently to the test set. All data not yet allocated to the test, validation, or calibration sets is allocated to the training set. The final partitioning of the data is shown in Fig. 3.12. The top row of Fig. 3.12 shows the percentage of the data in each partition for SNM calculations while the bottom row shows the same for PNM. The three columns correspond, from left to right, the number of closed shells determined by the number of single particle states [108], the flow parameter, and the perturbative convergence estimate. At each point on the x-axis of Fig. 3.12, the percentages across the four data partitions sum to 100%.

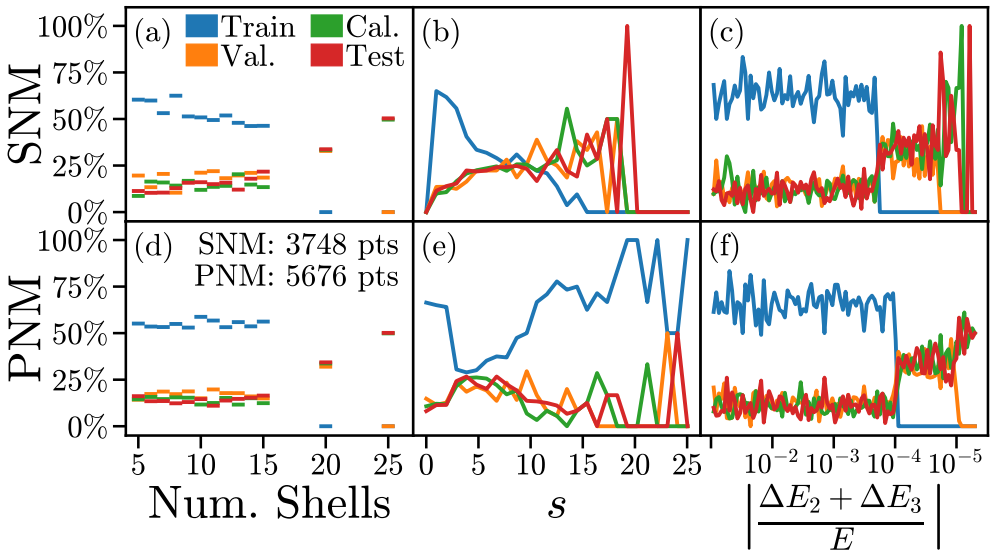

Figure 3.12 The stratified partitioning of the IMSRG samples used for training, validating, calibrating, and testing the PMM–IMSRG emulator. The training data (blue) is taken only from relatively unconverged samples with 15 or fewer closed shells. Validation data (orange) are drawn from up to 20 closed shells and include nearly-converged samples. No fully converged samples or data from the largest number of closed shells, 25, are present in the training or validation sets. The calibration (green) and testing (red) datasets span the full range of available IMSRG samples equally. All four datasets are entirely disjoint. Due to the comparative difficulty in obtaining IMSRG results for SNM, the number of SNM samples (3748) is roughly two-thirds that of PNM samples (5676). The number of closed shells is determined by the number of single-particle states and is shown here instead of N_s to allow better visualization between SNM and PNM [108]. Note that s was not used to partition the data as it does not perfectly correlate with convergence.

3.12.1.5 Training, Model Selection, and Calibration

The implementation, training, calibration, and deployment of the emulator is accomplished through the open-source PyPMM package[57] [1]. We train the emulator via gradient descent as discussed in Section 3.4. The training loss function is an annealed average of the weighted mean squared error (wMSE) and the variance of the weighted absolute error (wVAR). The wMSE over the training set T is given by

$$\mathrm{wMSE}(T) \equiv \underset{\left[(X_H;\, s);\, E_0^{\mathrm{IMSRG}}\right] \in T}{\mathrm{mean}} \frac{\left(E_0^{\mathrm{IMSRG}} - E_0^{\mathrm{PMM}}(X_H;\, s)\right)^2}{w(X_H)^2} \tag{3.117}$$

[57] I am currently both the lead and sole developer of PyPMM.

where $w(X_H)$ is the inverse square root of the proportion of the training set which is represented by the specific ν and N_s values in X_H,

$$w(X_H) \equiv \left(\frac{|T|}{|\{X'_H \in T | (\nu, N_s) = (\nu', N'_s)\}|} \right)^{1/2}, \tag{3.118}$$

where $|\cdot|$ denotes the size of a set. This weighting ensures that data taken at certain ν and N_s do not dominate the model training simply due to over- or under-representation in the training set. Similarly, the wVAR over the training set is given by

$$\text{wVAR}(T) \equiv \underset{\left[(X_H;\, s);\, E_0^{\text{IMSRG}}\right] \in T}{\text{var}} \frac{\left| E_0^{\text{IMSRG}} - E_0^{\text{PMM}}(X_H;\, s) \right|}{w(X_H)^2}. \tag{3.119}$$

The complete loss function used during training at Adam epoch t is then

$$\mathcal{L}(T) = \frac{\text{wMSE}(T) + \gamma(t)\text{wVAR}(T)}{1 + \gamma(t)}, \tag{3.120}$$

where $\gamma(t) \in [0, 1]$ is the annealing given by

$$\gamma(t) = \text{clip}\left(\frac{t - t_{\text{wMSE}}}{t_{\text{transition}}}, 0, 1 \right), \tag{3.121}$$

where t_{wMSE} is a hyperparameter which determines how many epochs of pure wMSE loss ($\gamma(t) = 0$) to with which to train the model before gradually—at a rate determined by the hyperparameter $t_{\text{transition}}$—transitioning to a loss composed of equal parts wMSE and wVAR ($\gamma(t) = 1$). This choice of loss function allows training to rapidly converge to an accurate but imprecise model before gradually encouraging a more precise, but still accurate, model. The validation loss, used both for quantifying training progress and comparing candidate models, is quantified by the top 95^{th} percentile of the absolute error on the validation set. This validation loss favors models with better worst-case performance over models that have good average performance but exhibit long tails in their error distributions.

The architecture (such as the choice for the universal functions), hyperparameters (such as the reduced-space dimensions n_1 and n_2), and loss function were chosen by a standard partial grid search. Several candidate models with different hyperparameters and data-driven function choices were trained and compared using the validation loss. The model with the

best validation loss across all trials was selected as the final emulator. A small amount of additional validation data was then created to manually verify the performance of the selected model. This additional data is not included in any of the aforementioned datasets. This model selection procedure led to $n_1 = 9$, $n_2 = 17$, and data-driven PMMs with mild smoothing (see Section 3.8) for f_j, α_i, and β_i. This yields a PMM emulator with fewer than 30 000 single-precision floating-point numbers.

We use the calibration dataset for uncertainty quantification and stratified bias corrections. Utilizing split-conformal predictions (Section 4.2) and the uncertainty heuristic described in Section 4.2.1, we are able to turn the emulator's point-wise predictions into rigorous confidence intervals for any desired significance level. The stratified bias corrections are fixed additive corrections to the model independent of the trainable parameters. These corrections ensure that the median error over the calibration set is exactly 0 for each of the four strata specified by the distinct values of ν and Y_p. While these corrections are beneficial, they are likely unnecessary as the largest relative correction—which occurred for SNM at NLO—was less than 0.055% of the range of the training data.[58,59]

3.12.1.6 Results

In this section we consider only the performance of the emulator in terms of accuracy and computational speedup. For applications of this emulator, see Ref. [142]. Figure 3.13 shows the accuracy of the emulator on the withheld test data at N^{2}LO and N^{3}LO. The test data includes both SNM and PNM, varies all considered LECs (c_D, c_E, D_2, F_2) and the nuclear density, and includes data of the most converged IMSRG calculations for the largest model sizes—neither of the final two were represented in the training or validation data. Each point in Fig. 3.13 shows the 95% confidence interval of the emulator-predicted energy versus the IMSRG calculation, demonstrating both high accuracy and precision.

[58] In the rescaled units where the training data energies belong to the range $[-1, 1]$, the largest correction was -0.00109.

[59] Once calculated, choosing whether or not to keep these corrections based on their values would constitute data leakage and introduce researcher bias—see Section 3.4.4—and thus these corrections remain part of the model despite their negligible contributions.

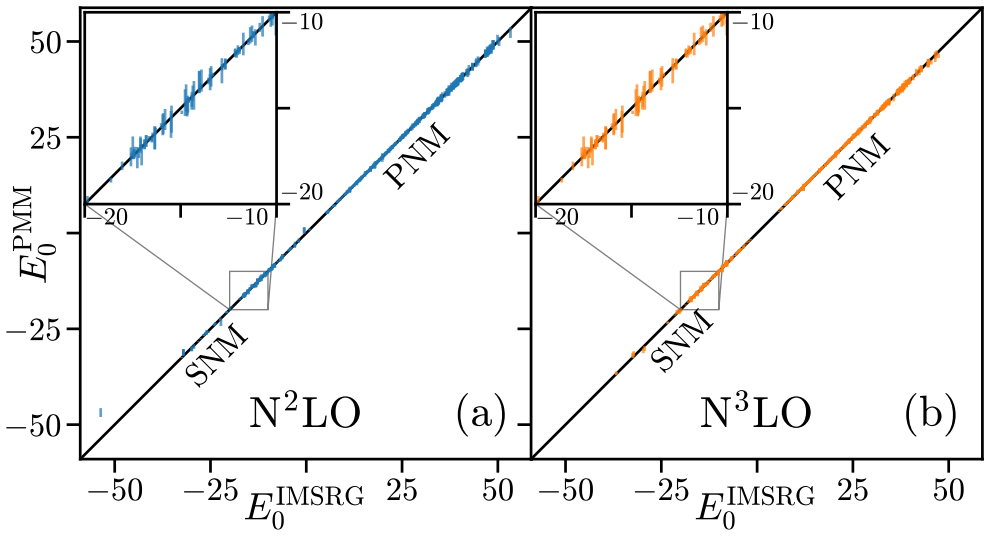

Figure 3.13 Comparison of PMM-predicted energies at the 95% confidence level and IMSRG calculations for the withheld test set for both PNM and SNM as all degrees of freedom, including LECs and density, are varied for both N^{2}LO (panel a) and N^{3}LO (panel b). Insets show additional detail around the empirical saturation energy. All values are in units of MeV.

We can quantify the trustworthiness of the UQ for this emulator by comparing the requested confidence level to the empirically observed coverage over the withheld test data. This is shown for all requested levels in Fig. 3.14. These results show that the emulator's UQ produces empirical coverages that track closely with the requested coverage. Importantly, for the withheld test data, at requested levels above 20% the maximum under-coverage is less than 0.35% and the maximum over-coverage is less than 2.5%. The UQ is biased slightly toward over-coverage, yielding slightly overly cautious uncertainty estimates—this is preferable to an overly confident model.

This emulator provides a computational speedup of about five orders of magnitude[60] compared to direct IMSRG calculations, enabling studies of the EOS such as Bayesian model calibration that otherwise would be intractable. Beyond raw speed, the emulator requires about six orders of magnitude less energy to perform a single prediction[61] and 300 times less memory to perform predictions.[62] Predictions can be made with the emulator on a

[60] IMSRG: up to 2 days vs. PMM: < 1 sec
[61] IMSRG: ~ 1 kWh vs. PMM: $< 10^{-6}$ kWh
[62] IMSRG: up to 600 GB vs. PMM: < 2 GB

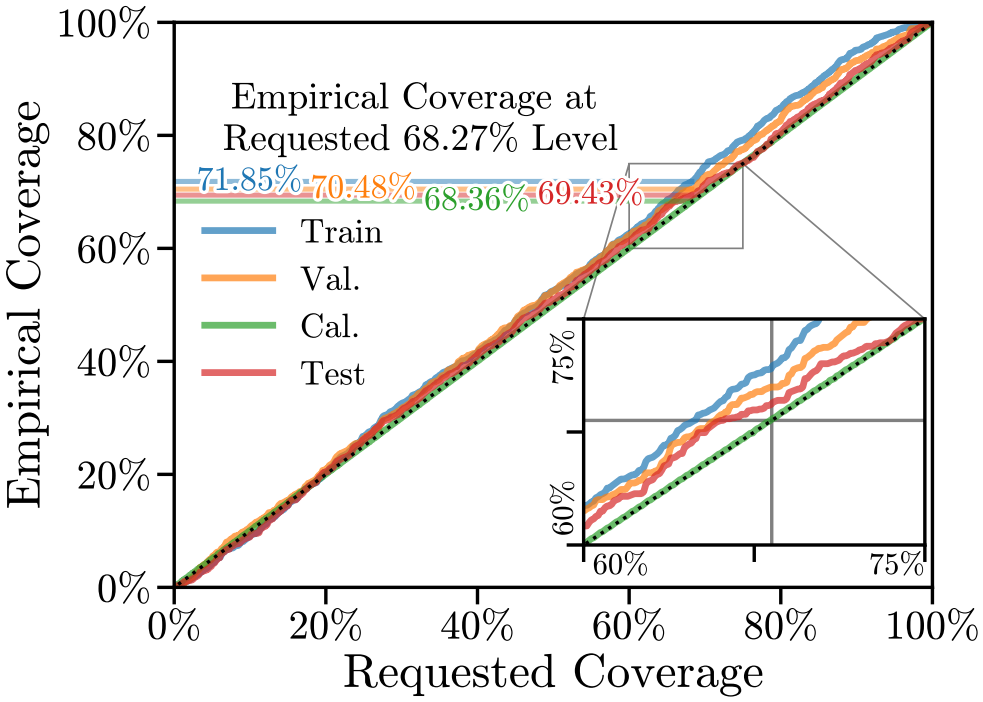

Figure 3.14 Actual coverage versus requested confidence level for the PMM emulator on each of the four data partitions. The observed coverages at the requested 68.27% level are shown as horizontal lines, and the coverage range from 60 to 75% is magnified in the inset to illustrate the small deviations from the ideal coverage.

low-power consumer laptop, compared to the expensive high-performance computing server required for IMSRG calculations. Even the offline costs—including those of time, electricity, and hardware—associated with training the model are reasonable and less than what was required to generate the IMSRG data.[63,64,65]

3.13 Summary

This chapter introduced parametric matrix models (PMMs) as an implicit reduced basis method (iRBM) for emulation with additional applications in general purely-data-driven machine learning. A comprehensive framework for PMMs was established, built on the central idea of allowing both the subspace and the basis of explicit reduced basis methods (eRBMs) to be implicit and instead fitting parameterized versions of reduced operators

[63]IMSRG: more than 1 week vs. PMM: 1 day
[64]IMSRG: ~ 1 MWh vs. PMM: < 6 kWh
[65]IMSRG: high-performance computing server vs. PMM: desktop computer with discrete graphics processing unit (GPU)

Method	Legend	Mathematically Parametric	Numerically Parametric	Nonlinear-Capable	Computationally Efficient	Data Efficient	Hyperparameter Efficient	Systematically Improvable	Interpretable	Physically Consistent	Intrusiveness	Quantified Uncertainties
	✓ Yes ✗ No ∼ Partial or conditional **I** Intrusive **N** Non-intrusive **H** Hybrid **A** Adaptive **E** Error bounds or estimates											
Galerkin												
Proper orthogonal decomposition		✓	✗	∼	✓	✓	✓	✓	✓	✓	**I**	**E**
Dynamic mode decomposition		∼	✗	∼	✓	✓	✓	∼	✓	✗	**N**	✗
Method in Section 2.11.1.3		✗	✗	✗	✓	✓	✓	✓	✓	✓	**I**	**E**
Eigenvector continuation		✓	✗	✗	✓	✓	✓	✓	✓	✓	**I**	**E**
Gaussian process regression		✓	✗	✓	✗	✗	∼	✗	∼	✗	**N**	✓
Physics-informed neural networks		✓	✓	✓	∼	∼	✗	∼	✗	∼	**H**	✗
Parametric matrix models		✓	✓	✓	✓	✓	✓	✓	✓	✓	**A**	**E**

Table 3.1 Same as Table 2.10 from the end of Chapter 2, but with the properties of parametric matrix models added.

directly to available data. Details for training PMMs as well as a guide to effective parameterization were provided. Applications of PMMs to the emulation of a variety of problems including standard parametric eigenproblems, linear partial differential equations, nonlinear eigenproblems, and nonlinear partial differential equations. A final application demonstrating the partially-data-driven-PMM emulation of in-medium similarity renormalization group (IMSRG) calculations of the nuclear equation of state (EOS).

CHAPTER 4

UNCERTAINTY QUANTIFICATION

A central motivation for emulation is quantifying the uncertainties of high-fidelity models, a process that without emulation would often be impractical or impossible due to the large number of model evaluations. For accurate and effective uncertainty quantification (UQ) of high-fidelity models via emulation, the emulator itself must be not only accurate but also have well-quantified uncertainties. Beyond this application, the trustworthiness of any machine learning method is a key—though sometimes overlooked—aspect that incorporates both accuracy and (un)certainty. Well-calibrated UQ can be the difference between an impressive method and a useful method. In this chapter, we briefly discuss two approaches to UQ in machine learning and emulation. The first method is often the "go-to" due to its relative simplicity. The second is particularly well-suited for UQ in parametric matrix model (PMM)-based emulators.

4.1 Bootstrap Aggregation

Bootstrap aggregation, or bagging,[1] is a specific case of the more general bootstrap and ensemble methods [158, 159]. This method is used to create an ensemble of models whose average predictions are more accurate than any individual constituent model while additionally providing uncertainty estimates via statistics over the ensemble predictions. In bootstrap aggregation, each model in the ensemble has the same architecture, training procedure, and loss function but is trained on a different random subset of the training data [119].

The core motivation for bagging is that if the error of an individual model is a random distribution centered near zero, and the errors of the individual models are uncorrelated, then on average these errors will cancel. In the worst possible case, where the error of each model is perfectly correlated, the ensemble will have the same error as any individual model, and so the bagging procedure will neither improve nor worsen the quality of the predictions. However, correlated errors will lead to overconfident uncertainty estimates,

[1] From "**b**ootstrap **agg**regat**ing**." [158]

which is a common failure mode of bagging. In practice, the errors of individual models are often correlated to some degree. Beyond UQ, bagging is often used as a form of regularization to reduce overfitting and improve generalization.

The process of bootstrap aggregation is as follows [119, 158]:

1. Given m training samples, choose a fixed model architecture, training procedure, loss function, and number of models K to include in the ensemble.

2. For each model $k \in \{1, 2, \ldots, K\}$:

    a) Randomly sample m training samples with replacement from the original training set to create a new training set for model k. This set will likely contain duplicates and also omit some of the original training samples.

    b) Train model k on this new training set using an independent random initialization of the model parameters (if applicable).

3. After training all K models, the ensemble prediction for a new input is some aggregation of the predictions of the individual models, such as the mean or median[2] of the individual predictions.

4. The uncertainty of the ensemble prediction can be estimated by computing the variance or standard deviation of the individual model predictions.

While both the training and prediction steps of bagging can be parallelized, the increased computational cost of training many models can be prohibitive. In the context of emulation, bagging also diminishes the interpretability of the emulator, as the ensemble predictions are no longer directly tied to the underlying model architecture.

[2]The aggregation method used is dependent on the type of prediction. For example the median or mean is common in regression tasks while the mode is common in classification tasks.

4.2 Conformal Prediction

Conformal prediction[3] is a framework that turns a pointwise prediction model into a set or interval prediction model with distribution-free finite-sample coverage guarantees that rely only on the exchangeability of data. Here, we consider only the core principle and process of split conformal prediction for regression. See Refs. [128, 160] for comprehensive introduction of conformal prediction.

The first core idea of split conformal prediction is that statistics of the predictions on a calibration set that is representative of the data the model will see in deployment should be nearly equal to the statistics of the predictions on the test set. A simplified example of this idea is that if the model achieves sub-1% error on the calibration set, then it should also achieve sub-1% error on the test set if the calibration set is representative of the test set. This division of calibration and test sets is where the "split" in split conformal prediction comes from. This relies on the assumption that the calibration and test sets are drawn from the same distribution and samples in each are therefore exchangeable.

The second, and more important, core principle is that of "conformalization." This is the process of transforming any heuristic uncertainty measure into a rigorous uncertainty measure [128]. The heuristic uncertainty measure, called the uncertainty heuristic $\mathcal{U}(\cdot)$, is a scalar-valued function that need only be monotonically related to the true uncertainty of the model. That is, the value of $\mathcal{U}(\cdot)$ should be larger for inputs where the model is less certain. The calibration set is used to calibrate—or conformalize—this heuristic into rigorous prediction intervals for any miscoverage rate $\alpha \in (0, 1)$.[4] Importantly, the conformalization process guarantees that the prediction intervals will have at least $1 - \alpha$ coverage on the calibration set (and thus the test set if exchangeability holds) regardless of the underlying model or uncertainty heuristic [127, 128].

Denoting the input features as X, the true output as $\hat{Y}$, and the trained model prediction

[3] Also called conformal inference.

[4] The miscoverage rate α is the probability that the true value is not contained in the prediction interval. The confidence level is $1 - \alpha$.

as $Y(X)$, the split conformal prediction process is broadly as follows [128]:

1. Devise the uncertainty heuristic $\mathcal{U}(X)$ which is a scalar-valued function that encodes some notion of uncertainty of the model prediction $Y(X)$.

2. Calculate the *score*—the residual divided by the uncertainty heuristic—for each sample in the calibration set
$$\mathcal{S}(X_i) = \frac{\left|Y(X_i) - \hat{Y}_i\right|}{\mathcal{U}(X_i)}. \tag{4.1}$$

3. For a desired miscoverage rate α (or confidence level $1 - \alpha$), define $\hat{q}$ as the
$$\frac{\lceil (1-\alpha)(n+1) \rceil}{n} \tag{4.2}$$
quantile of the calibration set scores $\{\mathcal{S}(X_i)\}$ where n is the number of samples in the calibration set and $\lceil \cdot \rceil$ is the ceiling function.

4. $\hat{q}$ is the multiplicative correction to the uncertainty heuristic which ensures that the prediction interval
$$[Y(X) - \hat{q}\mathcal{U}(X),\ Y(X) + \hat{q}\mathcal{U}(X)] \tag{4.3}$$
has at least $1 - \alpha$ coverage for any new input X.

Note that all of the above steps except for the final step can be performed as part of the offline phase. Only step 4 must be performed online for each new input X and so only requires the additional cost of the uncertainty heuristic evaluation and the multiplication by $\hat{q}$. For multi-dimensional outputs, the uncertainty heuristic is typically also multi-dimensional—matching the output dimension—and this process is applied element-wise to each output.

While the coverage of the intervals is guaranteed, the tightness of the intervals is reliant on the ability of the uncertainty heuristic to accurately reflect the true uncertainty of the model. Consider the case of a completely uninformative uncertainty heuristic. In this case $\hat{q}$ would be so large that every prediction interval would encompass the entire range of possible outputs. These intervals would have perfect coverage, as guaranteed, but would be useless

for UQ. It is therefore important to devise an uncertainty heuristic that accounts for all sources of uncertainty in the model prediction.

4.2.1 Uncertainty Heuristic for Parametric Matrix Models

The analyticity and adaptive smoothness (see Section 3.7.1) of explicit reduced basis method (eRBM)-based and data-driven PMMs make the incorporation of all sources of sensitivity and uncertainty into a single uncertainty heuristic relatively straightforward. For any PMM, we incorporate two sources of model sensitivity and one source of data uncertainty into the uncertainty heuristic. Denoting the predictions of the trained PMM as $Y(X)$, we include

1. The sensitivity of the trained PMM to perturbations in the trainable parameters,

$$S_\theta(X) = \frac{1}{\ell(\theta)} \left\| \left. \frac{\partial Y}{\partial \theta} \right|_X \odot \theta \right\|_2^2, \tag{4.4}$$

   where $\odot$ is the elementwise product, θ is the vector of all trainable parameters, and $\ell(\theta)$ is the number of trainable parameters. This term accounts for the relative sensitivity of the model to each of its trainable parameters.

2. The sensitivity of the trained PMM to perturbations in the continuous input features,

$$S_X(X) = \frac{1}{\ell(X_{\text{cont}})} \left\| \left. \frac{\partial Y}{\partial X_{\text{cont}}} \right|_X \odot \delta(X_{\text{cont}}) \right\|_2^2, \tag{4.5}$$

   where X_{cont} represents the subset of input features that are continuous, $\ell(X_{\text{cont}})$ is the number of continuous input features, and $\delta(X_{\text{cont}})$ is a vector of the characteristic scales of each continuous input feature, which is taken to be the interquartile range of each feature in the calibration set. This term accounts for the relative sensitivity of the model to each of its continuous input features.

3. The dissimilarity of the input X to the training data, $S_d(X)$. The dissimilarity is defined by the mean unweighted Gower's distance between X and all points in the training set within a distance threshold equal to the median unweighted Gower's distance between all pairs of points in the training set [161]. For the characteristic range of each

continuous feature in the calculation of Gower's distance, we again use the interquartile range. Gower's distance is a similarity metric that is useful for both continuous and discrete features simultaneously. The distance threshold drastically reduces the computational cost of making predictions for new inputs and simultaneously avoids the influence of outliers. Any input feature that has an infinite or undefined value is ignored in the calculation of Gower's distance. This term accounts for the uncertainty due to the model being applied to inputs that are dissimilar to the training data.

The uncertainty heuristic is then defined as the normalized sum of these three terms. Each term is normalized by its median absolute deviation (MAD) over the calibration set,

$$\underset{C}{\mathrm{MAD}}(S_i) = \underset{X \in C}{\mathrm{median}} \left| S_i(X) - \underset{X \in C}{\mathrm{median}}\, S_i(X) \right|, \tag{4.6}$$

where C is the calibration set, to ensure that each term contributes equally to the uncertainty heuristic. The final uncertainty heuristic is then

$$\mathcal{U}(X) = \frac{S_\theta(X)}{\underset{C}{\mathrm{MAD}}(S_\theta)} + \frac{S_X(X)}{\underset{C}{\mathrm{MAD}}(S_X)} + \frac{S_d(X)}{\underset{C}{\mathrm{MAD}}(S_d)}. \tag{4.7}$$

By computing the MAD of each term over the calibration set in the offline phase, as well as building a data structure for efficient nearest neighbor search for the dissimilarity term,[5] the online cost of evaluating the uncertainty heuristic is determined by the cost of evaluating the derivatives of the PMM which are computed via automatic differentiation (AD).

This uncertainty heuristic produces exceedingly well-calibrated prediction intervals for PMM-based emulators, as demonstrated in Section 3.12.1. Along with the native interpretability of PMM-based emulators, this UQ solidifies the trustworthiness of PMM-based emulators. The open-source pyPMM package—written as a part of this thesis—implements this uncertainty heuristic and the split conformal prediction process [1].

[5] Such as a k-d tree as described in Ref. [162] and provided in the SciPy library [163].

CHAPTER 5

CONCLUSION AND OUTLOOK

Nuclear and many-body physics today are limited by the computational resources available. The complexities of our modern models of these systems necessitate complicated and expensive algorithms to solve these models at the highest levels of accuracy. Simultaneously, these models have free parameters that must be fit to experimental observations, and uncertainties in both these free parameters and model predictions must be rigorously quantified. This all requires hundreds of thousands to millions of evaluations of these expensive models, which today is intractable. Fast and efficient, but also accurate and precise, surrogate models—or emulators—bridge this gap.

This thesis presented a slice of the current state of emulation in nuclear and many-body physics. After establishing why we need emulation and what we want from it, several popular methods were explored. Particular attention was paid to potentially the oldest technique, Galerkin's Method. It was shown that many emulation methods in use today can be interpreted as variations or special cases of Galerkin's Method—assisting in a loosely unified analysis. However, all such methods lack a desirable property: numerical parametricity. In contrast, the method of physics-informed neural networks (PINNs) is guaranteed this property, albeit at the cost of many others and potentially practical effectiveness altogether.

We introduced parametric matrix models (PMMs), an emulation and general machine learning (ML) technique predicated on the idea of introducing numerical parametricity directly into Galerkin's Method. The interpretation of PMMs as Galerkin's Method with implicit subspaces and bases allowed for a formal mathematical framework for analyzing PMM-based emulators as well as an explicit step-by-step process to constructing such emulators. We saw, through examples from simple linear systems to intricate models with missing information, that this method is maximally adaptable, exceedingly efficient, and controllably accurate.

As part of this thesis, I developed the open-source pyPMM package [1]. This pack-

age facilitates the construction, training, sharing, and deployment of PMMs in a modular, extendable, and graphics processing unit (GPU)-optimized framework thanks to the JAX package [116].

Research is currently ongoing across several groups to apply PMM-based emulation to ever-larger and more difficult problems. In particular, extensions of the emulator for the nuclear equation of state (EOS) in Section 3.12.1 to natively emulate in-medium similarity renormalization group (IMSRG) calculations of finite nuclei, multiple PMM emulators for various nuclear lattice chiral effective field theory (χEFT) calculations, and PMM emulation of large-scale nuclear reaction networks are directions all being pursued currently or in the immediate future. Although the method is mature enough now for research to be mainly focused on applications, there are many questions to be answered and algorithmic improvements to be made. For instance, the optimal pre-training initialization of PMMs parameters is currently unknown and could drastically reduce the cost of training. The extent to which purely-data-driven PMMs are competitive with modern artificial neural network (ANN) methods and what exactly their scaling laws are in these contexts is under active research. The answers to these questions may have impacts on partially-data-driven PMM-based emulators, which are perhaps the most powerful application of the method.

PMMs provide a unique, adaptive, and powerful tool for emulation in physics, enabling scientific progress to continue in the face of computational bottlenecks.

APPENDIX A

MATRIX-FREE OPERATORS

A key observation in many numerical methods used across scientific computing is that rarely are the individual matrix elements of linear operators necessary [65, 164–166]. Instead, often only the *action* of the operator is required. For an operator A and an element of the vector space in which it acts on, v, this is denoted by the usual left multiplication, Av. In numerical contexts, this is often given the more practical term of *matrix-vector products*. Many core software libraries for scientific computing already support the representation of operators via functions on vectors [65, 163]. Even when representing operators as matrices element-by-element, one must have considered the matrix-vector product to derive the matrix elements. In this sense, representing operators by their action is more natural.

When representing elements of a vector space numerically, the ubiquitous choice is to represent them as vectors (one dimensional arrays). Doing so necessitates the choice of a basis, which will also determine the representation of any operators—regardless of whether they are represented by explicit matrices or by their actions. Consider the standard quantum harmonic oscillator in one dimension. Choosing the position basis, the Hamiltonian is written as[1]

$$H = -\frac{1}{2m}\frac{\partial^2}{\partial x^2} + \frac{1}{2}m\omega^2 x^2, \tag{A.1}$$

and the state as $\Psi(x) \equiv \int \Psi(x)\delta(x-u)du$. In contrast, in the energy basis the Hamiltonian is written as

$$H = \omega\left(a^\dagger a + \frac{1}{2}\right), \tag{A.2}$$

and the state as $|\Psi\rangle \equiv \sum_{k=0} c_k|k\rangle$, where $k = 0, 1, \ldots$ and $a^\dagger a|k\rangle = k|k\rangle$. Note that both representations of the state vector are expressed as an integral (in the case of a continuous basis) or sum (in the case of a discrete basis) of some state-specific coefficients times the basis states. These coefficients are what we store programmatically to represent the states numerically, leaving the basis entirely implicit. In the first example, we would first approximate

[1] With $\hbar = 1$.

the continuous basis via discretization,

$$\begin{aligned}
&[-\infty, \infty] \approx [x_1, x_2, \ldots, x_N], \qquad x_k \equiv x_{\min} + (k+1)\Delta x, \\
&\implies \Psi(x) \approx \sum_k \underbrace{\Psi(x_k)}_{\text{coeff.}} \underbrace{s_k(x)}_{\text{basis st.}}, \\
&\text{where } s_k(x) \equiv \begin{cases} \frac{1}{\sqrt{\Delta x}} & x_k - \frac{\Delta x}{2} \leq x < x_k + \frac{\Delta x}{2} \\ 0 & \text{otherwise} \end{cases},
\end{aligned} \tag{A.3}$$

then numerically the one dimensional array representing the state would be the column vector

$$\Psi^{(x)} \doteq [\Psi(x_1)\ \Psi(x_2)\ \cdots\ \Psi(x_N)]^T. \tag{A.4}$$

Similarly, for the energy basis we would first truncate the summation so as to only retain the lowest N states,

$$|\Psi\rangle \approx \sum_{k=0}^{N-1} c_k |k\rangle, \tag{A.5}$$

and then store the vector of coefficients,

$$\implies \Psi^{(E)} \doteq [c_0\ c_1\ \cdots\ c_{N-1}]^T. \tag{A.6}$$

These approximations are necessary to represent what is fundamentally an infinitely large Hilbert space with finite computational resources. Note that in both cases, the basis states are orthonormal:

$$\begin{aligned}
\int s_i(x) s_j(x) dx &= \delta_{ij}, \\
\langle i | j \rangle &= \delta_{ij}.
\end{aligned} \tag{A.7}$$

From here, the usual process is to examine the "matrix elements" of the Hamiltonian in the given basis. For the position basis, we must additionally approximate the continuous derivative by a suitable finite-difference approximation,

$$\frac{\partial^2}{\partial x^2} f(x) \approx \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{(\Delta x)^2}. \tag{A.8}$$

Splitting Eq. (A.1) into a kinetic term $T^{(x)}$ and a potential term $V^{(x)}$, the matrix element of the kinetic energy term of Eq. (A.1) is

$$\begin{aligned} T_{ij}^{(x)} &\doteq -\frac{1}{2m}\int s_i(x)\frac{\partial^2}{\partial x^2}s_j(x)dx \\ &\approx -\frac{1}{2m(\Delta x)^2}\int s_i(x)[s_j(x+\Delta x)-2s_j(x)+s_j(x-\Delta x)]dx \\ &= -\frac{1}{2m(\Delta x)^2}[\delta_{i,\,j-1}-2\delta_{ij}+\delta_{i,\,j+1}], \end{aligned} \tag{A.9}$$

and the matrix element of the potential term is

$$\begin{aligned} V_{ij}^{(x)} &\doteq \frac{1}{2}m\omega^2\int s_i(x)x^2 s_j(x)dx \\ &= \delta_{ij}x_i^2. \end{aligned} \tag{A.10}$$

And so the Hamiltonian in this discrete position basis can be written as the tridiagonal matrix[2]

$$H^{(x)} \doteq -\frac{1}{2m(\Delta x)^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} + \frac{1}{2}m\omega^2\begin{bmatrix} x_1^2 & & & & \\ & x_2^2 & & & \\ & & x_3^2 & & \\ & & & \ddots & \\ & & & & x_N^2 \end{bmatrix}. \tag{A.11}$$

And likewise in the energy basis,

$$\begin{aligned} H_{nm}^{(E)} &\doteq \langle n|H|m\rangle \\ &= \omega\big\langle n\big|a^\dagger a\big|m\big\rangle + \frac{\omega}{2}\langle n|m\rangle \\ &= \omega\left(n+\frac{1}{2}\right)\delta_{nm} \\ \implies H^{(E)} &\doteq \omega\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & N-1 \end{bmatrix} + \frac{\omega}{2}I_{N\times N}, \end{aligned} \tag{A.12}$$

[2]We have ignored boundary conditions here, and therefore implicitly assumed Dirichlet boundary conditions where $\Psi(x)=0$ for $x < x_{\min}-\frac{1}{2}$ or $x \geq x_{\max}+\frac{1}{2}$, which is equivalent to placing the potential within an infinite well at the boundaries.

where $I_{N\times N}$ is the $N \times N$ identity matrix. This is the typical "dense-matrix" way one would go about implementing this system. These matrices multiply their respective numerical state vector representations in Eqs. (A.4) and (A.6)—despite only being computed from their multiplication on the basis vectors.

Consider instead describing this Hamiltonian in each of these bases by its action on the numerical state vector representations, Eqs. (A.4) and (A.6) directly. Each element of the result $H\Psi$ can only ever be a linear combination of the elements of Ψ. There are many ways to derive this action, one of which is to carry out the matrix-element derivation as before and reinterpret the results as an action. The more natural approach is to interpret the states and operators as one would for analytical, as opposed to numerical, expressions. For the position basis, we know the kinetic energy term is a scalar multiple of the second derivative which we approximate with finite differences,

$$\Psi(x) \xrightarrow{T} -\frac{1}{2m}\Psi''(x) \approx -\frac{1}{2m}\frac{\Psi(x+\Delta x) - 2\Psi(x) + \Psi(x-\Delta x)}{(\Delta x)^2}, \tag{A.13}$$

and similarly the potential energy term is just a multiplication by $m\omega^2 x^2/2$,

$$\Psi(x) \xrightarrow{V} \frac{1}{2}m\omega^2 x^2 \Psi(x). \tag{A.14}$$

Since the numerical representation of the state vector is simply the evaluation of this $\Psi(x)$ on a grid of positions with spacing Δx, we know element-by-element the result of applying either of these operators to an arbitrary coefficient vector,

$$\begin{aligned} \left(T^{(x)}\Psi^{(x)}\right)_i &= -\frac{1}{2m(\Delta x)^2}\left(\Psi^{(x)}_{i+1} - 2\Psi^{(x)}_i + \Psi^{(x)}_{i-1}\right), \\ \left(V^{(x)}\Psi^{(x)}\right)_i &= \frac{1}{2}m\omega^2 x_i^2 \Psi^{(x)}_i. \end{aligned} \tag{A.15}$$

Equation (A.15) is known as the *matrix-free* representation of the operators T and V in the position basis. Both of these expressions can be implemented efficiently with vectorized operations available in many libraries.[3] Note that acting on a standard basis vector reproduces

[3]For example—with periodic boundary conditions for simplicity—the action of the kinetic energy operator in the position basis can be written in PYTHON+NUMPY as the anonymous function `T = lambda v:  -(numpy.roll(v, +1) - 2 * v + numpy.roll(v, -1)) / (2 * m * dx**2)`.

the corresponding column of the dense representation,[4]

$$T\begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix} = -\frac{1}{2m(\Delta x)^2}\begin{bmatrix}-2\\1\\0\\\vdots\\0\end{bmatrix}, \qquad T\begin{bmatrix}0\\1\\0\\0\\\vdots\\0\end{bmatrix} = -\frac{1}{2m(\Delta x)^2}\begin{bmatrix}1\\-2\\1\\0\\\vdots\\0\end{bmatrix}. \tag{A.16}$$

Due to the energy basis being the basis in which the Hamiltonian is diagonal, expressing the action of the Hamiltonian in that basis is less involved,

$$\left(H^{(E)}\Psi^{(E)}\right)_n = \omega\left(n+\frac{1}{2}\right)\Psi_n^{(E)}. \tag{A.17}$$

However, the individual creation and annihilation operators, $a^\dagger$ and a, are more interesting, since

$$\begin{aligned} a|n\rangle &= \begin{cases} 0 & n=0 \\ \sqrt{n}\,|n-1\rangle & \text{otherwise} \end{cases} \\ a^\dagger|n\rangle &= \begin{cases} 0 & n=N-1 \\ \sqrt{n+1}\,|n+1\rangle & \text{otherwise} \end{cases}, \end{aligned} \tag{A.18}$$

for their actions on $|\Psi\rangle$ in Eq. (A.5) we have

$$\begin{aligned} |\Psi\rangle &\xrightarrow{a} \sum_{k=1}^{N-1} c_k\sqrt{k}\,|k-1\rangle = \sum_{k=0}^{N-2} c_{k+1}\sqrt{k+1}\,|k\rangle \\ |\Psi\rangle &\xrightarrow{a^\dagger} \sum_{k=0}^{N-2} c_k\sqrt{k+1}\,|k+1\rangle = \sum_{k=1}^{N-1} c_{k-1}\sqrt{k}\,|k\rangle, \end{aligned} \tag{A.19}$$

[4]Taking $\Psi_i = 0$ for $i < 0$ or $i > N$.

where the final expression for each makes it clear that the matrix-free representations are

$$\begin{aligned}\left(a\Psi^{(E)}\right)_n &= \begin{cases} 0 & n = N-1 \\ c_{n+1}\sqrt{n+1} & \text{otherwise} \end{cases}, \\ \left(a^\dagger\Psi^{(E)}\right)_n &= \begin{cases} 0 & n = 0 \\ c_{n-1}\sqrt{n} & \text{otherwise} \end{cases}. \end{aligned} \tag{A.20}$$

Again, these expressions can be implemented efficiently using vectorized operations.

Compared to the dense matrix representation,[5] matrix-free implementations can be significantly faster and less memory-intensive. For the examples above, the direct (naive) dense matrix approach would require $\mathcal{O}(N^2)$ multiplications per matrix-vector product and require the storage of $\mathcal{O}(N^2)$ elements for the operator.[6] Compared to the matrix-free implementations which, in these examples, all require $\mathcal{O}(N)$ multiplications for matrix-vector products and $\mathcal{O}(N)$ space for the operator.[7] Automatic differentiation (AD) is compatible with matrix-free implementations [167] and has been implemented for JAX in Ref. [168].

[5] As well as implementations which leverage sparsity.

[6] Non-naive sparse approaches would require $\mathcal{O}\big((1-s)N^2\big)$ space where s is the *sparsity* of the operator.

[7] The matrix-free representation of $T^{(x)}$ requires only $\mathcal{O}(1)$ space.

APPENDIX B

LONG-RANGE SPIN MODEL

The Lipkin–Meshkov–Glick (LMG) model,[1] named after the authors of the paper which introduced it [169], is a standard "toy model" in nuclear and many-body theory. Following the original description of the model, we consider L particles distributed between two L-fold degenerate energy levels. The separation between the energy levels is ε. The state of each particle can be described by the quantum numbers $\sigma \in \{-1, 1\}$ and $p \in \{1, 2, \ldots, L\}$. σ describes which of the two levels the particle occupies, $+1$ for the upper level and -1 for the lower level. p labels the individual particle. These particles are subject to an infinite-range two-body interaction which scatters pairs of particles between the two levels without changing p. This system can be described by the Hamiltonian

$$\begin{aligned} H = \frac{1}{2}\varepsilon \sum_{p\sigma} \sigma a^\dagger_{p\sigma} a_{p\sigma} + \frac{1}{2} V \sum_{pp'\sigma} a^\dagger_{p\sigma} a^\dagger_{p'\sigma} a_{p'(-\sigma)} a_{p(-\sigma)} \\ + \frac{1}{2} W \sum_{pp'\sigma} a^\dagger_{p\sigma} a^\dagger_{p'(-\sigma)} a_{p'\sigma} a_{p(-\sigma)}, \end{aligned} \tag{B.1}$$

where $a^\dagger_{p\sigma}$ and $a_{p\sigma}$ are the usual creation and annihilation operators for a single particle in the (p, σ) state, and V and W parameterize the strength of each type of interaction [169]. The first term is the energy contribution of the single particle states, here $-\varepsilon/2$ or $\varepsilon/2$. The second term represents the interaction that sends two different particles from the same level to the opposing level. The third term exchanges particles on different levels, sending one particle to the lower level from the upper level and the other to the upper level from the lower level. Up to an additive constant—which merely shifts the single particle energy levels—this is equivalent to describing the system as L distinguishable spin-1/2 particles under the re-parameterized Hamiltonian with infinite-range interactions,

$$H = -\frac{J}{L} \sum_{i<j}^{L} \left(\gamma_x \sigma^x_i \sigma^x_j + \gamma_y \sigma^y_i \sigma^y_j \right) - B \sum_{i}^{L} \sigma^z_i, \tag{B.2}$$

[1] Also commonly referred to simply as the Lipkin model.

where σ_k^u is the usual Pauli u operator for particle k, B parameterizes the single-particle energy levels, and J, γ_x, and γ_y parameterize the interaction.[2] To completely disambiguate the notation,

$$\begin{aligned} |\sigma_{L-1}, \ldots, \sigma_1, \sigma_0\rangle &\equiv |\sigma_{L-1}\rangle \otimes \cdots \otimes |\sigma_1\rangle \otimes |\sigma_0\rangle, \\ \sigma_k^u &\equiv \underbrace{I}_{L-1} \otimes \underbrace{I}_{L-2} \otimes \cdots \otimes \underbrace{\sigma^u}_{k} \otimes \cdots \otimes \underbrace{I}_{1} \otimes \underbrace{I}_{0}, \end{aligned} \tag{B.3}$$

which means

$$\sigma_k^u |\sigma_{L-1}, \ldots, \sigma_1, \sigma_0\rangle = (I|\sigma_{L-1}\rangle) \otimes \cdots \otimes (\sigma^u|\sigma_k\rangle) \otimes \cdots \otimes (I|\sigma_1\rangle) \otimes (I|\sigma_0\rangle). \tag{B.4}$$

This formulation bears striking resemblance to the Ising model, and in practice the LMG model is sometimes referred to as the "infinite-range Ising model". This individual-particle representation would require basis states described by L quantum numbers, $|\sigma_1\sigma_2\cdots\sigma_L\rangle$, which each could take on two possible values. Thus the Hilbert space would be of dimension 2^L.

Alternatively, it is more common to see the LMG model refer only to the Hamiltonian expressed in the quasi-spin basis,

$$H = \varepsilon J_0 + \frac{1}{2}V\left(J_+^2 + J_-^2\right) + \frac{1}{2}W(J_+J_- + J_-J_+), \tag{B.5}$$

where

$$J_+ = \sum_p a_{p+1}^\dagger a_{p-1}, \qquad J_- = \sum_p a_{p-1}^\dagger a_{p+1}, \qquad J_0 = \frac{1}{2}\sum_{p\sigma} \sigma a_{p\sigma}^\dagger a_{p\sigma}. \tag{B.6}$$

The basis states are then represented by just two quantum numbers,

$$\begin{aligned} J &\in \left\{ \frac{L}{2}, \frac{L}{2} - 1, \ldots, \begin{pmatrix} 0 \text{ if } L \text{ even} \\ \frac{1}{2} \text{ if } L \text{ odd} \end{pmatrix} \right\}, \\ J_0 &\in \left\{ -\frac{J}{2}, -\frac{J}{2} + 1, \ldots, \frac{J}{2} \right\}. \end{aligned} \tag{B.7}$$

The Hamiltonian does not change J since the operator $J^2 = \frac{1}{2}(J_+J_- + J_-J_+) + J_0^2$ commutes with the Hamiltonian. Because the ground state of the non-interacting ($V = 0$, $W = 0$)

[2] This is equivalent to the original representation of the LMG model with $\varepsilon = -2B$, $V = -J(\gamma_x - \gamma_y)/L$, and $W = -J(\gamma_x + \gamma_y)/L$—with an additional additive constant $J(\gamma_x + \gamma_y)/2$.

Hamiltonian has $J = L/2$—as the non-interacting ground state must have $J_0 = -L/2$—it is common to consider only the *sector* with $J = L/2$. This sector is completely described by only $L + 1$ basis states, significantly fewer than the full Hilbert space.

While the LMG model exhibits interesting phase-transition physics, it is fundamentally a mean-field model [170]. However, a beyond-mean-field generalization of Eq. (B.2) exists that exhibits much of the same interesting physics and is physically realizable on a quantum simulator [20]. By allowing the pairwise interactions to vary—therefore no longer necessarily being infinite range—one arrives at

$$H = -\frac{1}{\mathcal{J}} \sum_{i<j}^{L} J_{ij} \left(\gamma_x \sigma_i^x \sigma_j^x + \gamma_y \sigma_i^y \sigma_j^y \right) - B \sum_{i}^{L} \sigma_i^z, \tag{B.8}$$

where $\mathcal{J}$ is a normalization factor introduced to ensure a well-defined thermodynamic limit—called the Kac normalization factor—given by

$$\mathcal{J} = \frac{1}{L-1} \sum_{i \neq j}^{L} J_{ij}. \tag{B.9}$$

For a general interaction J_{ij}, the only reasonable choice of basis is the individual particle basis, usually specified by the z-projection of each spin, $|\sigma_1^z \cdots \sigma_N^z\rangle$.

Reference [20] includes details of the quantum-simulator implementation of this model. My contributions to Ref. [20] included predicting critical exponents for non-equilibrium critical phenomena and implementing numerical simulations of the experimental system.

Here we focus on the computational aspects of this model, specifically the matrix-free implementation of the Hamiltonian and relevant operators (see Appendix A). Since the dimensionality of the Hilbert space is 2^L, intractable for even modest L, either approximate methods[3] must be used or great care must be taken to implement the matrix-vector products as efficiently as possible.

A dense-matrix implementation of the Hamiltonian would require storing 4^L matrix elements, cost $\mathcal{O}\left(4^L\right)$ per matrix-vector product, and cost $\mathcal{O}\left(8^L\right)$ to compute the ground state

[3]Such as Density Matrix Renormalization Group (DMRG).

or time dynamics. Sparse methods are not easily applicable for an arbitrary J_{ij}. However, with a matrix-free implementation of the Hamiltonian combined with a suitable eigensolver or time evolution method (see Section 2.11.1.3) the memory cost can be reduced to $\mathcal{O}(1)$, and the matrix-vector product, ground state computation, and time dynamics costs can all be reduced to $\mathcal{O}\big(L^2 2^L\big)$. We start with a change in notation: Let $\sigma_i \in \{0, 1\}$, where 0 labels the $\langle \sigma_i^z \rangle = -1$ state and 1 labels the $\langle \sigma_i^z \rangle = +1$ state. Additionally, begin the indexing of the particles at 0. Now, consider the basis expansion of a general state,

$$\begin{aligned} |\Psi\rangle &= \sum_{\sigma_{L-1},\ldots,\sigma_1,\sigma_0 \in \{0,1\}} c_{\sigma_{L-1},\ldots,\sigma_1,\sigma_0} |\sigma_{L-1}, \ldots, \sigma_1, \sigma_0\rangle \\ &= c_{00\cdots00}|00\cdots00\rangle + c_{00\cdots01}|00\cdots01\rangle \\ &\qquad + c_{00\cdots10}|00\cdots10\rangle \\ &\qquad + \cdots \\ &\qquad + c_{11\cdots11}|11\cdots11\rangle, \end{aligned} \tag{B.10}$$

which strongly suggests the re-labeling of the basis by the base-10 number represented by the binary string of σ_i^z. For example, $|0011\rangle \to |0011_2\rangle = |3_{10}\rangle$. This leads us to mapping non-negative integers to the quantum numbers by $k = (\sigma_{L-1} \cdots \sigma_1 \sigma_0)_2$. Writing the general state in this way yields

$$|\Psi\rangle = \sum_k^{2^L} c_k |k\rangle, \qquad \text{where} \qquad |k\rangle = |(\sigma_{L-1} \cdots \sigma_1 \sigma_0)_2\rangle. \tag{B.11}$$

Not only is this a natural way to order the coefficients of the numerical representation of the state (c_k), but it is simultaneously a convenient basis for describing the action of the σ_k^u operators. Examine the action of each of the single-particle σ^u on the single-particle basis states $|0\rangle$ and $|1\rangle$,

$$\begin{aligned} \sigma^x|0\rangle &= |1\rangle, & \sigma^x|1\rangle &= |0\rangle, \\ \sigma^y|0\rangle &= -i|1\rangle, & \sigma^y|1\rangle &= i|0\rangle, \\ \sigma^z|0\rangle &= -|0\rangle, & \sigma^z|1\rangle &= |1\rangle. \end{aligned} \tag{B.12}$$

That is, σ^x "flips" the state, σ^y flips the state and multiplies by $\mp i$, and σ^z multiplies by ∓ 1. For the full L-particle state then, the actions of σ_k^u on the state $|n\rangle$ are

$$\begin{aligned}
\sigma_k^x|n\rangle &= \sigma_k^x|(\sigma_{L-1}\cdots\sigma_1\sigma_0)_2\rangle \\
&= |(\sigma_{L-1}\cdots\overline{\sigma_k}\cdots\sigma_1\sigma_0)_2\rangle \\
&= |n\oplus(1\ll k)\rangle, \\
\sigma_k^y|n\rangle &= \sigma_k^y|(\sigma_{L-1}\cdots\sigma_1\sigma_0)_2\rangle \\
&= (-1)^{\sigma_k+1}i|(\sigma_{L-1}\cdots\overline{\sigma_k}\cdots\sigma_1\sigma_0)_2\rangle \\
&= (-1)^{\sigma_k+1}i|n\oplus(1\ll k)\rangle, \\
\sigma_k^z|n\rangle &= \sigma_k^z|(\sigma_{L-1}\cdots\sigma_1\sigma_0)_2\rangle \\
&= (-1)^{\sigma_k+1}|(\sigma_{L-1}\cdots\sigma_k\cdots\sigma_1\sigma_0)_2\rangle \\
&= (-1)^{\sigma_k+1}|n\rangle,
\end{aligned} \tag{B.13}$$

where $\overline{b}$ is the negation of b,[4] $\oplus$ is the bitwise XOR operation,[5] and $a\ll b$ denotes the logical left-shift of a by b bits.[6] Bitwise operations like these are among the fastest operations for computers. In practice, $(-1)^{\sigma_k+1}$ is replaced with a much faster boolean statement,

$$(-1)^{\sigma_k+1}\equiv\begin{cases}-1 & \text{if } \sigma_k \\ +1 & \text{otherwise}\end{cases}. \tag{B.14}$$

Similar to the single-particle operators, we can interpret σ_k^x as flipping the k^{th} particle, σ_k^y as flipping the k^{th} particle and multiplying by $\mp i$, and σ_k^z as multiplying by ∓ 1. Extending this to $\sigma_i^u\sigma_j^u$ with $i\neq j$ and $u\in\{x,y\}$, which appear in Eq. (B.8), yields

$$\begin{aligned}
\sigma_i^x\sigma_j^x|n\rangle &= |n\oplus(1\ll i)\oplus(1\ll j)\rangle, \\
\sigma_i^y\sigma_j^y|n\rangle &= -(-1)^{\sigma_i+\sigma_j}|n\oplus(1\ll i)\oplus(1\ll j)\rangle,
\end{aligned} \tag{B.15}$$

[4] $\overline{0}=1$ and $\overline{1}=0$.
[5] For example, $3\oplus 2 = 11_2\oplus 01_2 = ((1\oplus 1)(1\oplus 0))_2 = 01_2 = 1$.
[6] For example $1\ll 3 = 0001_2\ll 3 = 1000_2 = 8$.

where $-(-1)^{\sigma_i+\sigma_j}$ is $+1$ if the bits at index i and j in the binary representation of n are different and -1 if they are the same.[7] Additionally for $\sum_i \sigma_i^z$,

$$\begin{aligned}\sum_i^L \sigma_i^z|n\rangle &= \sum_i^L \sigma_i^z|(\sigma_{L-1}\cdots\sigma_1\sigma_0)_2\rangle \\ &= -\left(\sum_i^L (-1)^{\sigma_i}\right)|n\rangle \\ &= (2w_H(n)-L)|n\rangle,\end{aligned} \tag{B.16}$$

where $w_H(n)$ is the number of ones in the binary representation of n, known as the Hamming weight of n.

We now have the action of all operators in the Hamiltonian on the basis states. Applying these to the general state vector, $|\Psi\rangle = \sum_k^{2^L} c_k|k\rangle$,

$$\begin{aligned}\sigma_i^x\sigma_j^x|\Psi\rangle &= \sum_k^{2^L} c_k|k\oplus(1\ll i)\oplus(1\ll j)\rangle, \\ \sigma_i^y\sigma_j^y|\Psi\rangle &= \sum_k^{2^L} -(-1)^{\sigma_i+\sigma_j}c_k|k\oplus(1\ll i)\oplus(1\ll j)\rangle, \\ \sum_i^L \sigma_i^z|\Psi\rangle &= \sum_k^{2^L}(2w_H(k)-L)c_k|k\rangle.\end{aligned} \tag{B.17}$$

We can then "reverse" these relations to get the action of the operators on the numerical representation of the state vector, $\Psi \doteq [c_0\, c_1\, \cdots\, c_{L-1}]^T$,

$$\begin{aligned}\left(\sigma_i^x\sigma_j^x\Psi\right)_k &= c_{k\oplus(1\ll i)\oplus(1\ll j)}, \\ \left(\sigma_i^y\sigma_j^y\Psi\right)_k &= -(-1)^{\sigma_i+\sigma_j}c_{k\oplus(1\ll i)\oplus(1\ll j)}, \\ \left(\sum_i^L \sigma_i^z\Psi\right)_k &= (2w_H(k)-L)c_k.\end{aligned} \tag{B.18}$$

Finally, combining these actions to form the action of the full Hamiltonian in Eq. (B.8) on

[7] A more efficient implementation to check if the bits at i and j are equal is checking the truth value of the expression $(n \gg i)\,\&\,1 = (n \gg j)\,\&\,1$ where & is the bitwise AND operation and $a \gg b$ denotes the logical right-shift of a by b bits. So $-(-1)^{\sigma_i+\sigma_j}$ would be written as $\begin{cases}-1 & \text{if } (n\ll i)\,\&\,1 = (n\ll j)\,\&\,1 \\ +1 & \text{otherwise}\end{cases}$.

the numerical state vector yields

$$\begin{aligned}(H\Psi)_k = &-\frac{1}{\mathcal{J}}\sum_{i<j}^{L} J_{ij}\big(\gamma_x - (-1)^{\sigma_i+\sigma_j}\gamma_y\big)c_{k\oplus(1\ll i)\oplus(1\ll j)} \\ &- B(2w_H(k) - L)c_k,\end{aligned} \tag{B.19}$$

or by using the more performant expression for the sign of the γ_y term,[8]

$$\begin{aligned}(H\Psi)_k = &-\frac{1}{\mathcal{J}}\sum_{i<j}^{L} J_{ij}\left[\gamma_x + \begin{pmatrix}-1 \text{ if } (k\gg i)\,\&\,1 = (k\gg j)\,\&\,1\\ +1 \text{ otherwise}\end{pmatrix}\gamma_y\right]c_{k\oplus(1\ll i)\oplus(1\ll j)} \\ &- B(2w_H(k) - L)c_k.\end{aligned} \tag{B.20}$$

Each term in the sum in this implementation for the action of the Hamiltonian requires only a handful of bitwise operations and array accesses which are all $\mathcal{O}(1)$. Since there are $\mathcal{O}(L^2)$ terms in the sum and $\mathcal{O}\big(2^L\big)$ elements in the numerical state vector, the complete action can be calculated in $\mathcal{O}\big(L^2 2^L\big)$ time, a significant improvement over the $\mathcal{O}\big(4^L\big)$ complexity of the dense-matrix implementation. Additionally, the Hamiltonian itself no longer requires the storage of any data besides the handful of parameters describing the interaction. Altogether, these efficiency improvements nearly double the achievable system size compared to the dense-matrix approach and allowed for the simulations of the experimental system in Ref. [20] up to $L = 25$ to be carried out on a low-power consumer-grade laptop.

[8]Note that, in either expression for the sign of the γ_y term, we are really checking to see if the i^{th} and j^{th} bits in $k \oplus (1 \ll i) \oplus (1 \ll j)$ are equal, but this is always equivalent to checking the same bits in k.

BIBLIOGRAPHY

[1] Patrick Cook. *pyPMM: The Python Package for Parametric Matrix Models*. Version 0.0.0+dev. June 2026. URL: https://github.com/Parametric-Matrix-Models/pyPMM (cit. on pp. ii, 144, 155, 156).

[2] E. Epelbaum, H.-W. Hammer, and Ulf-G. Meißner. "Modern theory of nuclear forces". In: *Rev. Mod. Phys.* 81 (4 Dec. 2009), pp. 1773–1825. DOI: 10.1103/RevModPhys.81.1773. URL: https://link.aps.org/doi/10.1103/RevModPhys.81.1773 (cit. on p. 3).

[3] K. Hebeler, S. K. Bogner, et al. "Improved nuclear matter calculations from chiral low-momentum interactions". In: *Phys. Rev. C* 83 (3 Mar. 2011), p. 031301. DOI: 10.1103/PhysRevC.83.031301. URL: https://link.aps.org/doi/10.1103/PhysRevC.83.031301 (cit. on p. 3).

[4] R. J. Furnstahl. "Turning the nuclear energy density functional method into a proper effective field theory: reflections". In: *The European Physical Journal A* 56.3 (Mar. 2020), p. 85. ISSN: 1434-601X. DOI: 10.1140/epja/s10050-020-00095-y. URL: https://doi.org/10.1140/epja/s10050-020-00095-y (cit. on p. 3).

[5] F. Marino, C. Barbieri, et al. "Nuclear energy density functionals grounded in ab initio calculations". In: *Phys. Rev. C* 104 (2 Aug. 2021), p. 024315. DOI: 10.1103/PhysRevC.104.024315. URL: https://link.aps.org/doi/10.1103/PhysRevC.104.024315 (cit. on p. 3).

[6] B. C. He and S. R. Stroberg. "Factorized approximation to the in-medium similarity renormalization group IMSRG(3)". In: *Phys. Rev. C* 110 (4 Oct. 2024), p. 044317. DOI: 10.1103/PhysRevC.110.044317. URL: https://link.aps.org/doi/10.1103/PhysRevC.110.044317 (cit. on p. 3).

[7] T. D. Morris, N. M. Parzuchowski, and S. K. Bogner. "Magnus expansion and in-medium similarity renormalization group". In: *Phys. Rev. C* 92 (3 Sept. 2015), p. 034331. DOI: 10.1103/PhysRevC.92.034331. URL: https://link.aps.org/doi/10.1103/PhysRevC.92.034331 (cit. on p. 3).

[8] Sota Yoshida. "IMSRG-Net: A machine-learning-based solver for the in-medium similarity renormalization group method". In: *Phys. Rev. C* 108 (4 Oct. 2023), p. 044303. DOI: 10.1103/PhysRevC.108.044303. URL: https://link.aps.org/doi/10.1103/PhysRevC.108.044303 (cit. on p. 3).

[9] T. Miyagi, S. R. Stroberg, et al. "Converged ab initio calculations of heavy nuclei". In: *Phys. Rev. C* 105 (1 Jan. 2022), p. 014302. DOI: 10.1103/PhysRevC.105.014302. URL: https://link.aps.org/doi/10.1103/PhysRevC.105.014302 (cit. on p. 3).

[10] N. Schunck, J. Dobaczewski, et al. "Solution of the Skyrme-Hartree-Fock-Bogolyubov equations in the Cartesian deformed harmonic-oscillator basis. (VII) HFODD (v2.49s): a new version of the program". In: *Comput. Phys. Commun.* 183 (2012), pp. 166–192. DOI: 10.1016/j.cpc.2011.08.013. arXiv: 1103.1851 [nucl-th] (cit. on p. 3).

[11] W. Ryssens. "Symmetry breaking in nuclear mean-field models". PhD thesis. Université Libre de Bruxelles, 2016. URL: http://hdl.handle.net/2013/ULB-DIPOT:oai:dipot.ulb.ac.be:2013/235692 (cit. on p. 3).

[12] Michael Bender, Rémi Bernard, et al. "Future of nuclear fission theory". In: *Journal of Physics G: Nuclear and Particle Physics* 47.11 (Oct. 2020), p. 113002. DOI: 10.1088/1361-6471/abab4f. URL: https://doi.org/10.1088/1361-6471/abab4f (cit. on p. 3).

[13] N. Schunck and L. M. Robledo. "Microscopic theory of nuclear fission: a review". In: *Reports on Progress in Physics* 79.11 (Oct. 2016), p. 116301. DOI: 10.1088/0034-4885/79/11/116301. URL: https://doi.org/10.1088/0034-4885/79/11/116301 (cit. on p. 3).

[14] Nicolas Schunck and David Regnier. "Theory of nuclear fission". In: *Progress in Particle and Nuclear Physics* 125 (2022), p. 103963. ISSN: 0146-6410. DOI: 10.1016/j.ppnp.2022.103963. URL: https://doi.org/10.1016/j.ppnp.2022.103963 (cit. on p. 3).

[15] Olga Russakovsky, Jia Deng, et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y (cit. on p. 4).

[16] J. A. Melendez, C. Drischler, et al. "Model reduction methods for nuclear emulators". In: *Journal of Physics G: Nuclear and Particle Physics* 49.10 (Sept. 2022), p. 102001. DOI: 10.1088/1361-6471/ac83dd. URL: https://doi.org/10.1088/1361-6471/ac83dd (cit. on p. 4).

[17] Michael Sipser. *Introduction to the Theory of Computation*. Third. Boston, MA: Course Technology, 2013. ISBN: 113318779X (cit. on p. 7).

[18] Volker Strassen. "Gaussian elimination is not optimal". In: *Numerische Mathematik* 13.4 (Aug. 1969), pp. 354–356. ISSN: 0945-3245. DOI: 10.1007/BF02165411. URL: https://doi.org/10.1007/BF02165411 (cit. on p. 8).

[19] Jared Kaplan, Sam McCandlish, et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361 (cit. on p. 11).

[20] Arinjoy De, Patrick Cook, et al. "Non-equilibrium critical scaling and universality in a quantum simulator". In: *Nature Communications* 16.1 (Aug. 2025), p. 7939. ISSN: 2041-1723. DOI: 10.1038/s41467-025-63398-y. URL: https://doi.org/10.1038/s41467-025-63398-y (cit. on pp. 11, 51, 166, 170).

[21] H. Nishimori and G. Ortiz. *Elements of Phase Transitions and Critical Phenomena*. Oxford Graduate Texts. OUP Oxford, 2011. ISBN: 9780199577224. URL: https://books.google.com/books?id=qiEUDAAAQBAJ (cit. on p. 11).

[22] Thomas Duguet, Andreas Ekström, et al. "Colloquium: Eigenvector continuation and projection-based emulators". In: *Rev. Mod. Phys.* 96 (3 Aug. 2024), p. 031002. DOI: 10.1103/RevModPhys.96.031002. URL: https://link.aps.org/doi/10.1103/RevModPhys.96.031002 (cit. on pp. 13, 15, 55, 56).

[23] Rahul Somasundaram, Cassandra L. Armstrong, et al. "Emulators for scarce and noisy data: Application to auxiliary field diffusion Monte Carlo for the deuteron". In: *Physics Letters B* 866 (2025), p. 139558. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2025.139558. URL: https://doi.org/10.1016/j.physletb.2025.139558 (cit. on p. 13).

[24] Wenqian Chen, Qian Wang, et al. "Physics-informed machine learning for reduced-order modeling of nonlinear problems". In: *Journal of Computational Physics* 446 (2021), p. 110666. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2021.110666. URL: https://doi.org/10.1016/j.jcp.2021.110666 (cit. on p. 14).

[25] B. G. Galerkin. "Rods and Plates, Series Occurring in Various Questions Concerning the Elastic Equilibrium of Rods and Plates". Russian. In: *Vestnik Inzhenerov i Tekhnikov (Engineers and Technologists Bulletin)* 19 (1915), pp. 897–908 (cit. on p. 17).

[26] W. J. Duncan. *Galerkin's method in mechanics and differential equations*. Research Report Reports and Memoranda No. 1798. National Advisory Committee for Aeronautics, United Kingdom, Aug. 1937. URL: https://reports.aerade.cranfield.ac.uk/handle/1826.2/1428 (cit. on p. 17).

[27] Mathematical Research, Inc. *Methods of the Galerkin Type*. Contractor Report NASA-CR-92216. Houston, Texas: National Aeronautics and Space Administration, United States, Jan. 1967. URL: https://ntrs.nasa.gov/api/citations/19680021795/downloads/19680021795.pdf (cit. on pp. 17, 19).

[28] Hans Petter Langtangen and Kent-Andre Mardal. *Introduction to Numerical Methods for Variational Problems*. Texts in Computational Science and Engineering. Springer Cham, 2019. ISBN: 978-3-030-23787-5. DOI: 10.1007/978-3-030-23788-2. URL: https://doi.org/10.1007/978-3-030-23788-2 (cit. on p. 17).

[29] A. C. Antoulas. "Special Topics in SVD-Based Approximation Methods". In: *Approximation of Large-Scale Dynamical Systems*. Advances in Design and Control. Society for Industrial and Applied Mathematics, 2009. Chap. 9. ISBN: 9780898716580. DOI: 10.1137/1.9780898718713.ch9. URL: https://books.google.com/books?id=HntlsoAR0NoC (cit. on pp. 18, 20).

[30] Clarence Rowley and Scott Dawson. "Model Reduction for Flow Analysis and Control". In: *Annual Review of Fluid Mechanics* 49 (Jan. 2017). DOI: 10.1146/annurev-fluid-010816-060042 (cit. on pp. 18, 20).

[31] Jean Céa. "Approximation variationnelle des problèmes aux limites". French. In: *Annales de l'Institut Fourier* 14.2 (1964), pp. 345–444. DOI: 10.5802/aif.181. URL: https://www.numdam.org/articles/10.5802/aif.181/ (cit. on p. 20).

[32] Stefan Volkwein Martin Kahlbacher. "Galerkin proper orthogonal decomposition methods for parameter dependent elliptic systems". eng. In: *Discussiones Mathematicae, Differential Inclusions, Control and Optimization* 27.1 (2007), pp. 95–117. URL: http://eudml.org/doc/271156 (cit. on p. 25).

[33] G Berkooz, P Holmes, and J L Lumley. "The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows". In: *Annual Review of Fluid Mechanics* 25.Volume 25, 1993 (1993), pp. 539–575. ISSN: 1545-4479. DOI: 10.1146/annurev.fl.25.010193.002543. URL: https://doi.org/10.1146/annurev.fl.25.010193.002543 (cit. on p. 25).

[34] Carl Eckart and Gale Young. "The Approximation of One Matrix by Another of Lower Rank". In: *Psychometrika* 1.3 (1936), pp. 211–218. DOI: 10.1007/BF02288367 (cit. on p. 26).

[35] L. Mirsky. "Symmetric Gauge Functions And Unitarily Invariant Norms". In: *The Quarterly Journal of Mathematics* 11.1 (Jan. 1960), pp. 50–59. ISSN: 0033-5606. DOI: 10.1093/qmath/11.1.50. eprint: https://academic.oup.com/qjmath/article-pdf/11/1/50/7295335/11-1-50.pdf. URL: https://doi.org/10.1093/qmath/11.1.50 (cit. on p. 26).

[36] C. Radhakrishna Rao. *Linear Statistical Inference and Its Applications*. 2nd ed. Wiley series in probability and statistics. Wiley Eastern, 1973. ISBN: 9780471708230. DOI: 10.1002/9780470316436. URL: https://books.google.com/books?id=wCvH0QEACAAJ (cit. on p. 26).

[37] C. Radhakrishna Rao. "Separation theorems for singular values of matrices and their applications in multivariate analysis". In: *Journal of Multivariate Analysis* 9.3 (1979), pp. 362–377. ISSN: 0047-259X. DOI: https://doi.org/10.1016/0047-259X(79)90094-0. URL: https://www.sciencedirect.com/science/article/pii/0047259X79900940 (cit. on p. 26).

[38] R. Bellman. *Introduction to Matrix Analysis: Second Edition*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1997. ISBN: 9780898713992 (cit. on p. 26).

[39] João Filipe Queiró. "On the interlacing property for singular values and eigenvalues". In: *Linear Algebra and its Applications* 97 (1987), pp. 23–28. ISSN: 0024-3795. DOI: https://doi.org/10.1016/0024-3795(87)90136-4. URL: https://www.sciencedirect.com/science/article/pii/0024379587901364 (cit. on p. 26).

[40] R. Schatten. *Norm Ideals of Completely Continuous Operators*. Ergebnisse der Mathematik und ihrer Grenzgebiete. 2. Folge. Springer Berlin Heidelberg, 2012. ISBN: 978-3-642-87654-7. DOI: 10.1007/978-3-642-87652-3. URL: https://doi.org/10.1007/978-3-642-87652-3 (cit. on p. 27).

[41] Saifon Chaturantabut and Danny C. Sorensen. "Discrete Empirical Interpolation for nonlinear model reduction". In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. 2009, pp. 4316–4321. DOI: 10.1109/CDC.2009.5400045 (cit. on pp. 27–29, 40).

[42] Hannah Lu and Daniel M. Tartakovsky. "Prediction Accuracy of Dynamic Mode Decomposition". In: *SIAM Journal on Scientific Computing* 42.3 (2020), A1639–A1662. DOI: 10.1137/19M1259948. eprint: https://doi.org/10.1137/19M1259948. URL: https://doi.org/10.1137/19M1259948 (cit. on p. 27).

[43] Boris Kramer and Karen E. Willcox. "Nonlinear Model Order Reduction via Lifting Transformations and Proper Orthogonal Decomposition". In: *AIAA Journal* 57.6 (2019), pp. 2297–2307. DOI: 10.2514/1.J057791. eprint: https://doi.org/10.2514/1.J057791. URL: https://doi.org/10.2514/1.J057791 (cit. on pp. 27, 29, 40).

[44] Cheng Huang, Jiayang Xu, et al. "Exploration of Reduced-Order Models for Rocket Combustion Applications". In: Jan. 2018. DOI: 10.2514/6.2018-1183 (cit. on p. 29).

[45] Chenjie Gu. "QLMOR: A Projection-Based Nonlinear Model Order Reduction Approach Using Quadratic-Linear Representation of Nonlinear Systems". In: *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 30.9 (Sept. 2011), pp. 1307–1320. ISSN: 0278-0070. DOI: 10.1109/TCAD.2011.2142184. URL: https://doi.org/10.1109/TCAD.2011.2142184 (cit. on pp. 29, 32).

[46] Danny Jammooa. "A Novel Framework for Machine Learning: Parametric Matrix Models and Their Application in Nuclear Physics, Scientific Computing, and General AI". PhD thesis. Michigan State University, 2025. DOI: 10.25335/drnc-5685. URL: https://doi.org/10.25335/drnc-5685 (cit. on p. 32).

[47] Mark H. Holmes. *Introduction to Scientific Computing and Data Analysis*. Texts in Computational Science and Engineering. Springer Cham, 2023. ISBN: 978-3-031-22430-0. DOI: 10.1007/978-3-031-22430-0. eprint: https://doi.org/10.1007/978-3-031-22430-0. URL: https://doi.org/10.1007/978-3-031-22430-0 (cit. on p. 33).

[48] Y. Lecun, L. Bottou, et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791 (cit. on p. 34).

[49] Jock Blackard. *Covertype*. UCI Machine Learning Repository. 1998. DOI: 10.24432/C50K5N (cit. on p. 34).

[50] Ben Hamner, dcthompson, and Jorg. *Predicting a Biological Response*. https://kaggle.com/competitions/bioresponse. Kaggle. 2012 (cit. on p. 34).

[51] J. Gamito, J. Khalouf-Rivera, et al. "Excited-state quantum phase transitions in the anharmonic Lipkin-Meshkov-Glick model: Static aspects". In: *Phys. Rev. E* 106 (4 Oct. 2022), p. 044125. DOI: 10.1103/PhysRevE.106.044125. URL: https://link.aps.org/doi/10.1103/PhysRevE.106.044125 (cit. on p. 34).

[52] Patrick Cook, Danny Jammooa, et al. "Parametric matrix models". In: *Nature Communications* 16.1 (July 2025), p. 5929. ISSN: 2041-1723. DOI: 10.1038/s41467-025-61362-4. URL: https://doi.org/10.1038/s41467-025-61362-4 (cit. on pp. 34, 69, 72, 73, 77, 82, 83, 115–117).

[53] Peter J. Schmid. "Dynamic mode decomposition of numerical and experimental data". In: *Journal of Fluid Mechanics* 656 (2010), pp. 5–28. DOI: 10.1017/S0022112010001217 (cit. on p. 41).

[54] Jonathan H Tu. "Dynamic mode decomposition: Theory and applications". PhD thesis. Princeton University, 2013 (cit. on p. 41).

[55] Quincy A. Huhn, Mauricio E. Tano, et al. "Parametric dynamic mode decomposition for reduced order modeling". In: *Journal of Computational Physics* 475 (2023), p. 111852. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2022.111852. URL: https://doi.org/10.1016/j.jcp.2022.111852 (cit. on pp. 41, 45).

[56] Francesco Andreuzzi, Nicola Demo, and Gianluigi Rozza. "A Dynamic Mode Decomposition Extension for the Forecasting of Parametric Dynamical Systems". In: *SIAM Journal on Applied Dynamical Systems* 22.3 (Aug. 2023), pp. 2432–2458. ISSN: 1536-0040. DOI: 10.1137/22m1481658. URL: http://dx.doi.org/10.1137/22M1481658 (cit. on pp. 41, 43, 45).

[57] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937. DOI: 10.1073/pnas.1517384113. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1517384113. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1517384113 (cit. on p. 42).

[58] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. 2nd ed. Cambridge University Press, 2022 (cit. on p. 42).

[59] Kathleen Champion, Bethany Lusch, et al. "Data-driven discovery of coordinates and governing equations". In: *Proceedings of the National Academy of Sciences* 116.45 (2019), pp. 22445–22451. DOI: 10.1073/pnas.1906995116. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1906995116. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1906995116 (cit. on p. 43).

[60] Nicholas Zolman, Christian Lagemann, et al. "SINDy-RL for interpretable and efficient model-based reinforcement learning". In: *Nature Communications* 16.1 (Nov. 2025), p. 10714. ISSN: 2041-1723. DOI: 10.1038/s41467-025-65738-4. URL: https://doi.org/10.1038/s41467-025-65738-4 (cit. on p. 43).

[61] Siva Viknesh, Younes Tatari, et al. "ADAM-SINDy: An efficient optimization framework for parameterized nonlinear dynamical system identification". In: *Phys. Rev. Res.* 8 (1 Jan. 2026), p. 013040. DOI: 10.1103/dwkk-5g2h. URL: https://link.aps.org/doi/10.1103/dwkk-5g2h (cit. on p. 43).

[62] Sara M. Ichinaga, Francesco Andreuzzi, et al. "PyDMD: A Python Package for Robust Dynamic Mode Decomposition". In: *Journal of Machine Learning Research* 25.417 (2024), pp. 1–9. URL: http://jmlr.org/papers/v25/24-0739.html (cit. on p. 43).

[63] Patrick Cook. *Tractable a priori Dimensionality Reduction for Quantum Dynamics*. 2025. arXiv: 2407.06340 [quant-ph]. URL: https://arxiv.org/abs/2407.06340 (cit. on pp. 46, 51).

[64] Cleve Moler and Charles Van Loan. "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later". In: *SIAM Review* 45.1 (2003), pp. 3–49. DOI: 10.1137/S00361445024180. eprint: https://doi.org/10.1137/S00361445024180. URL: https://doi.org/10.1137/S00361445024180 (cit. on p. 46).

[65] R.B. Lehoucq, D.C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics, 1998. ISBN: 9780898714074. URL: https://books.google.com/books?id=iMUea23N_CQC (cit. on pp. 49, 158).

[66] Gerard L. G. Sleijpen and Henk A. Van der Vorst. "A Jacobi-Davidson iteration method for linear eigenvalue problems". In: *SIAM Review* 42.2 (2000), pp. 267–293. ISSN: 0036-1445 (cit. on p. 49).

[67] Roman Geus. "The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems with application to the design of accelerator cavities". Diss., Technische Wissenschaften ETH Zürich, Nr. 14734, 2002. Doctoral Thesis. Zürich: ETH Zurich, 2002. DOI: 10.3929/ethz-a-004469464 (cit. on p. 49).

[68] Alan R. Tackett and Massimiliano Di Ventra. "Targeting specific eigenvectors and eigenvalues of a given Hamiltonian using arbitrary selection criteria". In: *Phys. Rev. B* 66 (24 Dec. 2002), p. 245104. DOI: 10.1103/PhysRevB.66.245104. URL: https://link.aps.org/doi/10.1103/PhysRevB.66.245104 (cit. on p. 49).

[69] Dillon Frame, Rongzheng He, et al. "Eigenvector Continuation with Subspace Learning". In: *Phys. Rev. Lett.* 121 (3 July 2018), p. 032501. DOI: 10.1103/PhysRevLett.121.032501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.121.032501 (cit. on pp. 55, 56).

[70] Avik Sarkar and Dean Lee. "Convergence of Eigenvector Continuation". In: *Phys. Rev. Lett.* 126 (3 Jan. 2021), p. 032501. DOI: 10.1103/PhysRevLett.126.032501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.126.032501 (cit. on pp. 55, 56).

[71] S. König, A. Ekström, et al. "Eigenvector continuation as an efficient and accurate emulator for uncertainty quantification". In: *Physics Letters B* 810 (2020), p. 135814. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2020.135814. URL: https://doi.org/10.1016/j.physletb.2020.135814 (cit. on pp. 55, 56).

[72] P. Demol, T. Duguet, et al. "Improved many-body expansions from eigenvector continuation". In: *Physical Review C* 101.4 (Apr. 2020). DOI: 10.1103/physrevc.101.041302. URL: https://doi.org/10.1103%2Fphysrevc.101.041302 (cit. on pp. 55, 56).

[73] R.J. Furnstahl, A.J. Garcia, et al. "Efficient emulators for scattering using eigenvector continuation". In: *Physics Letters B* 809 (2020), p. 135719. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2020.135719. URL: https://doi.org/10.1016/j.physletb.2020.135719 (cit. on pp. 55, 56).

[74] C. Drischler, M. Quinonez, et al. "Toward emulating nuclear reactions using eigenvector continuation". In: *Physics Letters B* 823 (2021), p. 136777. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2021.136777. URL: https://doi.org/10.1016/j.physletb.2021.136777 (cit. on pp. 55, 56, 136).

[75] Dong Bai and Zhongzhou Ren. "Generalizing the calculable R-matrix theory and eigenvector continuation to the incoming-wave boundary condition". In: *Phys. Rev. C* 103 (1 Jan. 2021), p. 014612. DOI: 10.1103/PhysRevC.103.014612. URL: https://link.aps.org/doi/10.1103/PhysRevC.103.014612 (cit. on pp. 55, 56).

[76] Avik Sarkar and Dean Lee. "Convergence of Eigenvector Continuation". In: *Phys. Rev. Lett.* 126.3 (2021), p. 032501. DOI: 10.1103/PhysRevLett.126.032501. arXiv: 2004.07651 [nucl-th] (cit. on pp. 55, 56).

[77] M Companys Franzke, Alexander Tichai, et al. "Excited states from eigenvector continuation: The anharmonic oscillator". In: *Physics Letters B* 830 (2022), p. 137101 (cit. on pp. 55, 56).

[78] Avik Sarkar. "Eigenvector Continuation: Convergence and Emulators". PhD thesis. Michigan State University, 2022. DOI: 10.25335/94pa-7819. URL: https://doi.org/doi:10.25335/94pa-7819 (cit. on pp. 55, 56).

[79] Dillon Kyle Frame. "*ab initio* simulations of light nuclear systems using eigenvector continuation and auxiliary field Monte Carlo". PhD thesis. Michigan State University, 2019. DOI: 10.25335/dprp-bx55. URL: https://doi.org/doi:10.25335/dprp-bx55 (cit. on pp. 55, 56).

[80] Sota Yoshida and Noritaka Shimizu. "Constructing approximate shell-model wavefunctions by eigenvector continuation". In: *Progress of Theoretical and Experimental Physics* 2022.5 (Apr. 2022), p. 053D02. ISSN: 2050-3911. DOI: 10.1093/ptep/ptac057. eprint: https://academic.oup.com/ptep/article-pdf/2022/5/053D02/43860928/ptac057.pdf. URL: https://doi.org/10.1093/ptep/ptac057 (cit. on p. 56).

[81] Nuwan Yapa and Sebastian König. "Volume extrapolation via eigenvector continuation". In: *Phys. Rev. C* 106 (1 July 2022), p. 014309. DOI: 10.1103/PhysRevC.106.014309. URL: https://link.aps.org/doi/10.1103/PhysRevC.106.014309 (cit. on p. 56).

[82] Nuwan Yapa, Kévin Fossez, and Sebastian König. "Eigenvector continuation for emulating and extrapolating two-body resonances". In: *Phys. Rev. C* 107 (6 June 2023), p. 064316. DOI: 10.1103/PhysRevC.107.064316. URL: https://link.aps.org/doi/10.1103/PhysRevC.107.064316 (cit. on p. 56).

[83] S. Wesolowski, I. Svensson, et al. "Rigorous constraints on three-nucleon forces in chiral effective field theory from fast and accurate calculations of few-body observables". In: *Phys. Rev. C* 104 (6 Dec. 2021), p. 064001. DOI: 10.1103/PhysRevC.104.064001. URL: https://link.aps.org/doi/10.1103/PhysRevC.104.064001 (cit. on p. 56).

[84] Xilin Zhang and R. J. Furnstahl. "Fast emulation of quantum three-body scattering". In: *Phys. Rev. C* 105 (6 June 2022), p. 064004. DOI: 10.1103/PhysRevC.105.064004. URL: https://link.aps.org/doi/10.1103/PhysRevC.105.064004 (cit. on p. 56).

[85] T. Djärv, A. Ekström, et al. "Bayesian predictions for $A = 6$ nuclei using eigenvector continuation emulators". In: *Phys. Rev. C* 105 (1 Jan. 2022), p. 014005. DOI: 10.1103/PhysRevC.105.014005. URL: https://link.aps.org/doi/10.1103/PhysRevC.105.014005 (cit. on pp. 56, 60).

[86] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Nov. 2005. ISBN: 9780262256834. DOI: 10.7551/mitpress/3206.001.0001. eprint: https://direct.mit.edu/book-pdf/2514321/book_9780262256834.pdf. URL: https://doi.org/10.7551/mitpress/3206.001.0001 (cit. on p. 61).

[87] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. "Kernel methods in machine learning". In: *The Annals of Statistics* 36.3 (June 2008), pp. 1171–1220. DOI: 10.1214/009053607000000677. URL: https://doi.org/10.1214/009053607000000677 (cit. on p. 61).

[88] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012 (cit. on p. 61).

[89] Thomas Beckers. *An Introduction to Gaussian Process Models*. 2021. arXiv: 2102.05497 [eess.SY]. URL: https://arxiv.org/abs/2102.05497 (cit. on p. 61).

[90] C. Drischler, R. J. Furnstahl, et al. "How Well Do We Know the Neutron-Matter Equation of State at the Densities Inside Neutron Stars? A Bayesian Approach with Correlated Uncertainties". In: *Phys. Rev. Lett.* 125 (20 Nov. 2020), p. 202702. DOI: 10.1103/PhysRevLett.125.202702. URL: https://link.aps.org/doi/10.1103/PhysRevLett.125.202702 (cit. on p. 61).

[91] A. C. Semposki, C. Drischler, et al. "From chiral effective field theory to perturbative QCD: A Bayesian model mixing approach to symmetric nuclear matter". In: *Phys. Rev. C* 111 (3 Mar. 2025), p. 035804. DOI: 10.1103/PhysRevC.111.035804. URL: https://link.aps.org/doi/10.1103/PhysRevC.111.035804 (cit. on p. 61).

[92] P. J. Millican, R. J. Furnstahl, et al. "Assessing correlated truncation errors in modern nucleon-nucleon potentials". In: *Phys. Rev. C* 110 (4 Oct. 2024), p. 044002. DOI: 10.1103/PhysRevC.110.044002. URL: https://link.aps.org/doi/10.1103/PhysRevC.110.044002 (cit. on p. 61).

[93] P. J. Millican, R. J. Furnstahl, et al. *Assessing Convergence Patterns Across Modern Nucleon-Nucleon Potentials*. 2025. arXiv: 2508.17558 [nucl-th]. URL: https://arxiv.org/abs/2508.17558 (cit. on p. 61).

[94] A. C. Semposki, R. J. Furnstahl, and D. R. Phillips. "Interpolating between small- and large-g expansions using Bayesian model mixing". In: *Phys. Rev. C* 106 (4 Oct. 2022), p. 044002. DOI: 10.1103/PhysRevC.106.044002. URL: https://link.aps.org/doi/10.1103/PhysRevC.106.044002 (cit. on p. 61).

[95] Amber Boehnlein, Markus Diefenthaler, et al. "Colloquium: Machine learning in nuclear physics". In: *Rev. Mod. Phys.* 94 (3 Sept. 2022), p. 031003. DOI: 10.1103/RevModPhys.94.031003. URL: https://link.aps.org/doi/10.1103/RevModPhys.94.031003 (cit. on p. 61).

[96] Motonobu Kanagawa, Philipp Hennig, et al. *Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences*. 2018. arXiv: 1807.02582 [stat.ML]. URL: https://arxiv.org/abs/1807.02582 (cit. on p. 61).

[97] Mikhail Belyaev, Evgeny V. Burnaev, and Yermek Kapushev. "Exact Inference for Gaussian Process Regression in case of Big Data with the Cartesian Product Structure". In: *arXiv: Methodology* (2014). URL: https://api.semanticscholar.org/CorpusID:88512895 (cit. on p. 62).

[98] Anis Fradi, Tien-Tam Tran, and Chafik Samir. "Decomposed Gaussian Processes for Efficient Regression Models with Low Complexity". In: *Entropy* 27.4 (2025). ISSN: 1099-4300. DOI: 10.3390/e27040393. URL: https://www.mdpi.com/1099-4300/27/4/393 (cit. on p. 62).

[99] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed. O'Reilly Media, 2022. ISBN: 9781098122461. URL: https://books.google.com/books?id=X5ySEAAAQBAJ (cit. on pp. 62, 92, 93, 96, 97, 117).

[100] Aston Zhang, Zachary C. Lipton, et al. *Dive into Deep Learning*. https://D2L.ai. Cambridge University Press, 2023 (cit. on pp. 62, 88–92, 96, 97, 117).

[101] George Em Karniadakis, Ioannis G. Kevrekidis, et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (June 2021), pp. 422–440. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00314-5. URL: https://doi.org/10.1038/s42254-021-00314-5 (cit. on p. 63).

[102] Hubert Baty. "A hands-on introduction to physics-informed neural networks for solving partial differential equations with benchmark tests taken from astrophysics and plasma physics". In: *arXiv preprint arXiv:2403.00599* (2024) (cit. on p. 63).

[103] Zhongkai Hao, Jiachen Yao, et al. "PINNacle: A comprehensive benchmark of physics-informed neural networks for solving pdes". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 76721–76774 (cit. on p. 65).

[104] Tamara G Grossmann, Urszula Julia Komorowska, et al. "Can physics-informed neural networks beat the finite element method?" In: *IMA Journal of Applied Mathematics* 89.1 (Jan. 2024), pp. 143–174. ISSN: 0272-4960. DOI: 10.1093/imamat/hxae011. eprint: https://academic.oup.com/imamat/article-pdf/89/1/143/58325885/hxae011.pdf. URL: https://doi.org/10.1093/imamat/hxae011 (cit. on p. 65).

[105] Nick McGreivy and Ammar Hakim. "Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations". In: *Nature Machine Intelligence* 6.10 (Oct. 2024), pp. 1256–1269. ISSN: 2522-5839. DOI: 10.1038/s42256-024-00897-5. URL: https://doi.org/10.1038/s42256-024-00897-5 (cit. on p. 65).

[106] Shiyi Chen and Gary D. Doolen. "Lattice Boltzmann Method For Fluid Flows". In: *Annual Review of Fluid Mechanics* 30 (1998), pp. 329–364. ISSN: 1545-4479. DOI: https://doi.org/10.1146/annurev.fluid.30.1.329. URL: https://www.annualreviews.org/content/journals/10.1146/annurev.fluid.30.1.329 (cit. on p. 71).

[107] Alberto Padovan, Blaine Vollmer, and Daniel J. Bodony. "Data-Driven Model Reduction via Non-intrusive Optimization of Projection Operators and Reduced-Order Dynamics". In: *SIAM Journal on Applied Dynamical Systems* 23.4 (2024), pp. 3052–3076. DOI: 10.1137/24M1628414. eprint: https://doi.org/10.1137/24M1628414. URL: https://doi.org/10.1137/24M1628414 (cit. on p. 75).

[108] Justin G. Lietz, Samuel Novario, et al. "Computational Nuclear Physics and Post Hartree-Fock Methods". In: *An Advanced Course in Computational Nuclear Physics: Bridging the Scales from Quarks to Neutron Stars*. Ed. by Morten Hjorth-Jensen, Maria Paola Lombardo, and Ubirajara van Kolck. Cham: Springer International Publishing, 2017, pp. 293–399. ISBN: 978-3-319-53336-0. DOI: 10.1007/978-3-319-53336-0_8. URL: https://doi.org/10.1007/978-3-319-53336-0_8 (cit. on pp. 79, 143, 144).

[109] Eric Douglas Flynn. "Nuclear Fission and Tunneling Phenomena". PhD thesis. Michigan State University, 2025. DOI: 10.25335/pdnw-zq48. URL: https://doi.org/10.25335/pdnw-zq48 (cit. on p. 79).

[110] C.J. Foot. *Atomic Physics*. Oxford Master Series in Physics. OUP Oxford, 2005. ISBN: 9780198506959. URL: https://books.google.com/books?id=kXYpAQAAMAAJ (cit. on p. 79).

[111] J. Pumplin, D. R. Stump, and W. K. Tung. "Multivariate fitting and the error matrix in global analysis of data". In: *Phys. Rev. D* 65 (1 Dec. 2001), p. 014011. DOI: 10.1103/PhysRevD.65.014011. URL: https://link.aps.org/doi/10.1103/PhysRevD.65.014011 (cit. on p. 84).

[112] Yann N Dauphin, Razvan Pascanu, et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/04192426585542c54b96ba14445be996-Paper.pdf (cit. on p. 86).

[113] W. Wirtinger. "Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen". In: *Mathematische Annalen* 97.1 (Dec. 1927), pp. 357–375. ISSN: 1432-1807. DOI: 10.1007/BF01447872. URL: https://doi.org/10.1007/BF01447872 (cit. on p. 87).

[114] Christoph Boeddeker, Patrick Hanebrink, et al. *On the Computation of Complex-valued Gradients with Application to Statistically Optimum Beamforming*. 2019. arXiv: 1701.00392 [math.NA]. URL: https://arxiv.org/abs/1701.00392 (cit. on pp. 87, 91).

[115] A. Hirose. *Complex-Valued Neural Networks: Advances and Applications*. IEEE Press Series on Computational Intelligence. Wiley, 2013. ISBN: 9781118590065. URL: https://books.google.com/books?id=B2_bucoS5VAC (cit. on p. 88).

[116] James Bradbury, Roy Frostig, et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.9.0+. 2018. URL: http://github.com/jax-ml/jax (cit. on pp. 88, 90, 91, 157).

[117] Jason Ansel, Edward Yang, et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. URL: https://docs.pytorch.org/assets/pytorch2-2.pdf (cit. on pp. 88, 90, 91).

[118] Martín Abadi, Ashish Agarwal, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on pp. 88, 90, 91).

[119] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, Cambridge, Massachusetts, 2016. URL: https://www.deeplearningbook.org/ (cit. on pp. 88, 92, 96, 117, 150, 151).

[120] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Preprint at https://arxiv.org/abs/1412.6980. 2017. URL: https://arxiv.org/abs/1412.6980 (cit. on p. 88).

[121] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0 (cit. on p. 90).

[122] Atilim Gunes Baydin, Barak A. Pearlmutter, et al. "Automatic Differentiation in Machine Learning: a Survey". In: *Journal of Machine Learning Research* 18.153 (2018), pp. 1–43. URL: http://jmlr.org/papers/v18/17-468.html (cit. on p. 90).

[123] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181 (cit. on p. 91).

[124] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. arXiv: 1211.5063 [cs.LG]. URL: https://arxiv.org/abs/1211.5063 (cit. on p. 91).

[125] P. Güttinger. "Das Verhalten von Atomen im magnetischen Drehfeld". German. In: *Zeitschrift für Physik* 73.3 (Mar. 1932), pp. 169–184. ISSN: 1434-601X. DOI: 10.1007/BF01351211. URL: https://doi.org/10.1007/BF01351211 (cit. on p. 91).

[126] Emil Hvitfeldt. *Feature Engineering A–Z*. https://github.com/EmilHvitfeldt/feature-engineering-az. 2024. URL: https://feaz-book.com/ (cit. on p. 93).

[127] V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic Learning in a Random World*. Springer US, 2005. ISBN: 9780387001524. URL: https://books.google.com/books?id=pEXc4C1ymakC (cit. on pp. 95, 152).

[128] Anastasios N. Angelopoulos and Stephen Bates. "Conformal Prediction: A Gentle Introduction". In: *Foundations and Trends in Machine Learning* 16.4 (Mar. 2023), pp. 494–591. ISSN: 1935-8237. DOI: 10.1561/2200000101. URL: https://doi.org/10.1561/2200000101 (cit. on pp. 95, 152, 153).

[129] Mario Lezcano-Casado and David Martínez-Rubio. *Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group*. 2019. arXiv: 1901.08428 [cs.LG]. URL: https://arxiv.org/abs/1901.08428 (cit. on p. 101).

[130] Jacob C Bridgeman and Christopher T Chubb. "Hand-waving and interpretive dance: an introductory course on tensor networks". In: *Journal of Physics A: Mathematical and Theoretical* 50.22 (May 2017), p. 223001. DOI: 10.1088/1751-8121/aa6dc3. URL: https://doi.org/10.1088/1751-8121/aa6dc3 (cit. on p. 106).

[131] S. Lang. *Algebra*. Graduate Texts in Mathematics. Springer New York, 2005. ISBN: 9780387953854. URL: https://books.google.com/books?id=Fge-BwqhqIYC (cit. on p. 108).

[132] L.D. Landau and E.M. Lifshitz. *Quantum Mechanics: Non-Relativistic Theory*. Course of theoretical physics. Butterworth-Heinemann, 1991. ISBN: 9780750635394 (cit. on p. 112).

[133] Michael Bender, Paul-Henri Heenen, and Paul-Gerhard Reinhard. "Self-consistent mean-field models for nuclear structure". In: *Rev. Mod. Phys.* 75 (1 Jan. 2003), pp. 121–180. DOI: 10.1103/RevModPhys.75.121. URL: https://link.aps.org/doi/10.1103/RevModPhys.75.121 (cit. on pp. 118, 119).

[134] J. Dobaczewski, A. Idini, et al. *TALENT Course: Density functional theory and self-consistent methods*. https://www.fuw.edu.pl/~dobaczew/TALENT_DFT_2016/main.pdf. TALENT Course https://fribtheoryalliance.org/TALENT/courses/course_04.php. Aug. 2016 (cit. on pp. 118, 119).

[135] Nicolas Schunck, ed. *Energy Density Functional Methods for Atomic Nuclei*. 2053-2563. IOP Publishing, 2019. ISBN: 978-0-7503-1422-0. DOI: 10.1088/2053-2563/aae0ed. URL: https://doi.org/10.1088/2053-2563/aae0ed (cit. on pp. 118, 119).

[136] Aurel Bulgac and Michael McNeil Forbes. "Time-dependent density functional theory". In: *Energy Density Functional Methods for Atomic Nuclei*. Ed. by Nicolas Schunck. 2053-2563. IOP Publishing, 2019, 4-1 to 4–34. ISBN: 978-0-7503-1422-0. DOI: 10.1088/2053-2563/aae0edch4. URL: https://doi.org/10.1088/2053-2563/aae0edch4 (cit. on p. 126).

[137] Shi Jin, Kenneth J. Roche, et al. "The LISE package: Solvers for static and time-dependent superfluid local density approximation equations in three dimensions". In: *Computer Physics Communications* 269 (2021), p. 108130. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2021.108130. URL: https://www.sciencedirect.com/science/article/pii/S0010465521002423 (cit. on p. 126).

[138] Aurel Bulgac, Piotr Magierski, et al. "Induced Fission of ^{240}Pu within a Real-Time Microscopic Framework". In: *Phys. Rev. Lett.* 116 (12 Mar. 2016), p. 122504. DOI: 10.1103/PhysRevLett.116.122504. URL: https://link.aps.org/doi/10.1103/PhysRevLett.116.122504 (cit. on p. 126).

[139] Antonio Bjelčić, Ibrahim Abdurrahman, and Kyle Godbey. *Angular and Kinetic Properties of Scission Neutrons within Time-dependent Density Functional Theory*. 2026. arXiv: 2606.09656 [nucl-th]. URL: https://arxiv.org/abs/2606.09656 (cit. on p. 126).

[140] E P Gross. "Structure of a quantized vortex in boson systems". In: *Nuovo Cimento* 20 (1961), pp. 454–477. DOI: 10.1007/BF02731494. URL: https://cds.cern.ch/record/343403 (cit. on p. 128).

[141] L P Pitaevskii. "Vortex lines in an imperfect Bose gas". In: *Sov. Phys. JETP* 13 (2 1961). Translated from Zh. Eksp. Teor. Fiz. 40, 646 (1961), pp. 451–454. URL: https://www.jetp.ras.ru/cgi-bin/e/index/e/13/2/p451?a=list (cit. on p. 128).

[142] Kang Yu, Patrick Cook, et al. "PMM–IMSRG emulator for the nuclear equation of state with quantified uncertainties". In preparation. 2026 (cit. on pp. 136, 137, 146).

[143] Agnieszka Sorensen et al. "Dense nuclear matter equation of state from heavy-ion collisions". In: *Prog. Part. Nucl. Phys.* 134 (2024), p. 104080. DOI: 10.1016/j.ppnp.2023.104080. arXiv: 2301.13253 (cit. on p. 136).

[144] Rajesh Kumar et al. "Theoretical and experimental constraints for the equation of state of dense and hot matter". In: *Living Rev. Rel.* 27.1 (2024), p. 3. DOI: 10.1007/s41114-024-00049-6. arXiv: 2303.17021 [nucl-th] (cit. on p. 136).

[145] Katerina Chatziioannou, H. Thankful Cromartie, et al. "Neutron stars and the dense matter equation of state". In: *Rev. Mod. Phys.* 97.4 (2025), p. 045007. DOI: 10.1103/ymsq-cfcw. arXiv: 2407.11153 (cit. on p. 136).

[146] Kshitij Agarwal, Johannes Jahan, et al. "Toward a Unified Understanding of the Dense Matter Equation of State". In: (Nov. 2025). arXiv: 2511.20378 (cit. on p. 136).

[147] Omokuyani Chibuzor Udiani. "Extending the In-Medium Similarity Renormalization Group to Nuclear Matter with Novel Insights on Unitary Coupled-Cluster Theory". PhD thesis. East Lansing, MI: Michigan State University, 2025. URL: https://d.lib.msu.edu/etd/52303 (cit. on pp. 136, 138).

[148] Omokuyani Udiani, Kang Yu, et al. "In-Medium Similarity Renormalization Group Calculations of Nuclear Matter". In preparation. 2026 (cit. on pp. 136, 138).

[149] C. Drischler, K. Hebeler, and A. Schwenk. "Chiral interactions up to next-to-next-to-next-to-leading order and nuclear saturation". In: *Phys. Rev. Lett.* 122.4 (2019), p. 042501. DOI: 10.1103/PhysRevLett.122.042501. arXiv: 1710.08220 (cit. on p. 137).

[150] D. R. Entem, R. Machleidt, and Y. Nosyk. "High-quality two-nucleon potentials up to fifth order of the chiral expansion". In: *Phys. Rev. C* 96.2 (2017), p. 024004. DOI: 10.1103/PhysRevC.96.024004. arXiv: 1703.05454 (cit. on p. 137).

[151] C.-J. Yang, A. Ekström, et al. "Power counting in chiral effective field theory and nuclear binding". In: *Phys. Rev. C* 103 (5 May 2021), p. 054304. DOI: 10.1103/PhysRevC.103.054304. URL: https://link.aps.org/doi/10.1103/PhysRevC.103.054304 (cit. on p. 137).

[152] K. Hebeler and A. Schwenk. "Chiral three-nucleon forces and neutron matter". In: *Phys. Rev. C* 82 (2010), p. 014314. DOI: http://dx.doi.org/10.1103/PhysRevC.82.014314. arXiv: 0911.0483 (cit. on p. 138).

[153] H. Hergert, S.K. Bogner, et al. "The In-Medium Similarity Renormalization Group: A novel ab initio method for nuclei". In: *Physics Reports* 621 (2016). Memorial Volume in Honor of Gerald E. Brown, pp. 165–222. ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2015.12.007. URL: https://www.sciencedirect.com/science/article/pii/S0370157315005414 (cit. on p. 138).

[154] Heiko Hergert, Scott K. Bogner, et al. "In-Medium Similarity Renormalization Group Approach to the Nuclear Many-Body Problem". In: *An Advanced Course in Computational Nuclear Physics: Bridging the Scales from Quarks to Neutron Stars*. Ed. by Morten Hjorth-Jensen, Maria Paola Lombardo, and Ubirajara van Kolck. Cham: Springer International Publishing, 2017, pp. 477–570. ISBN: 978-3-319-53336-0. DOI: 10.1007/978-3-319-53336-0_10. URL: https://doi.org/10.1007/978-3-319-53336-0_10 (cit. on p. 138).

[155] H. Hergert. "In-Medium Similarity Renormalization Group for Closed and Open-Shell Nuclei". In: *Phys. Scripta* 92.2 (2017), p. 023002. DOI: 10.1088/1402-4896/92/2/023002. arXiv: 1607.06882 (cit. on p. 138).

[156] S. Ragnar Stroberg, Scott K. Bogner, et al. "Non-Empirical Interactions for the Nuclear Shell Model: An Update". In: *Annu. Rev. Nucl. Part. Sci.* 69 (2019), pp. 307–362. DOI: 10.1146/annurev-nucl-101917-021120. arXiv: 1902.06154 (cit. on p. 138).

[157] Xin Zhen, Rongzhe Hu, et al. "Non-perturbative calculations of nuclear matter using in-medium similarity renormalization group". In: *Phys. Lett. B* 862 (2025), p. 139350. DOI: 10.1016/j.physletb.2025.139350. arXiv: 2501.02479 (cit. on p. 138).

[158] Leo Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (Aug. 1996), pp. 123–140. ISSN: 1573-0565. DOI: 10.1007/BF00058655. URL: https://doi.org/10.1007/BF00058655 (cit. on pp. 150, 151).

[159] B. Efron. "Bootstrap Methods: Another Look at the Jackknife". In: *The Annals of Statistics* 7.1 (1979), pp. 1–26. DOI: 10.1214/aos/1176344552. URL: https://doi.org/10.1214/aos/1176344552 (cit. on p. 150).

[160] Roel Hulsman. *Distribution-Free Finite-Sample Guarantees and Split Conformal Prediction*. 2022. arXiv: 2210.14735 [stat.ML]. URL: https://arxiv.org/abs/2210.14735 (cit. on p. 152).

[161] J. C. Gower. "A General Coefficient of Similarity and Some of Its Properties". In: *Biometrics* 27.4 (1971), pp. 857–871. ISSN: 0006341X, 15410420. URL: http://www.jstor.org/stable/2528823 (visited on 03/25/2026) (cit. on p. 154).

[162] Songrit Maneewongvatana and David M. Mount. *Analysis of approximate nearest neighbor searching with clustered point sets*. 1999. arXiv: cs/9901013 [cs.CG]. URL: https://arxiv.org/abs/cs/9901013 (cit. on p. 155).

[163] Pauli Virtanen, Ralf Gommers, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. on pp. 155, 158).

[164] Magnus R. Hestenes and Eduard Stiefel. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49 (1952), pp. 409–435. URL: https://api.semanticscholar.org/CorpusID:2207234 (cit. on p. 158).

[165] Dong C. Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical Programming* 45.1 (Aug. 1989), pp. 503–528. ISSN: 1436-4646. DOI: 10.1007/BF01589116. URL: https://doi.org/10.1007/BF01589116 (cit. on p. 158).

[166] D.A. Knoll and D.E. Keyes. "Jacobian-free Newton–Krylov methods: a survey of approaches and applications". In: *Journal of Computational Physics* 193.2 (2004), pp. 357–397. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2003.08.010. URL: https://www.sciencedirect.com/science/article/pii/S0021999103004340 (cit. on p. 158).

[167] Nicholas Krämer, Pablo Moreno-Muñoz, et al. "Gradients of functions of large matrices". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 49484–49518 (cit. on p. 163).

[168] Nicholas Krämer. *Matfree: Matrix-free linear algebra in JAX*. version v0.6.1. 2026. URL: https://github.com/pnkraemer/matfree (cit. on p. 163).

[169] H.J. Lipkin, N. Meshkov, and A.J. Glick. "Validity of many-body approximation methods for a solvable model: (I). Exact solutions and perturbation theory". In: *Nuclear Physics* 62.2 (1965), pp. 188–198. ISSN: 0029-5582. DOI: https://doi.org/10.1016/0029-5582(65)90862-X. URL: https://www.sciencedirect.com/science/article/pii/002955826590862X (cit. on p. 164).

[170] Paraj Titum and Mohammad F. Maghrebi. "Nonequilibrium Criticality in Quench Dynamics of Long-Range Spin Models". In: *Phys. Rev. Lett.* 125 (4 July 2020), p. 040602. DOI: 10.1103/PhysRevLett.125.040602. URL: https://link.aps.org/doi/10.1103/PhysRevLett.125.040602 (cit. on p. 166).