Physics-informed emulation
with applications in soft-tissue mechanics

David Dalton

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

SCHOOL OF MATHEMATICS AND STATISTICS

COLLEGE OF SCIENCE AND ENGINEERING

MARCH 2024

Abstract

The development of mathematical models in the past generation has been transformative for a number of scientific disciplines. Mathematical models allow for causal effects to be examined, relationships between variables to be quantified, and hypotheses to be explored through the generation of synthetic data. One example of particular relevance for this work is the field of soft-tissue mechanics, where recent developments in modelling theory and practice have the potential for application in quantitative personalised medicine.

The sophistication of modern mathematical models extends far beyond the limits of analytical tractability, which means that numerical physics *simulators* are required for solutions to be found. Computer simulation algorithms have also seen tremendous development in the past generation. However, challenges remain with traditional simulation approaches, particularly their high computational costs in certain cases. An alternative method which can alleviate some of these challenges is *emulation*, whereby the simulator is replaced by a cheaper, data-driven surrogate model. Emulation however has its own drawbacks, in particular lack of explicit incorporation of known physical laws. In this work we make use of a new generation of methods which incorporate aspects of both physics simulators and data-driven emulators, which we call *physics-informed emulation*.

We first consider a model describing the behaviour of the left-ventricle (LV) of the heart in diastole, performing emulation with a Graph Neural Network (GNN) surrogate model. In contrast to more traditional emulation approaches, a GNN allows for the exact computational mesh representation of the LV to be modelled, without any approximations. Numerical experiments are performed which demonstrate that our approach can achieve strong out of sample predictive accuracy, while offering massive savings over the simulator at prediction time. We next extend the GNN emulation framework to a range of soft-tissue mechanical models, involving varying constitutive laws, boundary conditions and geometries. Furthermore, the emulator is trained by application of the Principle of Minimum Potential Energy, which has the advantage that no simulation data is required for training. The possible significance of this work is in enabling soft-tissue mechanical

models to be deployed for real-time clinical decision support. Finally, we switch focus to consider how to integrate noisy observation data, linear PDE information and boundary conditions into a Gaussian process (GP) surrogate modelling framework. We demonstrate via theoretical analysis and numerical experiments that this integration can be done seamlessly and efficiently, and is especially useful for solving inverse problems.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank firstly my supervisors Prof. Dirk Husmeier and Dr. Hao Gao for their support, guidance and at times patience during the past four years. I also need to thank a number of researchers from the SofTMech group at Glasgow that I have been fortunate to work with, namely, Yuzhang Ge, Dr. Alan Lazarus and Dr. Arash Rabbani. Special thanks to Alan for his help during the first year of my PhD.

Finally, thanks to the University of Glasgow for funding my PhD studies.

Declaration

I declare that, except where explicit reference is made to the contribution of others, this thesis is the result of my own work and has not been submitted for any other degree at the University of Glasgow or any other institution. This work was carried out during my PhD studies under the supervision of Prof. Dirk Husmeier and Dr. Hao Gao. The contents of Chapters 2 and 3 have resulted in two journal publications, which are listed below. Additional publications I obtained which are not presented in this thesis include [1]–[8].

- Dalton, David; Gao, Hao; Husmeier, Dirk (2022) [9]. "Emulation of Cardiac Mechanics using Graph Neural Networks". In *Computer Methods in Applied Mechanics and Engineering*.

- Dalton, David; Husmeier, Dirk; Gao, Hao (2023) [10]. "Physics-Informed Graph Neural Network Emulation of Soft-Tissue Mechanics". In *Computer Methods in Applied Mechanics and Engineering*.

David Dalton

Chapter 1
Introduction

The objective of this chapter is to give the reader a general overview of both the historical context of the work presented in this thesis and some relevant material from the more recent literature. A less formal style is favoured here which emphasises conceptual understanding, with technical details kept to a minimum. All such details are instead explained comprehensively in separate methods sections presented in each subsequent chapter.

1.1 Mathematical Modelling

Mathematical modelling involves representing real-world systems or phenomena using equations or algorithms, with the goal of *abstracting* the potentially complex structure of the system under consideration [11]. Such abstractions have a number of potential advantages over purely empirical approaches, primarily the fact that they allow for the fundamental underlying principles driving the system to be elucidated and understood [12]. This is essential for modern scientific applications, where increasingly complex processes are being modelled [13]. In addition, mathematical models enable for the effect of different variables to be quantitatively explored, including any potentially intricate interaction effects. This allows for the formulation and testing of hypotheses in a manner which is not possible if only observational data were considered. The obvious disadvantage of this approach is that with too much abstraction, the model may be too coarse-grained to capture the essential structure of the real process. In this sense then, the *art* of modelling is to balance considerations between fidelity and practicality, and in turn allow for the construction of less wrong and more useful models, to paraphrase George Box [14].

A landmark moment in the development of mathematical modelling was Isaac Newton's derivation of the orbits of the planets in 1687 [15]. The behaviour of the celestial bodies has been the subject of universal human fascination for millennia, with cultures across the entire planet having carefully documented the motion of the stars, planets and comets during all of recorded history [16]. The apex of this type of analysis had come earlier in the 17^{th} century, with the publication by Johannes Kepler of his three laws for planetary motion [17]. Derived via careful examination of astronomical records, Kepler's laws accurately described the motion of the planets of the solar system. However, these laws were purely *empirical* - there was no underlying *causal* model explaining why the laws held. Newton's approach was much different, and it completely revolutionised the means by which the solar system could be understood. Coupling his famous three laws of mechanics with a theory of universal gravitation, Newton derived Kepler's empirical laws from first principles. A key advantage of Newton's approach is that it allowed for out-of-sample, data-independent predictions to be made. These predictions could then be compared with empirical observations, allowing for discrepancies to be highlighted for further investigation. A dramatic confirmation of the unique advantages of this approach came in the mid 19^{th} century, when astronomers identified a deviation in the observed orbit of Uranus with that predicted under Newton's mathematical model. This led some scientists to postulate that the discrepancy was due to the existence of a large mass in the solar system which had yet to be identified [18]. In 1846 Urbain Le Verrier was able to derive the location and mass of an object that would reconcile the predictions of the model with the astronomical observations. Later that year, the object was identified by telescope to within 1° of Le Verrier's prediction - we now call it the planet Neptune [19].

Various types of mathematical model exist, ranging from simple algebraic models like the classic Leontief input-output economic model [20], to game-theoretic [21] and agent based models [22]. In this work, however, we will be concerned primarily with mathematical models based on *differential equations* [23], which are ubiquitous across virtually all fields of science [24]. Differential equations (DEs) can be categorised into a range of different subtypes, including for example *ordinary* (ODEs), *partial* (PDEs), *stochastic* (SDEs) and *differential-algebraic* (DAEs). At their core, however, all differential equation models postulate a functional form for some differential operation of the process

of interest. This functional form is usually derived using physical principles such as the conservation of energy or momentum balance. A simple example is Newton's second law, which postulates a relationship between the force acting on an object and the acceleration (i.e. second derivative of position) it experiences. Other classic examples of differential equations include the harmonic oscillator, which describes the motion of a mass connected to a spring [25], the Lotka-Volterra equations for modelling the evolution of predator and prey populations [26], and finally the Schrödinger equation which governs the wave function of a quantum-mechanical system [27].

The complexity of the physical systems being modelled has dramatically increased since the pioneering work of Newton. For more complex systems, highly simplifying assumptions are in general not adequate to capture the behaviour of the true process. For example, Newton was able to derive highly accurate celestial orbits without having to account for the specific geometries of the planets and the sun - instead, he considered each as simply a point mass. By contrast, in the soft-tissue simulation models considered in this work, it is essential that patient specific geometries are accounted for to maintain model accuracy. This is because each geometry will be unique to the individual patient, and changes in the geometry affect the stress-strain behaviour of the body under loading [28]. While more sophisticated models can produce greater physical realism, they generally lack the analytical tractability of simpler models. For this reason, essential to the development of mathematical modelling in the past generation has been the contemporaneous development of computer engineering and algorithms, which allows for numerical *simulation* of sophisticated models to be performed.

1.2 Computer Simulation

Modern mathematical models provide a conceptual framework in terms of abstract equations and algorithms - it is only with computer simulation, however, that these abstractions are brought to life [29]. The main advantage of simulation methods is that they allow researchers to push their models beyond the limits of analytical tractability, and

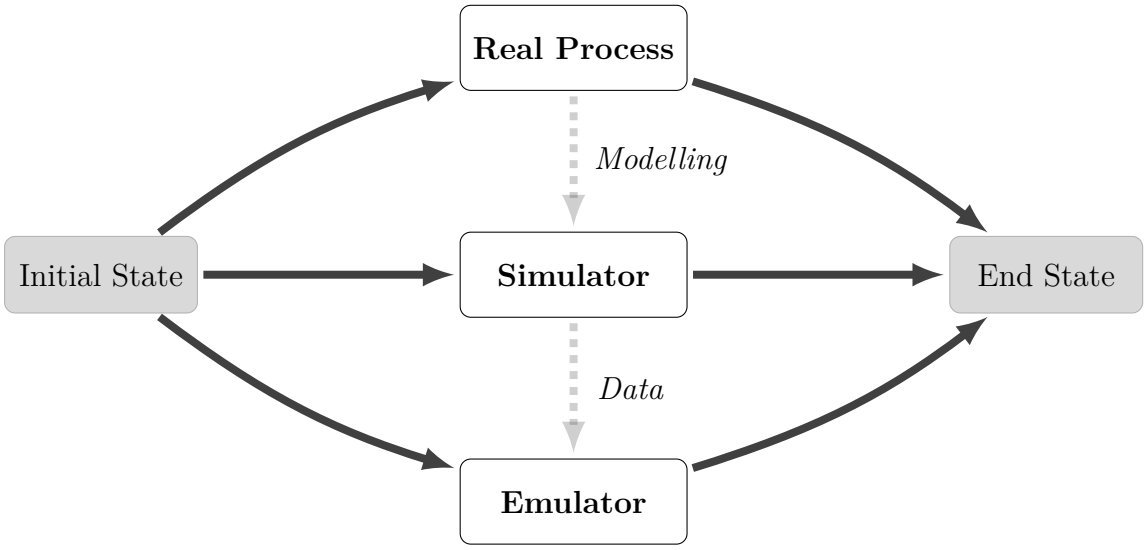


Figure 1.1: A real-world process maps a physical system from an initial state to an end state. While we can observe the states of the system, the functional form of the process itself is not known. The objective of mathematical modelling is to *model* and abstract properties of the physical-process. This in turn allows a computer simulator to be built which approximates the forward map of the real process. Finally, a black-box emulator is an approximation of the simulator, i.e. an approximation of an approximation. This is based purely on *data*, i.e. none of the modelling assumptions used to build the simulator are explicitly encoded into the emulator.

consider features of the system of interest that would not be possible otherwise, such as nonlinearities [30], multiscale interactions [31] and chaotic temporal dynamics [32]. This is illustrated in Figure 1.1, where, through modelling and abstraction, we can build a computer simulator which acts as a "digital twin" for the real-world process [33].

Contemporary computer simulation methods have as their foundation three complementary disciplines that have seen massive development in the past decades, namely; computer *hardware*, computer *software*, and numerical simulation *algorithms*.

The earliest Turing-complete digital electronic computers include the ENIAC (1945) [34] and UNIVAC (1951) [35]. Even at this early stage, the primary focus of these computers was in the simulation of physical processes, such as nuclear physics [36, Chapter 13], weather forecasting [37], and fluid dynamics [38]. By the 1960s and 1970s, computers became more widespread as costs decreased and accessibility increased, for example with the development of graphical user interfaces (GUIs) [39]. In turn, computer simulation became more widely applied in the hard sciences [40]. The 1990s and 2000s saw the

advent of affordable and powerful desktop computers, which in turn assisted the development of computer simulation into new fields, including biology [41] and the social sciences [42]. These developments have only grown in the past decade, aided by the increasing availability of high performance computer clusters and cloud computing platforms [43].

Advances in computer hardware have been coupled with commensurate advances in the ease with which computers can be programmed to perform numerical simulations. Programming the earliest computers was a laborious process, where punchcards and later switchboards were required [44]. Higher level computer *languages* gradually became available however which lowered the entry level required to program computers, most notably FORTRAN (1957) [45], C (1972) [46] and C++ (1979) [47]. This has culminated today in a multitude of modern languages and frameworks which make it convenient to write efficient, differentiable programs which run not only on central processing units (CPUs), but also graphics processing units (GPUs) and tensor processing units (TPUs). An example of this is the excellent Python library JAX for differentiable and jit compilable code [48], which is used to perform all numerical experiments presented in this work.

Finally, the past several decades have also seen massive developments in the quality and scale of numerical algorithms used to perform computer simulations. An early example is the Markov-chain Monte-Carlo (MCMC) method, which allows intractable integral equations to be solved approximately by using random samples drawn from a Markov process. Originally developed for use in nuclear physics [49], [50], this approach has now revolutionised the practice of Bayesian statistical inference [51]. For systems described by differential equations, a wide range of algorithms have been developed that facilitate computer simulation. This includes the finite-element method (FEM) [52], finite-difference method (FDM) [53], finite-volume method (FVM) [54], and material-point method (MPM) [55], to give some examples. The common theme in all these methods is in discretising a set of continuous model equations, which can then be naturally handled on a digital computer. In this work, all simulations are performed using the FEM, which is discussed in detail in Chapter 3.

Traditional numerical simulation approaches such as the FEM have many advantages; including rigorous mathematical foundations [56], suitability for complex application domains [57], and wide availability of user-friendly software for practical implementation [58]. There are however some drawbacks. For example, it can be difficult to incorporate noisy observational data [59], and also the computational expense can be very high for simulating more complex systems [60]. In this work, we will consider an alternative approach called *emulation* that can alleviate these issues.

1.3 Emulation

Emulation is a technique for approximating a computationally expensive computer simulator with a cheaper machine learning model [61]. The objective is to create a surrogate that can accurately replicate the behaviour of the simulator, but which is significantly more efficient at prediction time, enabling experiments and analyses to be performed that would otherwise be too expensive in reasonable time frames. Additionally, an emulator can naturally accommodate noisy observations of the real-world process itself, which can be difficult for a traditional numerical simulator [59].

The pioneering work of Kennedy and O'Hagan (2001) [62] is now seen as seminal in the emulation community, where it was outlined how a class of stochastic processes called *Gaussian processes* (GPs) could be used as surrogates for costly computer simulation models. GPs are described in detail in Chapter 4. At first glance, this approach may seem counter-intuitive, as it means assuming that the *deterministic* simulator follows a *stochastic* process. However, this is exactly the Bayesian paradigm for performing inference, whereby all variables about which we are uncertain are assigned a prior probability measure. The beauty of the GP formalism is that it allows us to place a probability measure directly on a function space, which is an appealing approach for learning the functional form (i.e. the input-output map) of a computer simulator. Closed form expressions exist for then updating the prior to a posterior given observation data. Once the GP emulation framework has been presented, the paper discusses techniques for calibration and validation of the surrogate model, before presenting several applications, including for example

environmental and engineering systems. More recently, the work of Gramacy (2020) [63] will likely prove seminal for the next generation of researchers in the field, in which a systematic illustration of the power of emulation using GPs is given, for tasks such as prediction, uncertainty quantification, optimisation and experimental design.

In order for an emulator to be useful in practice, it must be able to generalise beyond its training sample. For soft-tissue mechanics, it is essential that an emulator can generalise to a new geometry (e.g. heart geometry), as this will vary from subject to subject. Because of the inherent dimensionality of a 3D soft-tissue body (generally represented with a computational mesh with thousands of nodes) this is difficult for a traditional emulation approach, as they are typically applied to lower dimensional problems. To overcome this issue, we make use of a graph neural network (GNN) emulator for soft-tissue mechanics in Chapters 2 and 3, which can generalise to new geometries. One problem with using a more complex emulation strategy, such as a GNN, is a corresponding loss in interpretability of its predictions. While this was not an objective of this thesis, it has been the subject of extensive research in recent years. Proposed GNN interpretation methods include CF-GNNExplainer [64], GNNInterpreter [65] and PGExplainer [66]. For a comprehensive discussion of this topic, we direct the reader to the survey paper [67].

The traditional approach to emulation based on GPs treats the underlying simulator as a *black-box* function, which maps inputs to outputs. This approach has some drawbacks, however. First and foremost, it requires the generation of a large data set from the simulator for training, which will be expensive to obtain. Clearly, it would be preferable if emulator training could be performed *without* requiring a simulation data set. Furthermore, only through the simulation data itself is information about the underlying system communicated to the emulator. In other words, prior knowledge about the system is not systematically incorporated into the surrogate model, but instead must be learned from the simulation data. This is illustrated in Figure 1.1, where the dashed line indicates that the emulator is related to the simulator only via a data set, which acts as a bottleneck through which all modelling assumptions and abstractions used to build the simulator must be learned implicitly.

Intuitively, an alternative modelling approach which combines the benefits of traditional emulation, such as computational efficiency, with the benefits of computer simulation methods, like explicit handling of known physical properties, seems appealing. In fact, models of this type have seen tremendous development during the past half-decade, in the rapidly growing discipline of *physics-informed machine learning* - for details, see Section 1.6.

1.4 Model validation

An essential component of both mathematical modelling and computer simulation is *validation*, that is evaluating how well the model represents the real-world process by identifying inconsistencies between the predicted and observed results [68]. One possibility for an inconsistency is that the model is (approximately) correct, and there is an issue with the observational data. The classic example of this was described in Section 1.1, where the predicted and observed orbits of Uranus conflicted because Neptune had not been identified and accounted for in the mathematical model. Alternatively, it could be the case that the model itself is not an accurate abstraction for the true process. A classic illustrative example of this is again provided by Newton's model of celestial mechanics. After the successful location of Neptune, astronomers began in earnest to identify more inconsistencies between the model's predictions and astronomical observations, in the hope that further celestial bodies could be found [69]. One source of inconsistency was in the orbit of Mercury, which Le Vellier himself predicted was due once more to the existence of another planet, which became known as Vulcan [70]. Vulcan was never found, however [69]. The reason was that, in this case, missing data was not the cause of the discrepancy - instead the *model itself* was incorrect [71]. This was not clear until the beginning of the following century with the celebrated work of Einstein on general relativity [72], which showed that the discrepancies in Mercury's orbit can be explained by the curvature of spacetime due to the Sun's mass [69].

This example illustrates another important aspect of mathematical modelling, which is that the *domain of validity* for a model should be clearly defined. For instance, it is now understood that Newton's three laws of motion are not universally valid, as for objects at atomic level, the laws of quantum mechanics hold [73]. Similarly, while Newton's theory of universal gravitation holds for many practical purposes, it breaks down for velocities close to the speed of light and/or in the presence of large gravitational potential, where it is superseded by Einstein's special and general theories of relativity [74].

Unfortunately, these important topics are beyond the scope of this thesis. However, relevant ideas are implicit in some of the work we present. For instance, in Chapter 2 we build an emulator for a simulation model of the LV in diastole, assuming a material model for the cardiac tissue called the Holzapfel-Ogden (HO) model, which has been validated in *ex vivo* mechanical studies [75]. Because of this, we appeal to transitivity to assert that if the emulator accurately replicates the results of the simulator, then it is also reliable against real data. However, in assuming the validity of this model, we are also implicitly assuming that the external loads placed on the cardiac tissue are physiologically realistic. On the other hand, for excessive loads which would cause the material to rupture, the model is not valid.

1.5 Soft-tissue mechanics: simulation and emulation

The complexity of soft-tissues and their role in various physiological functions make it challenging to analyse their mechanical properties through experimental techniques alone. Consequently, mathematical modelling and computer simulation have the potential to deliver unique insights in this area. This has inspired the development of the field of *soft-tissue mechanics*, which involves modelling the mechanical behaviour of biological tissues, such as muscles, tendons, ligaments [76].

The most common modelling approach taken in the field is the continuum mechanics framework [77], whereby soft-tissues are treated as continuous and deformable materials. In this framework, the goal is to find mathematical descriptions both for the forces under which the soft-tissue body is placed (e.g. blood pressure), and the constitutive relationship of the material, i.e. its mechanical response under loading. Equations of motion can

then be derived in terms of momentum balance and mass conservation, which are solved numerically. These numerical simulation results capture the deformation, stress, and strain distribution within the tissue, providing insights into its mechanical behaviour. In all soft-tissue models considered in this work, the continuum mechanic framework will be used, with simulations performed using the FEM.

Advancements in computational power, numerical algorithms, and imaging technologies have contributed to the increasing accuracy and complexity of soft-tissue mechanics models and simulators in recent years. This includes coupled multi-physics simulations, such as fluid-structure interaction [78] or electro-mechanical coupling [79], and models of tissue growth and regeneration over longer time scales [80]. Such models have highly promising potential for applications in personalised medicine. By integrating patient-specific data, for example medical images or biomechanical measurements, with mathematical models and simulations, researchers can predict tissue response, assess treatment options, and optimise interventions [81]. Soft-tissue mechanics simulations could also play a crucial role in surgical planning, by simulating surgical procedures, evaluating different surgical approaches, and predicting postoperative outcomes [82]. This could aid in preoperative decision-making and reduce the risk associated with surgical interventions.

Despite the clear potential benefits offered by soft-tissue mechanical modelling, the deployment of these models for real-time clinical applications has been limited to date. One reason in particular is the prohibitive expense incurred in running physiologically realistic computer simulations. In general, soft-tissue models are comprised of a system of coupled, non-linear PDEs defined on a complex geometry. Iterative numerical techniques are then required to perform a simulation, which can be time-consuming. For example, a key model considered in this work will be a model of the left ventricle (LV) of the heart in *diastole*, the passive stage of the cardiac cycle during which the LV fills with blood. One potential application for this model is in the context of an inverse problem, where stiffness properties of the cardiac tissue are estimated non-invasively from imaging scans [60]. However, this procedure involves calibrating the output of the simulator with observed

experimental data, and accurate calibration via an iterative optimisation routine could require hundreds or thousands of simulations to be performed. Since a single simulation of the LV model can take up to 10 minutes [60], clearly this procedure can not be performed fast enough for real-time results to be obtained.

For this reason, alternative means are required for real time deployment of complex soft-tissue mechanical models. There has recently been significant interest in the use of emulators or digital twins for this purpose - see for instance the review papers by Qian et al. [83] and Laubenbacher et al. [84]. In the present work, surrogate models are used for the *forward problem* of predicting the deformation of the LV from start to end diastole. Surrogate models based on GPs, neural networks and polynomial chaos expansion have been already been widely used in cardiac mechanics for forward problems [85]–[88]. Emulators have also been commonly used for *inverse problems*, where the objective is to identify certain parameters of the physical model by incorporating observational data. To give two specific examples, Lazarus et al. (2022) [89] used a neural-network surrogate model to identify stiffness levels in the cardiac tissue given volume and circumferential strain observations, while Caforio et al. (2023) [90] performed parameter inference using a physics-informed neural network. While we do not consider the inverse problem in cardiac mechanics in this work, we do consider inverse problems involving linear PDEs in Chapter 4.

There are several factors currently prohibiting cardiac mechanic emulators from deployment in clinical-settings. Perhaps the key problem is the lack of precise, real-time, automated methods for extracting data from cardiac imaging scans. For in-clinic deployment, a cardiac model would need to be calibrated against patient-specific data. Currently, the state-of-the-art is manual segmentation, which is slow and prone to human error. There is active research ongoing in this important area (in which we have participated [91], [4]), however this is not the subject of the present thesis. Regarding emulation itself, a particular limitation of existing surrogate models is their handling of the cardiac geometry. It is essential that an emulator for cardiac mechanics can generalise to unseen geometries. This is because the heart geometry of each subject will be unique, and differences in cardiac geometry can yield different behaviour under loading [28]. Existing studies accounted for variations in cardiac geometry with a low order

representation found using principal component analysis (PCA) [92]–[94]. This means that the emulator operates on an approximation to the geometry used by the simulator, not the true cardiac geometry extracted from imaging scans. We address this issue in Chapter 2 with the use of a graph neural network (GNN) emulator, which can operate on the exact cardiac geometry without approximations. Another limitation of most current approaches is their reliance on data-driven training. This means, for example, that performing in-clinic corrections via transfer learning (as discussed in [95, Chapter 6]) would be time-consuming to perform, as it would require simulation data to be generated in advance. Also, data-driven may not fully capture the true underlying physics of the model, as it is not trained to do so explicitly. We address this issue in Chapter 3 through the use of a physics-informed emulator training strategy. This precludes the use of simulation data, meaning real-time transfer learning can be performed. Furthermore, we show experimentally that this training routine yields emulation results that approximate the underlying physics more closely. Finally, the impact of uncertainty must be quantified and mitigated before cardiac mechanic emulators can be deployed in-clinic. For instance, even the geometry itself is uncertain due to noise in the imaging scans and imperfections in the current manual extraction procedure. The level of uncertainty here and its effect on emulator results should be both quantified and mitigated. A possible mitigation approach would be to build an ensemble of emulators, which would reflect the underlying uncertainty in the geometry. Despite the importance of this issue, once again this is beyond the scope of the present work.

Reduced-order models (ROMs) constitute an alternative approach by which real-time solutions can be found for soft-tissue mechanical models [96]. While the present work does not consider ROM techniques, we will give a brief overview here for completeness. ROMs aim to represent the dynamics of a high-dimensional system using a much smaller set of variables, capturing the essential variations at lower computational cost. In the context of cardiac mechanics, for example, this involves reducing the complexity of modelling the entire heart geometry and dynamics, while retaining key physiological features. A full-order model (FOM) (i.e. a simulator) of cardiac mechanics describes the dynamics of the state variables of the cardiac system (such as the deformation field). An ROM describes these dynamics using a reduced basis, which is typically found by applying

proper orthogonal decomposition (POD) to snapshots of the dynamical process. Using the reduced basis, a reduced form of the cardiac-mechanical equations can be derived, which can be solved in a computationally efficient manner. ROMs and emulators have different strengths and weakness. For example, the forward map of an emulator tends to be faster, while ROMs can be more accurate when dealing with patient-specific cardiac simulations [97].

1.6 Physics-Informed machine learning

Physics-informed machine learning (PIML) merges techniques from computational physics and machine learning (ML) to improve the modelling and understanding of physical systems. The idea of PIML is to design models which are a hybrid between physically derived mathematical models and data-driven ML models. PIML most commonly refers to the use of ML training techniques that incorporate into the inference procedure known laws or principles of the physical system under consideration. These laws are typically in the form of PDEs, but other formulations (for example energy formulations) are also possible. Unlike with traditional computer simulators, noisy observation data can also be easily integrated here into the inference. In this thesis we follow the precedent of [59] and extend the umbrella of PIML to also include approaches which explicitly encode known physical principles into the design of the ML model itself. In this context, the principles are typically invariances or equivariances of the system under certain operations [98].

The PIML literature is growing by the day - for an overview, we recommend recent survey papers [59], [99]. Here, however, we will concentrate on three specific and landmark PIML publications that are especially relevant for the work presented in this thesis.

- Battaglia et al. (2018) [100] surveys the literature on Graph Neural Networks (GNNs), a class of neural network architecture which has seen rapid advancements in the past several years. GNNs allow inductive biases to be explicitly encoded into the model itself, meaning they do not have to be learned from data. This can

enable systems described by an extremely large number of particles to be accurately modelled without overfitting, in contrast to a fully connected neural network. For instance, in a fluid dynamics model, each particle represents a particle of fluid. For more details on GNNs, see Section 1.7.2 below.

- Raissi et al. (2019) [101] is a groundbreaking paper in the field of PIML, in which physics-informed neural networks (PINNs) are introduced. PINNs incorporate both noisy observation data and PDE information into a single coherent inference framework that allows for forward and inverse problems to be solved. PINNs have seen massive development in the years since this publication, and have been used to model processes across the entire spectrum of physics [59].

- Raissi et al. (2017) [102] is an earlier work by the same authors, in which again the objective was to develop a coherent inference framework which integrated observational data with PDE information. Here, however, Gaussian processes (GPs) are considered in place of neural networks. This work builds upon the well known property that Gaussian processes are closed under linear operations, which makes them natural candidates to model linear PDEs. The authors show how to leverage this property to seamlessly integrate data and PDEs using a GP, allowing for efficient and effective learning of any unknown PDE parameters as well as the unknown function itself.

1.7 Background Methods

1.7.1 Fully-connected neural networks

Fully-connected neural networks (FCNNs), also known as multi-layer perceptrons (MLPs), are flexible, nonlinear function approximators [103, Chapter 13]. FCNNs can trace their origins to the pioneering work of [104], [105] - today, they are perhaps the most essential component of modern machine learning systems. In the present work, FCNNs are used in the context of *regression*. Recall that the objective of regression is to learn the form of an unknown underlying function (denoted f), given an observed dataset of input-output

pairs:

$$\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}. \tag{1.1}$$

The following noise model linking the observations and the true value of the underlying function is typically assumed

$$y^{(i)} = f(\boldsymbol{x}^{(i)}) + \varepsilon^{(i)}, \quad i = 1, \ldots, N, \tag{1.2}$$

where $\varepsilon^{(i)}$ is a random value. In this context, an FCNN can be used to define a forward map which approximates the true latent function across the input space, i.e.

$$NN(\boldsymbol{x}; \boldsymbol{\omega}) = \hat{y} \approx f(\boldsymbol{x}), \tag{1.3}$$

where $\boldsymbol{\omega}$ are trainable parameters (see Section 1.7.1.1 below for more details).

The forward map of the FCNN is composed of a series of affine transformations, followed by a nonlinear activation. Specifically, if we set $\boldsymbol{z}^{(0)} = \boldsymbol{x}$, then the FCNN firstly processes L hidden layers $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(L)}$ as

$$\boldsymbol{z}^{(l)} = \varphi(\mathbf{W}^{(l)}\boldsymbol{z}^{(l-1)} + \boldsymbol{b}^{(l)}) \text{ for } l = 1, \ldots, L, \tag{1.4}$$

where φ is called the activation function (which is applied element-wise), $\mathbf{W}^{(l)}$ is called the weight matrix of layer l, while $\boldsymbol{b}^{(l)}$ is called the bias vector. After all the hidden layers have been processed, the final prediction is then made as

$$\hat{y} = \mathbf{w}^{(L+1)} \cdot \boldsymbol{z}^{(L)} + b^{(L+1)}. \tag{1.5}$$

In order to use an FCNN, the *architecture* of the network must be specified, by deciding both the number of hidden layers (i.e. the value of L), and their dimensionality. These values are referred to as the *depth* and *width* respectively of the network. It is possible to prove that an FCNN with a single hidden layer is a universal function approximator, given appropriate choice of activation function and no limit on the network's width [106].

Intuitively, this means it is possible to choose the weights and biases such that any continuous function can be approximated to arbitrary precision. In practice, however, deep network architectures with more than one hidden layer tend to yield greater accuracy and generalisation capability than single layer networks [107].

In addition to the network architecture, the form of the activation function φ used when processing the hidden layers must also be specified. The smoothness properties of the activation function affect the smoothness properties of the FCNN itself, and can also induce different training behaviour - for more details, see the review paper [108]. Two commonly used activation functions include the *ReLU* and *tanh* functions, which are defined as

$$\textbf{tanh: } \varphi(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1}, \tag{1.6}$$

$$\textbf{ReLU: } \varphi(x) = \text{Max}(0, x). \tag{1.7}$$

These functions are plotted in Figure 1.2 (a), over the interval $[-1, 1]$. It is essential that a *non-linear* activation function is used to allow the FCNN to model non-linear functions - for an example of this, see Figure 1.2 (d).

1.7.1.1 Training of weights and biases

Once the architecture and activation function of the FCNN have been chosen, it remains to specify its weights and biases, denoted collectively as

$$\boldsymbol{\omega} = (\mathbf{W}^{(1)}, \boldsymbol{b}^{(1)} \ldots, \mathbf{W}^{(L)}, \boldsymbol{b}^{(L)}, \mathbf{w}^{(L+1)}, b^{(L+1)}). \tag{1.8}$$

Intuitively, the value of $\boldsymbol{\omega}$ should be chosen so that the approximation in Eq. (1.3) is as close as possible, across the entire input domain of interest. To this end, $\boldsymbol{\omega}$ can be tuned to minimise the mean squared prediction error on the training data:

$$L(\boldsymbol{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - NN(\boldsymbol{x}^{(i)}; \boldsymbol{\omega}) \right)^2. \tag{1.9}$$

The nonlinearity of φ means that the objective function in Eq. (1.9) is not convex with respect to $\boldsymbol{\omega}$, and no closed-form solution is available [109, Chapter 8]. Because of this, a numerical optimisation scheme is used instead. Perhaps the simplest approach that can be used is gradient descent (GD), whereby an initial value of the parameters $\boldsymbol{\omega}^{(0)}$ is proposed (generally randomly generated), before this is sequentially updated via a series of steps to designed to reduce the loss function $L(\boldsymbol{\omega})$. Each update step $t > 0$ takes the form

$$\boldsymbol{\omega}^{(t)} = \boldsymbol{\omega}^{(t-1)} - \eta \nabla L(\boldsymbol{\omega}), \tag{1.10}$$

where $\eta > 0$ is called the learning rate. For the training of deep neural networks, GD has some drawbacks, however. Firstly, all data points are required to compute the gradient of the loss function (i.e. $\nabla L(\boldsymbol{\omega})$) at each update step, which can be expensive. This can however be overcome through the use of mini-batches, where a stochastic estimate of the gradient is obtained using a subsample of the available training data [103, Section 8.4.1]. A more fundamental issue for GD is that it can struggle in the presence of both suboptimal local minima and saddle points of the loss function [110]. Several extensions of basic GD have been proposed to overcome this issue - one example is called adaptive moment estimation (Adam) [111], which is used in all the optimisation problems considered in this thesis. Adam makes use of a momentum like variable to adapt the learning rate during optimisation, allowing saddle points to be crossed. For an illustration of an FCNN used to perform regression (with three different choices of activation function), see panels (b), (c) and (d) of Figure 1.2. In this toy example, we set $f(x) = x \sin(x)$, and use $N = 50$ training data points with Gaussian observation noise.

It is important to distinguish two types of loss when training a machine learning model, such as an FCNN. Firstly, there is the *training loss* on the discrete set of observational data. Then, there is the *generalisation loss* at points not seen in the training phase. Minimising training loss does not imply that generalisation loss is also minimised, as it is possible for the model to "overfit" to the noise values $\varepsilon^{(i)}$ in the observation model from

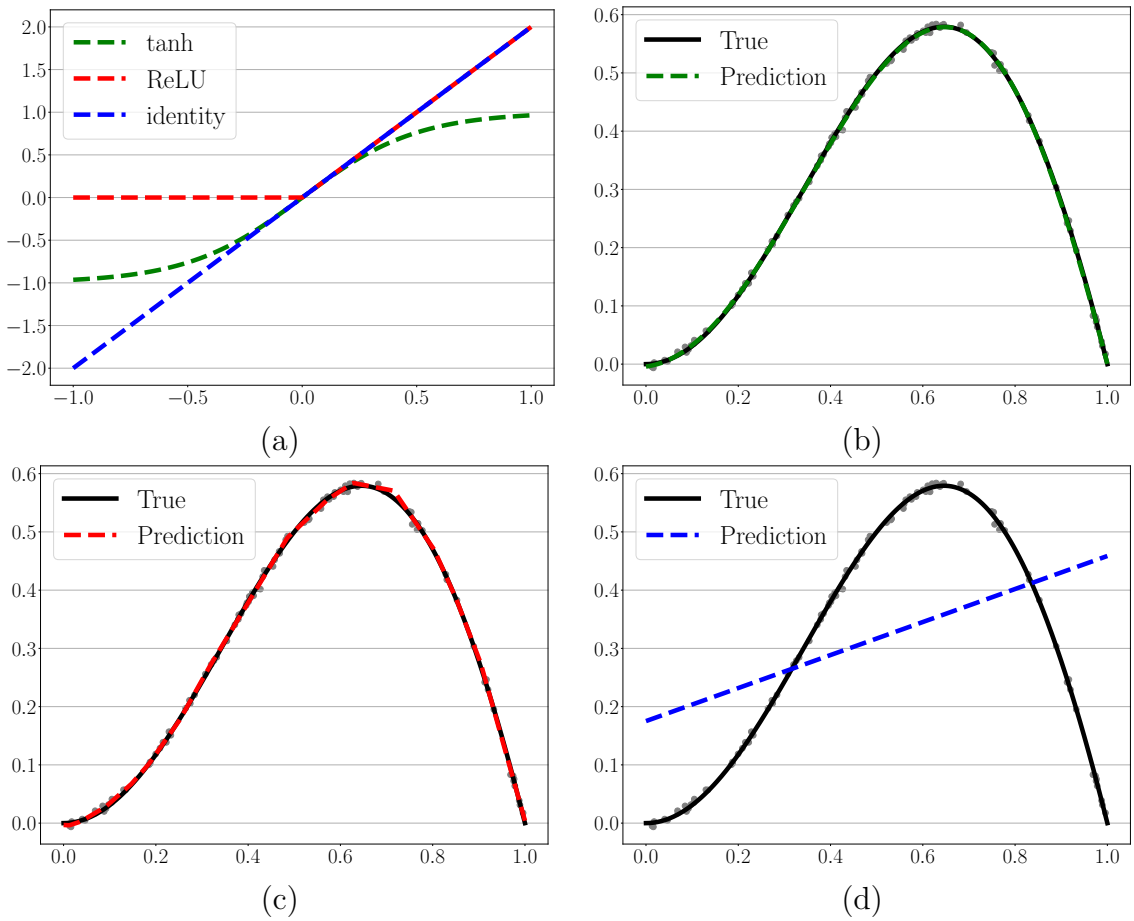


Figure 1.2: Panel (a) shows three possible activation functions φ. Regression results on toy data with FCNNs using each of these choices of φ are shown in remaining panels. Note that the linearity of the identity function induces an FCNN which is affine in the input variable x.

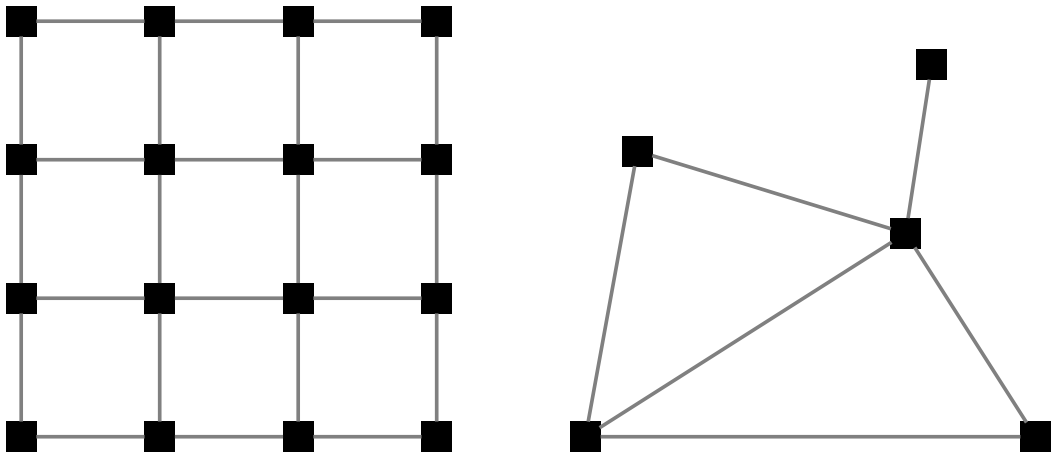
Eq. (1.2), rather than learning the form of the underlying function f [103, Section 1.2.3]. Numerous approaches have been proposed to alleviate the overfitting problem - in this work, we use the simple approach of reserving part of the training data as an independent validation set, on which the issue of overfitting can be monitored.

1.7.2 Graph neural networks

Graph neural networks (GNNs) have emerged in recent years as a powerful framework for performing machine learning tasks on graph-structured data [103, Chapter 23]. Mathematically, a *graph* is a tuple of the form $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$. Here the individual elements (called nodes) n_i of the graph are contained in the set $\mathcal{V} = \{n_1, n_2, \ldots, n_{|\mathcal{V}|}\}$, while directed edge connections between pairs of nodes (denoted $n_i \rightarrow n_j$) are contained in the set $\mathcal{E}$, called the graph topology. If necessary, this basic graph structure can be imbued with additional details such as global features - for more details, see Section 2.2.1.

Two example graphs are shown in Figure 1.3, where the nodes are represented as black squares, and the edges (plotted without directionality) as grey lines. While these two graphs may look similar at first glance, there is an important difference in their structures, which in turn affects the class of neural network that can operate on them. For the graph in panel (a), the graph has a grid-like shape, with each node having the same type of neighbourhood structure (top/bottom, left/right)[1]. This type of graph arises for example in the digital representation of images, in which case the nodes correspond to pixels. For machine learning tasks related to image processing (such as object detection / classification), *convolutional* neural networks (CNNs) are considered the default choice [103, Chapter 14]. Convolutional filters are the key mechanisms of a CNN. These filters are encoded as trainable weight matrices, and are applied sequentially across the image by the use of matrix multiplications applied to the neighbourhood of each pixel. Importantly, because of the grid-like nature of the underlying graph, each pixel has the same neighbourhood structure and so the same filter can be applied across the image. For the graph in panel (b), however, the neighbourhood structure of each node is not fixed. For instance, the node in the centre has four neighbours, while the upper right node has only one. This inconsistency means that a convolutional filter cannot be directly applied to this graph, meaning that a CNN is not appropriate here. However, it is exactly this type of irregular graph structure that a GNN is designed to operate on. A GNN can in fact be

[1]Note that while this is not the case for boundary nodes, this can be easily overcome through the use of padding [103, Section 14.2.1.4].



(a) Grid-like graph (b) Irregular graph

Figure 1.3: Plots of two example graphs, where the nodes are represented as black squares, and the edges as grey lines.

considered as a generalisation of a CNN to operate on graphs without a fixed, grid-like neighbourhood structure. Graphs with this type of complex relational structure arise in many areas, such as chemistry [112], biology [113], physics [114], social networks [115] and recommendation systems [116].

A wide range of different GNN architectures have been proposed. For instance, *spectral* GNN approaches use the spectral domain of the graph Laplacian matrix to define graph convolutions. This can be done in a spectrum-based manner where the eigendecomposition of the Laplacian is explicitly computed [117], or in a spectrum-free manner where the decomposition is not explicitly computed [118]. An alternative approach is to perform graph convolutions in a *spatial* manner. This is a more direct analogue to the convolution operations of a CNN. The main idea of these approaches is to create fixed-size graph patches for each node, overcoming the irregular neighbourhood structure of general graphs. This can be done, for example, in a sampling manner (such as the GraphSAGE model [119]), or using attention mechanisms (such as the Graph attention network [120]). For the modelling of physical systems, in particular, an alternative method has become popular called *message-passing* GNNs. Since this is the framework that is deployed in Chapters 2 and 3 for emulation in soft-tissue mechanics, we discuss message-passing GNNs in more detail below.

1.7.2.1 Message-passing GNNs

Message-passing GNNs approach graph-based machine learning problems via an information diffusion mechanism, whereby the sequential application of message-passing steps between neighbouring nodes allows information to be propagated around the graph, before a final decode stage solves the task at hand. Here we will consider the task of nodal regression, which is most pertinent to the work of this thesis. This type of approach dates back to the pioneering work of Gori et al. (2005) [121]. Battaglia et al. (2018) [100] outlines the modern evolution of the approach, in which GNNs process learned embeddings over both nodes and features, allowing tasks such as node regression, link prediction, and graph classification to be performed. While there are many different versions of message-passing GNNs, in general, each follows a three stage architecture, which is detailed below.

Stage One: Encoder. In the first stage, the nodes and edges of the graph are given feature vectors. We denote the node features as $\boldsymbol{v}_i$ and the edge features as $\boldsymbol{e}_{i\to j}$. The form of these features will depend on the specific problem of interest. For example, in Chapter 2, we make use of a message passing GNN for emulation of cardiac mechanics. In this case, the nodes represent parts of the cardiac tissue, and the node feature vector includes the fibre orientation of the muscle fibres at that node, while the edge features give the distance between pairs of nodes. Once the node and edge features have been assigned, they are then *encoded* into a higher dimensional latent space:

$$\boldsymbol{v}_i^0 = \text{NODE-ENCODER}\left(\boldsymbol{v}_i\right) \text{ for all } n_i \in \mathcal{V}, \tag{1.11}$$

$$\boldsymbol{e}_{i\to j}^0 = \text{EDGE-ENCODER}\left(\boldsymbol{e}_{i\to j}\right) \text{ for all } \{n_i \to n_j\} \in \mathcal{E}. \tag{1.12}$$

The idea is that by embedding into this latent space, we obtain more expressive representations for the processor and decode stages.

Stage Two: Processor. In the processor stage, the encoded node and edge representations are sequentially updated for $k = 1, \ldots, K$ *message passing* steps, where K is user specified. Each step k begins by calculating a message along each edge as follows:

$$\boldsymbol{m}_{i\to j}^k = \text{EDGE-PROCESSOR}\left(\boldsymbol{e}_{i\to j}^{k-1}, \boldsymbol{v}_i^{k-1}, \boldsymbol{v}_j^{k-1}\right) \text{ for all } \{n_i \to n_j\} \in \mathcal{E}. \tag{1.13}$$

The messages are then used to update the node and edge representations as

$$\boldsymbol{v}_i^k = \boldsymbol{v}_i^{k-1} + \text{NODE-PROCESSOR}\left(\boldsymbol{v}_i^{k-1}, \sum_{j \in \mathcal{N}_i^{\mathcal{G}}} \boldsymbol{m}_{j\to i}^k\right) \text{ for all } n_i \in \mathcal{V}, \tag{1.14}$$

$$\boldsymbol{e}_{i\to j}^k = \boldsymbol{e}_{i\to j}^{k-1} + \boldsymbol{m}_{i\to j}^k \text{ for all } \{n_i \to n_j\} \in \mathcal{E}, \tag{1.15}$$

where $\mathcal{N}^{\mathcal{G}_i}$ is the set of all nodes in $n_j \in \mathcal{V}$ which are connected to node n_i in $\mathcal{E}$. After K rounds of message passing have been performed, the final representations at each node can make use of information from all nodes that are no more than K steps away in the graph.

Stage Three: Decoder. The third stage is to *decode* the final learned node representations, depending on the task of interest. In this work, we are interested in node-decoding, whereby a prediction for the displacement of the node from the start to the end of diastole is made (see Chapter 2 for further details). Node decoding takes the form:

$$\hat{\boldsymbol{y}}_i = \text{NODE-DECODER}\left(\boldsymbol{v}_i^K, \sum_{j \in \mathcal{N}_i^{\mathcal{G}}} \boldsymbol{e}_{i\to j}^K\right) \text{ for all } n_i \in \mathcal{V}, \tag{1.16}$$

where $\hat{\boldsymbol{y}}_i$ denotes the GNN prediction of the quantity of interest at node n_i.

The form of the forward map defined by a message-passing GNN is determined by the encoding, processing and decoding functions respectively. These functions are each specified to be individual FCNNs. The weights and biases of all these FCNNs can then be trained in exactly the same manner as described in Section 1.7.1.1 above, with appropriate choice of loss function given the specific task of interest. The latent space dimensionality of the node and edge embeddings can be tuned by experimentation - this number should be chosen to be high enough so that the embedding is sufficiently expressive, but low enough to prevent overfitting. As for the number of message passing steps K, emulator accuracy can be expected to increase as more steps are included, but likely with diminishing marginal gains (see Figure 5 (d) of **meshGraphNets** for instance). Again, experimentation can be applied to see when performance begins to plateau.

1.7.3 Physics-informed neural networks

Physics-informed neural networks (PINNs) are neural networks which incorporate physical laws or principles into their training routine [101]. These laws or principles can be encoded in different forms - using, for example, PDEs, integro-differential equations, fractional equations or conservation laws [122]. PINNs are today used to solve forward, inverse and optimisation problems, for systems across the spectrum of biology, chemistry and physics [99]. In its most basic form, the idea behind a PINN is to approximate the solution function to a PDE (plus any initial/boundary conditions) using a neural network. This idea is illustrated below using a simple toy problem - the extension to more complex applications is then discussed in Section 1.7.3.1.

Consider the problem of finding $u : [0, 1] \to \mathbb{R}$ such that

$$-\partial_{xx} u(x) = g(x) = x^2 \text{ for all } x \in (0, 1), \tag{1.17}$$

subject to homogeneous Dirichlet boundary conditions, i.e $u(0) = u(1) = 0$. This is a specific form of Poisson's equation in one dimension [123, Section 2.2]. The solution u can be found by integrating twice and applying the boundary conditions to find the constants of integration, yielding the expression

$$u(x) = (x^4 - x)/12. \tag{1.18}$$

A common theme in physics is that there exist different ways of representing the same physical system. For example, classical mechanics can be represented in terms of forces (i.e. Newtonian representation) or in terms of the Principle of Least Action (i.e. Lagrangian mechanics) [25]. In the case of the 1D Poisson equation, there exists an alternative formulation of the problem in terms of the below functional Π, which is called the *total potential energy*:

$$\Pi(u) = \langle \partial_x u, \partial_x u \rangle - \langle g, u \rangle = \int_0^1 (\partial_x u(x))^2 \, dx - \int_0^1 g(x) u(x) dx. \tag{1.19}$$

Specifically, it can be shown that the function u which satisfies Eq. (1.17) and the given boundary conditions is exactly the function u which minimises Π [123, Theorem 17]. While these two formulations are mathematically equivalent, there is a key computational difference between them, as only the first derivative of u needs to be evaluated to compute Π, i.e. the required derivative information has been *weakened* relative to Eq. (1.17), which is called the *strong form* of the PDE.

In general, differential equations of practical interest cannot be solved analytically, meaning that numerical algorithms are required to find approximate solutions. Classical approaches include the FDM and FEM, mentioned in Section 1.2. A PINN is an alternative approximate solution method, whereby a neural network is proposed as the solution function u. For this specific problem, we propose the following approximation $\hat{u}$:

$$\hat{u}(x;\boldsymbol{\omega}) = x(1-x)NN(x;\boldsymbol{\omega}), \tag{1.20}$$

where $NN(x;\boldsymbol{\omega})$ is an FCNN (as described in Section 1.7.1). Note that the application of the transformation $x(1-x)$ to the output of the network ensures that $\hat{u}$ satisfies the boundary conditions $u(0) = u(1) = 0$, independently of the value of the parameters $\boldsymbol{\omega}$. Therefore, the boundary conditions do not need to be explicitly handled in the PINN training routine (see below). For more details on the explicit enforcement of boundary conditions in PIML modelling, see Chapter 4.

The free parameters $\boldsymbol{\omega}$ must be tuned in order for the approximation $\hat{u}$ to accurately capture the true solution u. To this end, a loss function needs to be defined for training. Unlike in Section 1.7.1.1, training is not performed against observational data, but against the Poisson equation itself. Because we have two formulations of the Poisson problem, two different PINN loss functions can be defined. Firstly, considering the strong form, a loss function can be specified as:

$$\textbf{Strong Form: } L(\boldsymbol{\omega}) = \frac{1}{N_g}\sum_{i=1}^{N_g}\left(g(x_i) - \partial_{xx}\hat{u}(x;\boldsymbol{\omega})\right)^2. \tag{1.21}$$

where the collocation points $x_1, x_2, \ldots, x_{N_g}$ are chosen to fill the interval $(0, 1)$. The key step required for this loss function to be directly computable is to implement the differential operator ∂_{xx} using automatic differentiation (AD), which allows it to be applied without requiring any hand derivations. In this example, and throughout the thesis, we make use of the AD framework provided by JAX. Secondly, considering the energy-formulation of the problem, a PINN loss function can be specified as

$$\textbf{Energy form: } L(\boldsymbol{\omega}) = \Pi(\hat{u}(\cdot; \boldsymbol{\omega}). \tag{1.22}$$

The integrals required to compute this loss function cannot be evaluated exactly, but can be approximated to essentially arbitrary accuracy using numerical quadrature.

To assess the performance of the PINN framework in this toy model, we trained the approximate solution from Eq. (1.20) on each PINN loss function, i.e. Eq. (1.21) and Eq. (1.22). In both cases, $NN(x; \boldsymbol{\omega})$ was implemented using the tanh activation function with two hidden layers each of width 16, and training was performed with Adam. $N_g = 50$ collocation points were used to evaluate the strong-form loss, while the energy-form loss was evaluated with 16 numerical quadrature points. Figure 1.4 shows the prediction results for the strong-form PINN (left column) against the energy-form PINN (right column), on both the solution function u (top row) and the forcing term g (bottom row). In this example, the strong form PINN incurs lower mean-absolute prediction error on both g (0.079 versus 7.440) and u (0.003 versus 0.007).

1.7.3.1 Extensions of the basic PINN framework

To implement a strong-form loss function for a general PDE operator (say $\mathcal{F}$), this simply requires ∂_{xx} to be replaced with $\mathcal{F}$ in Eq. (1.21). To implement an energy-form loss function for more complex examples, will generally require a more sophisticated numerical integration scheme (see Section 3.2.4 for example). In practical examples, (noisy) observations of the solution function itself u may also be available. This type of observational

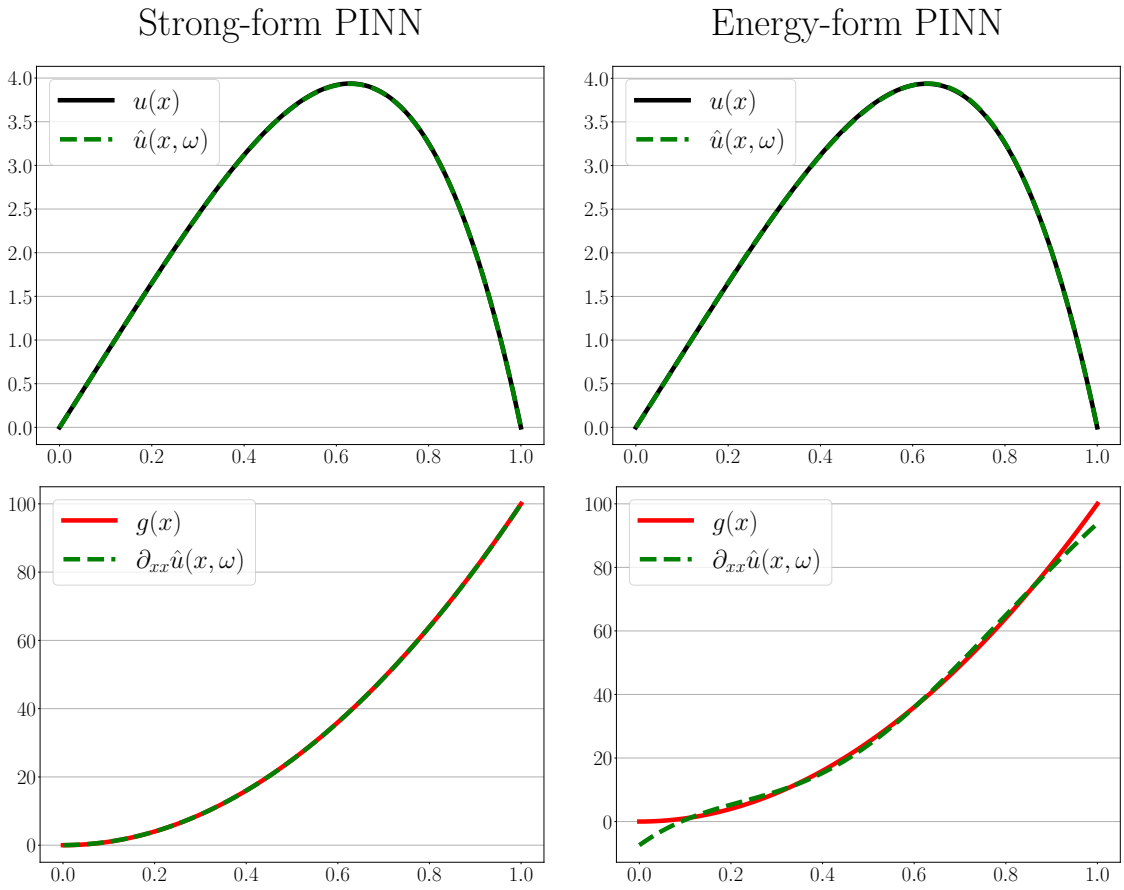


Figure 1.4: PINN results for 1D Poisson equation (see Eq. (1.17)). The left column shows the results of a PINN trained using the strong form of the problem (see Eq. (1.21)) and the right column shows the results of a PINN trained using the energy form of the problem (see Eq. (1.22)). The top row shows the true versus predicted value of the solution function u (see Eq.(1.18)), while the bottom row shows the true versus predicted values of the forcing term g (see Eq.(1.17)).

data can be easily incorporated into the inference framework by using a loss function which includes both a physics-informed term and a data-driven term (see Eq. (4.49), for instance). This is particularly useful for inverse problems, whereby any known PDE parameters can be jointly learned with u [101].

Beyond the strong-form and energy-form PINNs discussed above, a wide range of different physics-informed models have been proposed in recent years, to address different research challenges. To give some examples, this includes Bayesian PINNs for uncertainty quantification [124], generative PINNs for solving problems involving SDEs [125], PINNs which take inspiration from existing numerical approaches such as the finite-difference [126] and Galerkin [127] methods, conservative PINNs for accounting for conservation laws [128] and variational PINNs which allow for explicit domain decomposition [129].

1.7.4 Gaussian process regression

A stochastic process $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}\}$ is called a *Gaussian process* if, for any finite input set $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)} \in \mathcal{X}$, the corresponding outputs $\boldsymbol{f} = (f(\boldsymbol{x}^{(1)}), f(\boldsymbol{x}^{(2)}), ..., f(\boldsymbol{x}^{(N)}))^\top$ are jointly Gaussian distributed, i.e. $p(\boldsymbol{f}) = \mathcal{N}(\boldsymbol{m}, \mathbf{K})$ [130, Section 2.2]. Note that here we use the same notation f to refer to the GP random function and the forward model (Eq. (1.2)) - the reason for this shared notation will become clear below. A GP is completely defined by its mean function $m(\cdot)$ and covariance function, $k(\cdot, \cdot)$. These are evaluated on any discrete set of input points to give the mean vector $\boldsymbol{m}$ and covariance matrix $\mathbf{K}$ of the corresponding multivariate Gaussian distribution over the function outputs:

$$\mathbb{E}\left(f(\boldsymbol{x}^{(i)})\right) = [\boldsymbol{m}]^{(i)} = m(\boldsymbol{x}^{(i)}), \tag{1.23}$$

$$\mathrm{Cov}\left(f(\boldsymbol{x}^{(i)}), f(\boldsymbol{x}^{(j)})\right) = [\mathbf{K}]^{(i,j)} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}). \tag{1.24}$$

Note that, in order for $\mathbf{K}$ to be a valid covariance matrix, the covariance function k must be positive-definite. Such functions are called *Mercer kernels* - see Definitions 4.2.1 and C.1 for further details.

Consider now having observed data $\mathcal{D}$ as in Eq. (1.1), where we will denote the vector of observed output values as $\boldsymbol{y}$. In order to use a GP to perform regression using this dataset, two probabilistic assumptions are required about the data generation process given in Eq. (1.2). Firstly, we will assume that the underlying function in Eq. (1.2) follows a GP, with specified mean and covariance functions. Secondly, a distributional assumption is required for the observation errors. We will assume independent errors, each following a Gaussian distribution of the following form

$$\varepsilon^{(i)} \sim \mathcal{N}(0, \sigma^2), \text{ for } i = 1, 2, \ldots, N. \tag{1.25}$$

The objective of regression is to learn from the training data an estimate of the underlying function f, so that predictions can be made for the unobserved outputs $\boldsymbol{f}_*$ at any set of N_* test input locations of interest $\boldsymbol{x}_*^{(1)}, \boldsymbol{x}_*^{(2)}, \ldots, \boldsymbol{x}_*^{(N_*)}$. The GP assumption for the underlying function plus the Gaussian noise model together imply that $p(\boldsymbol{f}^*, \boldsymbol{y})$ is a multivariate

Gaussian distribution (see Eq. (4.7)). In this framework, the regression task reduces to the problem of finding the conditional distribution $p(\boldsymbol{f}^* | \boldsymbol{y})$. Using the properties of the multivariate Gaussian [103, Section 3.2.3], it can be shown that this distribution is also a Gaussian - see Proposition 4.2.1 for further details. Performing GP regression then requires only the mean and covariance functions respectively to be chosen. In practice, it is common to use a mean function set to zero [63]. The choice of covariance function is key, as this choice encodes assumptions about the properties of the underlying forward map, properties which can include stationarity, smoothness, periodicity, for instance. Assuming that our input space is the real numbers, two possible choices are the *rational-quadratic* and *linear* kernels:

$$\textbf{Linear: } k_{\text{Lin}}(x, x') = \sigma_b^2 + \sigma_v^2 (x - c)(x' - c) \tag{1.26}$$

$$\textbf{Rational-Quadratic: } k_{\text{RQ}}(x, x') = \sigma_v^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right)^{-\alpha} \tag{1.27}$$

To illustrate the impact that the choice of kernel has on the random functions generated by the GP, sample functions drawn from zero-mean GPs with linear and rational quadratic kernels, respectively, are shown in the top row of Figure 1.5. The linear kernel yields functions which are affine in x, while the rational-quadratic kernel yields smooth functions which are non-linear in x. In general, kernels will have tunable hyperparameters. For instance, the linear kernel has hyperparameters $(\sigma_b^2, \sigma_v^2, c)$ and the RQ kernel, has hyperparameters $(\sigma_v^2, \alpha, \ell^2)$. Given fixed observations $\boldsymbol{y}$, a commonly used approach is to tune the hyperparameter values to maximise $p(\boldsymbol{y})$, which is called the marginal likelihood in this context [130, Chapter 5].

1.7.4.1 Derivative observations

Because differentiation is a linear operator, GPs are closed under differentiation [130, Chapter 9]. This allows inference to be performed based on observations of both the value of the function itself and its derivatives. Again assuming that the input space is the real numbers, a GP prior $f(x) \sim \mathcal{GP}(0, k(\cdot, \cdot))$ implies the following covariance function

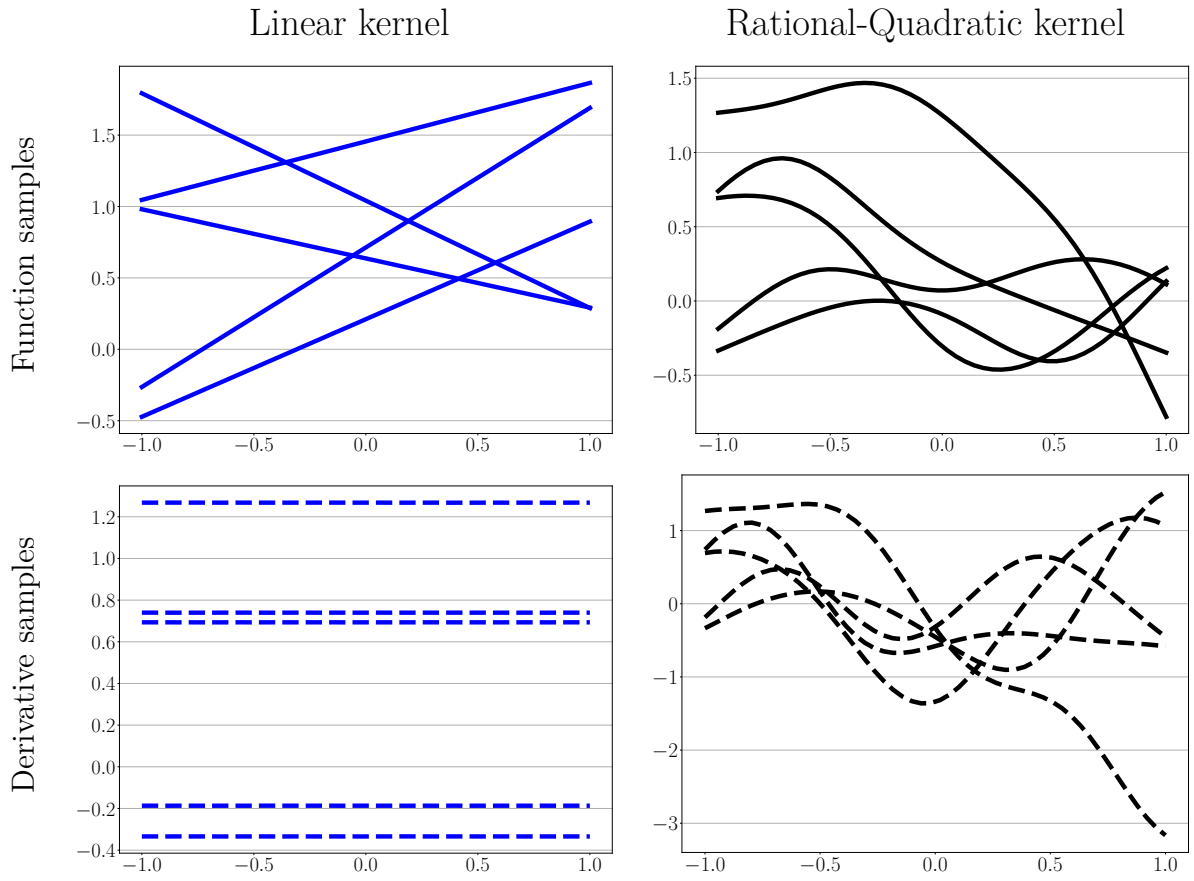


Figure 1.5: Samples from GPs with linear (Eq. (1.26)) and rational-quadratic (Eq. (1.27)) kernels, respectively. The top row shows samples in function space, and the bottom row shows samples in derivative space, where the kernel for the derivative samples was found as in Eq. (1.28).

on the derivative process

$$\mathrm{Cov}\left(\frac{\partial f(x)}{\partial x}, \frac{\partial f(x')}{\partial x'}\right) = \frac{\partial^2 k\,(x, x')}{\partial x \partial x'}. \tag{1.28}$$

and the following mixed covariance between the function and its derivative:

$$\mathrm{Cov}\left(f(x), \frac{\partial f(x')}{\partial x'}\right) = \frac{\partial k\,(x, x')}{\partial x'}. \tag{1.29}$$

For more details, see [131, Section 2.2]. The bottom row of Figure 1.5 shows samples of the derivative GPs induced by applying Eq. (1.28) to the linear and rational-quadratic kernels, respectively. These results can be extended beyond simple differentiation to more complex linear differential operators over higher dimensional spaces - this is made use of in Chapter 4, for parameter inference in systems described by linear PDEs.

1.8 Thesis outline

The contributions of this work are in the development of new approaches in the emulation of physical systems modelled with PDEs, with particular emphasis on the context of soft-tissue mechanics. The emulators we use are based on the latest research in PIML, and we therefore call this *physics-informed emulation*. Our contributions are presented in Chapters 2-4, which we summarise below.

- Chapter 2: here we introduce a Graph Neural Network (GNN) architecture for emulation of left-ventricle (LV) mechanics, trained in a data-driven manner. The advantage of a GNN here over more traditional emulation approaches is that it can operate directly on the high dimensional computational mesh used to represent the LV geometry, without requiring any approximations. Furthermore, the GNN delivers significant computational savings at prediction time over the simulator. We validate our method using extensive numerical experiments, where the emulator displays strong out-of-sample predictive performance.

- Chapter 3: here we extend the GNN framework above to make use of physics-informed training. This is where we train the emulator by minimisation of a potential energy functional, meaning that no costly simulation dataset needs to be generated. Once again, our approach is validated using numerical experiments, using a range of mechanical models and constitutive laws.

- Chapter 4: here we introduce a hard-constrained Gaussian process modelling approach for solving forward and inverse problems involving linear PDEs, which allows for the explicit enforcement of a versatile range of boundary constraints commonly used in PDE modelling. Again, we present extensive numerical experiments to test our approach, the results of which illustrate that explicit boundary enforcement greatly increases the robustness of GP inference in the presence of observation noise. Additionally, we present theoretical analyses of our method, which explore the representational capacity of our proposed model and its connection to neural networks.

Chapter 5 then concludes with a summary and discussion of the work presented in the thesis and an outlook on potential future research directions.

Chapter 2

Emulation of Cardiac Mechanics using Graph Neural Networks

Dalton, David; Gao, Hao; Husmeier, Dirk (2022) [9]. "Emulation of Cardiac Mechanics using Graph Neural Networks". In *Computer Methods in Applied Mechanics and Engineering*, https://doi.org/10.1016/j.cma.2022.115645.

Abstract

Recent progress in Graph Neural Networks (GNNs) has allowed the creation of new methods for surrogate modelling, or emulation, of complex physical systems to a high level of fidelity. The success of such methods has yet to be explored however in the context of soft-tissue mechanics, an area of research which has itself seen substantial developments in recent years. The present work explicates on this by introducing an emulation framework based on a multi-scale, message-passing GNN, before applying it to the modelling of passive left-ventricle mechanics. Through numerical experiments, it is demonstrated that the proposed method delivers strong predictive accuracy when benchmarked against the results of the non-linear finite-element method (FEM), and significantly outperforms an alternative emulator based on a fully connected neural network. Furthermore, large computational gains are achieved at prediction time against the FEM.

2.1 Introduction

We are entering the era of simulation intelligence (SI) [132]. SI describes a new paradigm for quantitative scientific methods, where traditional numerical simulation methods are combined with the latest advances in scientific machine learning. The leverage afforded by machine learning methods allows experiments that were previously computationally intractable to be performed, causal relationships between variables to be studied, model uncertainty to be quantified, and noisy real-world data to be integrated into the model framework. A key element of SI is surrogate modelling, or *emulation*. An emulator is a statistical machine learning model which approximates a numerical forward model, while incurring much lower computational expense at prediction time [62]. Commonly used emulation approaches include neural networks [103, Chapter 13], polynomial chaos [133] and Gaussian processes [63]. Emulation is widely used for forward [134] and inverse [135] problems, as well as design [136] and optimisation [137] tasks. Traditional methods for emulation however struggle to model high dimensional physical systems, such as those represented by high-fidelity meshes or large numbers of particles, due to the so-called curse of dimensionality. Furthermore, it can be difficult to build known symmetries or invariances of the system under consideration into the structure of traditional surrogate models. In recent years, new approaches have been developed for emulation of physical systems using Graph Neural Networks (GNNs) [103, Chapter 23]. A GNN can be seen as a generalisation of a Convolutional Neural Network (CNN) [103, Chapter 14] to operate on data with a non-Euclidean structure. This is critical for emulating complex or multiscale systems, where the topology of the particles or nodes in the system under consideration can be highly non-Euclidean. Furthermore, by accounting for the intrinsic topology of a given system, GNN emulators have demonstrated the ability to perform accurate modelling to very high levels of fidelity, beyond what is possible with traditional emulation approaches. Consider for example GraphCast, a GNN emulator for weather prediction which is considered the world's most accurate medium range weather forecaster [138]. GraphCast makes use of a discretisation of the earth's surface that is made up of more than one million grid points. If a traditional, fully connected neural network were to be used on this data, an architecture with hundreds of billions of para-

meters would potentially be required, which would be computationally infeasible to fit. Example application domains of GNN emulators include computational fluid dynamics [139], particle-based systems [114], cloth and elastic simulations **meshGraphNets**, rigid and deformable bodes [140] as well computer graphics [141].

One potential application domain of GNN emulation that has received less attention to date is the mechanics of biological soft tissue, that is, muscles, neurons, skin, cartilages, which usually exhibit remarkable complexity with multiconstituent, hierarchical and heterogeneous structures, and their mechanical responses to various loading conditions can be critical to maintain the biomechanical homeostasis for optimal mechanical functioning of the organism [142]. When injury or disease occurs in biological soft tissues, an imbalanced stress/strain micro-environment can be induced due to the loss of the complex organisation of biomolecules, such as the loss of myocytes after myocardial infarction. Therefore, there is a critical need for accurate and fast quantification of biomechanic factors, such as stress and strain, in soft tissues. One approach is to develop high-fidelity computational biomechanics models, which integrate knowledge of physiology, pathology and the fundamental laws of mechanics into one framework [143], with the potential to offer patient-specific diagnostic insights beyond what is available from typical in/ex vivo analyses [81]. An excellent example is computational cardiac mechanics, which is now being translated into industry and the clinic [144]–[146]. Other topics include the modelling of liver [147], mitral valve [148], arteries [149] and brain [150], to list some examples. Personalised biomechanics models need to be solved numerically, one such method being the nonlinear finite element method (FEM) [151]. FEM-based models can require significant computational resources due to potentially millions of unknown variables, interactions among different subsystems, integration across different temporal and spatial scales, and limited experimental data for model calibration [143]. This computational bottleneck constitutes a large obstacle in translating biomechanical models into the clinic for use in real-time estimation, where thousands of model simulations can be required to obtain accurate calibration with experimental data [144].

2.1.1 Contributions

To address these issues, recent efforts have focused on integrating machine learning and biomechanics modelling together to create robust predictive models, explore massive parameter spaces, and provide real-time solutions [152]. This chapter adds to the growing literature in this area by presenting a new GNN emulation framework, before evaluating its accuracy in the modelling of passive left ventricle (LV) mechanics. This is a challenging emulation problem, as different LV anatomies can vary substantially in terms of size, asymmetry, and wall thickness, for example, as well as exhibiting different material properties. In addition, the emulator must deliver significant computational savings at prediction time to be useful for real-time estimation. The GNN is implemented using a message-passing approach on an augmented, multi-scale graph representation of the LV. Experimental results demonstrate that the proposed method displays strong emulation accuracy across different LV anatomies, as well as the ability to specialise on a given anatomy of interest in a transfer learning setting. In addition, the emulator can make forward predictions up to five orders of magnitude more quickly than the numerical forward simulator.

The chapter is laid out as follows; first, Section 2.2 described the methods we use, including the proposed emulation framework. Section 2.3 describes the application of this framework to an illustrative two-dimensional beam deformation problem, before Section 2.4 describes the emulation results for passive LV mechanics. Section 2.5 finally then concludes.

2.2 Methods

2.2.1 General Mechanics and Graph Representation Framework

We consider two quasi-static or static mechanical systems in this work. The first is a 2-D beam with linear isotropic material property, clamped at one end and deformed under its own weight (see Section 2.3). The second is a real human left ventricular model in diastole, characterised by a nonlinear constitutive law (see Section 2.4). In general, the two models

can be formulated as a nonlinear boundary-value problem (BVP) of elastostatics, i.e.

$$\begin{cases} \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{b} = \mathbf{0} & \text{in } \Omega, \\ \boldsymbol{u} = \boldsymbol{u}_d & \text{on } \partial\Omega_d, \\ \boldsymbol{\sigma}\boldsymbol{n} = \boldsymbol{t} & \text{on } \partial\Omega_\sigma, \end{cases} \tag{2.1}$$

in which $\boldsymbol{\sigma}$ is the Cauchy stress tensor that is related to the strain tensor through a chosen constitutive law, $\boldsymbol{b}$ is the body force, $\boldsymbol{u}$ is the displacement field that is the unknown to be solved numerically, $\boldsymbol{u}_d$ is the prescribed displacement boundary conditions on the boundary $\partial\Omega_d$, and $\boldsymbol{t}$ is the applied traction density to the boundary $\partial\Omega_\sigma$. The whole computational domain Ω needs to be discretised with n simplicial or quadrilateral/hexahedral elements to allow the above BVP to be solved numerically, denoted as $\{\mathcal{T}_h\}_{h>0}$. Specifically,

$$\Omega \approx \{\mathcal{T}_h\}_{h>0} = \bigcup_{e_l=1}^{n} T_{e_l}, \tag{2.2}$$

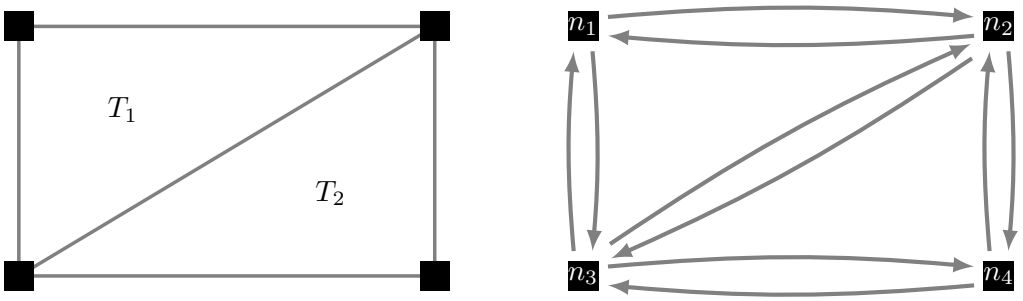
where $T \in \mathcal{T}_h$ can be tetrahedron or hexahedron for a 3-D finite-element (FE) mesh, and triangle or quadrilateral for a 2-D mesh. For example, Figure 2.1 (a) displays a simple rectangular domain Ω discretised with 2 triangles (T_1 and T_2).

In this work we use Graph Neural Network (GNN) emulators to find an approximation to the solution of the above general BVP for the two mechanical systems considered. In order to apply a GNN emulator to a given geometry, the discretised FE mesh (2.2) must be represented in the form of a *graph*, $\mathcal{G}$. We define a graph to be the following 3-tuple:

$$\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E}, \boldsymbol{\theta}). \tag{2.3}$$

The set of nodes of the graph $\mathcal{V}$ is defined as

$$\mathcal{V} \triangleq \bigcup_{i=1}^{|\mathcal{V}|} \{(n_i, \boldsymbol{x}_i)\}, \tag{2.4}$$



(a) FE Mesh $T_1 \cup T_2$ (b) Graph Representation $\mathcal{G}$

Figure 2.1: Panel (a) shows a simple 2-D FE mesh with two triangular elements (T_1 and T_2). Panel (b) shows the nodes $\mathcal{V}$ (indexed n_1 to n_4) and the topology $\mathcal{E}$ (shown as directed arrows) induced by the FE mesh.

which are extracted directly from the nodes of the FE mesh. Each node consists of a scalar index value n_i, the ordering of which follows that of the underlying FE mesh, and a coordinate vector $\boldsymbol{x}_i$, which gives the spatial coordinates of the corresponding node in the FE mesh, for $i = 1, 2, ..., |\mathcal{V}|$. As a shorthand notation, we will use the scalar index variable n_i alone when referring to the nodes in $\mathcal{V}$. The creation of $\mathcal{V}$ for the simple FE mesh in Figure 2.1 (a) is illustrated in panel (b), where the four FE nodes are converted to the set of graph nodes

$$\mathcal{V} = \{(n_1, \boldsymbol{x}_1), (n_2, \boldsymbol{x}_2), (n_3, \boldsymbol{x}_3), (n_4, \boldsymbol{x}_4)\}. \tag{2.5}$$

The set $\mathcal{E}$ is the graph topology, each element of which denotes a directed edge relationship between a pair of nodes in $\mathcal{V}$. A directed edge from node n_i to node n_j is denoted $\{n_i \rightarrow n_j\}$[2]. The directed edges in $\mathcal{E}$ are found by converting each undirected edge in the FE mesh into two directional graph edges which point in opposite directions. This is done by creating a directed edge from node n_i to all nodes $n_j \in \mathcal{V}$ that are in its *FE neighbourhood*, $\mathcal{N}_i^{FE}$, for all $n_i \in \mathcal{V}$. We define $\mathcal{N}_i^{FE}$ to be the set of all nodes in the graph that share an edge with node n_i in the FE discretisation from Eq. (2.2), i.e. the one-ring neighbourhood of n_i. For example, in Figure 2.1 (b), we have that the FE neighbourhood of node n_1 is

$$\mathcal{N}_1^{FE} = \{n_2, n_3\}. \tag{2.6}$$

[2]In the computer graphics literature, "directed edges" are commonly referred to as "half-edges".

We introduce the following shorthand notation to refer to all directed edges that have node n_i as sender:

$$\{n_i \rightarrow \mathcal{N}_i^{FE}\} = \bigcup_{n_j \in \mathcal{N}_i^{FE}} \{n_i \rightarrow n_j\}. \tag{2.7}$$

The entire graph topology $\mathcal{E}$ is then defined to be the union of the above over all nodes $n_i \in \mathcal{V}$,

$$\mathcal{E} \triangleq \bigcup_{i=1}^{|\mathcal{V}|} \{n_i \rightarrow \mathcal{N}_i^{FE}\}. \tag{2.8}$$

Thus for the FE mesh from Figure 2.1 (b), the specific edge set $\mathcal{E}$ is

$$\{n_1 \rightarrow n_2,\ n_2 \rightarrow n_1,\ n_2 \rightarrow n_4,\ n_4 \rightarrow n_2,\ n_3 \rightarrow n_4,\ n_4 \rightarrow n_3,\ n_2 \rightarrow n_3,\ n_3 \rightarrow n_2,\ n_1 \rightarrow n_3,\ n_3 \rightarrow n_1\}. \tag{2.9}$$

In this example, we have that $|\mathcal{E}| = 10$, which is almost as high as the cardinality of the completely connected directed graph on four nodes. However, for larger FE meshes, the cardinality of the graph topology $\mathcal{E}$ extracted in this manner will be significantly lower than that of the completely connected graph. Note that additional edges can be added if desired, beyond those directly extracted from the FE mesh in the manner described above. For example, the edge relations $\{n_1 \rightarrow n_4\}, \{n_4 \rightarrow n_1\}$ could be added to the graph topology $\mathcal{E}$ derived from the FE mesh in Figure 2.1.

The third element of $\mathcal{G}$ is a vector denoted $\boldsymbol{\theta}$, which represents any global attributes of interest. In the present work, $\boldsymbol{\theta}$ represents the material stiffness properties of the geometry under consideration, assumed constant for all nodes $n_i \in \mathcal{V}$, and hence $\boldsymbol{\theta}$ is a global attribute. However, this assumption can be relaxed to allow stiffness levels to vary across the graph.

Finally therefore, the entire graph is

$$\mathcal{G} = \left(\bigcup_{i=1}^{|\mathcal{V}|} \{(n_i, \boldsymbol{x}_i)\}, \quad \bigcup_{i=1}^{|\mathcal{V}|} \{n_i \rightarrow \mathcal{N}_i^{FE}\}, \quad \boldsymbol{\theta} \right). \tag{2.10}$$

For both the beam and left-ventricle (LV) mechanical systems considered for emulation in this work, the objective of the surrogate model is to be able to generalise to initial states not seen in the training data, in terms of the nodal coordinates from $\mathcal{V}$ and global material parameters $\boldsymbol{\theta}$. However, the topology $\mathcal{E}$ of the graph representations is assumed constant

for all graphs in each respective system. For example, each LV anatomy is represented by an FE mesh with the same number of nodes and the same finite element structure. This is reasonable because each LV has a consistent shape, whereby a central cavity (the LV chamber) is surrounded by a smooth 3D surface (the heart wall, or myocardium). This shape consistency is exploited by our in-house segmentation software, which converts cardiac imaging scans into a shared half-ellipsoidal mesh template. This means each mesh has the same number of nodes and connectivity structure. Each LV will be different geometrically, however, which means that the coordinate values assigned to each node will be unique to each LV. For further details of this segmentation procedure, we direct the reader to [60].

2.2.2 Neural Networks

A fully connected neural network, also known as the multi-layer perceptron (MLP) is a powerful tool for function approximation. Recall that an MLP $f_{\boldsymbol{\omega}} : \mathbb{R}^{D_{\text{in}}} \to \mathbb{R}^{D_{\text{out}}}$ is comprised of the sequential composition of affine functions with a nonlinear activation function $\varphi : \mathbb{R} \to \mathbb{R}$. A network with more than one unobserved, or hidden layer is referred to as a *deep* network. A deep network with two hidden layers takes the form

$$\begin{aligned} \boldsymbol{z}_1 &= \varphi\left(\mathbf{W}_0 \boldsymbol{x} + \boldsymbol{b}_0\right) \\ \boldsymbol{z}_2 &= \varphi\left(\mathbf{W}_1 \boldsymbol{z}_1 + \boldsymbol{b}_1\right) \\ \boldsymbol{y} &= \mathbf{W}_2 \boldsymbol{z}_2 + \boldsymbol{b}_2, \end{aligned} \tag{2.11}$$

where φ acts element-wise, $\mathbf{W}_t$ is the weight matrix of layer t and $\boldsymbol{b}_t$ the bias, for $t = 0, 1, 2$. Various forms for φ have been proposed [153], one of which is the tanh function:

$$\varphi(x) = \tanh(x) = \frac{2}{1 + \exp(-2x)} - 1. \tag{2.12}$$

The form of the forward map defined by an MLP is determined by the collection of all weights and biases of the network. A point estimate of these parameters is typically used for prediction, trained with a stochastic gradient-based update scheme on a dataset of input-output exemplars [103, Chapter 13].

Consider using an MLP to model very high-dimensional data, for example, emulating a complex physical system to a high level of fidelity. To capture the interactions and structure of the data, a network with large depth and/or width may be required. Because the internal weight matrices of the MLP are dense, the number of trainable parameters of the network will also be very large. This can make MLPs difficult to train to achieve good generalisation performance in this case. If however the high dimensional data being modelled has some intrinsic topology, this issue can be alleviated by designing a neural network architecture that accounts for this topology. For example, a convolutional neural network (CNN) applies a translation-invariant, discrete convolution to its internal layers that is applicable to data with Euclidean topology, instead of dense weight matrices. As convolution is a linear operator, a CNN can be equivalently written as a special case of an MLP, where the weight matrices exhibit a sparse, Toeplitz-like structure. This sparsity can lead to a massive reduction in the number of trainable parameters compared to an MLP, allowing CNNs to model high-dimensional image data, for example, without overfitting [103, Chapter 13].

The idea of a CNN is generalised by a Graph Neural Network (GNN), which refers to a class of deep learning architectures that allow graph-structured data to be modelled. GNNs have been applied in a wide range of contexts, including social network analysis [154], drug discovery [155] molecular chemistry [112], traffic prediction [156], and surrogate modelling of physical systems [114], [157], to give a few examples. Numerous approaches have been proposed for the design of GNN architectures. One such approach is spatial graph convolutions, which seeks to extend the convolution operations of a CNN, which operate on the regular (grid-like) local neighbourhoods of image data, to the irregular local neighbourhoods of arbitrary graph data. Spectral methods are another approach, where graph convolution operations are performed by operating in the spectral domain of the graph Laplacian matrix. In this work however, we use a GNN emulator that implements a *message-passing* approach. Message passing GNNs naturally extend to both large graphs and graphs not seen in the training data, and have proven successful for the emulation of physical systems **meshGraphNets**, [114], [158], [141]. A message-passing GNN tackles the problem of learning the physical dynamics of a system via an information diffusion mechanism around the graph topology. The nodes of the system are assigned a learned

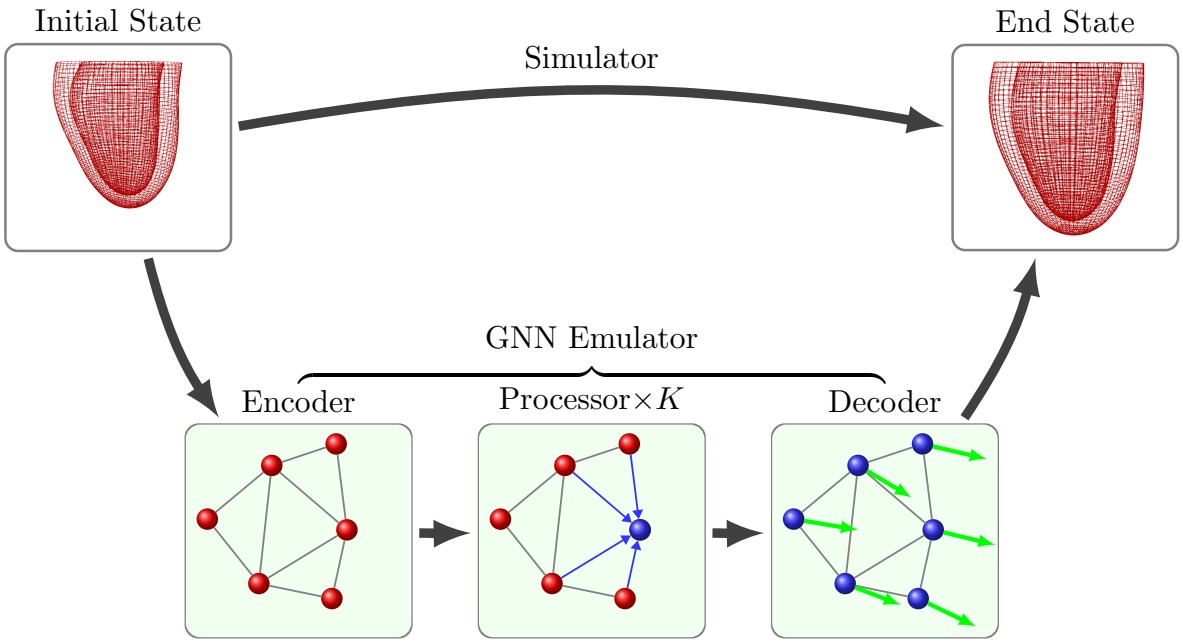


Figure 2.2: Schematic illustration of GNN emulation. The emulator makes use of a three-stage, encode-process-decode framework to map the initial state of the physical system to its end state.

representation using a sequence of processor steps, where messages are exchanged between neighbouring nodes, before a prediction for the forward dynamics of the system is made. A more general form of the message-passing framework was proposed in [100], where representations of nodes, edges, and the entire graph can be learned simultaneously in a three-stage, encode-process-decode approach. In this work a variant of the three-stage approach is used for emulation. Figure 2.2 gives a high-level illustration of the framework: for specific details, the reader can see Section 2.2.4.

The input to the GNN emulator is the initial state of the physical system under consideration in the form of a graph. In Figure 2.2, the input is the geometry of a left ventricle at the start of diastole, which is the emulation problem considered in Section 2.4. The first stage of the emulator is the encoder. This is where any node and edge-wise numerical features of the graph, such as boundary condition information, are embedded into a higher dimensional latent space. The second stage is to process these embeddings by performing a series of message-passing steps, whereby the node and edge representations are sequentially updated by exchanging information between neighbouring nodes in accordance with the graph topology. The last stage is the decoder stage, whereby the final node embeddings are used to predict the end state of the system. Each of the three

stages: encode, process and decode, are performed using individual, trainable MLPs. The parameters of the MLPs, which we refer to collectively as $\boldsymbol{\omega}$, control the predictions of the GNN. A pointwise estimate of $\boldsymbol{\omega}$ can be found using a gradient-based update scheme on a set of training data, in the same manner as for a single MLP.

2.2.3 Augmented Graph Generation

The GNN architecture introduced in Section 2.2.4 uses the message passing framework from Figure 2.2 to perform emulation, whereby information is propagated between the graph nodes, before a prediction is made for the forward displacement of the given body. Instead of working with the graph representation $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{\theta})$ directly extracted from the FE mesh in the manner discussed in Section 2.2.1, improved emulation performance can be achieved by working with an *augmented* graph representation of the mesh, $\tilde{\mathcal{G}}$. We define an augmented graph as

$$\tilde{\mathcal{G}} \triangleq (\mathcal{V} \cup \tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \boldsymbol{\theta}), \tag{2.13}$$

where the augmented nodes $\tilde{\mathcal{V}}$ and topology $\tilde{\mathcal{E}}$ are created iteratively over L steps as in Algorithm 1. The set $\tilde{\mathcal{V}}$ is defined as

$$\tilde{\mathcal{V}} \triangleq \bigcup_{l=1}^{L} Roots^l, \tag{2.14}$$

where $Roots^l$ is a set of virtual nodes derived from the FE nodes, for $l = 1, 2, ..., L$. The root nodes are defined constructively in Algorithm 1. In summary, each layer of root nodes $Roots^l$ is found by clustering the root nodes from the previous layer $Roots^{l-1}$ into a set of lower cardinality, meaning that successive layers provide an increasingly coarse representation of the underlying FE mesh. The augmented topology $\tilde{\mathcal{E}}$ is defined as

$$\tilde{\mathcal{E}} \triangleq \mathcal{E} \cup \bigcup_{l=1}^{L} \left\{ \mathcal{E}^l_{intra} \cup \mathcal{E}^l_{inter} \right\}. \tag{2.15}$$

At each step l, $\mathcal{E}^l_{intra}$ describes the *intra-layer* connections between nodes in $Roots^l$, while $\mathcal{E}^l_{inter}$ describes the *inter-layer* connections between nodes in $Roots^l$ and the nodes in the previous layer, $Roots^{l-1}$. Again, exact details of the construction of $\mathcal{E}^l_{intra}$ and $\mathcal{E}^l_{inter}$ are given in Algorithm 1. The advantage of the augmented graph representation $\tilde{\mathcal{G}}$ over the original graph $\mathcal{G}$ is that the virtual nodes $\tilde{\mathcal{V}}$ provide a shortcut by which information can propagate along the extended graph topology $\tilde{\mathcal{E}} \supset \mathcal{E}$.

Algorithm 1 details the construction of the augmented nodes and topology for a given input graph $\mathcal{G}$. Two other variables are required as inputs, the first of which is κ, the number of nearest neighbours to consider when defining the intra-layer connectivity $\tilde{\mathcal{E}}^l_{intra}$ for each layer of virtual nodes up to the final layer. The second is a decreasing sequence of natural numbers $\boldsymbol{\eta}$, which gives the cardinality of each layer of virtual nodes that are to be generated. These variables can be adjusted depending on the application context; specific details for the two experimental domains considered in this work are given in Sections 2.3.2 and 2.4.4 respectively. In general, the value of κ is specified so that the average number of neighbours each virtual node has in $\mathcal{E}^l_{intra}$ is approximately equal to the average number of neighbours each real node has in $\mathcal{E}$. The node cardinalities in $\boldsymbol{\eta}$ are selected so that the number of nodes decreases by approximately the same factor for each additional level, and such that this factor is at least as great as the number of real nodes in each finite element.

Figure 2.3 presents a visual explanation of the augmented graph generation algorithm, for one of the 2-D beam geometries considered for emulation in Section 2.3. Panel (a) of the figure displays the original graph $\mathcal{G}$. The positions of the $|\mathcal{V}| = 96$ nodes in $\mathcal{V}$ are shown as black squares, and each pair of directed edges in $\mathcal{E}$ is shown as a single grey line. Panel (b) then shows the coordinates of the virtual nodes $\tilde{\mathcal{V}} = Roots^1 \cup Roots^2$ generated by Algorithm 1 with $\boldsymbol{\eta} = [24, 6]$ and $\kappa = 4$, along with the original FE nodes. The first layer of $\eta_1 = 24$ virtual nodes are displayed as red circles, and the second and final layer of $\eta_2 = 6$ nodes are shown as green triangles.

Algorithm 1 begins by initialising the 0^{th} layer of root nodes, $Roots^0$, to be the original FE nodes $\mathcal{V}$. The multiscale augmented graph is then built over L steps. At step $l = 1$, the initial layer of virtual nodes $Roots^1$ is created by first grouping the coordinates $\boldsymbol{x}_i$ of the real nodes $\mathcal{V} = Roots^0 = Leaves^1$ into the clusters $\mathcal{C} = \{c_1, c_2, ..., c_{\eta_1}\}$ using κ-means.

Algorithm 1 Augmented Graph Generation

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{\theta})$, original graph; κ, nearest neighbours considered in $\tilde{\mathcal{E}}_{intra}$; $\boldsymbol{\eta} = [\eta_1, ...\eta_L]$, cardinalities of virtual node layers

Output: $\mathcal{G} = (\mathcal{V} \cup \tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \boldsymbol{\theta})$, augmented graph

Initialise $Roots^0 = \mathcal{V}$

for $l = 1, ...L$ **do**

  $Leaves^l = Roots^{l-1}$

  Group $Leaves^l$ into η_l clusters: $\mathcal{C} = \{c_1, c_2, ..., c_{\eta_l}\}$ using k-means

  Create new root node at each cluster centre: $\boldsymbol{x}_c = \frac{1}{|c|}\Sigma_{i \in c}\boldsymbol{x}_i$ for all $c \in \mathcal{C}$

  Collect all root nodes: $Roots^l = \bigcup_{c \in \mathcal{C}}\{(n_c, \boldsymbol{x}_c)\}$

  Connect roots to leaves: $\tilde{\mathcal{E}}^l_{inter} = \bigcup_{c \in \mathcal{C}}\{n_c \rightarrow n_i, n_i \rightarrow n_c \mid n_i \in c\}$

  if $l < L$ **then**

    Find neighbourhoods: $\mathcal{N}_c^{Roots} = \{n_{c'} \mid \boldsymbol{x}_{c'} \in \kappa\text{-nn}(\boldsymbol{x}_c, Roots^l)\}$ for all $c \in \mathcal{C}$

    Connect roots to roots: $\tilde{\mathcal{E}}^l_{intra} = \bigcup_{c \in \mathcal{C}}\{n_c \rightarrow n_{c'}, n_{c'} \rightarrow n_c \mid n_{c'} \in \mathcal{N}_c\}$

  else

    Connect roots to roots: $\tilde{\mathcal{E}}^l_{intra} = \bigcup_{c \in \mathcal{C}}\{n_c \rightarrow n_{c'} \mid n_{c'} \in \mathcal{C} \backslash c\}$

  end if

end for

Define augmented nodes: $\tilde{\mathcal{V}} = \bigcup_{l=1}^{L} Roots^l$

Define augmented topology: $\tilde{\mathcal{E}} = \mathcal{E} \cup \bigcup_{l=1}^{L}\{\mathcal{E}^l_{intra} \cup \mathcal{E}^l_{inter}\}$

The coordinates of the virtual nodes $Roots^1$ are then found as the centre of these η_1 clusters:

$$Roots^{l=1} = \bigcup_{c \in \mathcal{C}}\{(n_c, \boldsymbol{x}_c)\} \quad \text{where} \quad \boldsymbol{x}_c = \frac{1}{|c|}\sum_{n_i \in c}\boldsymbol{x}_i. \tag{2.16}$$

Note that each root node is also assigned a scalar index variable, denoted n_c, which is defined to be consistent with the index variables n_i of the real nodes $\mathcal{V}$. Once the roots have been found, the inter-layer topology $\tilde{\mathcal{E}}^1_{inter}$ between root and leaf nodes is defined by creating a pair of directed edges $\{n_i \rightarrow n_c, n_c \rightarrow n_i\}$ between each root node $n_c \in Roots^1$ and all leaf nodes $n_i \in Leaves^1$ which were assigned to cluster c by k-means,

$$\tilde{\mathcal{E}}^{l=1}_{inter} = \bigcup_{c \in \mathcal{C}}\{n_c \rightarrow n_i, n_i \rightarrow n_c \mid n_i \in c\}. \tag{2.17}$$

Figure 2.3 (c) displays the construction of the root nodes $Roots^1$ and inter-layer connections $\tilde{\mathcal{E}}^1_{inter}$ for the beam example. The coordinates $\boldsymbol{x}_c$ of each of the $\eta_1 = 24$ root nodes are shown as red circles, which were found by clustering the 96 FE nodes. The connections are shown as grey lines between each FE node and its corresponding cluster centre node.

For each root node $n_c \in Roots^1$, its local *root neighbourhood* $\mathcal{N}_c^{Roots}$ is defined to be its κ-nearest neighbours among other root nodes:

$$\mathcal{N}_c^{Roots} = \{n_{c'} \mid \boldsymbol{x}_{c'} \in \kappa\text{-nn}(\boldsymbol{x}_c, Roots^{l=1})\}. \tag{2.18}$$

The distance metric used in the nearest-neighbour calculations is the usual Euclidean metric. The intra-layer topology $\tilde{\mathcal{E}}^1_{intra}$ between the root nodes in $Roots^1$ is then constructed by creating a pair of directed edges $\{n_c \to n_{c'}\}, \{n_{c'} \to n_c\}$ between each root node n_c and its κ nearest neighbours in $\mathcal{N}_c^{Roots}$:

$$\tilde{\mathcal{E}}^{l=1}_{intra} = \bigcup_{c \in \mathcal{C}} \{n_c \to n_{c'}, n_{c'} \to n_c \mid n_{c'} \in \mathcal{N}_c\}, \tag{2.19}$$

Figure 2.3 (d) shows the creation of $\tilde{\mathcal{E}}^1_{intra}$ for the beam example as grey lines, using $\kappa = 4$ nearest neighbours. Note that the neighbourhoods defined by the κ-nn operation are not symmetric, that is $n_{c'} \in \mathcal{N}_c^{Roots} \not\Leftrightarrow n_c \in \mathcal{N}_{c'}^{Roots}$. This asymmetry is illustrated in Figure 2.4 (a), which shows the top right corner of the beam geometry from Figure 2.3 (d). Two nodes $n_c, n_{c'} \in Roots^1$ are highlighted. The four root nodes in $\mathcal{N}_c^{Roots}$ are shown with grey arrows, while the neighbours $\mathcal{N}_{c'}^{Roots}$ are shown with black arrows. We see that $n_{c'} \in \mathcal{N}_c^{Roots}$ but $n_c \notin \mathcal{N}_{c'}^{Roots}$. The GNN emulation architecture introduced in Section 2.4 requires that each edge relation $\{n_i \to n_j\}$ in $\tilde{\mathcal{E}}$ has a twin relation $\{n_j \to n_i\}$, so that the symmetry from Eq. (2.27) can be enforced. For this reason, we also add the additional connection $\{n_{c'} \to n_c\}$ to $\tilde{\mathcal{E}}^l_{intra}$, shown as a black arrow in Figure 2.4 (b). The inclusion of additional connections in this manner means that each root node will have *at least* κ neighbours in $\tilde{\mathcal{E}}^l_{intra}$, for $l = 1, 2, ..., L-1$.

Subsequent steps $l = 2, ..., L$ of Algorithm 1 proceed in the same manner as the first step, whereby the root nodes at the previous layer, $Roots^{l-1}$ become the leaf nodes of step l, $Leaves^{l}$ and are clustered into η_l root nodes. The inter-layer and intra-layer node topologies $\tilde{\mathcal{E}}^{l}_{inter}$ and $\tilde{\mathcal{E}}^{l}_{intra}$ are then calculated as before. This is with the exception of the final layer of virtual nodes $Roots^{L}$, which is fully connected:

$$\tilde{\mathcal{E}}^{L}_{intra} = \bigcup_{c \in \mathcal{C}} \{n_c \rightarrow n_{c'} \mid n_{c'} \in Roots^{l=1} \backslash (n_c, \boldsymbol{x}_c)\}. \tag{2.20}$$

Once the last layer of nodes has been created, the augmented graph $\tilde{\mathcal{G}}$ is returned. The third row of plots in Figure 2.3 shows the creation of the second and final layer of virtual nodes and connections for the beam example. In Panel (e), the virtual nodes from layer 1 are clustered into $\eta_2 = 6$ cluster centres to form $Roots^2$, and $\tilde{\mathcal{E}}^{2}_{inter}$ is created as before. Then, the final intra-layer topology $\tilde{\mathcal{E}}^{2}_{intra}$ in panel (f) is fully connected.

Our use of a multi-scale augmented graph for emulation is conceptually similar to earlier work [140], [159]. Recall however that the goal of this work is to build an emulator that can generalise to geometries not seen in the training data. As discussed though in Section 2.2.1, we assume that the graph representations of the geometries in each system we consider share a common topology, $\mathcal{E}$. We therefore desire that the augmented graph representations of the FE meshes also share a common topology, $\tilde{\mathcal{E}}$. However this is not guaranteed to be the case if Algorithm 1 were applied separately to the graph $\mathcal{G}$ induced by the FE mesh of each individual geometry. For example, consider the graphs $\mathcal{G}_0$ and $\mathcal{G}_1$ displayed in the top row of Figure 2.5. The two graphs clearly share a common topology, that is $\mathcal{E}_0 = \mathcal{E}_1$. However, the augmented graphs $\tilde{\mathcal{G}}_0$ and $\tilde{\mathcal{G}}_1$ that result from applying Algorithm 1 to each graph respectively with $\boldsymbol{\eta} = [2]$ do not share a common topology. This is illustrated in the second row of plots in Figure 2.5, where we can see that $\tilde{\mathcal{E}}_0 \neq \tilde{\mathcal{E}}_1$. The discrepancy occurs because the generation of the virtual nodes depends solely on the (different) coordinate values from the graph nodes $\mathcal{V}_0$ and $\mathcal{V}_1$, not the common graph topology $\mathcal{E}_0 = \mathcal{E}_1$. To ensure then that the augmented graph representations of different geometries have a shared topology $\tilde{\mathcal{E}}$, this can be constructed with respect to a single, representative graph using Algorithm 1. Any subsequent graphs can then be augmented in a manner similar to Algorithm 1, whereby each layer of virtual

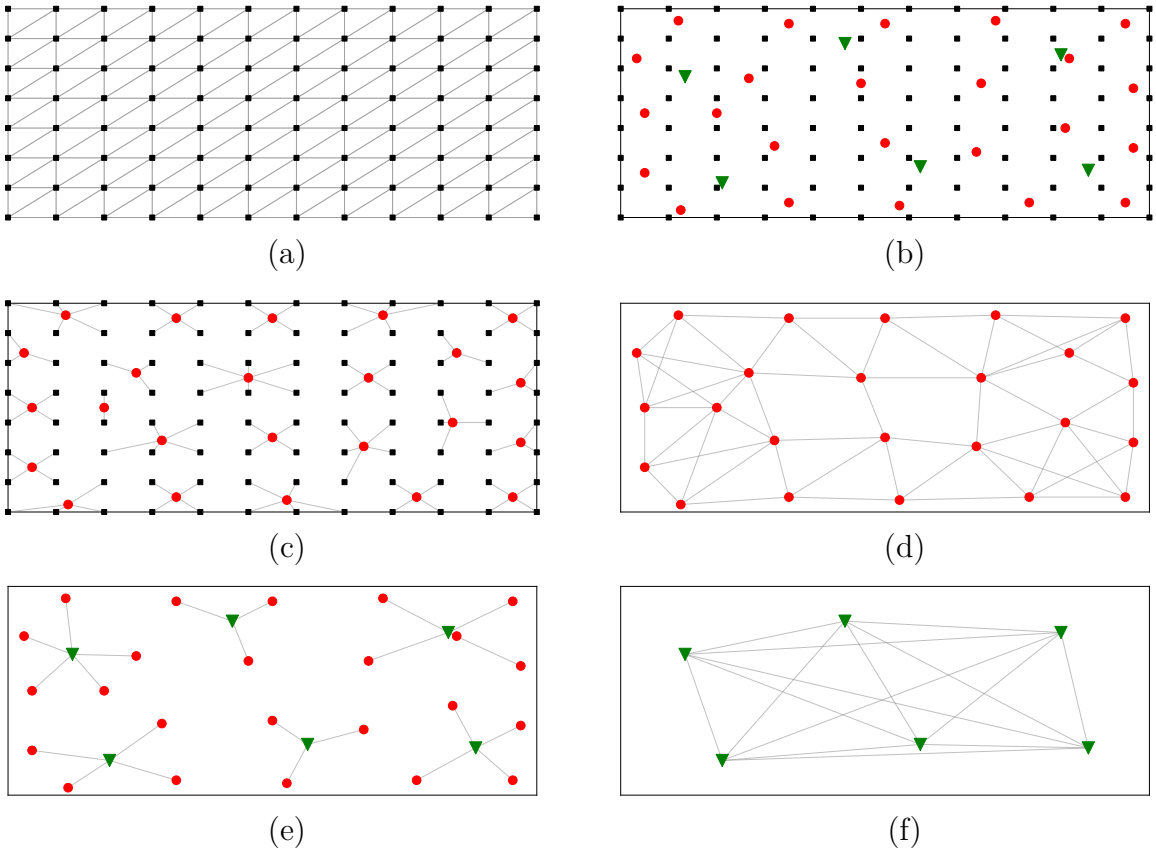
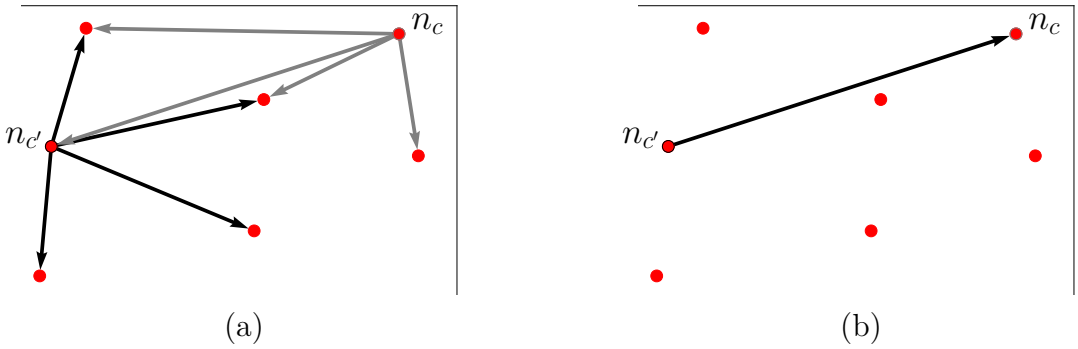


Figure 2.3: Illustration of Algorithm 1 applied to a 2-D beam geometry, with $\kappa = 4$ and $\boldsymbol{\eta} = [24, 6]$. Panel (a) shows the nodes and edges of the input graph and panel (b) plots the real nodes in black with the augmented nodes in red and green. The second and third row of figures illustrate steps one and two of the algorithm respectively - for a detailed description, see the text.

nodes is constructed by pooling nodes in the previous layer. However, the pooling is performed with respect to the clusters implied by the augmented topology $\tilde{\mathcal{E}}$ found on the representative graph, rather than reapplying κ-means clustering, meaning that each augmented graph shares this common topology. The nodes of the reference graph, denoted $\bar{\mathcal{V}}$ can be found by "averaging" over the node coordinates of the geometries from the available training data. Specifically, for each $n_i \in \bar{\mathcal{V}}$, the corresponding coordinate value $\bar{\boldsymbol{x}}_i$ is found as $\bar{\boldsymbol{x}}_i = \frac{1}{N_{tr}} \sum_j \boldsymbol{x}_i^{(j)}$, where we have introduced a superscript j to index over the N_{tr} available training geometries.



(a) (b)

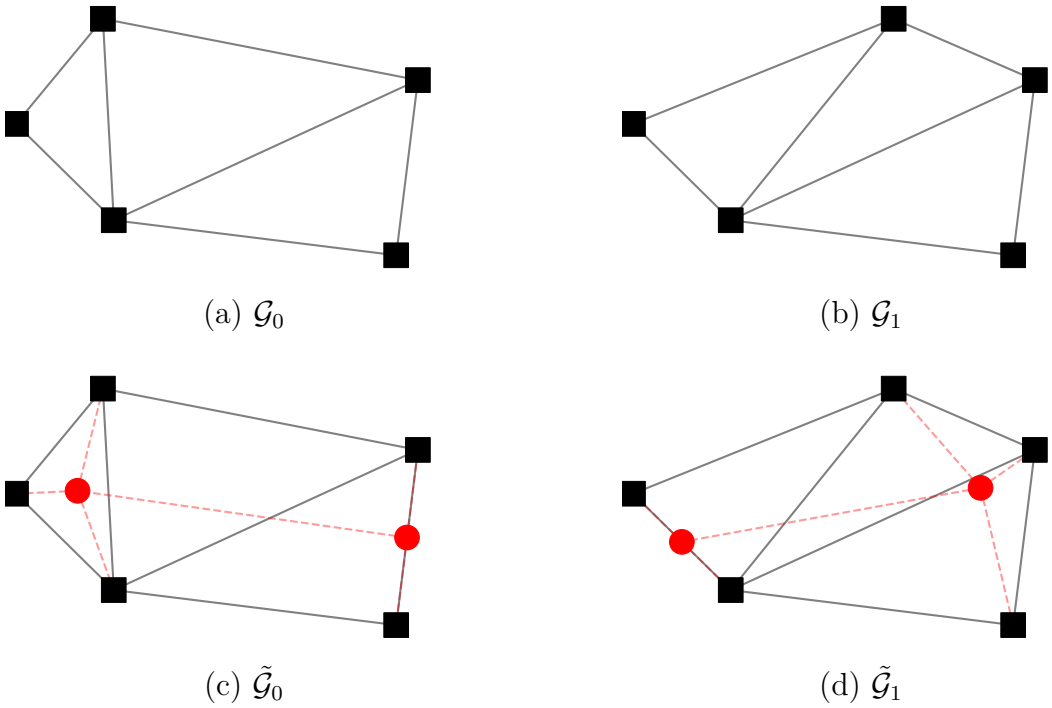
Figure 2.4: Panel (a) shows the root neighbourhood for node n_c, $\mathcal{N}_c^{Roots}$, with grey arrows, and the neighbourhood for node $n_{c'}$, $\mathcal{N}_{c'}^{Roots}$, with black arrows. We see that $n_{c'} \in \mathcal{N}_c^{Roots}$ but $n_c \notin \mathcal{N}_{c'}^{Roots}$. Therefore, an additional edge $\{n_{c'} \rightarrow n_c\}$ is created, so that the symmetry from Eq. (2.27) can be enforced in the message-passing stage of the emulator.

The virtual node positions could potentially be improved by using a different clustering approach, in particular through the use of a metric which accounted for the nonconvex shape of the left ventricle. However, we felt the simple κ-means approach was sufficient, as the main focus of this graph augmentation procedure was not the geometry of the virtual nodes, but their *topology*, i.e. the additional edges which can be used as shortcut connections for the message-passing.

The final data-processing stage before the surrogate model can be applied is to assign numerical feature vectors to the individual nodes and edges of the augmented graph representations. As detailed in Algorithm 2, these feature vectors are iteratively processed by the GNN emulator over a series of message passing steps, before the final processed values are used to predict the forward displacement of the body under consideration. Specifically, each real or virtual node $n_i \in \mathcal{V} \cup \tilde{\mathcal{V}}$ is *assigned* ($\twoheadrightarrow$) a feature vector, denoted $\boldsymbol{v}_i$,

$$n_i \twoheadrightarrow \boldsymbol{v}_i \text{ for all } n_i \in \mathcal{V} \cup \tilde{\mathcal{V}}. \tag{2.21}$$

The real nodes in $\mathcal{V}$ are first assigned a feature vector. The form of these features will depend on the application context, but could contain, for example, boundary condition information, or for an inhomogeneous material, local fibre structure information. Importantly, the absolute positions $\boldsymbol{x}_i$ are not encoded in the node features. Instead, relative node positions are encoded in the edge features defined below, ensuring that the outputs of the emulator are invariant to translations of the system in space. Each node in the



(a) $\mathcal{G}_0$ (b) $\mathcal{G}_1$

(c) $\tilde{\mathcal{G}}_0$ (d) $\tilde{\mathcal{G}}_1$

Figure 2.5: The upper row of figures shows two graphs $\mathcal{G}_0$ and $\mathcal{G}_1$ which share a common topology, that is $\mathcal{E}_0 = \mathcal{E}_1$. The bottom row shows the augmented graphs $\tilde{\mathcal{G}}_0$ and $\tilde{\mathcal{G}}_1$ found by running Algorithm 1 with $\boldsymbol{\eta} = [2]$. Virtual nodes are shown in red and additional edges are shown as dashed red lines. In this case, we have that $\tilde{\mathcal{E}}_0 \neq \tilde{\mathcal{E}}_1$.

first layer of virtual nodes is then assigned a feature vector by taking the mean of the features from all real nodes with which it shares an edge relation. Subsequent virtual node layers $l = 2, ...L$ are then iteratively assigned features in the same manner. Note that for boundary node Boolean indicator variables, we could have used min() rather than mean() aggregation to ensure that a virtual node will have a non-zero boundary indicator variable only if all of its neighbours in the previous layer are boundary nodes. These neighbourhoods will in general include interior points however and by using min() aggregation then the virtual nodes would have no boundary condition information in their feature vectors. See, for example, Figure 2.3, where no virtual nodes lie on the boundary of the beam geometry. By using mean() aggregation instead, boundary condition information is included in the virtual nodes, where the boundary indicator variable can take values between 0 and 1.

The edge relations $\{n_i \rightarrow n_j\} \in \tilde{\mathcal{E}}$ are also assigned numerical features, denoted $\boldsymbol{e}_{i\rightarrow j}$. For reasons discussed in Section 2.2.4, only one of each pair of directed edges $\{n_i \rightarrow n_j\}, \{n_j \rightarrow n_i\} \in \tilde{\mathcal{E}}$ needs to be assigned an edge feature vector. This is achieved by assigning features to those edges where the index of the sender node n_i is greater than that of the receiving node n_j (note there are no self-loops in the graph). That is, edge features are assigned as

$$\{n_i \rightarrow n_j\} \twoheadrightarrow \boldsymbol{e}_{i\rightarrow j} \text{ for all } \{n_i \rightarrow n_j\} \in \tilde{\mathcal{E}} \text{ where } i > j. \tag{2.22}$$

The form of the assigned edge features was the same for both systems considered here (beam and left ventricle);

$$\boldsymbol{e}_{i\rightarrow j} = (\boldsymbol{x}_j - \boldsymbol{x}_i, ||\boldsymbol{x}_j - \boldsymbol{x}_i||_2), \tag{2.23}$$

containing both the difference and distance between the coordinates of the sender node and receiving node respectively.

2.2.4 GNN Emulation Architecture

Algorithm 2 presents the `DeepGraphEmulator` GNN architecture, which defines a forward map of the form

$$\left(\tilde{\mathcal{G}}, \boldsymbol{z}^{\text{global}}\right) \mapsto \hat{\boldsymbol{U}}. \tag{2.24}$$

The first input variable $\tilde{\mathcal{G}}$ is the augmented graph representation of the geometry under consideration, the construction of which is described in Section 2.2.3 above. The second, optional input is $\boldsymbol{z}^{\text{global}}$, a vector-embedding of the global shape of the real nodes of the given geometry. The form of $\boldsymbol{z}^{\text{global}}$ will be context-dependent, and could be obtained for example using dimensionality reduction or parameterised methods. The GNN emulator then makes a prediction $\hat{\boldsymbol{U}}$ for the node-wise displacement of the body from its initial to end-state using a three-stage, encode-process-decode approach, which is illustrated schematically in Figure 2.2. Each stage is controlled by individual, trainable MLPs. In

Algorithm 2 `DeepGraphEmulator`

Input: $\tilde{\mathcal{G}} = (\mathcal{V} \cup \tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \boldsymbol{\theta})$, $\boldsymbol{z}^{\text{global}}$
Output: $\hat{\boldsymbol{U}} = \{\hat{\boldsymbol{u}}_i \text{ for all } n_i \in \mathcal{V}\}$

Encoder:
$\boldsymbol{v}_i^0 = f_V(\boldsymbol{v}_i)$ for all $n_i \in \mathcal{V} \cup \tilde{\mathcal{V}}$
$\boldsymbol{e}_{i\to j}^0 = f_E(\boldsymbol{e}_{i\to j})$ for all $\{n_i \to n_j\} \in \tilde{\mathcal{E}}$ where $i > j$

Processor:
for $k = 1 : K$ **do**
$\boldsymbol{m}_{i\to j}^k = g_E^k\left(\boldsymbol{e}_{i\to j}^{k-1}, \boldsymbol{v}_i^{k-1}, \boldsymbol{v}_j^{k-1}\right)$ for all $\{n_i \to n_j\} \in \tilde{\mathcal{E}}$ where $i > j$
$\boldsymbol{m}_{i\to j}^k = -\boldsymbol{m}_{j\to i}^k$ for all $\{n_i \to n_j\} \in \tilde{\mathcal{E}}$ where $i < j$
$\boldsymbol{v}_i^k = \boldsymbol{v}_i^{k-1} + g_V^k\left(\boldsymbol{v}_i^{k-1}, \sum_{j\in\mathcal{N}_i^{\mathcal{G}}} \boldsymbol{m}_{j\to i}^k\right)$ for all $n_i \in \mathcal{V} \cup \tilde{\mathcal{V}}$
$\boldsymbol{e}_{i\to j}^k = \boldsymbol{e}_{i\to j}^{k-1} + \boldsymbol{m}_{i\to j}^k$ for all $\{n_i \to n_j\} \in \tilde{\mathcal{E}}$ where $i > j$
end for
$\boldsymbol{z}_i^{\text{local}} = (\boldsymbol{v}_i^K, \sum_{j\in\mathcal{N}_i^{\mathcal{G}}} \boldsymbol{e}_{i\to j}^K)$ for all $n_i \in \mathcal{V}$

Decoder:
$\boldsymbol{z}^{\boldsymbol{\theta}} = f_P(\boldsymbol{\theta})$
$\hat{u}_i^d = h^d(\boldsymbol{z}^{\boldsymbol{\theta}}, \boldsymbol{z}^{\text{global}}, \boldsymbol{z}_i^{\text{local}})$ for all $n_i \in \mathcal{V}$ for $d = 1, ..., D$
$\hat{\boldsymbol{u}}_i = (\hat{u}_i^1, ..., \hat{u}_i^D)$ for all $n_i \in \mathcal{V}$

---

total, there are $(2K + 3 + D)$ such MLPs internal to the emulator: two encoder MLPs, f_V and f_E, two processor MLPs for each of K rounds of message passing, $\{g_V^k, g_E^k\}_{k=1}^K$, one global parameter embedding MLP, f_P, and D node-decode MLPs, $\{h^d\}_{d=1}^D$, where D is the dimensionality of the system.

The first stage of the `DeepGraphEmulator` algorithm, the encoder, embeds each node feature vector $\boldsymbol{v}_i$ and edge feature vector $\boldsymbol{e}_{i\to j}$ from the augmented graph representation of the physical system $\tilde{\mathcal{G}}$ into a higher M-dimensional vector space,

$$\boldsymbol{v}_i^0 = f_V(\boldsymbol{v}_i) \text{ for all } n_i \in \mathcal{V} \cup \tilde{\mathcal{V}}, \tag{2.25}$$

$$\boldsymbol{e}_{i\to j}^0 = f_E(\boldsymbol{e}_{i\to j}) \text{ for all } \{n_i \to n_j\} \in \tilde{\mathcal{E}} \text{ where } i > j. \tag{2.26}$$

This embedding increases the expressive power of the final learned node and edge representations. The node encoding is performed using the MLP f_V and the edge encoding with the MLP f_E.

The node and edge embeddings are then updated sequentially in the processor stage over $k = 1, 2, ...K$ message-passing steps. At each step k, the processor first calculates an M-dimensional message vector $\boldsymbol{m}_{i\to j}^k$ for each directed edge $\{n_i \to n_j\} \in \tilde{\mathcal{E}}$. In performing the message-passing, we follow the approach of [160] and enforce the symmetry

$$\boldsymbol{m}_{i\to j}^k = -\boldsymbol{m}_{j\to i}^k \tag{2.27}$$

in the messages passed between each twin pair of directed edges $\{n_i \to n_j\}, \{n_j \to n_i\} \in \tilde{\mathcal{E}}$. This symmetry is inspired by interpreting the messages as *forces* and then incorporating Newton's third law of motion. It is enforced by first computing the message vector for one edge from each twin pair in $\tilde{\mathcal{E}}$ using the MLP g_E^k as follows:

$$\boldsymbol{m}_{i\to j}^k = g_E^k\left(\boldsymbol{e}_{i\to j}^{k-1}, \boldsymbol{v}_i^{k-1}, \boldsymbol{v}_j^{k-1}\right) \text{ for all } \{n_i \to n_j\} \in \tilde{\mathcal{E}} \text{ where } i > j. \tag{2.28}$$

For remaining edges $\{n_i \to n_j\} \in \tilde{\mathcal{E}}$ where $i < j$, the message value $\boldsymbol{m}_{i\to j}$ is found by simply negating the message $\boldsymbol{m}_{j\to i}$ found using g_E^k above:

$$\boldsymbol{m}_{i\to j}^k = -\boldsymbol{m}_{j\to i}^k \text{ for all } \{n_i \to n_j\} \in \tilde{\mathcal{E}} \text{ where } i < j. \tag{2.29}$$

Note that a consequence of this method of enforcing the symmetry in Eq. (2.27) is that the output $\hat{\boldsymbol{U}}$ of the emulator is not invariant under a permutation of the node and edge indices of the input graph $\tilde{\mathcal{G}}$, as is typically the case for GNNs [100]. Permutation invariance and the message-passing symmetry could both be maintained at each step by, for example, using g_E^k to compute an initial message $\acute{\boldsymbol{m}}_{i\to j}^k$ on each edge $\{n_i \to n_j\} \in \tilde{\mathcal{E}}$, before calculating the final message values as $\boldsymbol{m}_{i\to j} = (\acute{\boldsymbol{m}}_{i\to j}^k - \acute{\boldsymbol{m}}_{j\to i}^k)/2$. We took the alternative approach given in Equations (2.28) and (2.29) in this work as it is more computationally efficient, because only half of the messages need to be explicitly computed using g_E^k. The updated node embedding vectors $\boldsymbol{v}_i^k$ are then found as the sum of the embedding from the previous step $k-1$, plus a learned value from the node-update MLP g_V^k,

$$\boldsymbol{v}_i^k = \boldsymbol{v}_i^{k-1} + g_V^k\left(\boldsymbol{v}_i^{k-1}, \sum_{j\in\mathcal{N}_i^{\mathcal{G}}} \boldsymbol{m}_{j\to i}^k\right) \text{ for all } n_i \in \mathcal{V} \cup \tilde{\mathcal{V}}. \tag{2.30}$$

The node-update MLP incorporates the message vectors by summing over all incoming messages to each node $n_i \in \mathcal{V} \cup \tilde{\mathcal{V}}$. The set of all nodes $n_j \in \mathcal{V} \cup \tilde{\mathcal{V}}$ which send a directed edge to node n_i is called its *graph neighbourhood* $\mathcal{N}_i^{\mathcal{G}}$, defined as

$$\mathcal{N}_i^{\mathcal{G}} \triangleq \{n_j \in \mathcal{V} \cup \tilde{\mathcal{V}} \mid \{n_j \rightarrow n_i\} \in \tilde{\mathcal{E}}\}. \tag{2.31}$$

The use of sum aggregation when updating the node representations is again motivated by interpreting the message vectors as forces, and then incorporating the elementary concept in classical mechanics that the net force acting on an object is the sum of all forces acting upon it[3]. The updated edge embeddings $\boldsymbol{e}_{i\rightarrow j}^k$ are found as the sum of the embedding from the previous step $k-1$, plus the computed message value $\boldsymbol{m}_{i\rightarrow j}$,

$$\boldsymbol{e}_{i\rightarrow j}^k = \boldsymbol{e}_{i\rightarrow j}^{k-1} + \boldsymbol{m}_{i\rightarrow j}^k \text{ for all } \{n_i \rightarrow n_j\} \in \tilde{\mathcal{E}}. \tag{2.32}$$

After K rounds of message passing are complete, the final local learned representation $\boldsymbol{z}_i^{\text{local}}$ is found for each real node $n_i \in \mathcal{V}$ as the concatenation of $\boldsymbol{v}_i^K$ with the sum of all edge embedding vectors $\boldsymbol{e}_{i\rightarrow j}^K$ in its local graph neighbourhood $\mathcal{N}_i^{\mathcal{G}}$,

$$\boldsymbol{z}_i^{\text{local}} = (\boldsymbol{v}_i^K, \sum_{j \in \mathcal{N}_i^{\mathcal{G}}} \boldsymbol{e}_{i\rightarrow j}^K) \text{ for all } n_i \in \mathcal{V}. \tag{2.33}$$

The importance of the virtual nodes and edges described in Section 2.2.3 to the processor stage of the emulator is illustrated in Figure 2.6, for the augmented beam geometry from Figure 1. Because messages can only move by one neighbourhood at a time, it can take a large number of steps for information to disseminate across a dense set of real nodes. By contrast, the augmented nodes are less dense, allowing information to be disseminated more efficiently around the graph, as shown for example by the blue arrows.

[3]An alternative message-aggregation function could however be selected here instead, however the selected function must be both permutation invariant and applicable to a variable number of inputs in order to be suitable for graphs with arbitrary topology [100, Section 3.2.2].

The final, decode stage of the emulator makes a prediction $\hat{\boldsymbol{u}}_i$ for the displacement for each real node $n_i \in \mathcal{V}$. This is done by first embedding the global parameters $\boldsymbol{\theta}$ into an M-dimensional vector $\boldsymbol{z}^{\boldsymbol{\theta}}$ using the MLP f_P,

$$\boldsymbol{z}^{\boldsymbol{\theta}} = f_P(\boldsymbol{\theta}). \tag{2.34}$$

Then, a prediction is made for the displacement at each node, using the MLPs $h^1, ..., h^D$ for each of the D dimensions of the system

$$\hat{u}_i^d = h^d(\boldsymbol{z}^{\boldsymbol{\theta}}, \boldsymbol{z}^{\text{global}}, \boldsymbol{z}_i^{\text{local}}) \text{ for all } n_i \in \mathcal{V} \text{ for } d = 1, ..., D. \tag{2.35}$$

The decoder MLPs take as input the global parameter embedding $\boldsymbol{z}^{\boldsymbol{\theta}}$, global shape embedding vector $\boldsymbol{z}^{\text{global}}$, and the local embedding vector $\boldsymbol{z}_i^{\text{local}}$. Note how the node-decode MLP makes a prediction for each node individually, and each dimension individually, without explicitly accounting for any correlation structure. This is because any such structure is implicitly accounted for by the processor stage of the emulator, and can hence be ignored in the final decoder stage. The output $\hat{\boldsymbol{U}}$ of the emulator is then the collection of all individual displacement predictions,

$$\hat{\boldsymbol{U}} = \{\hat{\boldsymbol{u}}_i \text{ for all } n_i \in \mathcal{V}\} \text{ where } \hat{\boldsymbol{u}}_i = (\hat{u}_i^1, ..., \hat{u}_i^D). \tag{2.36}$$

Performing emulation with `DeepGraphEmulator` requires in particular the structure of the internal MLPs, the dimensionality M of the internal embedding vectors, and finally the number of message-passing steps K to be specified. Greater values of M and K allow for more information to be incorporated in the final decoder stage of the emulator. However, this also leads to longer training and prediction times, as well as increases the potential for overfitting. As discussed above, there are $(2K + 3 + D)$ MLPs internal to the GNN. In addition, the outputs of all MLPs with the exception of the final node decode MLPs $\{h^d\}_{d=1}^D$ are processed with a LayerNorm [161]. The trainable parameters denoted $\boldsymbol{\omega}$ of the GNN then consist of the weights and biases of all internal MLPs, as well as the LayerNorm parameters.

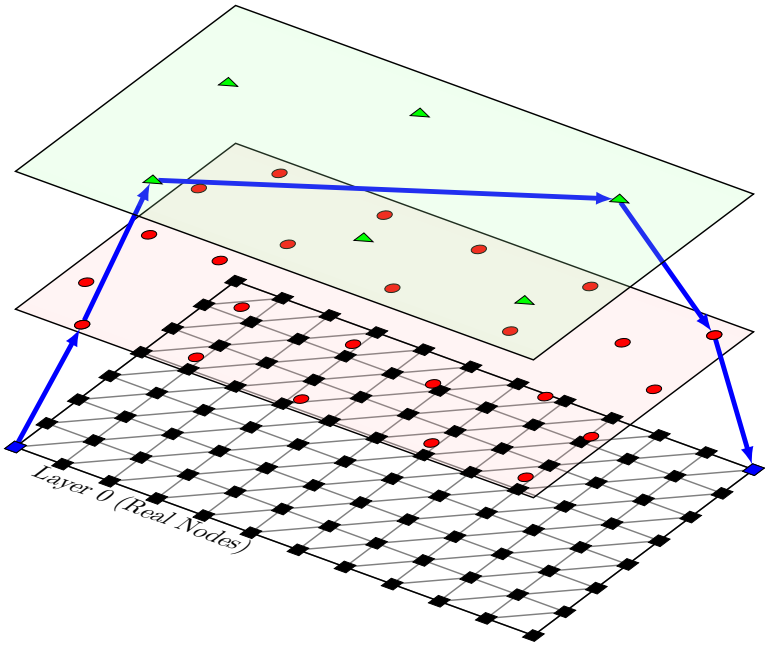


Figure 2.6: A large number of message passing steps may be required to disseminate information around a dense set of FE nodes. Introducing additional layers of coarse, virtual nodes allows information to disseminate more quickly using a "shortcut" through the augmented space (illustrated by the blue arrows).

Before moving on to describe the experimental results for the two systems considered for emulation, we first discuss the motivation behind two of the design choices of the architecture that are particularly advantageous for emulation of LV mechanics. Firstly, note the inclusion of the global shape embedding vector $\boldsymbol{z}^{\text{global}}$ in the decode stage. This allows the decoder MLP h to assimilate information specific to each individual node $n_i \in \mathcal{V}$ ($\boldsymbol{z}_i^{\text{local}}$) together with information about the global shape of the geometry when predicting the forward displacement of the LV, which is useful as each LV anatomy will have a unique shape. Secondly, consider how the parameter vector $\boldsymbol{\theta}$ is only included at the final, decode stage of the GNN. This means that, when considering repeated forward evaluations for a fixed initial geometry under different input parameter values, the encode and process state can be precomputed exactly once initially. Subsequent evaluations then only require the decode step to be performed. Since the processor stage constitutes the bulk of the computational requirements of the emulator, forward evaluations of the surrogate model

can be made highly efficiently in this case. Again, this is useful in LV emulation, as the emulator must deliver significant computational savings over the numerical simulator to be useful for real-time estimation tasks, which in general will be required for the fixed LV geometry of a given subject.

2.3 Beam Deformation Application

We first illustrate the application of the proposed GNN emulation framework to modelling the displacement of a 2-D beam. Specific details of the mechanics problem are given in Section 2.3.1, as well as a description of how training, validation and testing datasets were generated. Section 2.3.2 details how the emulator was implemented, and finally Section 2.3.3 presents and discusses the emulation results.

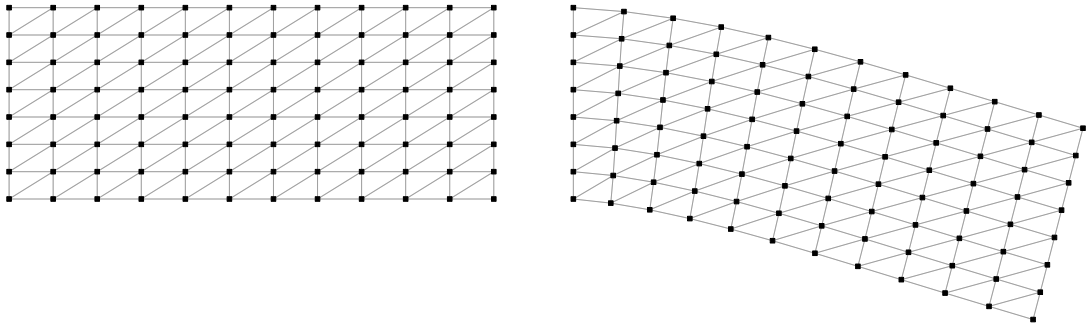
2.3.1 Mathematical Details and Data Generation

In this section, we will introduce a 2-D clamped beam that is deformed under its own weight, this is a simple 2-D linear elasticity model based on a similar example from FEniCS [162]. In brief, Figure 2.7 (a) shows an example of a beam geometry in its reference configuration, while Figure 2.7 (b) displays the beam in its current, displaced configuration with the left side fixed. Denote the Cauchy stress tensor $\boldsymbol{\sigma}$, the linear strain tensor $\boldsymbol{\epsilon} = \frac{1}{2}\left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right)$ where $\boldsymbol{u}$ is the displacement field. By assuming the material is isotropic and linear, we have

$$\boldsymbol{\sigma} = \lambda \operatorname{tr}(\boldsymbol{\epsilon})\boldsymbol{I} + 2\mu\boldsymbol{\epsilon}, \tag{2.37}$$

where $\boldsymbol{\theta} = (\lambda, \mu)$ gives the Lamé parameters, and $\boldsymbol{I}$ is the identity matrix. From Eq. (2.1), the balance equations for this beam (Ω) can be written as

$$\begin{aligned} -\nabla \cdot \boldsymbol{\sigma} &= \boldsymbol{f} \qquad \text{in} \quad \Omega \\ \boldsymbol{u} &= \boldsymbol{0} \qquad \text{at} \quad \partial\Omega^d, \end{aligned} \tag{2.38}$$



(a) Initial State (b) End State

Figure 2.7: Panel (a) shows an example discretised beam geometry in its initial, reference configuration. Panel (b) then shows the end-state of the beam geometry after a simulation following Eq. (2.38) is run.

where $\boldsymbol{f} = (0, -\rho g)$ is the gravity force density with ρ the density of the beam and g the gravity acceleration, and $\partial\Omega^d$ is the clamped end that is the left side of the beam. No traction is applied to the rest of the boundaries. Detailed numerical implementation can be found in the FEniCS documentation [162].

Three hundred individual beam geometries were generated for emulation, all of which shared a common triangular finite-element mesh with 96 nodes and 154 elements. Each beam's length was found by uniformly randomly sampling within the range $[40, 70]$ mm, and its width was set as a randomly chosen fraction of the length within the range [0.20, 0.40]. A forward simulation was run for each beam according to Eq. (2.38), with a uniformly random sample of $\boldsymbol{\theta} \in [4, 10]^2$, where the Lamé parameters are in units of stress [162]. The first 225 simulations were used as training data on which the emulator could be fit, the following 25 for validation and the remaining 50 were reserved as an independent test set. Note that this sampling procedure ensured each data point had unique beam geometry and Lamé parameter values.

2.3.2 Implementation Details

Emulation was performed for several different values of the number of message-passing steps K, dimensionality M of the internal embedding vectors within the GNN, and cardinality of the virtual node layers $\boldsymbol{\eta}$, to quantify the impact of these hyper-parameters on emulation accuracy. In practice, these values could be selected using a held-out validation set of simulations. The global shape embedding input to Algorithm 2 for each individual

beam geometry was set to be the two-dimensional vector giving the beam's length and width. All encoder, processor and decoder MLPs internal to the emulator were implemented with two hidden layers each of width 128, using tanh activation function[4]. For each value of $\boldsymbol{\eta}$ considered, a single augmented graph topology $\tilde{\mathcal{E}}^{Beam}$ for use on all training, validation and test beams was found by applying Algorithm 1 to a reference beam geometry. The reference beam was generated using the "averaging" approach described in Section 2.2.3. Emulation results for an alternative choice of representative graph are given in Section A.1, with the results obtained being very similar to those presented in Section 2.3.3 below. The augmented nodes for each individual training, validation and test beam were then generated in the manner discussed in Section 2.2.3. For all beam graph representations, each edge $\{n_i \rightarrow n_j\}$ where $i > j$ was assigned a feature vector $\boldsymbol{e}_{i\rightarrow j}$ as given in Eq. (2.23). Each real node was assigned a single, Boolean feature variable, which indicated if it lay on the Dirichlet boundary $\partial\Omega^d$ from Eq. (2.38). Augmented nodes were then assigned features sequentially as detailed in Section 2.2.3. Before training, all input features and output displacement values were normalised to mean zero and unit variance. Training was performed using the Adam optimiser [111] with a batch size of one for 300 epochs using a fixed learning rate of 5×10^{-5}, after which the loss value on the validation set was found to plateau. The loss function used for training the GNN parameters $\boldsymbol{\omega}$ was the mean prediction error between nodal displacements outputted by the simulator $\boldsymbol{u}_i$ and those predicted by the emulator $\hat{\boldsymbol{u}}_i$,

$$\mathcal{L}(\boldsymbol{\omega}) = \frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} || \ \boldsymbol{u}_i - \hat{\boldsymbol{u}}_i \ ||_2, \tag{2.39}$$

where $|\mathcal{V}| = 96$ is the number of nodes in the FE representation of each beam. The 25 simulations in the validation dataset were used to monitor emulation performance, with the trained network parameters which gave the lowest error on this data saved for use in predicting the displacements of the 50 test simulations.

[4]Initial experiments with smaller MLPs gave slightly poorer predictive performance. We found two hidden layers of width 128 to be a good comprise between predictive accuracy and computational efficiency.

2.3.3 Results and Discussion

Figure 2.8 compares the test set prediction errors when emulation is performed on the non-augmented graph representations of the beam geometries against results on the augmented graphs, for $K = 1, ..., 6$ message passing steps[5]. When operating on the non-augmented graphs, at least $K = 5$ message passing steps are required to ensure the bulk of emulation errors are less than 0.1 mm, whereas almost identical performance can be obtained in only $K = 2$ steps when using the augmented representations. For reference, the average magnitude of the true nodal forward displacement vectors across the test data was 4.2 mm. Figure 2.9 (a) displays emulation errors for different values of virtual node cardinality input vector $\boldsymbol{\eta}$, with results exhibiting low sensitivity with respect to this hyperparameter vector, and panel (b) displays errors for different choices of latent embedding dimension M. Errors tend to fall as M increases, however for $M = 60$, we see a small increase in the distribution of tail prediction errors which could be evidence of slight overfitting. By using arguments similar to Takens' embedding theorem for dynamical systems (see e.g. [163]), it is reasonable to assume there exists an optimal value for M, below which insufficient information underfitting may occur and beyond which more scope for overfitting is introduced.

The disparity between the augmented and non-augmented emulation errors for low values of K from Figure 2.8 exists because the emulator requires higher values of K to distinguish between the internal nodes of the beam geometries when using the non-augmented graph representations. In particular, due to the regularity of the nodes in each beam geometry, as illustrated in Figure 2.7 (a), any nodes more than K steps away from a boundary node will have the exact same local information available to them after K message passing steps. This means they will share a common value of $\boldsymbol{z}_i^{\text{local}}$ (Eq. (2.33)) and hence will be indistinguishable to the node-decode MLPs (Eq. (2.35)). Even then for this 2-D beam model, for a high fidelity FE mesh a large value of K would be required to discriminate between internal nodes and achieve accurate emulation. By introducing virtual nodes and edges, this issue is alleviated. The use of augmented graph representations

[5]For ease of reference, all beam emulation results are tabulated in Section A.1.

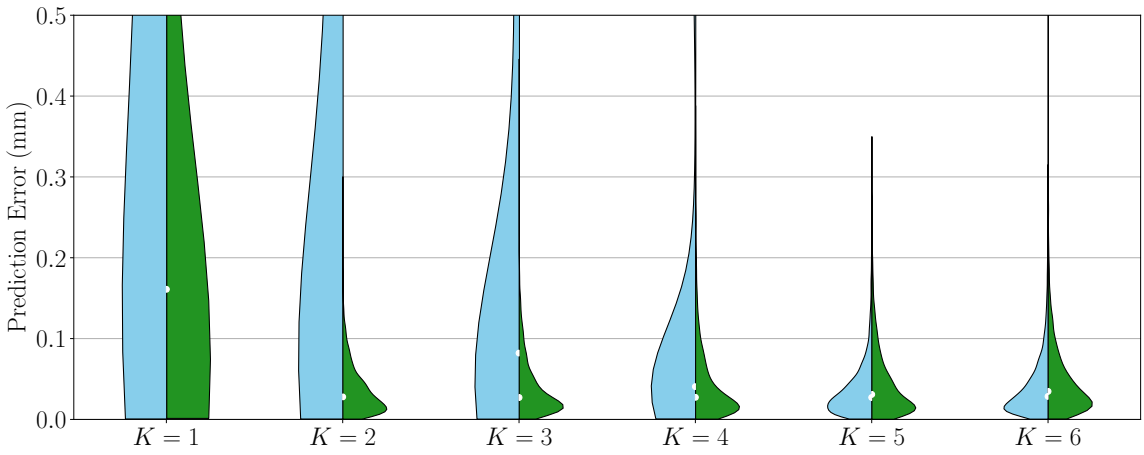


Figure 2.8: Violin plots of test-set nodal GNN emulation prediction errors (in mm) for $K = 1, ..., 6$ message passing steps with the dimensionality of the internal embedding vectors $M = 40$. The left hand side of each violin plot in blue gives the distribution of errors when the non-augmented graph representations are used for emulation, that is without any virtual nodes or edges. The right hand side of each plot in green shows the error distribution when the augmented graphs are used, generated using Algorithm 1 with nearest neighbour parameter $\kappa = 4$ and node cardinality vector $\boldsymbol{\eta} = [24, 6]$.

has other advantages also, beyond what can be seen in this simple example. For instance, when considering simulations involving the contact of different bodies, the virtual nodes and edges allow the contact information to be rapidly propagated across the entire body under contact [140].

The top row of Figure 2.10 compares the predictions of the GNN emulator with $K = 2$ against the outputs of the simulator for two beam geometries from the independent test set. Note the difference in size between the two beams, which is a result of the random sampling approach used to generate them. The beam in the left column was the simulation for which the emulator obtained the first-quartile value of mean prediction error (0.03 mm), and the beam on the right the third quartile error value (0.05 mm). For both simulations, the emulator delivers accurate predictions across the entire geometry. The violin plots from Figure 2.8 illustrate the accuracy / efficiency trade-off of the choice of K. That is, K must be chosen such that the node-decode MLPs have sufficient information to make accurate predictions. However, higher values of K also introduce the potential for the network to overfit to its training data. In this case, as more than $K = 2$ steps of message passing are performed using the augmented graph representations, the emulator begins to overfit. Nevertheless, the exact value of K is far less critical than when perform-

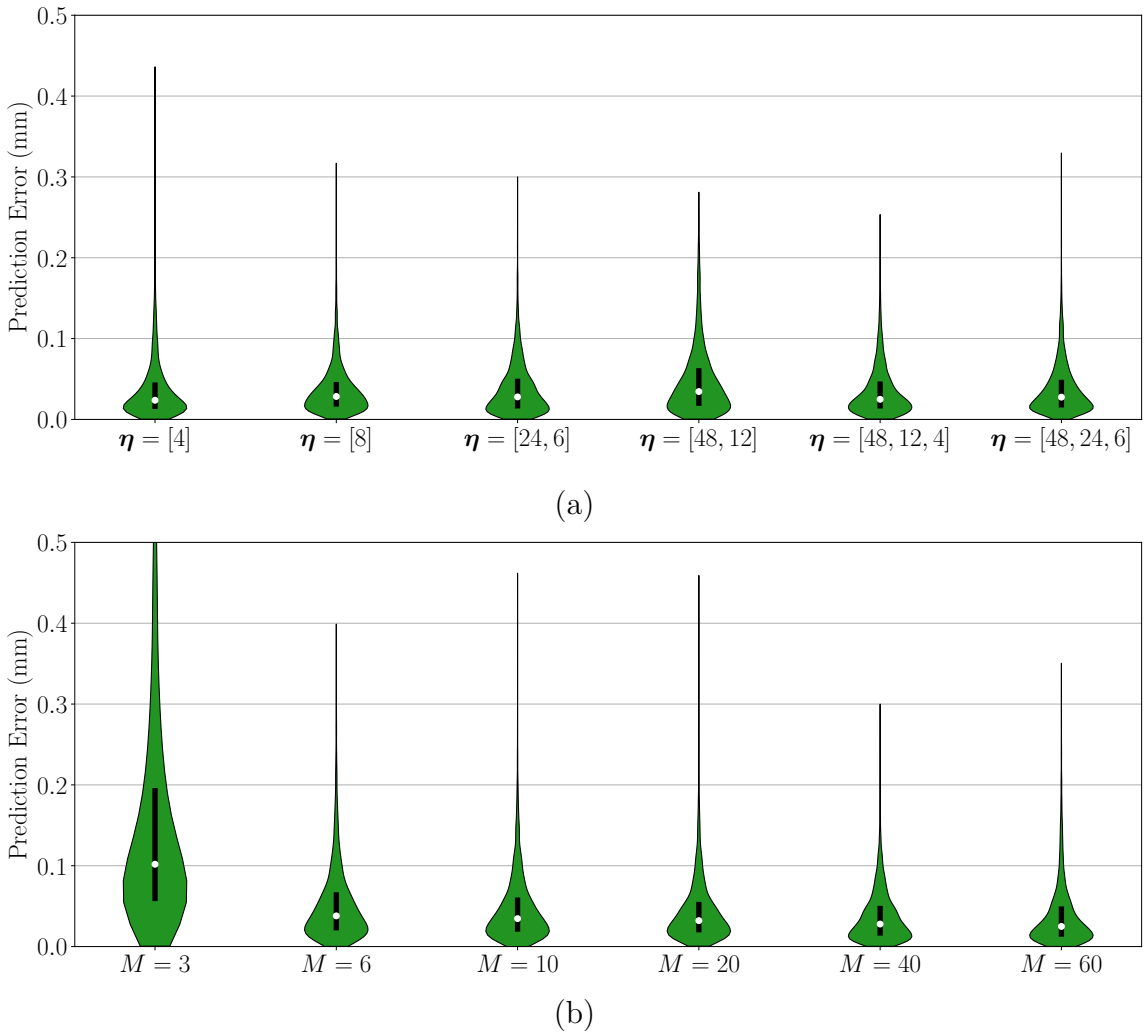


Figure 2.9: Violin plots of test-set nodal GNN emulation prediction errors for different hyperparameter input values. Panel (a) shows the distribution of errors for six different values of $\boldsymbol{\eta}$, with $K = 2$ and $M = 40$. Panel (b) plots the distribution of emulation error for $M = 3, 6, 10, 20, 40, 60$, with $K = 2$ and virtual node layer cardinality vector $\boldsymbol{\eta} = [24, 6]$.

ing emulation using the non-augmented graphs. The overfitting effect is visualised in the final two rows of Figure 2.10, which compares the predictions of the GNN emulator with $K = 2$ against an emulator with $K = 6$. Despite the slight overfitting, the predictions of the two emulators appear qualitatively similar, with increased deviations for nodes farther from the clamped left side.

Finally, Figure 2.11 displays the node coordinate values $\{\boldsymbol{x}_i\}_{i=1}^{|\mathcal{V}|}$ of the same geometries from Figure 2.10 in the top row of panels, while the bottom row displays two-dimensional projections of the corresponding local embedding vectors $\{\boldsymbol{z}_i^{\text{local}}\}_{i=1}^{|\mathcal{V}|}$. The embeddings were generated using the GNN with $M = 40$ (which implies that each $\boldsymbol{z}_i^{\text{local}}$ was of length 80) and $K = 2$. The projection values were found using Principal Component Analysis

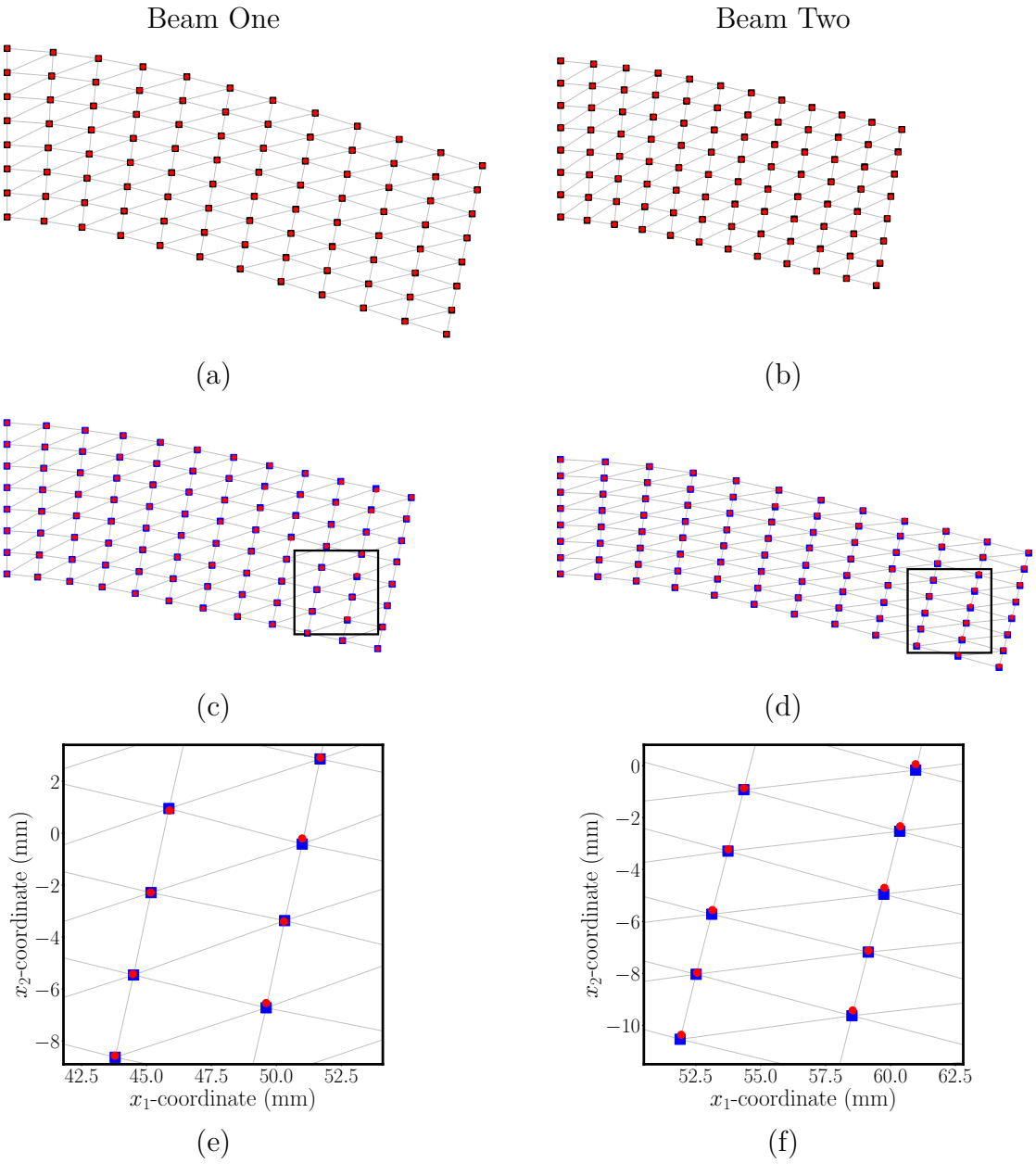


Figure 2.10: Illustration of emulation performance for two test-set beam simulations. The beam in the left column had an initial width/height of 56.5/22.0 mm, and the simulation was run with Lamé parameters $\lambda = 7.03$, $\mu = 5.94$. The beam in the right column had initial dimensions of 66.1/16.7 mm with $\lambda = 6.52$, $\mu = 5.98$. The top row of plots shows the predictions of the GNN emulator with $K = 2$ rounds of message passing as red circles against the outputs of the simulator as black squares. The second row displays emulator predictions with $K = 2$ again as red circles, against predictions with $K = 6$ as blue squares. The final row of figures displays an enlarged version of the nodes inside the black squares from the figures in the second row, with coordinate values provided for reference.

(PCA). The PCA projection matrix was learned using as input data the collection of individual values of $\boldsymbol{z}_i^{\text{local}}$ across all the beams in the test data. The first two components, labelled PC 1 and PC 2, were found to explain over 44% of the variance in this data. The columns of the nodal coordinates in geometry space are coloured with a shading that is shared with the PCA projections, allowing for clusters in the projection values to be compared with coordinate position values. In panel (c), four distinct node clusters can be seen. Firstly, the boundary nodes in black are clustered far away from all other nodes, with high values of PC 1 and low values of PC 2. The second cluster consists of blue nodes near the left hand clamped boundary of the beam, which congregate at the top left of the scatter plot. The third cluster near the top centre of the plot consists mainly of red nodes from the right hand side of the beam. The final cluster consists of mostly nodes from the middle of the beam geometry which have low values of PC 1 and PC 2. Interestingly, the blue nodes which are nearest the clamped boundary in geometry space are furthest from the clamped boundary nodes in the projected space. The results in panel (d) are very consistent in relative terms with panel (c), while in absolute terms the projections in (d) tend to have slightly lower PC 1 value.

2.4 Passive Left Ventricle Mechanics Application

This section describes the application of the proposed GNN emulation framework to the mechanics of the left-ventricle (LV) in diastole, the passive stage of the cardiac cycle. Mathematical and numerical details of the mechanics problem are given in Section 2.4.1. Section 2.4.2 describes the creation of simulation data on which the emulator was trained, validated and tested. A summary of existing work on the emulation of cardiac-mechanics is given in Section 2.4.3 before implementation details are given in Section 2.4.4. Emulation results are then presented in Section 2.4.5 and finally discussed in Section 2.4.6.

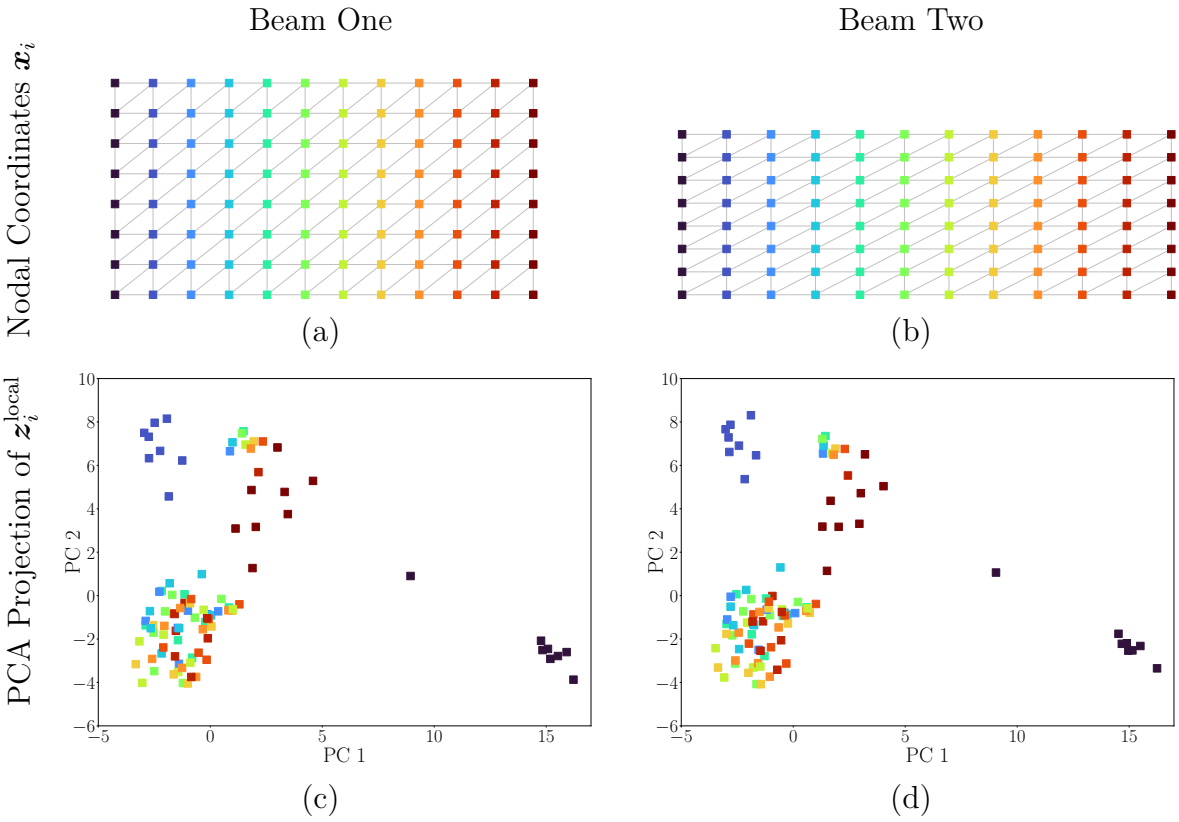


Figure 2.11: Comparison of nodal coordinates $\{\boldsymbol{x}_i\}_{i=1}^{|\mathcal{V}|}$ in geometry space (top row of panels) against two dimensional projections of the local embedding vectors $\{\boldsymbol{z}_i^{\text{local}}\}_{i=1}^{|\mathcal{V}|}$ (bottom row of panels), for two beam geometries. The projection values were found using Principal Component Analysis (PCA). The columns of the nodal coordinates in geometry space are coloured with a shading that is shared with the PCA projections, allowing for clusters in the projection values to be compared with coordinate position values.

2.4.1 Mathematical Details

The passive myocardium is modelled as an incompressible, hyperelastic and transversely isotropic material, which is reduced from the Holzapfel-Ogden HO model [75], motivated by the results of previous uncertainty quantification study [164]. The adopted strain energy density function is given by

$$\Psi = \frac{a}{2b}\left(e^{b(I_1-3)} - 1\right) + \frac{a_\text{f}}{2b_\text{f}}\left(e^{b_\text{f}(\max(I_{4\text{f}},1)-1)^2} - 1\right) + \frac{1}{2}\mathcal{P}(J-1)^2, \tag{2.40}$$

where $\boldsymbol{\theta} = (a,\ b,\ a_\text{f},\ b_\text{f})$ are the material parameters. Parameters a and b describe the isotropic response of the myocardium, a_f and b_f characterise the reinforced stiffness in the myofibre direction. The principal invariant I_1 and transversely isotropic invariant $I_{4\text{f}}$ are defined as $I_1 = \text{trace}(\boldsymbol{C})$ and $I_{4\text{f}} = \boldsymbol{f}_0 \cdot (\boldsymbol{C}\boldsymbol{f}_0)$, with $\boldsymbol{C} = \boldsymbol{F}^\text{T}\boldsymbol{F}$ the right Cauchy-Green

tensor, $\boldsymbol{F}$ the deformation gradient, and $\boldsymbol{f}_0$ the unit vector in the myofibre direction in the reference configuration. The max() term in Eq. (2.40) ensures that the myofibres only support extension but not compression, while the final term in the equation enforces the incompressibility constraint with the Lagrange multiplier $\mathcal{P}$ and $J = \det(\boldsymbol{F})$.

The filling process of the LV in diastole is described by a quasi-static, pressure-loaded boundary-value problem over the computational domain occupied by the geometry of the LV. A linearly ramped pressure from 0 to 8 mmHg is applied to the endocardial surface (Γ^{endo}), and the basal plane of the geometry is fixed in the longitudinal and circumferential directions, with only radial expansion allowed during the filling process. Note a cylindrical coordinate system is introduced to the nodes in the basal plane. The system of equations at the current configuration (Ω) is given by:

$$\begin{cases} \nabla \cdot \boldsymbol{\sigma} = 0 & \text{in } \Omega, \\ \boldsymbol{\sigma} \cdot \boldsymbol{n} = -p\,\boldsymbol{n} & \text{on } \Gamma^{\text{endo}}, \\ \mathrm{u}_\theta = \mathrm{u}_z = 0 & \text{on } \Gamma^{\text{base}}, \end{cases} \tag{2.41}$$

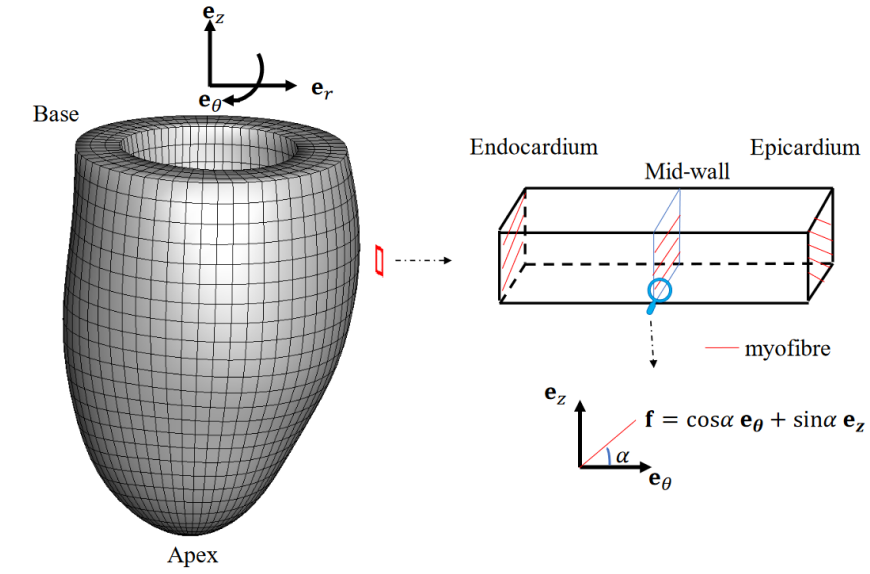
where $\boldsymbol{n}$ is the normal direction of the endocardial surface Γ^{endo} and p is the loaded pressure at Γ^{endo}. The vectors u_θ and u_z are the displacement components along the circumferential direction and z-axis at the basal surface Γ^{base}, respectively. The above FE-based LV model (Eq. (2.41)) is then solved using ABAQUS (Simulia, Providence, RI, USA), a general-purpose finite-element package. Further details of the FE simulation of the passive LV dynamics are given in [165].

Note that here we restrict our analysis to the passive stage of the cardiac cycle. However, emulation could also be used to model the active stage of the cardiac cycle, when the LV contracts. For instance, see the recent work by Naghavi et al. (2024) [166].

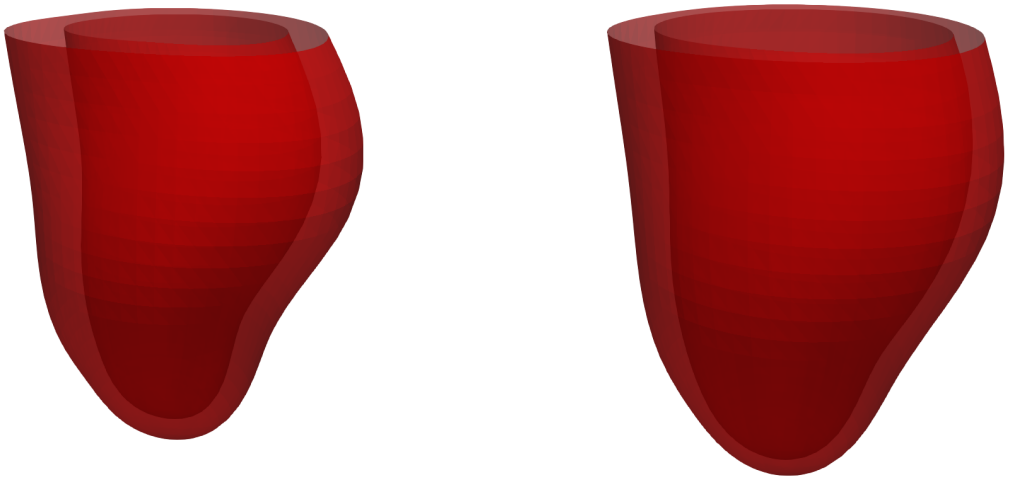
2.4.2 Simulation Data Generation

The emulation study was conducted using 3000 randomly generated, synthetic LV geometries. Synthetic geometries were used in the absence of a large dataset of real geometries. For full details of the generation procedure, see Appendix A.3. In brief, the synthetic LV geometries were generated by sampling in a 40-dimensional latent space. This space was found by applying principal component analysis (PCA) to a population of 65 real LV geometries, extracted from cardiac imaging scans. The variation of the sampler along each axis was set to be slightly more than the empirical variation observed on the real data. The advantage of using a high-dimensional latent space is that the synthetic geometries exhibited a wide variation of size, wall thickness and skewness, for example. However, they were not guaranteed to be physically realistic. For example, a geometry can be generated where the inner and outer wall intersect. To prevent such problems, we applied a post-processing step after geometry generation to remove such unrealistic geometries. Each LV geometry was represented using a three-layer, hexahedral mesh with 6874 nodes and 4995 finite-elements, with myofibre structure described by a rule based method (RBM) [165]. This mesh is slightly less dense than would be considered in practical cases involving real patients (see the experiments in [92] for example). We used a less dense mesh here to accelerate the extensive numerical experiments we performed.Figure 2.12 (a) illustrates both the form of the hexahedral mesh and the RBM for fibre generation.

To create simulation datasets on which the GNN emulator could be trained, validated and tested, a forward mechanics simulation was run for all 3000 synthetic LVs. Panel (b) in Figure 2.12 shows an example LV geometry in its reference configuration, while panel (c) shows the form of the geometry after a forward simulation is run. Each simulation was performed with a unique value of the material parameter vector $\boldsymbol{\theta} = [a, b, a_{\mathrm{f}}, b_{\mathrm{f}}]$. The parameter values were found by running a Sobol sequence of length 3000 within the set $[0.1, 10]^4$, where a and a_{f} are measured in kPa while b and b_{f} are dimensionless quantities. A Sobol sequence is a low-discrepancy sequence, commonly used in the design of computer experiments to ensure a space-filling coverage of the experimental domain of interest [167]. The Sobol sequence was run on the *log-scale* rather than the uniform scale. Sampling on the log-scale favours material parameter vectors $\boldsymbol{\theta}$ with lower stiffness



(a)



(b) Initial State (c) End State

Figure 2.12: Panel (a) shows the hexahedral FE mesh structure of a LV along with the RBM generated myofibre field, where myofibre orientation varies smoothly from endocardium to epicardium. Panel (b) shows the endocardial and epicardial surfaces of an example LV geometry in its initial reference configuration. Panel (c) then shows the deformed LV geometry after a simulation following Eq. (2.41) is run.

levels, which results in an increase in the magnitude of the simulated displacement of the myocardium from start to end-diastole. By contrast, samples on the uniform scale result in displacements that are smaller in magnitude than would be expected for real data [164]. The first 2250 simulation results were used as training data, and the following 150 were used as a validation set. The remaining simulation results were used as a test set. Note that 29 of the forward mechanics simulations failed to converge. Furthermore, some of the remaining simulations exhibited extremely large displacements beyond what would

be expected to be observed in-vivo. For this reason, we excluded from the test set any simulation results where the cavity volume of the deformed geometry exceeded 850 mL, which left 567 test data points. For reference, the mean end-diastolic volume for healthy males has been found to be 160 mL, with standard deviation of 27 mL [168, Table 2].

An additional dataset of 350 simulations was generated for a fixed LV geometry, randomly chosen from the test set. Again, each simulation was run with a different material parameter input vector $\boldsymbol{\theta}$, the values of which were found by continuing the Sobol sequence beyond the last index from the above data. The fixed LV geometry simulations were used to quantify how *transfer learning* on a specific geometry of interest can affect emulation performance. Transfer learning involves taking a pre-trained machine learning model and fine-tuning it for a new, but related application domain [169]. Details of the transfer-learning approach used in this instance are given in Section 2.4.4.2. The first 85 simulations were used for training, with 15 reserved for validation. Four simulations failed to converge, leaving 246 test points.

2.4.3 Existing Work on Cardiac-Mechanic Emulation

Emulation has been widely applied in LV mechanics, due to the numerical expense incurred in simulating soft-tissue mechanical systems. As discussed in Section 2.2.2 however, traditional emulation methods such as MLPs are not suited to emulating LV mechanics due to the excessive dimensionality of the simulations that are required for high fidelity results. Instead, a common approach in the literature has been to emulate certain numerical properties of the displaced geometry outputted by the simulator, such as LV inner cavity volume (LVV), instead of directly emulating the displacement of the myocardium itself. Consider for example the additional simulation data described in the last paragraph of Section 2.4.2, that were created for a fixed LV geometry. If we calculate the LVV resulting from each simulation, a dataset where inputs are the four material parameters in $\boldsymbol{\theta}$ and outputs being scalar LVV can be created, which is appropriate to model with a traditional surrogate modelling technique. This is the approach taken in [170], where Gradient Boosted trees were used to emulate blood pressure, myocardial stresses and LVV through the full cardiac cycle. In [171], polynomial chaos emulators were used

to conduct a sensitivity analyses of the passive mechanics of the LV, considering outputs such LVV and apex lengthening. Finally, [135] and [172] used Gaussian process emulators for LVV and circumferential stains to perform fast parameter inference for the material properties of the underlying constitutive law.

All of the above works performed emulation for a single, fixed LV geometry. In order for a traditional emulation method to be able to generalise to different LV anatomies, a low-order representation of the true LV is required as input to the emulator. This representation is typically found using PCA. In [89] for example, a 5 dimensional representation of the LV geometry was found using PCA, while in [86] a 2 dimensional PCA representation of the LV was used. Low-order parameterised representations for the LV geometry have also been considered for training emulators [87]. However, [173] has shown that such parameterised methods incur higher reconstruction error than PCA.

To benchmark the emulation performance of the GNN surrogate model, we compare its predictive accuracy for LVV with that of an MLP emulator on the test data set of 567 simulations described in Section 2.4.2. The MLP learns a forward map of the form

$$\left(\boldsymbol{\theta}, \boldsymbol{z}^{\text{global}}\right) \mapsto \text{LVV} \tag{2.42}$$

on the training data, whereby a prediction of LVV is made directly given input values for the global material parameters $\boldsymbol{\theta}$ and global shape coefficients $\boldsymbol{z}^{\text{global}}$ respectively. We have previously found that MLPs achieve strong performance for emulation of LV mechanics [2]. In addition, the computational efficiency of MLPs allows for gradient-based MCMC approaches to be used for inverse problems, see [89] for example.

2.4.4 Implementation Details

Emulation for the LV model in diastole was performed using the same augmented graph topology for each LV geometry, denoted $\tilde{\mathcal{E}}^{LV}$. A common augmented topology was used as all LV geometries share a common FE mesh structure and hence share a common non-augmented graph topology, $\mathcal{E}^{LV}$, which is derived as in Eq. (2.8). $\tilde{\mathcal{E}}^{LV}$ was generated by applying Algorithm 1 to a reference graph with nodes $\bar{\mathcal{V}}$ and graph topology $\mathcal{E}^{LV}$, where $\kappa = 5$ and $\boldsymbol{\eta} = [813, 126, 13]$. The node set $\bar{\mathcal{V}}$ was found using the "averaging"

approach discussed in Section 2.2.3. The value of κ was chosen so virtual nodes would have approximately the same number of intra-layer neighbours in $\tilde{\mathcal{E}}^{LV}$ as each real node has neighbours in the FE mesh (5.46 on average). The value of $\boldsymbol{\eta}$ was selected so that the cardinality of each successive layer of virtual nodes decreased by approximately a factor of 8, which is the number of real nodes in each hexahedral finite-element. Note that the usual Euclidean metric was used for the operations performed in Algorithm 1. Performance could potentially be improved by instead using a metric which accounts for the non-convexity and curvature of the myocardium, however we leave this to future work. Each real node $n_i \in \mathcal{V}$ was assigned a feature vector $\boldsymbol{v}_i$ for processing by the emulator, as discussed in Section 2.2.3. The feature vector for each node included two Boolean indicator variables, which were equal to one if the node lay on Γ^{base} or Γ^{endo} respectively, and zero otherwise. Additionally, the 3-D fibre orientation vectors generated by the rule-based method at each node were assigned to $\boldsymbol{v}_i$. Finally, each node feature further included the radial direction of the underlying node in the FE mesh (the vector $\mathbf{e}_r$ in Figure 2.12 (a)), which helped the emulator learn the boundary condition on Γ^{base}. Edge features were assigned as in Eq. (2.23). Emulation was performed with fixed $M = 40$, which was the value that gave the lowest prediction error for the beam dataset. The structure of the MLPs internal to the GNN was the same as described in Section 2.3.2, as were the training details, with the exception that training was performed using a fixed learning rate of 5×10^{-5} for 3000 epochs, after which loss on the validation dataset was found to plateau. Because the LV geometries under analysis are synthetic, generated by sampling coefficients in a 40-dimensional latent space obtained using PCA (see Section A.3 for details), we used these principal component coefficients for input to the emulator as the global embedding vector $\boldsymbol{z}^{\text{global}}$. Note that for real LV geometry data, the global coefficients could be found by applying the PCA projection matrix as in Eq. (A.1). To analyse the impact of this hyper-parameter on emulation performance, emulation was performed with $\{0, 8, 16, 24, 32, 40\}$ PC coefficients inputted to $\boldsymbol{z}^{\text{global}}$. Emulation performance was also quantified for $K = \{1, 2, 3, 4, 5, 6\}$. Preliminary experiments with regularisation techniques including weight decay and dropout yielded no increase in emulation accuracy.

Note that our use of a stochastic training approach (Adam [111] with batch size of one) can be seen as an *implicit* regularisation method. This is because stochastic training is practically similar to training with added noise, which can help prevent overfitting [174] and is equivalent to Tikhonov regularisation [175].

2.4.4.1 MLP Emulator Implementation Details

As discussed in Section 2.4.3, we compare the performance of the GNN emulator in predicting LVV on the varying geometry dataset against an MLP emulator, which directly predicts LVV using a forward map of the form of Eq. (2.42). The MLP was implemented with two hidden layers, each of width 128, and tanh activation function. A point estimate for the weights and biases was found by optimising the squared error between the true LVV values and those predicted by the MLP on the training data set of 2250 simulations. Optimisation was performed using the Adam algorithm [111] with a batch size of 200 for 1000 steps, after which validation set loss was found to plateau. We have found that the predictive accuracy of the MLP emulator can be significantly improved by performing pre-transformations of the two inputs $\boldsymbol{\theta}$ and $\boldsymbol{z}^{\text{global}}$ before training and evaluation. Firstly, the material parameters $\boldsymbol{\theta}$ were inputted on normalised *log-scale*, rather than uniform scale. Secondly, the individual principal component coefficients in $\boldsymbol{z}^{\text{global}}$ were not normalised independently before training. Instead, all coefficients were normalised with respect to the variance in the training data along the direction of the *second* principal component. These transformations were informed by our previous experience in emulating LV mechanics [2].

2.4.4.2 Transfer Learning Details

The GNN emulator is designed to be able to perform forward evaluations (without retraining) for arbitrary values of the LV geometry and material parameter inputs respectively. In practice however, particularly in clinical applications, predictions for different material parameters may only be required for one specific LV geometry of interest. In this case, the original emulator can be fine-tuned using additional forward simulations of this specific LV geometry, which is referred to as transfer learning. Transfer learning is more efficient and can lead to more accurate predictions than re-training the emulator from scratch

on the additional simulations [169]. We performed transfer learning by taking a GNN emulator that was pre-trained on the original data, and fine-tuning its parameters on the dataset of 85 forward simulations for a randomly chosen, fixed LV geometry. See Section 2.4.2 for details of this fixed geometry dataset. The transfer learning was performed with the Adam optimiser, using a learning rate of 1×10^{-5} for 1500 epochs. We used a lower learning rate than used on the original dataset to decrease the stochasticity observed in the trace plots of the loss on the validation set of 15 simulations, however we found that the learning rate does not have a significant impact on emulation performance. Furthermore, only the weights and biases of the three node-decode MLPs in the final stage of the GNN were updated during the transfer learning. All other trainable parameters were held fixed to the values determined by the original dataset.

2.4.5 Results

2.4.5.1 General Dataset

Figure 2.13 displays violin plots of the test-set nodal prediction errors (in mm) for the GNN emulator for different hyperparameter input values[6]. Panel (a) shows the distribution of errors with $0, 8, 16, 24, 32$ PCs retained in $\boldsymbol{z}^{\text{global}}$ with $K = 5$ message passing steps. We see that emulation error decreases monotonically as more components are included. Panel (b) shows the distribution of errors for $K = 1, ..., 6$ with 32 coefficients retained in $\boldsymbol{z}^{\text{global}}$. Again we see a decrease in error as more message passing steps are performed, but this effect is less pronounced than in panel (a). For subsequent experiments, we proceeded with the GNN emulator with $K = 5$ and 32 PC coefficients retained in $\boldsymbol{z}^{\text{global}}$. This emulator obtained a median nodal-prediction error of 0.27 mm on the independent test set, with interquartile range $[0.15, 0.50]$ mm. For reference, the average magnitude of the true nodal forward displacement vectors across the test data was 11.0 mm. We chose $K = 5$ as the difference in emulation performance was marginal compared with $K = 6$. We do not retain all 40 PCs in $\boldsymbol{z}^{\text{global}}$, as we believe this better represents the case for real data where a perfect low order representation of the LV anatomy will not be available.

[6]For ease of reference, all LV emulation results are tabulated in Section A.2.

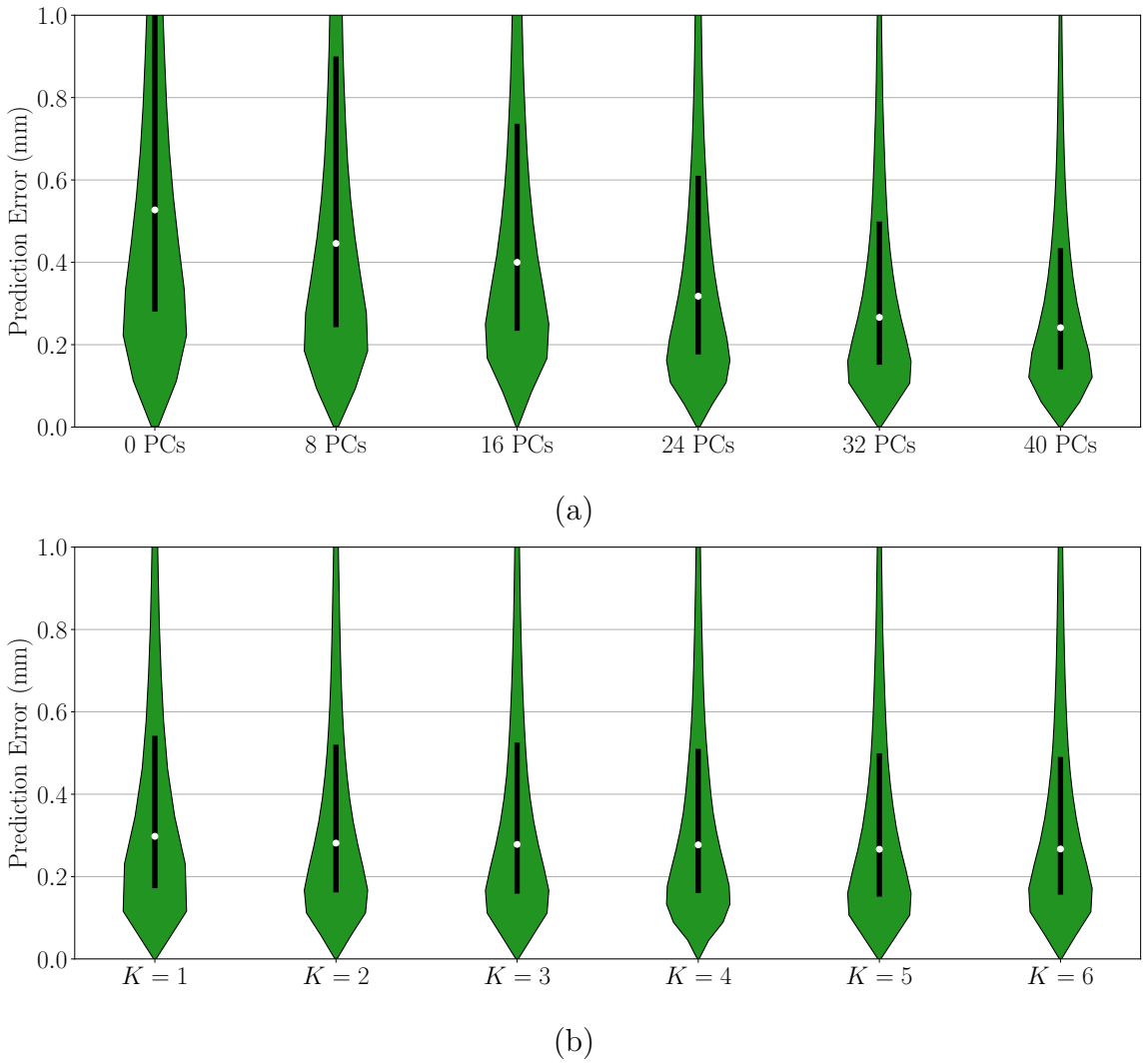


Figure 2.13: Violin plots of test-set nodal GNN emulation prediction errors (in mm) for different hyperparameter input values. Panel (a) shows the distribution of errors with $\{0, 8, 16, 24, 32\}$ PCs retained in $\boldsymbol{z}^{\text{global}}$ using $K = 5$ message passing steps. Panel (b) shows the distribution of errors for $K = 1, ..., 6$ with 32 PCs retained in $\boldsymbol{z}^{\text{global}}$.

2.4.5.2 Comparison with MLP Emulation

Figure 2.14 presents a comparison of test set LVV prediction errors for the MLP and GNN emulators, measured in terms of the absolute percentage error between the predicted and true LVV values. The GNN emulators predict the value of LVV by explicitly calculating the volume inside the deformed endocardial surface derived from the forward prediction of the GNN. The MLP emulators directly predict the value of LVV at end-diastole with a forward map following Eq. (2.42), where 32 PCs were retained inside $\boldsymbol{z}^{\text{global}}$. The violin plots of MLP 1 and GNN 1 give emulation results when the pre-transformations of inputs $\boldsymbol{\theta}$ and $\boldsymbol{z}^{\text{global}}$ detailed in Section 2.4.4.1 are *not* applied before training, while the violin plots

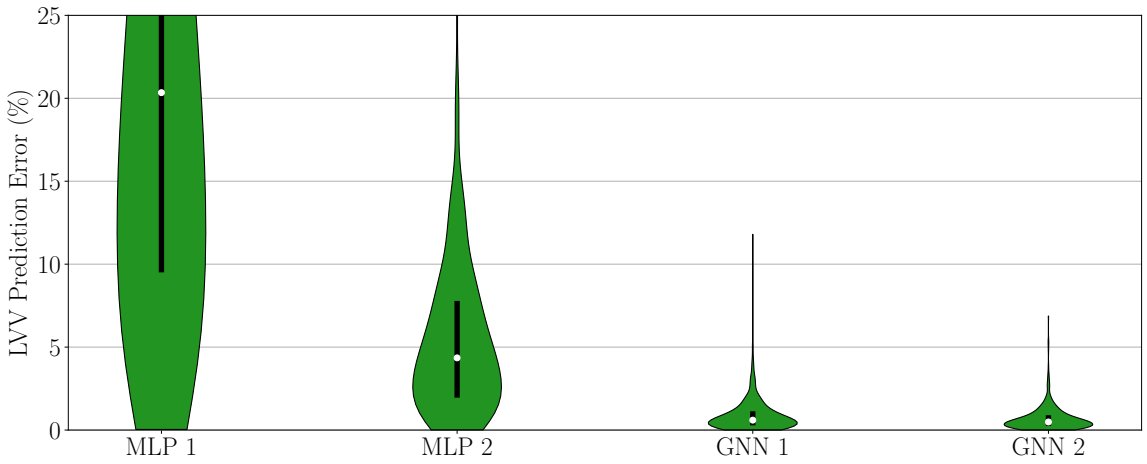


Figure 2.14: Comparison of test set LVV prediction errors for MLP and GNN emulation. Performance is evaluated in terms of the absolute percentage error between the predicted and true LVV values. The violin plots of MLP 1 and GNN 1 give emulation results when the pre-transformations of inputs $\boldsymbol{\theta}$ and $\boldsymbol{z}^{\text{global}}$ detailed in Section 2.4.4.1 are *not* applied before training. The violin plots of MLP 2 and GNN 2 then give the emulation results when these transformations are applied.

of MLP 2 and GNN 2 give the emulation results when these transformations are applied. Without the input transformations, the MLP incurs very large errors in the prediction of LVV, with median absolute error of 20.3%. When the input transformation is performed, error for the MLP approach is significantly reduced to interquartile range $[1.95\%, 7.8\%]$. However, these errors are still substantially larger than those of the two GNN emulators. In particular, GNN 2 achieves median prediction error of 0.49% and interquartile range $[0.24\%, 0.89\%]$. This again presents an improvement over the case where the input pre-transformations are not applied, however the difference however is much less pronounced in this case. GNN 2 achieves achieves prediction errors approximately one order of magnitude lower than MLP 2. In addition, the prediction errors of the GNN were less heavily skewed - the largest error for the GNN was 6.8%, whereas the MLP incurred an error of over 50% for one test point. This outlier point was run with a large value of material parameter a (9.03 kPa) and very low value of b (0.10). For reference, the error of the GNN emulator for this point was 1.6%.

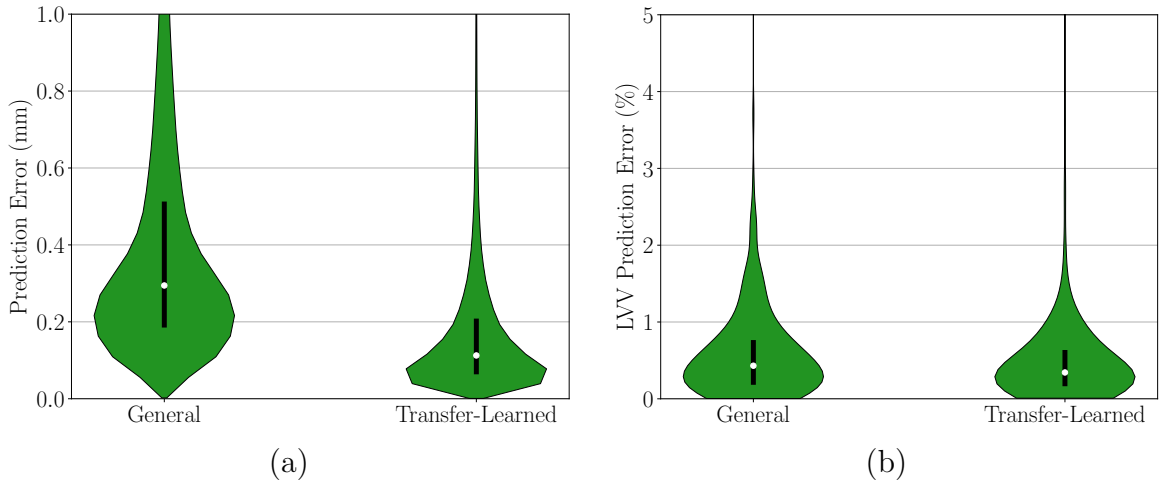


Figure 2.15: Comparison of emulation performance of the general emulator and the emulator transfer-learned on the specific LV geometry of interest. Panel (a) shows violin of prediction errors in the nodal displacement of the deformed geometry (mm), and (b) shows violin plots of errors in the prediction of LVV (%).

2.4.5.3 Transfer Learning on a Fixed LV Geometry

We also explore the effect of transfer-learning on the accuracy of the GNN emulator. Initialising on the trained parameter values from the above GNN emulator with $K = 5$ and 32 PCs retained in $\boldsymbol{z}^{\text{global}}$, we transfer-learned the GNN on a set of 85 simulations for a randomly chosen, fixed LV from the held out set of 567 geometries. Figure 2.15 (a) compares the test-set nodal prediction accuracy of the original and transfer-learned emulators and Figure 2.15 (b) compares their predictive accuracy for LVV. Nodal prediction errors tend to fall by greater than a factor of 2, with the transfer-learned emulator incurring a median node prediction error of 0.11 mm compared to 0.29 mm for the original emulator. The increase in LVV prediction accuracy was more slight, with the transfer-learned emulator achieving a median error of 0.34%, compared to 0.43% for the original emulator.

To further characterise emulation performance, we consider the distribution of prediction errors across the LV geometry surface in Figures 2.16. We selected for plotting the three test simulations where the transfer learned GNN attained the 25^{th} percentile (0.2%), median (0.3%) and 75^{th} percentile (0.6%) error values in the prediction of LVV. The original GNN incurred errors of 1.4%, 0.1% and 0.2% respectively for these simulations. Figure 2.16 displays the three ground truth LV geometries outputted from the simulator in red, against the predictions of the original and transfer-learned GNNs respectively in

blue. The first row is for the predictions of the original GNN, and the second for the transfer-learned GNN. The first column shows the simulation where the transfer-learned surrogate achieved the 25^{th} percentile error in LVV, the second the median, and the third the 75^{th} percentile. For the original GNN, we tend to see slightly higher prediction errors towards the apex of the LV in panel (a). For all other predictions for both the original and transfer-learned GNNs, it is difficult to discern a spatial pattern in the predictions errors, as predictions are highly accurate across the entire geometry.

Using the same fixed LV geometry as above, additional forward simulations were run over a grid of values in $[a, b]$ space with fixed $a_{\text{f}} = b_{\text{f}} = 1$ (approximately the median values from the training data), and predictions were made using both the general and transfer-learned emulators. Constraining the input space to be two dimensional allows plots of the distribution of prediction errors to be produced, which are displayed in Figure 2.17. The first column of Figure 2.17 shows the results for the original emulator trained on the varying-geometry dataset, and the second column the results for the emulator transfer-learned on this specific LV anatomy. The top row displays displacement prediction error (in mm) over $[a, b]$ space, and the second row shows absolute percentage LVV prediction error. The transfer learned emulator tends to attain higher accuracy across the entire space in both cases. For displacement error, both emulators incur highest errors at the corner of the space where a and b are low. For LVV, the distribution of emulation errors over $[a, b]$ space is less smooth. For the general emulator, highest errors occur for very low values of a and b, while for the transfer learned emulator, highest errors are incurred for low values of a where b is in the range $[5, 10]$.

2.4.5.4 Visual Analysis of Latent Learned Embeddings

Figure 2.18 displays a comparison of node coordinate values $\{\boldsymbol{x}_i\}_{i=1}^{|\mathcal{V}|}$ against 2-D projections of the corresponding local embedding vectors $\{\boldsymbol{z}_i^{\text{local}}\}_{i=1}^{|\mathcal{V}|}$, for two randomly selected LV geometries from the test set. The top row of panels shows short-axis views of the two geometries, where the slice is taken mid-way up the myodardium. This plot is analogous to Figure 2.11 for the beam data, however in this case we restrict our analysis to the epicardial points on a single short-axis slice rather than considering all nodes in the FE mesh, allowing for 2-D plots to be made. PCA was applied to the local embedding

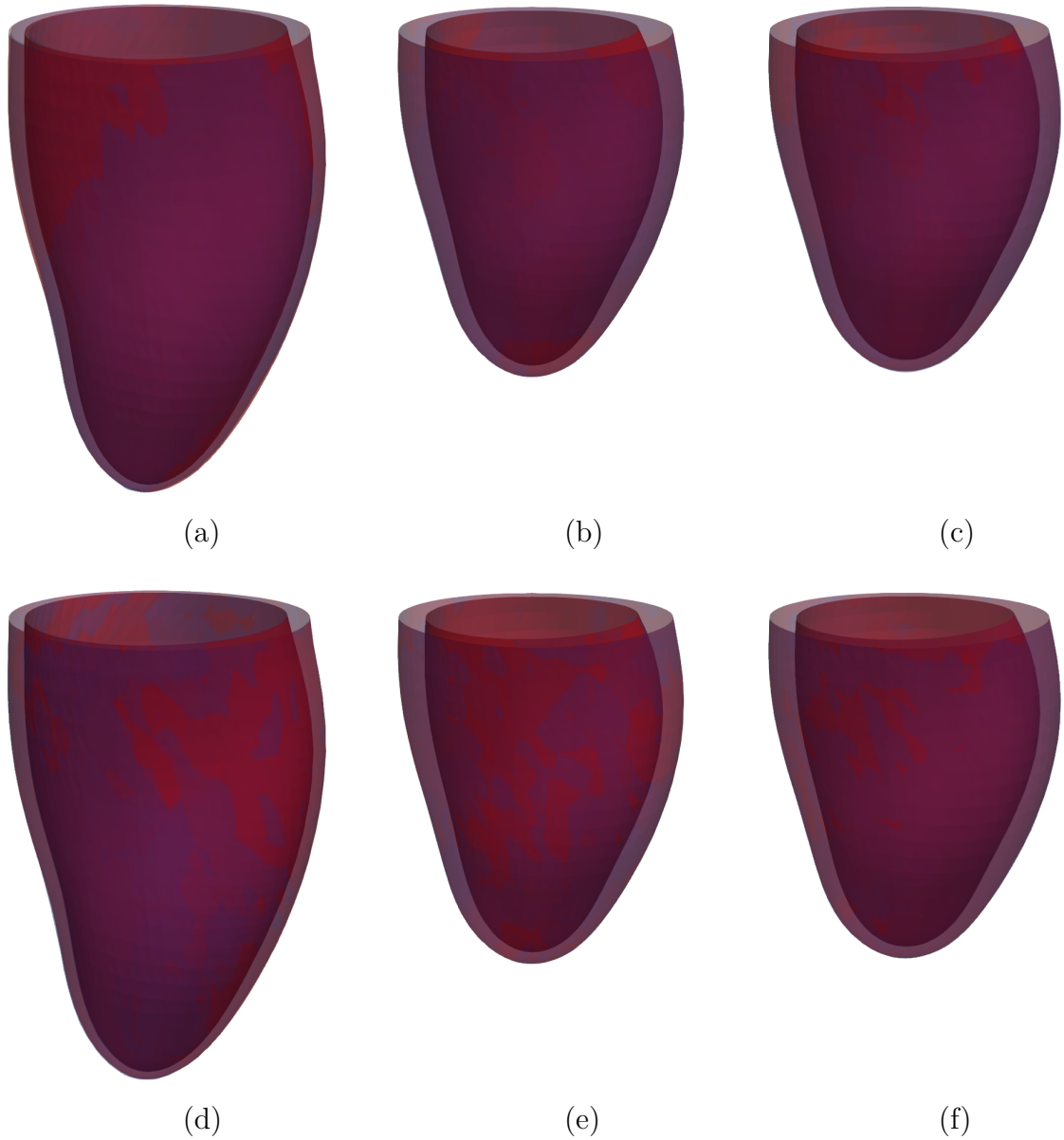


Figure 2.16: Three example test-set end-diastole geometries outputted by the simulator (red) against emulator predictions (blue). The top row shows the predictions of the original, varying-geometry emulator in blue, and the second row shows the predictions of the fixed-geometry emulator. Note that the simulated inflation of the LV in the left column is larger than would be expected to be observed *in-vivo*.

vectors $\boldsymbol{z}_i^{\text{local}}$ along this slice of the epicardium, across all 567 geometries from the varying geometry test data set. The local embedding vectors were found using the GNN with $K = 5$ message passing steps and $M = 40$. The first two PCs accounted for over 50% of the variance observed in the data. The projection values in the bottom row of panels are coloured with the same shading as the nodal coordinate values in the top row. For both LV geometries, the projection values exhibit a similar periodicity to that seen in the nodal

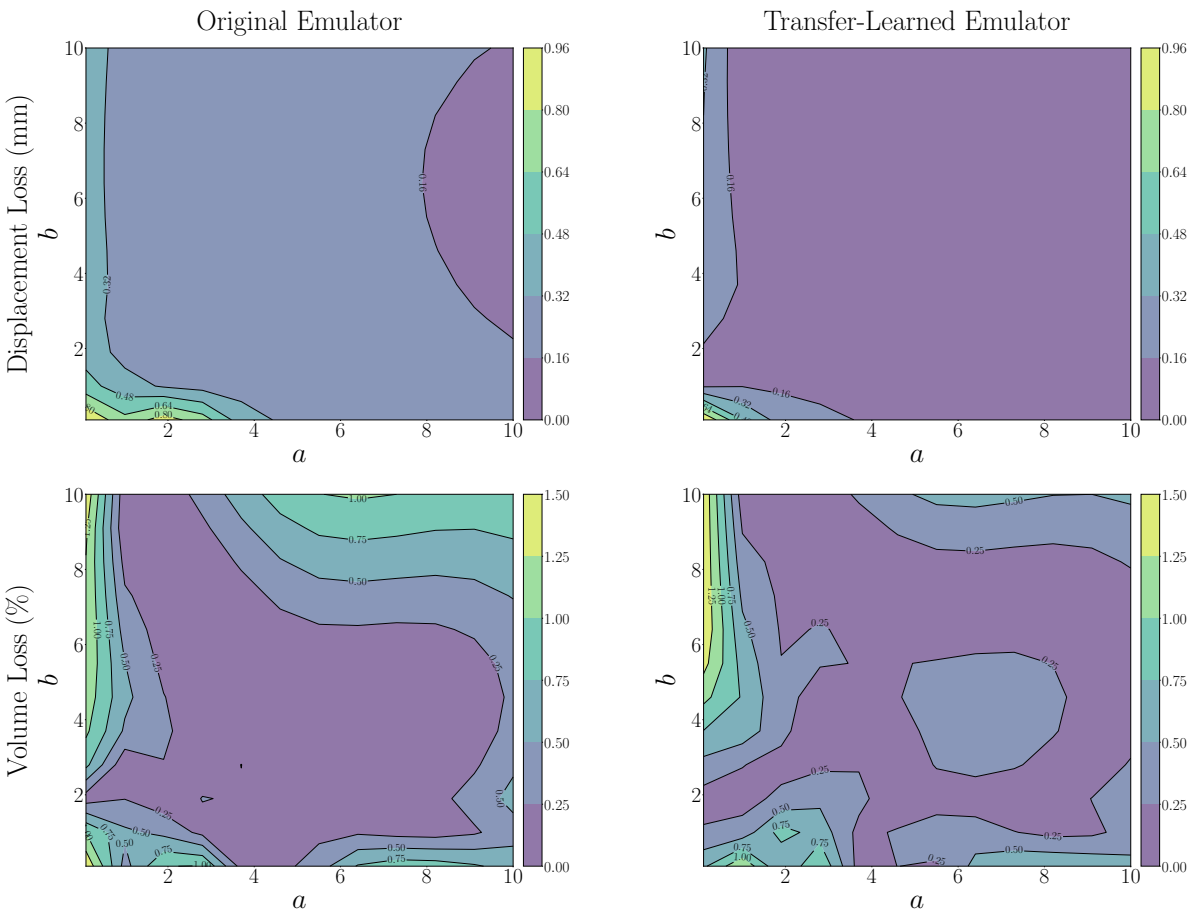


Figure 2.17: Distribution of emulation error over $[a, b]$ space, with fixed $a_\mathrm{f} = b_\mathrm{f} = 1$ and fixed LV geometry. The top row shows heatmaps of mean displacement errors over $[a, b]$ space (in mm), while the second row shows heatmaps of absolute percentage error in LVV prediction. The left column shows the results when using the original emulator for predictions, while the second column shows results when using the emulator transfer-learned on the fixed LV anatomy.

coordinate values. For the blue points with the lowest values of PC1 however, we see in both cases a more acute curve in the surface of the projection values than in the nodal coordinate values. Note however that the projection values also account for the myofibre structure of the LV, which is not shown in the nodal coordinate plots.

2.4.5.5 Prediction Speed

Finally, the wall-clock forward prediction times of the GNN surrogate model and the numerical simulator are compared in Table 2.1. Note that simulation times for the numerical solver are dependent on the values of the material stiffness parameters, with lower stiffness configurations generally leading to longer prediction times. The values indicated were run

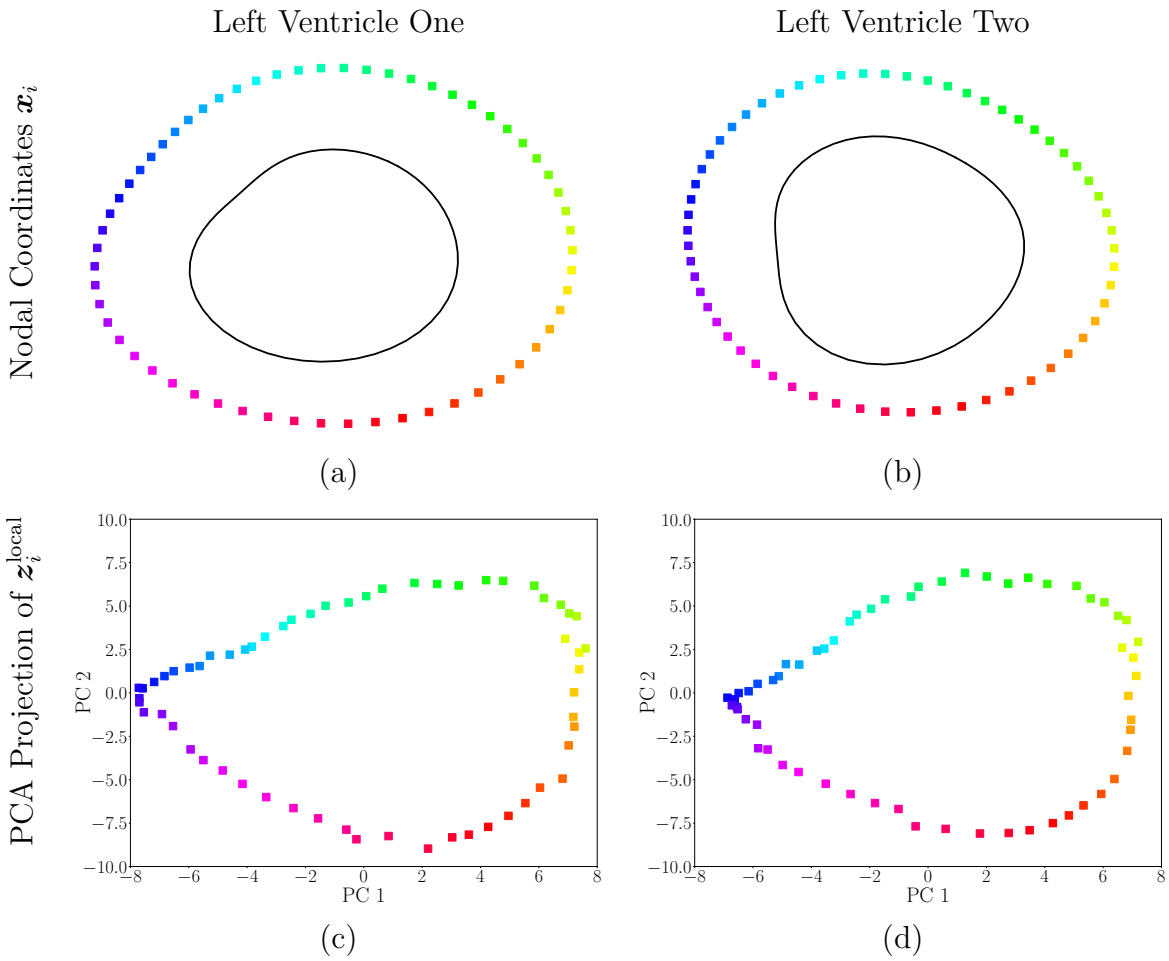


Figure 2.18: Comparison of nodal coordinates $\boldsymbol{x}_i$ in geometry space (top row of panels) against two dimensional projections of the local embedding vectors $\boldsymbol{z}_i^{\text{local}}$ (bottom row of panels), for two LV geometries. The top row of panels show short-axis views of the two geometries, with 60 coordinates along the epicardial surface of each LV displayed as points with a cyclic colour shading. For reference, the endocardial surface of each LV is also displayed as a black curve. The projection values of the epicardial points are found using Principal Component Analysis (PCA), and they are coloured in the bottom row of panels with the same shading as the nodal coordinates. We observe the same periodicity in the projection values as is seen in the coordinate values.

with $\boldsymbol{\theta} = [1, 1, 1, 1]$, approximately the median values from the simulation data set. The first row of Table 2.1 compares the prediction times of the emulator when run on a CPU (Dual Intel Xeon Gold 6138 2.0GHz), and the second row gives GPU (Dual Quadro RTX 8000) prediction times. In addition, two forward evaluation times for the emulator are presented for each type of hardware. The column "Varying Geom." in Table 2.1 shows the prediction time when all three stages from Algorithm 2 are performed. We have however designed the emulator such that repeated predictions for a fixed LV geometry over different material parameter values $\boldsymbol{\theta}$ are particularly efficient. Repeated forward

evaluations of this type are required for example in the context of an inverse problem where the material stiffness properties of a patient's LV are estimated from experimental data. Stiffness estimates obtained in this manner have been found to deliver insights into cardiac function that are relevant for clinical diagnostic purposes [60]. As discussed in Section 2.2.4, for a fixed LV geometry, the message passing stage of the network can be pre-computed exactly once to find $\boldsymbol{z}_i^{\text{local}}$ for each node $n_i \in \mathcal{V}$, which constitutes the bulk of the computations of the emulator. The column "Fixed Geom." in Table 2.1 shows the evaluation times of the emulator once this pre-computation has been performed, where only the final node-prediction MLP has to be evaluated. On CPU, the varying-geometry emulator requires 2.0×10^{-1} s to make a forward evaluation, while in the case of a fixed geometry, predictions can be made in only 9.6×10^{-3} s. By contrast, 78 s are typically required for the numerical simulator to converge. Forward evaluations of the emulator are even faster on the GPU, requiring 4.5×10^{-3} s in the general case, and 2.4×10^{-4} s for a fixed LV geometry. In particular for the fixed geometry case, this gives a speed up of over five orders of magnitude compared to the simulator.

Table 2.1: Prediction times (seconds) for forward simulator and GNN emulators. The first row shows emulator prediction times when run on CPU, and the second row shows GPU prediction times.

Emulator		**Simulator**	**Speedup Factor**	
Varying Geom.	Fixed Geom.	$a=b=a_{\text{f}}=b_{\text{f}}=1$	Varying Geom.	Fixed Geom.
2.0×10^{-1}	9.6×10^{-3}	78	3.7×10^{2}	8.1×10^{3}
4.5×10^{-3}	2.4×10^{-4}	78	1.7×10^{4}	3.3×10^{5}

2.4.6 Discussion

In this work we have introduced an emulation framework based on a novel GNN architecture. The GNN uses a three stage, encode-process-decode approach to perform emulation (Algorithm 2). The processor stage involves the processing and propagation of messages around an augmented form of the graph structure induced by the underlying FE mesh. The augmented structure involves the use of additional virtual nodes and edges, generated as in Algorithm 1, to increase message-passing efficiency. We then applied the framework to two separate mechanics problems, the first of which was the toy problem of a 2-D beam with linear and isotropic elastic properties, clamped at one end and deformed under its

own weight due to gravity. The second was a 3-D model of the LV in diastole, carried out using synthetic LV geometries with myocardium characterised by a non-linear and transversely isotropic constitutive law, and myofibre field generated by an RBM. Experiments were performed to evaluate the accuracy and computational efficiency of the proposed emulation framework.

2.4.6.1 General Dataset

Figure 2.13 shows that the prediction errors of the GNN emulator decrease both as more PCs are included in $\boldsymbol{z}^{\text{global}}$ and as more message passing steps K are performed. This effect is more pronounced for increasing PCs than for increasing K. The importance of including additional PCs in $\boldsymbol{z}^{\text{global}}$ shows that information about the global shape of the LV geometry helps to improve the generalisation performance of the emulator, which is likely due to the high level of variation observed between different LV anatomies.

2.4.6.2 LVV Benchmark against MLP

The GNN emulation approach is substantially more accurate than an MLP emulator in the prediction of LVV, achieving a reduction in prediction errors of approximately one order of magnitude, as illustrated in Figure 2.14. Note also in Figure 2.14 the sensitivity of MLP emulation performance to an application specific pre-transformation of the input variables. Without this transformation, informed by our experience in modelling this type of simulation data, the emulation performance is extremely poor. On the other hand, the GNN emulation results are much less sensitive to this transformation, suggesting that the GNN requires less domain specific feature engineering to obtain accurate results than a traditional emulator. Note that the GNN and MLP emulators also differ substantially, both in terms of number of trainable parameters and training times. Specifically, the MLP emulator has 21,377 parameters and took approximately 1 minute to train (Dual Intel Xeon Gold 6138 CPU), while the GNN has over 520,000 parameters and took 27 hours to train (Dual Quadro RTX 8000 GPU).

2.4.6.3 Transfer Learning on Fixed LV Geometry

Generalisation error can be expected to increase for test geometries further from the bulk of the training examples. The use of transfer learning in the case of a fixed LV geometry can lead to improved emulation performance. Figure 2.15 (a) demonstrates that nodal emulation prediction errors decrease by approximately a factor of three when using less than a hundred additional training simulations. LVV predictions from Figure 2.15 (b) also improve, but the effect was less pronounced. Recall that LVV was not incorporated in the loss function used when transfer learning. If a quantity such as LVV is of particular interest, the loss function could be adjusted to target this quantity specifically. The additional training input locations were chosen using a space-filling design. This could be improved using an active-learning approach, where the input locations are selected to target those areas of the input space where the emulator is less accurate [63, Chapter 6]. For example, considering the heatmap in the bottom right of Figure 2.17. An active-learning approach that accounts for prediction errors over this space would select additional simulation data $a < 1$ and $5 < b < 10$ to target the region of higher error.

2.4.6.4 Prediction Speed

The objective of the present work is to allow for patient-specific computational cardiac models to be eventually applied in clinical decision support roles, for which rapid evaluation times are required. Note that while generating the simulation datasets and training the GNN emulator is computationally expensive, the advantage of our approach is that these computations can be pre-computed in advance as no re-training is required for the emulator to be applied to an arbitrary LV anatomy of interest. A further increase in performance can be achieved however in the case of a fixed LV of interest using a small number of additional forward simulations, as discussed in the previous subsection. The results in Table 2.1 show that the emulator delivers significant gains in computation time over the simulator at the prediction stage, i.e. once training is complete. In the case of performing simulations for a fixed LV geometry, the GNN emulator can perform over 4000 forward evaluations per second on GPU. The GNN emulator can also leverage automatic

differentiation to compute exact derivatives (to machine precision) at negligible additional computational cost. This combination of rapid and accurate predictions available with gradient information means that our approach is suitable for performing numerical experiments such as forward and inverse problems in real-time.

2.4.6.5 Limitations and Future Work

The approach presented in this work could be extended for more realistic and patient-specific cardiac simulations. Specifically, the myofibre field of each synthetic LV geometry considered here was assigned by an RBM, see Section A.3. Each real LV geometry however will have a unique myofibre field, the structure of which can have a significant impact on the mechanical response of the LV in diastole [85]. In addition, we have assumed that the material properties of each LV are constant across all nodes. Future work could relax this assumption to allow stiffness values to vary for different regions of the LV, which can be achieved by simply inputting a local value $\boldsymbol{\theta}_i$ in the decoder stage of Algorithm 2 rather than a global value $\boldsymbol{\theta}$. This would be especially useful for emulation of the passive mechanics of post myocardial infarction (MI) patients, as the myocardium can exhibit significantly higher stiffness levels in the MI region. In this case however, removing the global material parameter information from the message passing stage of the GNN as we have done in Algorithm 2 may produce sub-optimal results. An indicator variable in each node feature vector $\boldsymbol{v}_i$ indicating healthy or diseased status could be sufficient though to achieve accurate emulation. Furthermore, this study has restricted its analysis to the passive phase of the cardiac cycle. Future work could consider the active phase and model the electrophysiologically driven contraction of the heart.

In addition, the architecture of the GNN itself could be more specifically tailored to the LV model. Note how we have used the exact same architecture for both the 2-D beam model and 3-D LV model. For both models, the operation of running a forward simulation of a given geometry commutes with the operation of translation of the geometry in space. We have designed the emulator to also satisfy this commutation property, through the use of relative displacements in the edge features from Eq. (2.23). For the LV model, we have an additional commutation relationship, which the GNN architecture from Algorithm 2 does not satisfy. That is, the operation of running a forward simulation of the LV in

diastole commutes with the operation of rotating the geometry around the z-axis. Future work could develop a GNN architecture which does satisfy this commutation relation, potentially improving emulation performance. Further insight into the possibilities offered by a more application specific emulator design can be drawn by considering the results in Figure 2.18, which show that the emulator has learned a periodicity in the local embedding vectors $\boldsymbol{z}_i^{\text{local}}$ that mirrors the periodicity seen in the coordinate values $\boldsymbol{x}_i$ for a slice of nodes on the epicardium. The periodicity in the latent space in turn helps to ensure a periodicity in the displacement predictions of the emulator, which we know to be true in advance as we do not consider ruptures of the myocardium. While it is promising that the emulator has learned this periodicity property automatically from the simulation data, emulation performance could potentially be improved by instead explicitly enforcing periodicity by adjusting the design of the emulator itself. This could be done by working with prolate spheroidal coordinates for example when performing emulation, rather than using the Cartesian coordinate system.

Extending the modelling framework to more complex and patient-specific simulations makes the general emulation problem more challenging in turn, as it increases the dimensionality of the input space of the emulator. In this case, it is likely that transfer-learning the general emulator for a specific LV anatomy and myofibre field of interest will be required to maintain highly accurate emulator predictions. However, transfer-learning in a *data-driven* manner as we consider in this chapter would require numerical forward simulations to be performed, which are expensive to obtain. An alternative would be to transfer-learn in a *physics-informed* manner [101]. With physics-informed training, the parameters of the surrogate model are learned by minimising a potential-energy [176] or PDE residual based loss function [177]. The advantage of physics informed loss functions is that they can be evaluated without requiring any numerical simulation data, potentially allowing for transfer learning to be performed rapidly in-clinic. Physics-informed training has been applied in cardiac mechanics using non-GNN based surrogate models [176], [94].

2.5 Conclusion

This work has introduced a novel GNN framework for emulation of FE based numerical simulators. The GNN uses a three-stage, encode-process-decode approach to perform emulation, implemented on an augmented graph representation of the underlying FE mesh. The GNN can easily generalise to different systems without retraining, both in terms of the geometry of the underlying mesh and any material constitutive parameters. The accuracy and efficiency of the emulator was compared against numerical simulations for two mechanics problems; a 2-D beam model and a 3-D model of the left-ventricle in diastole. Results are presented showing the emulator delivers accurate out-of-sample predictions comparable to the simulator, while being able to make forward evaluations several orders of magnitude more quickly. These results illustrate the promise of our GNN emulation framework compared to traditional approaches, and constitute an initial step towards eventual application of GNN emulation for clinical decision support.

Data and Code Availability

The GNN (Algorithm 2) was implemented in Python using the JAX [48] and Flax [178] libraries, while the MLP emulator from Section 2.4 was implemented using the scikit-learn machine learning library [179]. All GNN emulation code is available on GitHub[7].

[7] `https://github.com/dodaltuin/passive-lv-gnn-emul`

Chapter 3

Physics-Informed Graph Neural Network Emulation of Soft-Tissue Mechanics

Dalton, David; Husmeier, Dirk; Gao, Hao (2023) [10]. "Physics-Informed Graph Neural Network Emulation of Soft-Tissue Mechanics". In *Computer Methods in Applied Mechanics and Engineering*, https://doi.org/10.1016/j.cma.2023.116351.

Abstract

Modern computational soft-tissue mechanics models have the potential to offer unique, patient-specific diagnostic insights. The deployment of such models in clinical settings has been limited however, due to the excessive computational costs incurred when performing mechanical simulations using conventional numerical solvers. An alternative approach to obtaining results in clinically relevant time frames is to make use of a computationally efficient surrogate model, called an emulator, in place of the numerical simulator. In this work, we propose an emulation framework for soft-tissue mechanics which builds on traditional approaches in two ways. Firstly, we use a Graph Neural Network (GNN) to perform emulation. GNNs can naturally handle the unique soft-tissue geometry of a given patient, without requiring any low-order approximations to be made. Secondly, the emulator is trained in a physics-informed manner to minimise a potential energy functional, meaning that no costly numerical simulations are required for training. We present results showing that our framework allows for highly accurate emulation for a range of soft-tissue mechanical models, while making predictions several orders of magnitude more quickly than the simulator.

Nomenclature

Ω_0	Reference configuration
Ω	Current configuration
$\boldsymbol{X}$	Coordinates in reference configuration
$\boldsymbol{x}$	Coordinates in current configuration
$\boldsymbol{u}$	Displacement
$\boldsymbol{\sigma}$	Cauchy stress tensor
$\boldsymbol{b}$	Body force in current configuration
Ω^d	Dirichlet boundary in current configuration
$\boldsymbol{u}_d$	Prescribed displacement on Dirichlet boundary
Ω^σ	Neumann boundary in current configuration
$\boldsymbol{n}$	Surface normal vector in current configuration
$\boldsymbol{t}$	Traction force on Neumann boundary
$\boldsymbol{F}$	Deformation gradient
$\boldsymbol{C}$	Right Cauchy-Green tensor
J	Determinant of $\boldsymbol{F}$
I_1	First invariant of $\boldsymbol{F}$
Ψ	Strain energy density function
Π	Total potential energy
$\mathcal{G}/\tilde{\mathcal{G}}$	Graph / augmented graph
$\mathcal{V}/\tilde{\mathcal{V}}$	Graph nodes / augmented graph nodes
$\boldsymbol{v}_i$	Node feature vector
$\mathcal{E}/\tilde{\mathcal{E}}$	Graph edges / augmented edges
$\boldsymbol{e}_{i\to j}$	Edge feature vector
$\boldsymbol{\theta}$	Global graph parameters / material parameters
$\boldsymbol{\omega}$	Trainable emulator parameters
$\boldsymbol{m}_{j\to i}^k$	Message from node j to node i at processor step k
$\hat{\boldsymbol{u}}$	Emulator predicted displacement
$\boldsymbol{U}$	Array of displacements from simulator
$\hat{\boldsymbol{U}}$	Array of displacements from emulator

3.1 Introduction

This chapter extends the work of Chapter 2 to consider physics-informed training, as there are disadvantages to the data-driven approach. Firstly, a large dataset may be required to train an accurate surrogate model, which will be expensive to obtain. Also, with a purely data-driven training, any *a-priori* known underlying physical properties of the system under consideration (which could include momentum conservation, for instance) are not explicitly incorporated into the emulator. Instead, such properties are only implicitly incorporated via the simulation data. Physics-informed machine learning approaches constitute an alternative approach to emulation, whereby properties or constraints of the underlying physical system are explicitly accounted for in the structure of the emulator, and/or into the training procedure. Early work in the field includes the use of both neural networks [180]–[182] and Gaussian processes [183], [184] for solving forward and inverse problems involving differential equations. A major development was the seminal work of Raissi et al. (2019) [101], which introduced physics-informed neural networks (PINNs). PINNs explicitly incorporate the underlying equations of the model (typically PDEs) into the training of the neural network surrogate model. PINNs can be trained on a loss function solely derived from the underlying PDEs, or in a multi-task learning framework where observational data is also included. PINNs were originally implemented using the strong form of the underlying PDEs, with all required partial derivatives computed using automatic differentiation, but have subsequently been extended to model PDE systems using variational [185] and energy minimisation [186], [187] approaches. PINNs have become one of the most highly researched areas of scientific machine learning, for purposes including forward, inverse, control and model discovery problems across a range of disciplines. To give a handful of examples, this includes computational fluid dynamics [188], hyperelasticity [186], electromagnetics [189] and molecular dynamics [190]. Several detailed survey papers on PINNs are also available, providing a comprehensive overview of the field [59], [99].

PINNs have also been applied in soft-tissue mechanics, for emulation of the dynamics of the left ventricle of the heart for the entire cardiac cycle [94], and during the passive diastolic phase [176], with both approaches making use of fully connected neural networks (FCNNs). One issue with applying FCNNs in soft-tissue mechanics is that the geometries of individual organs or vessel networks vary across the population, meaning that a different computational mesh representation must be generated for each subject. This makes it difficult to construct an accurate emulator using a traditional approach, especially one that can generalise to geometries not seen in the training phase, due to the so-called curse of dimensionality. In Buoso et al. (2021) [94], a low rank approximation to both the left ventricle (LV) geometry and displacement field is required to overcome this issue and allow the FCNN to generalise to new LV geometries, while in Zhang et al. (2022) [176], only a simple cuboidal geometry is considered, not a real heart geometry.

3.1.1 Contributions

This chapter builds upon Chapter 2 by making use of physics-informed training through application of the principle of minimum total potential energy, in place of a data-driven approach based on numerical simulation data. Training on a potential energy objective function is enabled by the use of transformation barrier functions, which explicitly incorporate known physical constraints and stabilise the objective. A range of realistic soft-tissue mechanics models are considered, including highly non-linear, fibre reinforced materials. Experimental results demonstrate that strong out-of-sample accuracy can be achieved, i.e. the emulator can generalise to input points not seen during the training phase. Additional experimental comparisons indicate that physics-informed training allows for a more physically realistic deformation to be consistently captured in comparison to data-driven training. Finally, significant computational savings at prediction time are made over the finite-element based simulator.

The chapter is laid out as follows; Section 3.2 first describes the methods used, including the underlying mechanics framework considered and the proposed physics-informed emulation approach. Section 3.3 then presents emulation results for a number of mechanics models. Section 3.4 discusses these results, before Section 3.5 concludes.

3.2 Methods

3.2.1 Mechanics Framework

Consider a continuum body made of a hyperelastic material. Let $\Omega_0 \subset \mathbb{R}^3$ be the reference configuration of the body, comprised of material points $\boldsymbol{X} = (X_1, X_2, X_3)^\top \in \Omega_0$. Under external loading, the body can deform into current configuration Ω, comprised of material points $\boldsymbol{x} = (x_1, x_2, x_3)^\top$. The material points in the current and reference configurations are related as $\boldsymbol{x} = \boldsymbol{\chi}(\boldsymbol{X}, t) = \boldsymbol{X} + \boldsymbol{u}$, with $\boldsymbol{\chi}$ the (invertible) motion map and $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{X}, t)$ the displacement. In this chapter we will assume the same quasi-static, nonlinear boundary value problem from Eq. (2.1), in which case the displacement field is found as that which satisfies momentum balance subject to prescribed boundary conditions.

An important quantity for quantifying the forward map from the reference to the current configuration is the deformation gradient tensor, which is defined as $\boldsymbol{F} \in \mathbb{R}^{3\times3} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} = \nabla \boldsymbol{u} + \boldsymbol{I}$, with $\boldsymbol{I}$ the identity tensor. For a hyperelastic material, the stress $\boldsymbol{\sigma}$ and strain (or $\boldsymbol{F}$) tensors can be related by a constitutive law, Ψ, i.e. $\boldsymbol{\sigma} = J^{-1}\left(\frac{\partial \Psi}{\partial \boldsymbol{F}}\right)^T$, where $J = \det(\boldsymbol{F})$. Three constitutive laws are considered in this work: Neo-Hookean [191], Holzapfel-Ogden [75] and Guccione [192][8].

The Neo-Hookean model is derived from first principles on the properties of cross-linked polymer chains, and is suitable for isotropic plastic and rubber-like materials [191]. The strain energy density for a compressible Neo-Hookean material is given by

$$\textbf{Neo-Hookean :} \quad \Psi\left(\boldsymbol{F}, \boldsymbol{\theta}\right) = \frac{1}{2}\lambda[\log(J)]^2 - \mu \log(J) + \frac{1}{2}\mu\left(I_1 - 3\right). \tag{3.1}$$

Here, $I_1 = \mathrm{tr}(\boldsymbol{C})$ is the first invariant with $\boldsymbol{C} = \boldsymbol{F}^\top \boldsymbol{F}$ the right-Cauchy-Green deformation tensor. The parameters λ and μ are the Lame material parameters, which are denoted collectively as $\boldsymbol{\theta}$. The Neo-Hookean model can be equivalently parameterised in terms of Young's modulus E and Poisson ratio ν, which are related to the Lame parameters as:

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}. \tag{3.2}$$

[8] Experiments involving the Guccione law are performed in Section B.1.

The Holzapfel-Ogden (H-O) model is a phenomenologically derived anisotropic hyperelastic material model for describing the passive response of the myocardium [75]. The fully incompressible strain energy density function of a transversely isotropic H-O material is given by

$$\textbf{H-O :} \quad \Psi\left(\boldsymbol{F}, \boldsymbol{\theta}\right) = \frac{a}{2b}\left(e^{b(I_1-3)} - 1\right) + \frac{a_\mathrm{f}}{2b_\mathrm{f}}\left(e^{b_\mathrm{f}(\max(I_{4\mathrm{f}},1)-1)^2} - 1\right). \tag{3.3}$$

Here I_1 is defined as above, while the transversely isotropic invariant $I_{4\mathrm{f}}$ is equal to $\boldsymbol{f}_0 \cdot (\boldsymbol{C}\boldsymbol{f}_0)$, where $\boldsymbol{f}_0 \in \mathbb{R}^{3\times 1}$ is the unit vector in the myofibre direction in the reference configuration. The material parameters are $\boldsymbol{\theta} = (a,\, b,\, a_\mathrm{f},\, b_\mathrm{f})$. The max() term in Eq. (3.3) ensures that the myofibres only support extension but not compression. The use of the transversely isotropic version of the H-O material is motivated by the results of an earlier sensitivity study of the full H-O law [164]. In our numerical implementation, we consider the myocardium nearly incompressible, which is a widely used practice in cardiac models, using the F-bar method to Eq.(3.3) to ensure numerical stability [193]. By decomposing $\boldsymbol{F}$ into a deviatoric ($\bar{\boldsymbol{F}} = J^{-1/3}\boldsymbol{F}$) and volumetric component ($J^{1/3}\boldsymbol{I}$), the F-bar method will relax the near-incompressibility constraint. Accordingly, we have

$$\bar{\boldsymbol{C}} = \bar{\boldsymbol{F}}^T\bar{\boldsymbol{F}}, \quad \bar{I}_1 = \mathrm{tr}(\bar{\boldsymbol{C}}), \quad \bar{I}_4 = \bar{\boldsymbol{F}}\boldsymbol{f}_0 \cdot \bar{\boldsymbol{F}}\boldsymbol{f}_0, \tag{3.4}$$

and the H-O model is modified as

$$\Psi\left(\boldsymbol{F}, \boldsymbol{\theta}\right) = \frac{a}{2b}\left(e^{b(\bar{I}_1-3)} - 1\right) + \frac{a_\mathrm{f}}{2b_\mathrm{f}}\left(e^{b_\mathrm{f}(\max(\bar{I}_{4\mathrm{f}},1)-1)^2} - 1\right) + \frac{\mathcal{P}}{2}(J-1)^2, \tag{3.5}$$

where the final term in the equation, known as the penalty function, enforces the near-incompressibility constraint of the myocardium via the Lagrange multiplier $\mathcal{P} \in \mathbb{R}^+$, which may also be referred to as the bulk modulus. In this study, $\mathcal{P}$ is pre-set to be a fixed value, however $\mathcal{P}$ can also be treated as an unknown field variable based on the multi-field variable principles [77, Chapter 8].

The Guccione model [192] also describes the mechanical response of the passive myocardium, with the strain energy density function given by

$$\textbf{Guccione :} \quad \Psi(\boldsymbol{F},\boldsymbol{\theta}) = \frac{C}{2}\left(e^{Q}-1\right) + \frac{\mathcal{P}}{2}(J-1)^2, \text{ with} \tag{3.6}$$
$$Q = b_{\mathrm{f}}\bar{E}_{11}^2 + b_{\mathrm{t}}\left(\bar{E}_{22}^2+\bar{E}_{33}^2+\bar{E}_{23}^2+\bar{E}_{32}^2\right) + b_{\mathrm{fs}}\left(\bar{E}_{12}^2+\bar{E}_{21}^2+\bar{E}_{13}^2+\bar{E}_{31}^2\right).$$

Here $\bar{\boldsymbol{E}} = \frac{1}{2}(\bar{\boldsymbol{C}} - \boldsymbol{I})$ is the Green-Lagrange strain tensor, and we have material parameter vector $\boldsymbol{\theta} = (C,\, b_{\mathrm{f}},\, b_{\mathrm{t}},\, b_{\mathrm{fs}})$.

3.2.2 Numerical Methods

In this work, we propose a GNN emulation approach for boundary value problems of the form of Eq. (2.1), with particular emphasis on applications in soft-tissue mechanics. Emulation results are benchmarked with numerical simulations obtained using the non-linear finite-element method. Before a numerical solution for the displacement field can be found using either the FEM simulator or GNN emulator, a number of processing steps are required.

Firstly, the nonlinear boundary-value problem (BVP) of Eq. (2.1) is very challenging to be solved analytically, in particular for complex geometries. In this study, we consider the quasi-static BVP to be a conservative mechanical system, which requires the existence of an energy functional Π for both the stresses of a deformable body and the loads. Such an assumption is commonly used in solid mechanics. Based on the stationary energy principle, and further treating $\boldsymbol{u}$ as the only unknown, then of all the admissible $\boldsymbol{u}$, we seek the solution that minimises the total potential energy. [77, Chapter 8]. In other words, the first variation of the total potential energy $\delta\Pi$ needs to vanish in static equilibrium. In the case of a hyper-elastic continuum, the total potential energy Π of Eq. (2.1) is given by:

$$\Pi = \int_{\Omega_0} \Psi dV - \int_{\Omega_0} \boldsymbol{b}_0 \cdot \boldsymbol{u} dV - \int_{\partial\Omega_0^{\sigma}} \boldsymbol{t}_0 \cdot \boldsymbol{u} dA, \tag{3.7}$$

in which $\boldsymbol{b}_0(\boldsymbol{X}) = \boldsymbol{b}(\boldsymbol{\chi}^{-1}(\boldsymbol{x}, t), t = 0)$ is the body force density with respect to the reference configuration, and $\boldsymbol{t}_0(\boldsymbol{X}) = \boldsymbol{t}(\boldsymbol{\chi}^{-1}(\boldsymbol{x}, t), t = 0)$ is the traction force density with respect to the reference configuration. Mathematically, the desired solution can be obtained by requiring the directional derivative (or *Gateaux* derivative) of Π with respect to $\boldsymbol{u}$ to vanish in all directions $\delta\boldsymbol{u}$, that is

$$\delta\Pi(\boldsymbol{u}, \delta\boldsymbol{u}) = \frac{d}{d\epsilon}\Pi(\boldsymbol{u} + \epsilon\delta\boldsymbol{u})|_{\epsilon=0} = 0. \tag{3.8}$$

Further details of the principle of stationary potential energy can be found in [77, Chapter 8].

The FEM is the most commonly used approach to numerically solve this type of BVP. Before a solution can be found using FEM, the reference configuration of the body under consideration needs to be discretised with n elements, denoted as $\{\mathcal{T}_h\}_{h>0}$. Specifically,

$$\Omega_0 \approx \{\mathcal{T}_h\}_{h>0} = \bigcup_{e_l=1}^{n} T_{e_l}, \tag{3.9}$$

where in this work we consider each T_{e_l} to be a tetrahedron. This means that in practice a finite-dimensional projection of the full displacement field is solved for when minimising Eq. (3.7). The continuous displacement field within each element is represented as a weighted sum of a finite set of basis functions

$$\boldsymbol{u}(\boldsymbol{X}) = \sum_{\alpha=1}^{n_e} \boldsymbol{u}_{\alpha i}^h N_\alpha(\boldsymbol{X}), \tag{3.10}$$

in which $\boldsymbol{u}_{\alpha i}^h$ is the displacement component along the i-th direction at the α-th node, n_e is the number of nodes in one finite element, and N_α is the finite element basis function at the α-th node.

3.2.3 Graph Neural Network Surrogate Model

This chapter makes use of the same graph neural network (GNN) surrogate modelling approach introduced in Chapter 2. Recall that this involves first converting the underlying computational finite-element mesh into a graph format (see Section 2.2.1). Then, an augmented version of the graph is built (see Section 2.2.3). Finally, a message passing GNN is used for emulation (see Section 2.2.4 and Algorithm 2). The only difference is that in this chapter, only fixed soft-tissue geometries are considered, and therefore the global PCA representation vector $\boldsymbol{z}^{\text{global}}$ from Algorithm 2 is not required.

3.2.3.1 GNN Surrogate Training

The GNN surrogate is comprised of $(3 + 2K + D)$ internal FCNNs: three encoders $\{f_V, f_E, f_P\}$, two processors for each of K rounds of message passing, $\{g_V^k, g_E^k\}_{k=1}^K$, and D decoders $\{h^d\}_{d=1}^D$, where D is the dimensionality of the system. One node-decode FCNN with D outputs could have been used instead here, however we found during experimentation in previous work that this lead to less accurate results [9]. A layer normalisation operation (LayerNorm) is applied after each FCNN (with the exception of the decoder FCNNs) and in the creation of $\boldsymbol{z}_i^{\text{local}}$ to normalise the intermediate values, which increases numerical stability during training [161]. The trainable parameters of the GNN, which we denote collectively as $\boldsymbol{\omega}$, consistent then of all the weights and biases of the internal FCNNs along with the LayerNorm parameters.

Neural network surrogate models are typically trained in a data-driven approach, using a loss function derived from a dataset of numerical forward simulations. Denote a simulation dataset as $\{(\mathcal{G}_j, \boldsymbol{U}_j)_{j=1}^N\}$, the data-driven approach to learning a point estimate of $\boldsymbol{\omega}$ is then

$$\textbf{Data-Driven :} \quad \boldsymbol{\omega}^* = \underset{\boldsymbol{\omega}}{\operatorname{argmin}} \sum_{j=1}^{N} L\left(\boldsymbol{U}_j(\boldsymbol{\theta}_j), \hat{\boldsymbol{U}}_j(\mathcal{G}_j; \boldsymbol{\omega})\right) \tag{3.11}$$

where $L : \mathbb{R}^{|\mathcal{V}|\times 3} \times \mathbb{R}^{|\mathcal{V}|\times 3} \to \mathbb{R}$ is a user-specified loss function, for example root mean-squared-error. Eq. (3.11) is generally minimised using an iterative, gradient based updating scheme. Instead of a batch approach, where the gradients are computed from all N data points, we have used the concept of mini-batches. This is equivalent to a stochastic gradient descent scheme, which tends to be faster, less susceptible to entrapment in local optima, and more robust with respect to overfitting [175].

Physics-informed training is an alternative approach, whereby instead of considering simulation data, training is performed against known properties of the physical system being modelled, which can be formulated using balance equations, PDE residuals, or conservation principles [59]. In this work, we make use of a loss function based on the total potential energy functional Π (Eq. 3.7), which is illustrated schematically in Figure 3.1. Specifically, a point-estimate of the network parameters is learned as:

$$\textbf{Physics-Informed :} \quad \boldsymbol{\omega}^* = \underset{\boldsymbol{\omega}}{\operatorname{argmin}} \sum_{j=1}^{N} \Pi \left(\hat{\boldsymbol{U}}_j(\mathcal{G}_j; \boldsymbol{\omega}), \boldsymbol{\theta}_j \right) \tag{3.12}$$

A clear difference between the two approaches is that physics-informed (PI) training is not dependent on simulator outputs $\boldsymbol{U}_j$, in contrast to data-driven (DD) training. Another difference between relates to computational costs. Because more the PI loss function requires more involved computations, the DD loss is faster to evaluate - we found the difference to be about one order of magnitude for the LV model. Further differences are explored in Section 3.3.1. Note that a surrogate model can also be trained in a *multi-task* manner, using a loss function which incorporates both a data fit term and a physics-informed term [101].

3.2.4 Implementation Details

3.2.4.1 FEM Simulation Details

All FEM simulations were performed using first-order tetrahedral elements. For any simulations involving a traction force, an iterative solution approach was taken whereby the pressure was linearly increased from zero to the final prescribed value.

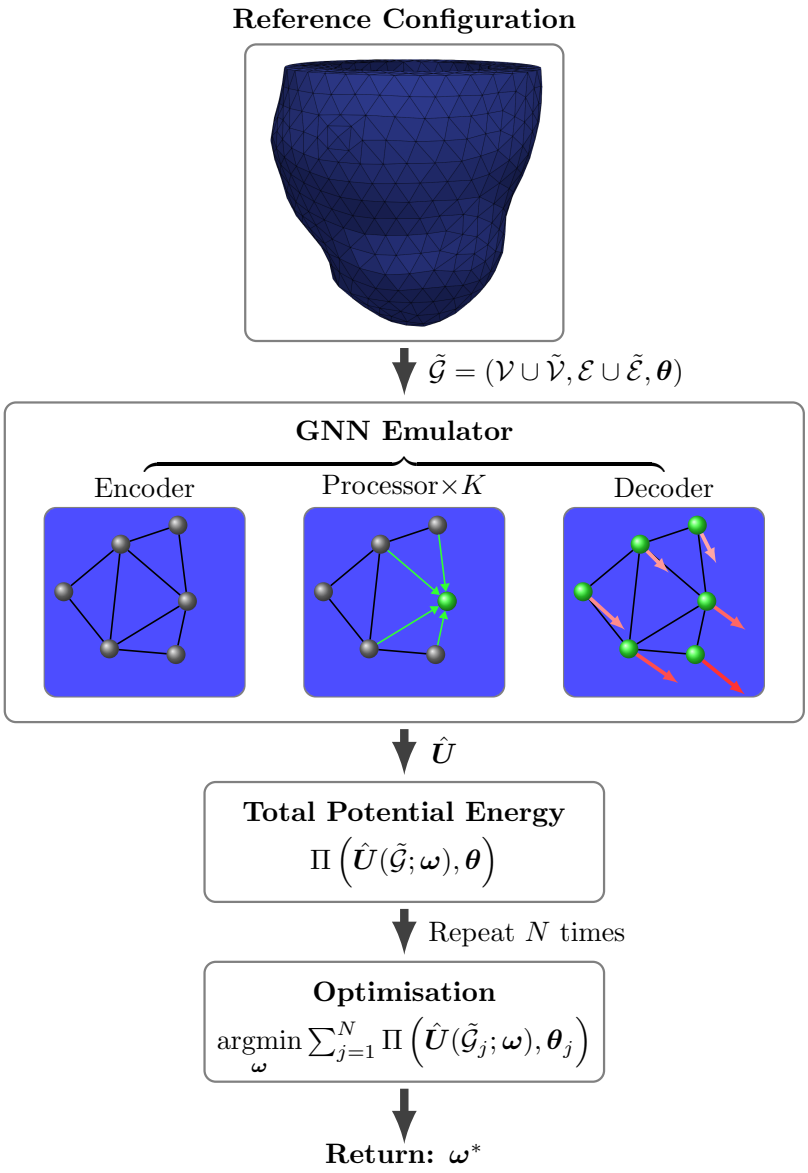


Figure 3.1: Schematic illustration of physics-informed training of $\boldsymbol{\omega}$, the tunable parameters of a GNN emulator.

3.2.4.2 Computation of Total Potential Energy

To calculate the total potential energy in Eq. (3.7), the deformation gradient $\boldsymbol{F}$ must first be found. Because first-order tetrahedral elements were used, $\boldsymbol{F}$ only needs to be computed at the centroid of each element. Denoting the coordinates of the four vertices of a tetrahedron in the current (reference) configuration as $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, $\boldsymbol{x}_k$, and $\boldsymbol{x}_l$ ($\boldsymbol{X}_i$, $\boldsymbol{X}_j$, $\boldsymbol{X}_k$ and $\boldsymbol{X}_l$), then $\boldsymbol{F}$ is found as

$$\boldsymbol{F} = \Big[\boldsymbol{x}_j - \boldsymbol{x}_i,\ \boldsymbol{x}_k - \boldsymbol{x}_i,\ \boldsymbol{x}_l - \boldsymbol{x}_i\Big] \Big[\boldsymbol{X}_j - \boldsymbol{X}_i,\ \boldsymbol{X}_k - \boldsymbol{X}_i,\ \boldsymbol{X}_l - \boldsymbol{X}_i\Big]^{-1}, \tag{3.13}$$

where the position vector for each node is treated as a column vector. Eq. (3.13) is the same as the method used in [94]. The integrals required for the calculation of Π are computed in the reference configuration Ω_0. As a result, we will need to express both the body force and the traction force in the reference configuration, $\boldsymbol{b}_0$ and $\boldsymbol{t}_0$, respectively. Here, the body force is taken to be gravity which is independent of the deformation, thus it remains the same. If the traction force is not a dead load but resulted from a pressure load P, then according to Nanson's formula, the traction force in the reference configuration can be expressed as

$$\boldsymbol{t}_0 = -J\, P \boldsymbol{F}^{-T} \boldsymbol{N}, \tag{3.14}$$

in which $\boldsymbol{N}$ is the normal of the boundary faces of the elements where a pressure load is applied. Finally the total potential energy (Eq. (3.7)) can be approximated as

$$\Pi \approx \sum_{el=1}^{n} \int_{\Omega_e^{el}} \Psi(\boldsymbol{F}) dV - \sum_{el=1}^{n} \int_{\Omega_e^{el}} \boldsymbol{b}_0 \cdot \boldsymbol{u} dV - \sum_{fl=1}^{n^f} \int_{\partial\Omega_e^{fl}} \boldsymbol{t}_0 \cdot \boldsymbol{u} dA, \tag{3.15}$$

where Ω_e^{el} is the domain occupied by the el-th tetrahedral element in the reference configuration, $\partial\Omega_e^{fl}$ is the fl-th face on the Neumann boundary with n^f the number of faces on the Neumann boundary. The differential volume dV and the differential area dA are computed in the reference configuration. The total potential energy is approximated using the first-order Gauss quadrature rule, that is

$$\Pi \approx \sum_{el=1}^{n} \Psi(\boldsymbol{F}^{el}) V_e^{el} - \sum_{el=1}^{n} \left(\boldsymbol{b}_0^{el} \cdot \boldsymbol{u}_e^{el}\right) V_e^{el} - \sum_{fl=1}^{n^f} \left(\boldsymbol{t}_0^{fl} \cdot \boldsymbol{u}_A^{fl}\right) A_e^{fl}, \tag{3.16}$$

in which $\boldsymbol{u}_e^{el}$ is the displacement vector at the centroid of the el-th element, and $\boldsymbol{u}_A^{fl}$ is the displacement vector at the centroid of the fl-th face, both of which are linearly interpolated from the nodal displacements $\boldsymbol{u}_i$. V_e and A_e are the volume and area of the associated element and face in the reference configuration, respectively, which can be precomputed in advance of emulator training.

3.2.4.3 Stabilising Potential Energy Computations

Training an emulator directly on the discretised potential energy functional Π from Eq. (3.16) is numerically difficult. During the early stages of training, the surrogate may produce physically unrealistic predictions, leading to extreme behaviour (such as divergences) in the computation of Π. In [176], the authors overcome this issue by initially training on a small subset of parameter space, before gradually expanding the training domain. In the present work, we make use of the entire material parameter domain throughout training, where numerical stability in Π is ensured by the use of certain barrier transformation functions, which we now detail.

Firstly, due to the invertibility of the forward map $\boldsymbol{\chi}$ and because volumes cannot be negative, we know $a-priori$ that any volumetric change ratios from the reference to current configuration must be strictly positive. Mathematically, this means that $J > 0$. An untrained emulator could however make a prediction to the displacement field which violates this condition. For this reason, we apply the following transformation to $\hat{J}$ before evaluating Π,

$$\hat{J}^{trans} = \begin{cases} \exp\Big(\beta_1 \hat{J} + \beta_2\Big) + \beta_3, & \text{if } \hat{J} \leq J^{ch} \\ \hat{J}, & \text{otherwise.} \end{cases} \tag{3.17}$$

Here J^{ch} is the point after which the transformation changes to the identity map, and $\{\beta_1, \beta_2, \beta_3\}$ are parameters of the transformation whose values must be specified. Elementary algebraic manipulations and the asymptotic behaviour of the exponential function can be combined to show that setting

$$\beta_1 = \frac{1}{J^{ch} - J^{min}}, \quad \beta_2 = \log\big(J^{ch} - J^{min}\big) - \frac{J^{ch}}{J^{ch} - J^{min}} \quad \text{and } \beta_3 = J^{min} \tag{3.18}$$

ensures $\hat{J}^{trans} > J^{min}$, and that the transformation is both continuous and differentiable at the change point J^{ch}. For all experiments, we set $J^{min} = 0.001$ and $J^{ch} = 0.05$, whereas for quasi-incompressible materials $J \approx 1$.

For the H-O and Guccione constitutive laws, a similar issue arises due to their use of the exponential function. Physically unrealistic emulator predictions during the initial phase of training can lead to large predicted values for I_1, I_{4f} or Q, which when exponentiated lead to overflow in the computation of Π. We prevent this behaviour through use of the following transformation

$$\hat{Z}_{trans} = \begin{cases} \tanh\left(\hat{Z} - Z^{ch}\right) + Z^{ch}, & \text{if } \hat{Z} \geq Z^{ch} \\ \hat{Z}, & \text{otherwise.} \end{cases} \tag{3.19}$$

where $\hat{Z} \in \{\hat{I}_1, \hat{I}_{4f}, \hat{Q}\}$ is the relevant value computed from the emulator's prediction and $Z^{ch} \in \{I_1^{ch}, I_{4f}^{ch}, Q^{ch}\}$ is the change point after which the non-linear transformation is applied. This transformation is continuous and differentiable at Z^{ch}, and ensures $\hat{Z}_{trans} < Z^{ch} + 1$. We do not have rigorous finite upper bounds for I_1, I_{4f} or Q. Instead, we set the bounds to be higher than is expected for the simulation results, but low enough to prevent overflow from becoming an issue. These specific values were $I_1^{ch} = 10$, $I_{4f}^{ch} = 8$ or $Q^{ch} = 15$.

Finally, we initialise the weights and biases in the final layer of the decoder FCNNs to zero. This ensures that the untrained emulator predicts zero displacement for all material parameter values, give further numerical stability in the initial stages of training.

3.2.4.4 Enforcing Dirichlet Boundary conditions

For each model considered, we explicitly enforce the outputs of the GNN emulator along the Dirichlet boundary Ω^d to satisfy the prescribed displacement values. This is achieved through an additional post-processing step after evaluation of the emulator, where the prescribed values are substituted in place of the predicted values on Ω^d. In principle, these boundary conditions could be softly enforced using a penalty method, however we found training was significantly easier when using explicit enforcement - similar results were reported in [177].

3.2.4.5 GNN Emulation Details

Performing emulation with the `DeepGraphEmulator` architecture (see Algorithm 2) requires the number of message-passing steps K, the dimensionality M of the internal embedding vectors, and the structure of the internal FCNNs to be specified. For all experiments, we fixed $K = 5$, $M = 40$ and used FCNNs with CELU activation function [194] and two hidden layers, each of width 128. The choice of these hyper-parameters is based on extensive numerical experiments conducted in previous work [9]. With these values, the emulator had approximately 510,000 trainable parameters for each model considered. Training is performed using Adam [111], with a batch size of one on a single GPU (Dual NVIDIA RTX A6000). The effect of changes in learning rate on emulator training is explored in Section B.1 - we find that a learning rate of 1×10^{-4} is optimal in the early stages of training. In data-driven training of neural networks, the use of techniques such as early stopping can be critical to prevent overfitting and ensure good generalisation performance. However we found during experimentation that this was not required for physics-informed training, where overfitting was not observed during longer training runs.

The node and edge features for each model considered were specified to have the following form

$$\boldsymbol{v}_i = (d_i) \text{ or } (d_i, \boldsymbol{f}_i) \tag{3.20}$$

$$\boldsymbol{e}_{i \to j} = (\boldsymbol{x}_j - \boldsymbol{x}_i, ||\boldsymbol{x}_j - \boldsymbol{x}_i||_2), \tag{3.21}$$

where d_i is a Boolean variable indicating if the node lies on the Dirichlet boundary, and $\boldsymbol{f}_i$ is the unit vector in the fibre direction. Fibre information is only included for models with non-isotropic constitutive law and spatially varying fibre field in the reference configuration. Node and edge features are normalised element-wise to mean zero and unit variance before being inputted to the emulator. Note how absolute positional information is not included in the node features, instead only relative positions are incorporated via the edge features. This ensures that the emulator is invariant under translations of the underlying geometry in space.

3.2.4.6 Data and Code Availability

All experiments were implemented in Python. FEM simulations were performed using FEniCS [162], and PI-GNN computations were performed using the JAX [48], Flax [178] and Optax [195] libraries. All data and code is available at `https://github.com/dodaltuin/soft-tissue-pignn`.

3.3 Numerical Experiments

We conducted numerical experiments to evaluate the performance of the PI-GNN emulation framework involving a total of five different mechanics models[9]. In each case, a fixed geometry is considered, and an emulator is trained over a specified material parameter domain using the GNN architecture detailed in Section 3.2.4.5. Complete details of each model are given in Sections 3.3.1-3.3.4, while Table 3.1 presents a summary. The column for $\boldsymbol{\eta}$ refers to the cardinalities of the layers of virtual nodes created from the base FE nodes - see Algorithm 1 for details. Benchmarking of emulation results is performed primarily in terms of error in prediction of the displacement $\boldsymbol{u}$, the deformation gradient $\boldsymbol{F}$, the first invariant I_1 and the total potential energy Π. Errors on these quantities are quantified using the metrics from Eq. (3.22), where the hat symbol denotes results from the emulator and $\|\cdot\|_F$ is the Frobenius norm. The error metric on $\boldsymbol{u}$ is an absolute value measured in (mm) while $err_{\boldsymbol{F}}$, err_{I_1} and err_{Π} are relative errors, measured in (%).

$$\textbf{Error Metrics :} \quad \begin{aligned} err_{\boldsymbol{u}} &= \|\boldsymbol{u}-\hat{\boldsymbol{u}}\|_2 \text{ (mm)}, & err_{\boldsymbol{F}} &= \frac{\|\boldsymbol{F}-\hat{\boldsymbol{F}}\|_F}{\|\boldsymbol{F}\|_F}\times 100 \text{ (\%)}, \\ err_{I_1} &= \left|\frac{I_1-\hat{I}_1}{I_1}\right|\times 100 \text{ (\%)}, & err_{\Pi} &= \left|\frac{\Pi-\hat{\Pi}}{\Pi}\right|\times 100 \text{ (\%)}. \end{aligned} \tag{3.22}$$

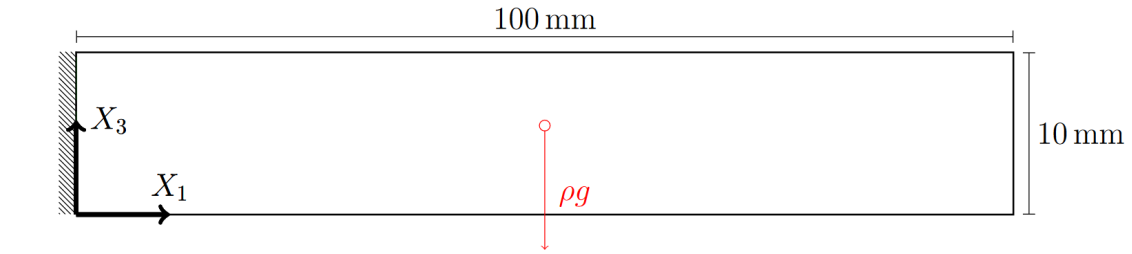
[9] Four additional models are considered in the supplementary material.

Model	Material	$\boldsymbol{\theta}$	$\boldsymbol{\eta}$	N_{node}	N_{elem}
`OnceClampedBeam`	Neo-Hookean	(λ, μ)	$[97, 24]$	390	1440
`TwiceClampedBeam`	Neo-Hookean	(λ, μ)	$[97, 24]$	390	1440
`TwistingCube`	Neo-Hookean	(E, ν)	Varying	Varying	Varying
`Liver`	Neo-Hookean	(λ, μ)	$[233, 58, 14]$	935	4408
`LeftVentricle`	Holzapfel-Ogden	$(a, b, a_{\mathrm{f}}, b_{\mathrm{f}})$	$[392, 98, 24]$	1570	6176

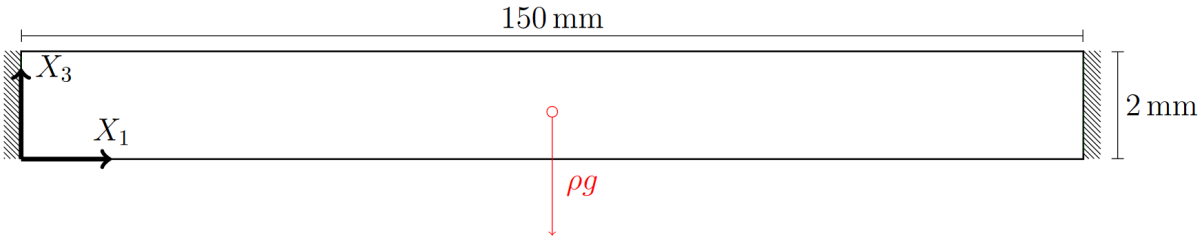
Table 3.1: Summary of models considered for emulation.

3.3.1 Data-Driven and Physics-Informed Training Comparison

The first model considered, `OnceClampedBeam` is illustrated in Figure 3.2 (a). This model involves a beam with $\Omega_0 = [0, 100] \times [0, 10]^2$ (mm), discretised using 390 nodes and 1440 elements. The beam is clamped at the left end $\partial\Omega_0^d = \{\boldsymbol{X} \in \Omega_0 : X_1 = 0\}$, and displaced under gravitation from its own weight. Specifically, $\boldsymbol{b} = (0, 0, -\rho g)^\top$, with ρ the density of the beam and g the acceleration due to gravity. No traction force is applied. A Neo-Hookean material model is assumed (see Eq. (3.1)), with material parameters $\lambda, \mu \in [5, 10]$ kPa. The second model considered, `TwiceClampedBeam`, is illustrated in Figure 3.2 (b). This is similar to the first model, except we have $\Omega_0 = [0, 150] \times [0, 10] \times [0, 2]$ (mm), and both ends of the beam are clamped, that is $\partial\Omega_0^d = \{\boldsymbol{X} \in \Omega_0 : X_1 = 0 \text{ or } X_1 = 150\}$.



(a) `OnceClampedBeam`



(b) `TwiceClampedBeam`

Figure 3.2: Illustration of the rectangular beam models considered as 2-D slices in the (X_1, X_3) plane (not to scale), where the dashed lines indicate a clamped Dirichlet boundary, ρ is the density of the material and g the acceleration due to gravity.

We use these models to compare the performance of a GNN emulator trained in a data-driven manner as in Eq. (3.11) (DD-GNN) with physics-informed training as in Eq. (3.12) (PI-GNN). For training of the DD-GNN, 200 simulations were first run from randomly sampled material parameter values. Training was then performed for 5000 epochs using a fixed learning rate of 1×10^{-4}. For consistency, the PI-GNN was trained at the exact same material parameter inputs, with the same number of training epochs and learning rate. For evaluation, an independent set of 100 simulations were performed for each model. The average magnitude of the nodal displacements $\|\boldsymbol{u}\|$ for `OnceClampedBeam` was 15.7 mm, with maximum value of 50.1 mm. For `TwiceClampedBeam`, the corresponding values were 2.3 mm and 6.0 mm respectively, where the disparity in displacement magnitudes is due to the additional clamped boundary constraints at both ends of the `TwiceClampedBeam` model.

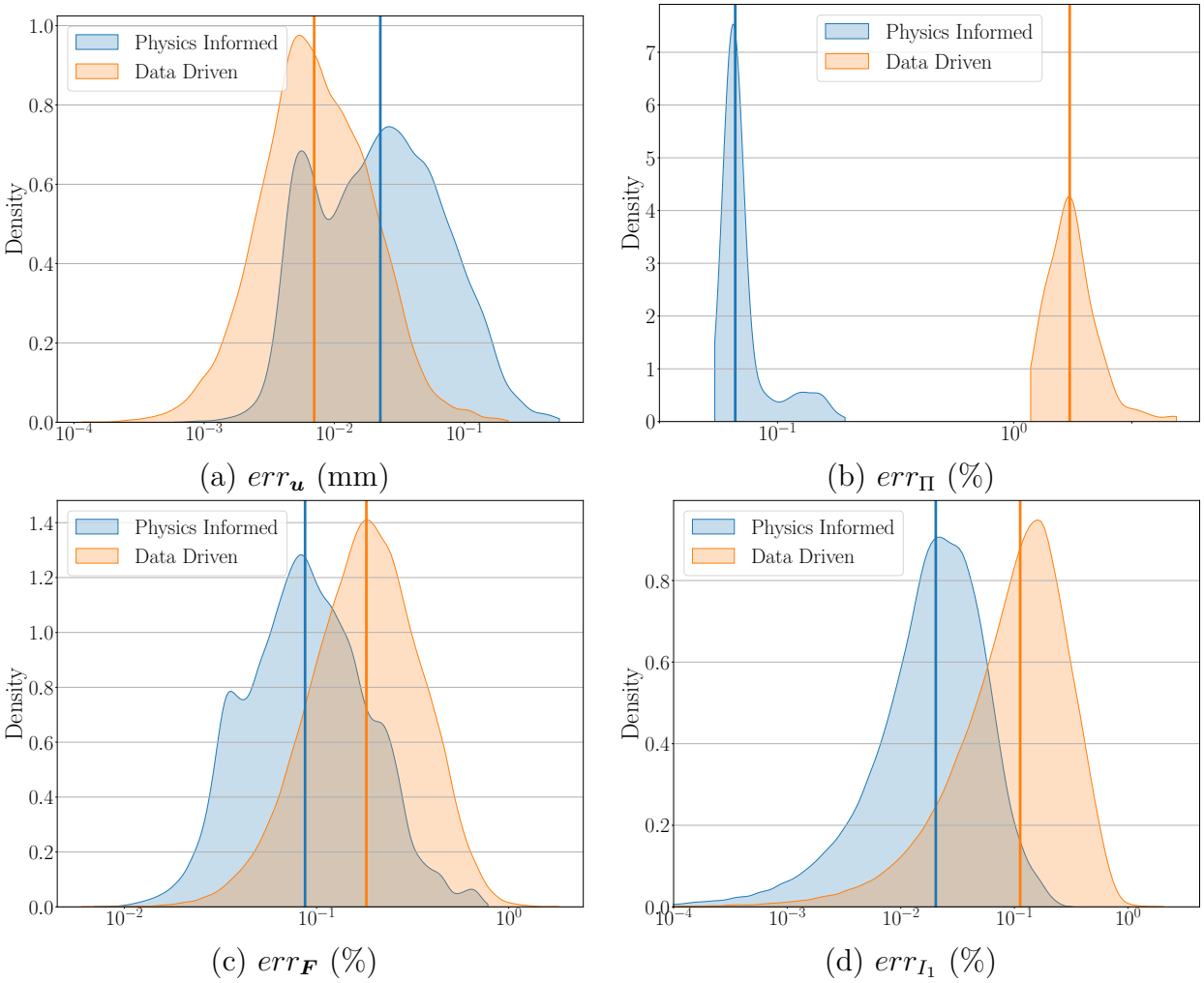


Figure 3.3: Comparison of data-driven and physics-informed emulation results on `OnceClampedBeam` model.

Figure 3.3 shows emulation results for the `OnceClampedBeam` model, using density plots which compare the test-set accuracy of the two training approaches in terms of the four error metrics from Eq. (3.22). Note that we present density plots on the log scale, as we believe this makes them more easy to interpret. Panel (a) indicates that the DD-GNN, which as been trained on displacement data, achieves lower error in prediction of the displacements, with median $err_{\boldsymbol{u}}$ value of 6.5×10^{-3} mm compared to 2.0×10^{-2} mm for the PI-GNN. Nevertheless, there is a high degree of overlap between the errors. By contrast, there is a sharp divergence between the distributions of err_{Π} for the two emulators in panel (b), as the median err_{Π} value incurred by the DD-GNN of 1.7 % percent is two orders of magnitude higher than the corresponding median value of 6.6×10^{-2} % seen with the PI-GNN. Furthermore, there is no overlap between the distributions, i.e. the lowest err_{Π} value for the DD-GNN is larger than the highest err_{Π} value with the PI-GNN. This shows that the predictions of the PI-GNN are consistently more physically realistic in terms of the potential energy state. The more realistic deformation captured by the PI-GNN is reflected in the prediction errors for $\boldsymbol{F}$ and I_1 displayed in panels (c) and (d), where we see lower errors obtained for the PI-GNN. Specifically, the DD-GNN incurs median error of 1.8×10^{-1} % for $err_{\boldsymbol{F}}$ against 8.7×10^{-2} % with the PI-GNN, while the median err_{I_1} value is 1.1×10^{-1} % for the data-driven approach is approximately one order of magnitude greater than for the PI-GNN, which had median err_{I_1} value of 2.0×10^{-2} %.

Figure 3.4 shows emulation results for the `TwiceClampedBeam` model. The pattern of results is similar to that seen in Figure 3.3, but here the data-driven method performs less well relative to the physics-informed approach. For errors in displacement space in panel (a), the PI-GNN achieves a slightly lower median error value of 9.6×10^{-3} mm versus 1.1×10^{-2} mm for the DD-GNN, albeit with large overlap between the two error distributions. For err_{Π} again we see no overlap between the errors, with the median value of err_{Π} for the DD-GNN of 5.3 % three orders of magnitude higher than the corresponding median value for the PI-GNN of 1.6×10^{-3} %. Finally, for the difference between the accuracy for the predictions of $\boldsymbol{F}$ and I_1 was even more pronounced in this model, with

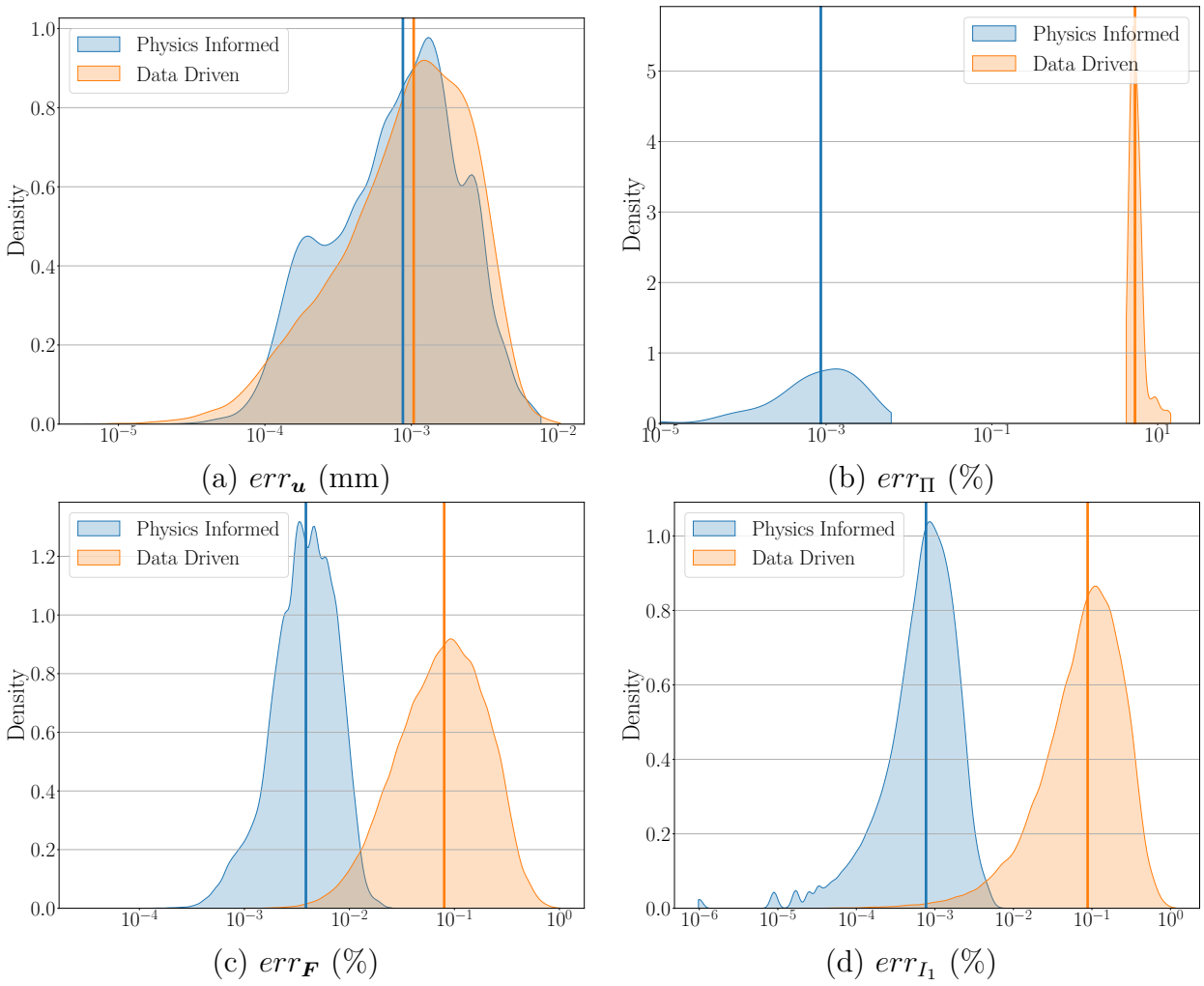


Figure 3.4: Comparison of data-driven and physics-informed emulation results on `TwiceClampedBeam` model.

PI-GNN errors typically one or two orders of magnitude lower. Specifically, the DD-GNN incurs median error of 8.0×10^{-1} % for $err_{\boldsymbol{F}}$ against 3.8×10^{-3} % with the PI-GNN, while median err_{I_1} value is 1.0×10^{-1} % for the data-driven approach, compared to 7.9×10^{-4} % for the PI-GNN.

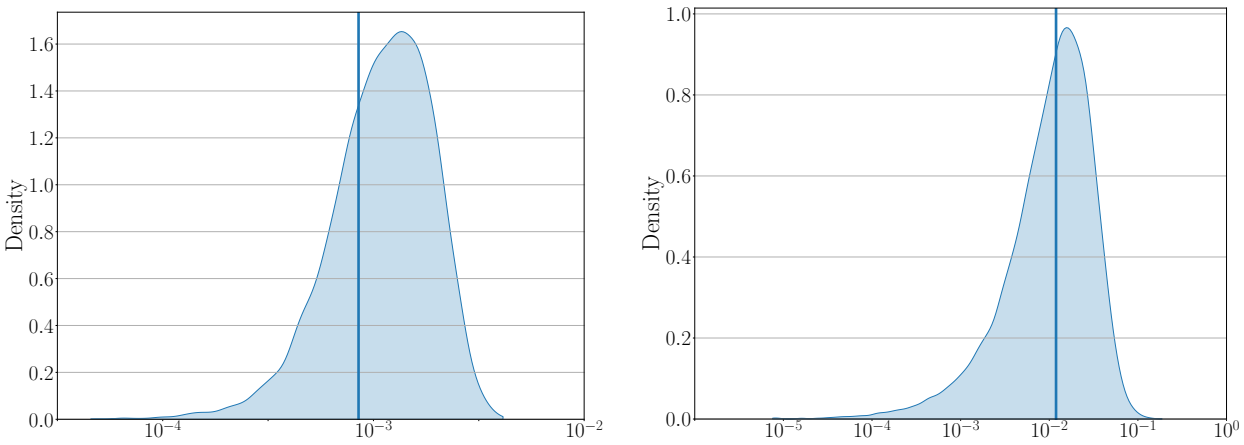
One possible reason why the DD-GNN performs better relative to the PI-GNN on the `OnceClampedBeam` model is the fact that the displacements $\boldsymbol{u}$ for this model had greater magnitude and variation across the test data compared to the `TwiceClampedBeam` model. The DD-GNN has the advantage of operating in normalised space by making use of summary statistics for the displacement field found on the training data, whereas the PI-GNN is trained in the original space, meaning the network weights have to be trained to values with larger magnitude to capture the variation in the data.

3.3.2 `TwistingCube`

We next consider emulation of a model `TwistingCube` involving a cuboidal geometry ($\Omega_0 = [0, 10]^3$ mm) discretised with 343 nodes and 1296 elements, based on a similar example from the FEniCS documentation [162]. Here, no external forces are explicitly applied. Instead, the cube is clamped at the left most end $\partial\Omega_0^{d_0} = \{\boldsymbol{X} \in \Omega_0 : X_1 = 0\}$, and the following rotational displacements are prescribed at the right end $\partial\Omega_0^{d_1} = \{\boldsymbol{X} \in \Omega_0 : X_1 = 1\}$:

$$\begin{aligned}\boldsymbol{u} =&(0,\\ &(0.5 + (x_2 - 0.5)\cos(\pi/3) - (x_3 - 0.5)\sin(\pi/3) - x_2)/2,\\ &(0.5 + (x_2 - 0.5)\sin(\pi/3) + (x_3 - 0.5)\cos(\pi/3) - x_3))/2).\end{aligned} \tag{3.23}$$

The Neo-Hookean material model is used (see Eq. (3.1)), parameterised in terms of Young's modulus E and Poisson ratio ν, with ranges $E \in [1, 25]$ and $\nu \in [0.1, 0.4]$. A PI-GNN emulator was trained for 1000 epochs over (E, ν) space. During the first 500 epochs, a fixed set of 200 material parameter configurations were used for training (selected using a Latin Hypercube Sampling (LHS) design), with learning rate of 1×10^{-4}. For the remaining epochs, the learning rate was reduced to 1×10^{-5}, and at each epoch, a new set of 200 material parameters were randomly sampled using a uniform distribution. To evaluate the trained emulator, 50 simulations were performed using the FEM to act as independent test set. Density plots for $err_{\boldsymbol{u}}$ and err_{I_1} on the test set are given in Figure 3.5. The density plot of $err_{\boldsymbol{u}}$ shown in Panel (a) shows that the displacement predictions of the emulator are extremely accurate, achieving median $err_{\boldsymbol{u}}$ value of 8.5×10^{-4} mm with no errors exceeding 7.0×10^{-3} mm (for reference, the mean and max values of $\|\boldsymbol{u}\|$ over the test data were 1.1 mm and 3.5 mm respectively). From panel (b), the errors in I_1 are also very low, with median err_{I_1} value of 1.2×10^{-2} %, while no errors are in excess of 2.0×10^{-1} %.

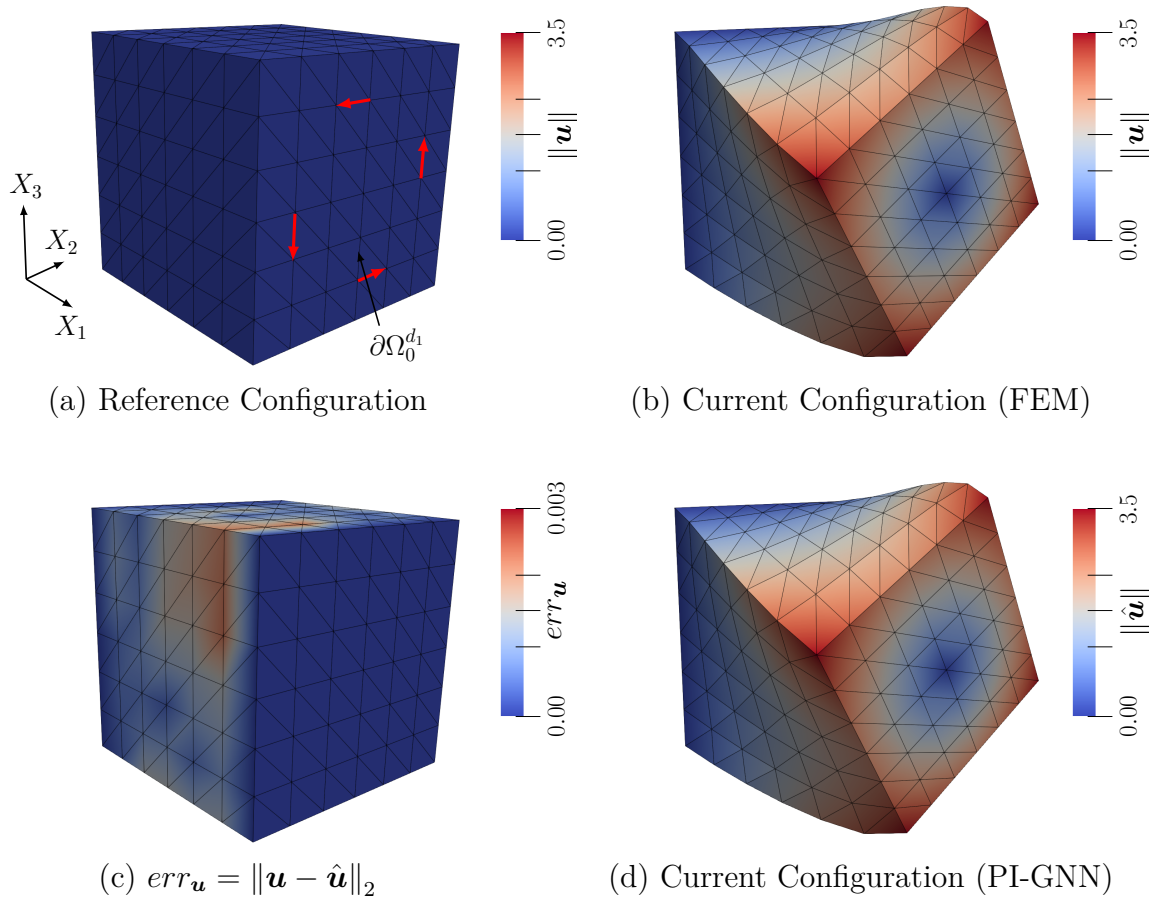


(a) $err_{\boldsymbol{u}}$ (mm) (b) err_{I_1} (%)

Figure 3.5: Error density plots for `TwistingCube` model (343 FE nodes). The vertical lines indicate median values.

The emulation results for the test-set simulation for which the PI-GNN achieved the median value of Mean($err_{\boldsymbol{u}}$) are visualised in Figure 3.6. Panel (a) shows the reference configuration of the cube, panel (b) the current configuration as given by the FEM simulation, while panel (d) shows the current configuration as predicted by the PI-GNN. No differences between the two results are apparent. Panel (c) shows the distribution of $err_{\boldsymbol{u}}$ on the surface of the reference geometry, and indicates that greater errors are incurred in the centre of the cube along the edges, although no prediction errors are observed in excess of 3.0×10^{-3} mm.

To investigate how the density of the FE mesh affects emulation results, we repeated the above experiments using four additional FE meshes, which had 1000 (4374), 4096 (20250), 13824 (73002) and 21952 (119098) nodes (elements) respectively. All other model and implementation details remained the same, with the exception of the mesh with 13824 (21952) nodes, where emulator training was performed for an additional 4000 (6500) steps to ensure convergence of the total potential energy. Table 3.2 gives the average $err_{\boldsymbol{u}}$ values obtained at each mesh density, where the errors are considered for those points where $X_1 = 0.5$, that is a slice in the (X_2, X_3) plane. Errors remain roughly constant for all mesh densities, indicating that the accuracy of the PI-GNN is not sensitive to mesh density. Note however that fewer training epochs are required to reach the same level of accuracy for less dense meshes. Table 3.2 also presents training and prediction times for the different meshes. Training and prediction times scale by roughly the same order



(a) Reference Configuration

(b) Current Configuration (FEM)

(c) $err_{\boldsymbol{u}} = \|\boldsymbol{u} - \hat{\boldsymbol{u}}\|_2$

(d) Current Configuration (PI-GNN)

Figure 3.6: Median out of sample emulation results for `TwistingCube` model (mm).

of magnitude. When the message passing stage of the GNN is precomputed however, prediction times for the decoder increase by less than a factor of three when the number of nodes rises by almost two orders of magnitude from 343 to 21952. This illustrates the advantage of our GNN architecture design (see Algorithm 2). By decoupling material parameter information from the message-passing stage of the GNN, predictions can be made extremely efficiently for a fixed input geometry once training is complete, even for very dense meshes. This is useful for inverse problems, for example, where numerous forward evaluations may be required to allow the material parameters for a given subject to be inferred from experimental data [135].

N_{node} N_{elem}	343 1296	1000 4374	4096 20250	13824 73002	21952 119098
Mean($err_{\boldsymbol{u}}$)	1.3×10^{-3}	1.1×10^{-3}	1.2×10^{-3}	1.1×10^{-3}	1.7×10^{-3}
Train Time	150 min	176 min	190 min	1166 min	2325 min
Prediction Time	1.7×10^{-3} s	1.9×10^{-3} s	4.1×10^{-3} s	1.1×10^{-2} s	1.6×10^{-2} s
Decoder Time	2.1×10^{-4} s	2.2×10^{-4} s	2.4×10^{-4} s	5.0×10^{-4} s	5.9×10^{-4} s

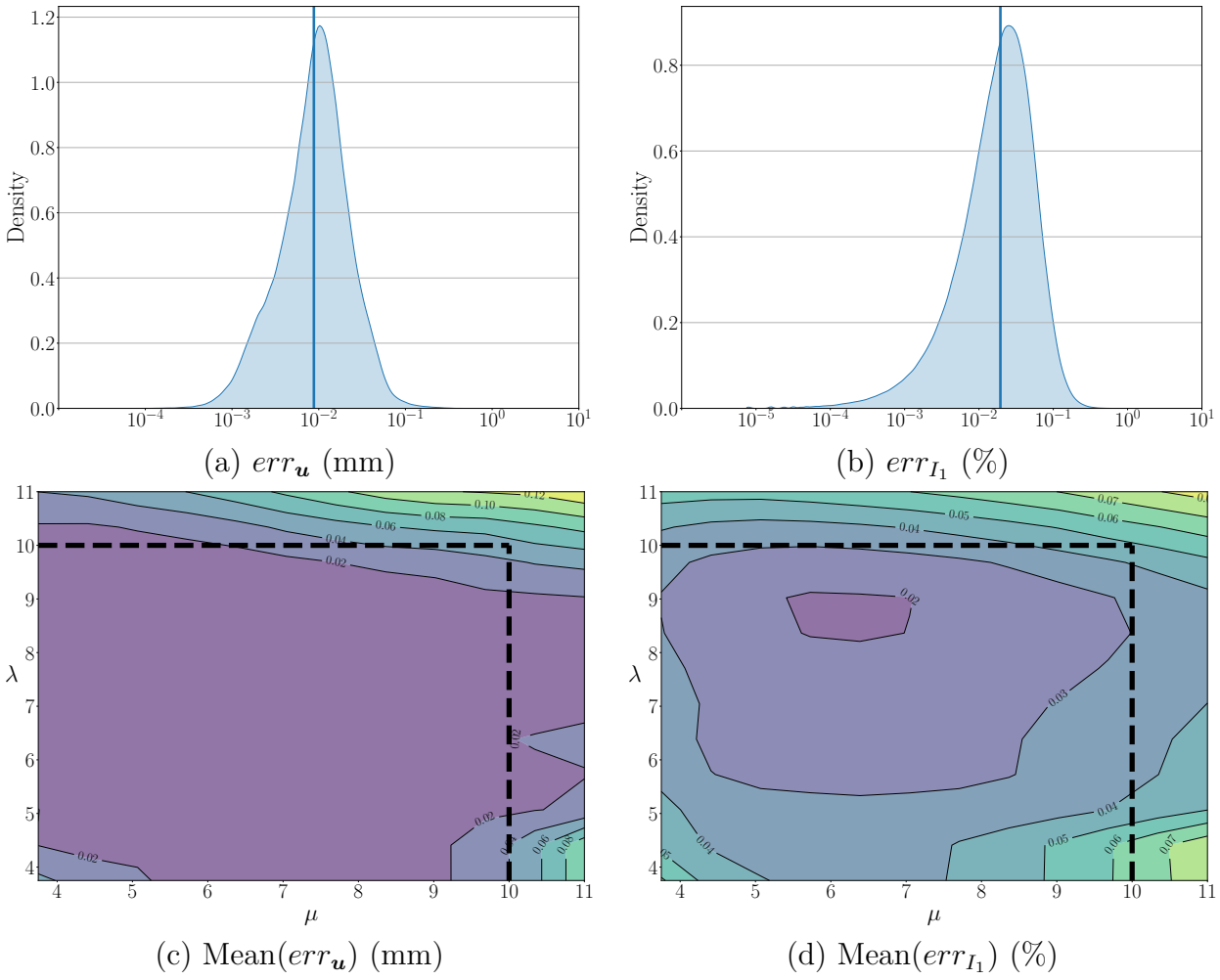
Table 3.2: Summary of emulation results for `TwistingCube` model using different mesh densities. The final row presents prediction times when only the final, decoder stage of Algorithm 2 is evaluated.

3.3.3 `Liver`

We next consider a model of a human liver, the reference configuration of which can be seen in Figure 3.8 (a). No traction force is applied, with external loading consisting solely of gravitational force. The Dirichlet boundary consists of half of the bottom surface of the geometry, where zero displacement is allowed. The geometry used is that made available by the authors of Zhang and Chauhan (2020) [196]. In this work, we use the same Dirichlet boundary surface as [196], and also assume the same Neo-Hookean material model (see Eq. (3.1)). We additionally re-scaled the geometry to have a length of 150 mm, approximately the average value for an adult human [197].

We trained a GNN emulator for the `Liver` model over parameter space $\boldsymbol{\theta} = (\lambda, \mu) \in [3.5, 10]^2$ kPa initially for 500 epochs with a fixed learning rate of 1×10^{-4} using a fixed set of 200 material parameter configurations. Training was then continued for an additional 500 epochs with learning rate 1×10^{-5}, with randomly sampled material parameters at each epoch. The out of sample prediction results of the trained emulator are presented in Figure 3.7. These results were evaluated on a set of 482 simulations performed using the FEM, each of which made use of a randomly sampled material parameter vector.

The emulator exhibits strong accuracy in prediction of the displacement values, as is illustrated in the density plot in Figure 3.7 (a). The median out of sample prediction error is 8.8×10^{-3} mm and only a tiny fraction of errors exceed one-tenth of a mm (for reference, the mean and max values of $\|\boldsymbol{u}\|$ over the test data were 7.9 mm and 58.5 mm respectively). From Panel (b), the errors in I_1 are also very low, with median err_{I_1} value of 1.9×10^{-2} %, while virtually no errors exceed half a percent. The two plots on the bottom row display loss heatmaps over material parameter space. We consider the

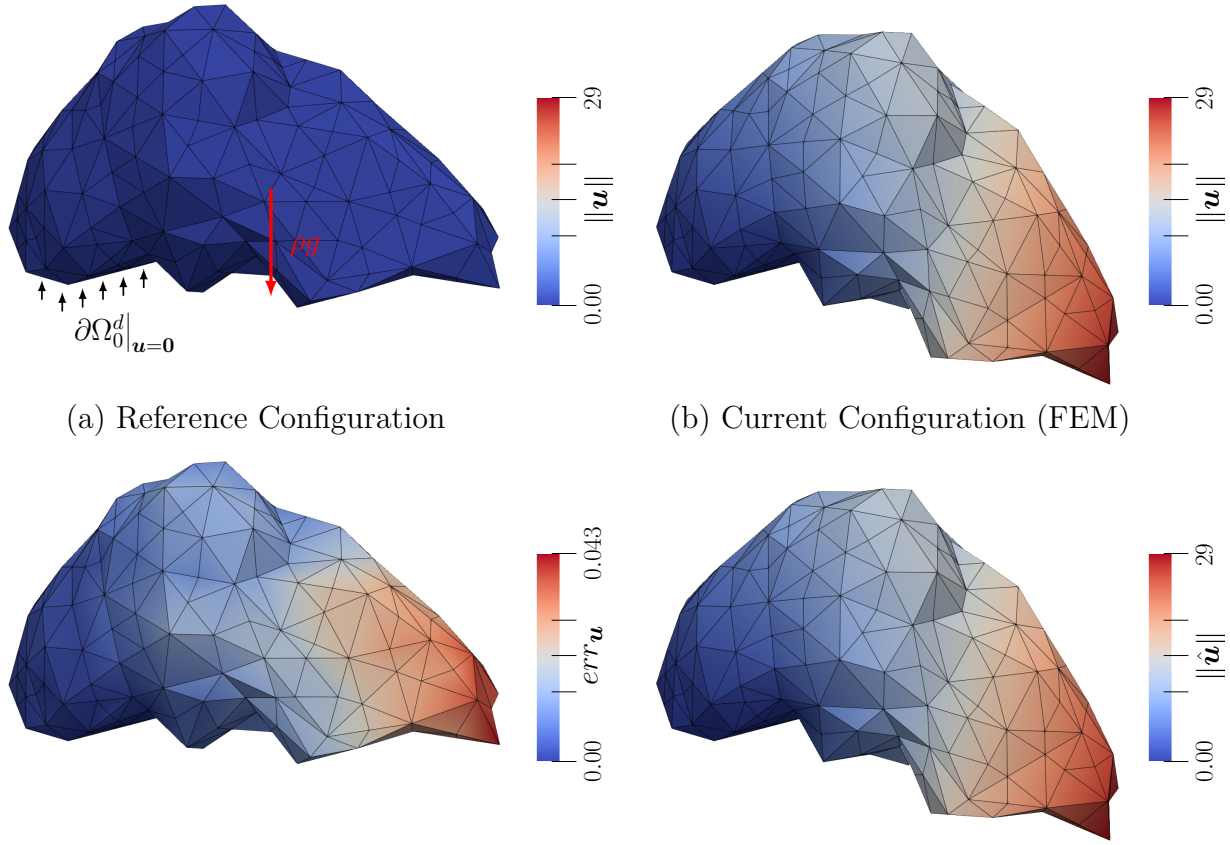


(a) $err_{\boldsymbol{u}}$ (mm) (b) err_{I_1} (%)

(c) Mean($err_{\boldsymbol{u}}$) (mm) (d) Mean(err_{I_1}) (%)

Figure 3.7: Emulation results for `Liver` model. The top row shows density plots of $err_{\boldsymbol{u}}$ and err_{I_1}, where the vertical lines indicate median error values. The bottom row displays loss heatmaps of $err_{\boldsymbol{u}}$ and err_{I_1} over the space of Lame parameter (i.e. (λ, μ)) configurations considered. The dashed lines indicate the boundary of the domain considered during training.

performance of the emulator under extrapolation by extending the heatmaps beyond the domain considered during training, the boundary of which is illustrated by the dashed lines. The plots exhibit broadly similar patterns, with very low errors within the training domain, with a smooth deterioration in prediction accuracy as we move outside this area. Figure 3.8 visualises results for the test data simulation where the emulator achieved median value of Mean($err_{\boldsymbol{u}}$). Panels (b) and (d) show the current configurations of the body outputted by the FEM and PI-GNN, respectively. It is difficult to visually discern any differences between the two results. The distribution of $err_{\boldsymbol{u}}$ values is given in Panel (c), which shows that errors are highest towards the end of the geometry where $\|\boldsymbol{u}\|$ is highest, however these errors do not exceed 4.3×10^{-2} mm.



(a) Reference Configuration

(b) Current Configuration (FEM)

(c) $err_{\boldsymbol{u}} = \|\boldsymbol{u} - \hat{\boldsymbol{u}}\|_2$

(d) Current Configuration (PI-GNN)

Figure 3.8: Median out of sample emulation results for `Liver` model (mm).

3.3.4 `LeftVentricle`

In the final emulation experiment, we consider a model for the passive mechanics of the left ventricle (LV) of the heart. The LV geometry considered is a real model, extracted from the cardiac magnetic resonance (CMR) imaging scans of a healthy volunteer at early diastole. Both long and short axis CMR scans were used for the reconstruction, which was performed using segmentation software developed in-house. For further details on LV geometry reconstruction, see [60], [80]. A layered myofibre structure is typically incorporated into LV models, but imaging of myofibres in-vivo remains a challenging problem. For this reason, we adopt a rule based method (RBM) to describe the layered myofibres in this work [165]. Here a fibre-sheet-normal local material coordinate system is defined, where we vary the fibre angle linearly from -90^o at endocardium to 90^o at epicardium. The geometry is approximately 75 mm in length from base to apex, and is discretised using 1570 nodes and 6176 elements. The passive filling of the myocardium is

modelled by a linearly increased cavity pressure applied to the inner surface of the LV, with no body force applied. The base of the LV is fully constrained with zero displacements. These boundary conditions are illustrated via a 2-D diagram of the base of the LV in Figure 3.9. The H-O material model is used to describe the mechanical response of the myocardium. The ranges considered for each material parameter (see Eq. (3.3)) were $a \in [0.1, 2.6]\,\text{kPa}$, $b \in [1., 4.2]$, $a_\text{f} \in [1.5, 5.18]\,\text{kPa}$ and $b_\text{f} \in [1, 4.46]$. These ranges were chosen so as to match the mean parameter values found on real data [60], with roughly double the level of variance around the mean. The final pressure loading on the endocardium is allowed to range from 4 to 10 mmHg, and it is incorporated in the GNN by concatenation to the material parameter vector $\boldsymbol{\theta}$. We set the parameter $\mathcal{P}$ to equal $25\,\text{kPa}$ ($5\,\text{kPa}$ was used in [176]) to allow some compressibility in the myocardium. Note that it is still debatable whether the myocardium shall be treated to be fully incompressible or compressible [198]. In future work, we will consider multi-field variational principles as an alternative approach to handle incompressibility - this is discussed further in Section 3.4.4. The reference configuration of the LV is shown in Figure 3.11 (a), while Panel (b) shows an FEM simulation result.

A PI-GNN emulator for the `LeftVentricle` model over the four dimensional material parameter space was trained for 15000 epochs, with a learning rate of 1×10^{-4} during the first half of training and 1×10^{-5} for the second half. More training epochs were used here over previous models based on examination of the traceplots of the potential energy. A test data set of 150 points was generated using the FEM to evaluate the out of sample performance of the trained PI-GNN, where each simulation was performed with a different randomly sampled material parameter vector and pressure loading value. Note that further details of the training procedure used are given in Section 3.2.4.

Density plots of emulation error on the test set are presented in Figure 3.10. From Panel (a), the bulk of displacement prediction errors fall within an order of magnitude of the median $err_{\boldsymbol{u}}$ value of $2.8 \times 10^{-2}\,\text{mm}$, however the tail of the distribution approaches $6.5 \times 10^{-1}\,\text{mm}$ (for reference, the mean and max values of $\|\boldsymbol{u}\|$ over the test data were $6.8\,\text{mm}$ and $26.3\,\text{mm}$ respectively). The distributions of $\boldsymbol{F}$ and err_{I_1} errors are both slightly more peaked around their median values of $1.9 \times 10^{-1}\,\%$ and $4.7 \times 10^{-2}\,\%$ respectively. We also consider the emulation error in predicting the volume V enclosed within

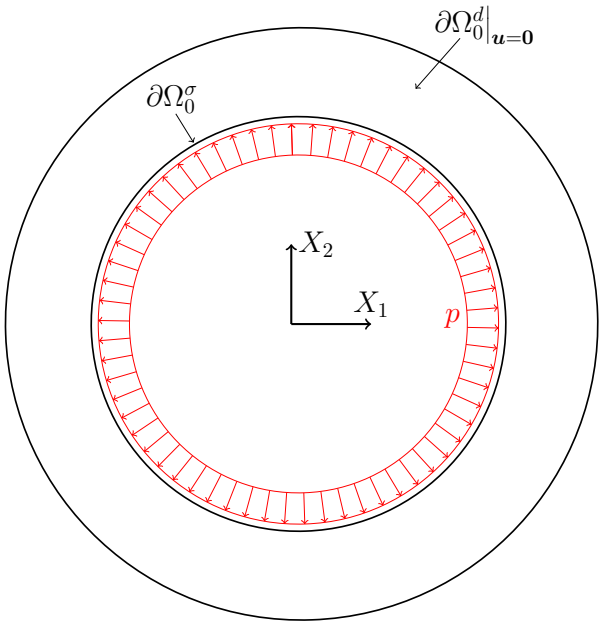


Figure 3.9: Short-axis illustration of boundary conditions for `LeftVentricle` model at the base of the geometry (idealised, symmetric geometry only used for illustration purposes). The figure shows in the (X_1, X_2) plane - the X_3 direction runs from the apex to the base (see Figure 3.11 (a)) . Here $\partial\Omega_0^d$ indicates the clamped base of the LV where zero displacements are allowed, $\partial\Omega_0^\sigma$ the inner surface of the LV (the endocardium) where outward pressure is applied, represented by p.

the cavity of the LV, using the error measure

$$err_V = \left| \frac{V - \hat{V}}{V} \right| \times 100, \tag{3.24}$$

as cavity volume is an important quantity for clinicians in the diagnosis of certain cardiovascular diseases [199]. The distribution of err_V from Panel (d) indicates strong agreement between the simulator and emulator on the test data, with median error value of 3.7×10^{-2} %, while the largest error incurred was less than half a percent.

Emulation results for the `LeftVentricle` model for the test simulation where the emulator achieved the median value of Mean($err_{\boldsymbol{u}}$) are visualised in Figure 3.11. Comparing the FEM and PI-GNN results from panels (b) and (d) respectively shows strong agreement between the two. The distribution of $err_{\boldsymbol{u}}$ values from panel (c) indicates that errors are highest further from the base of the LV, and approach 1×10^{-1} mm at the apex.

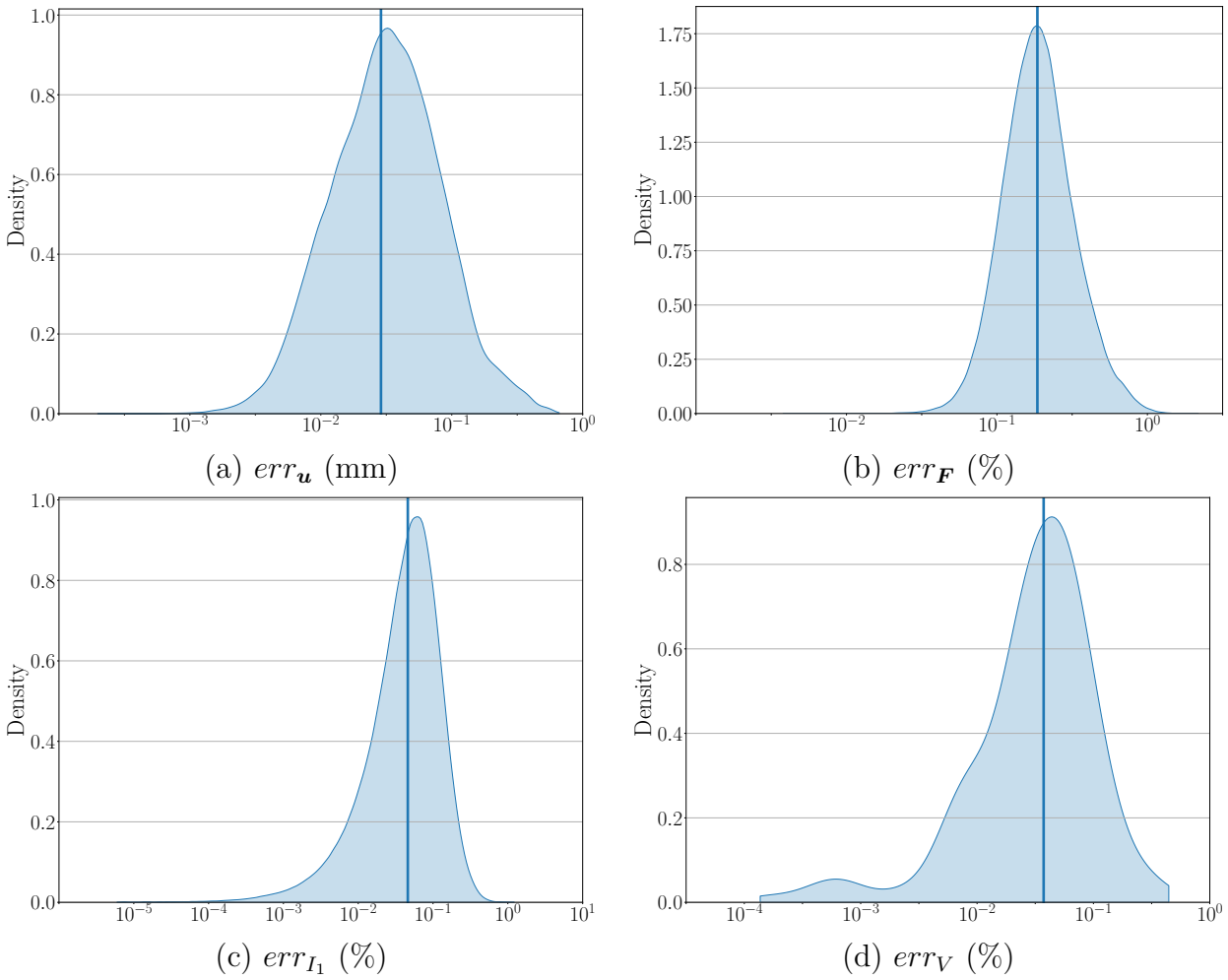
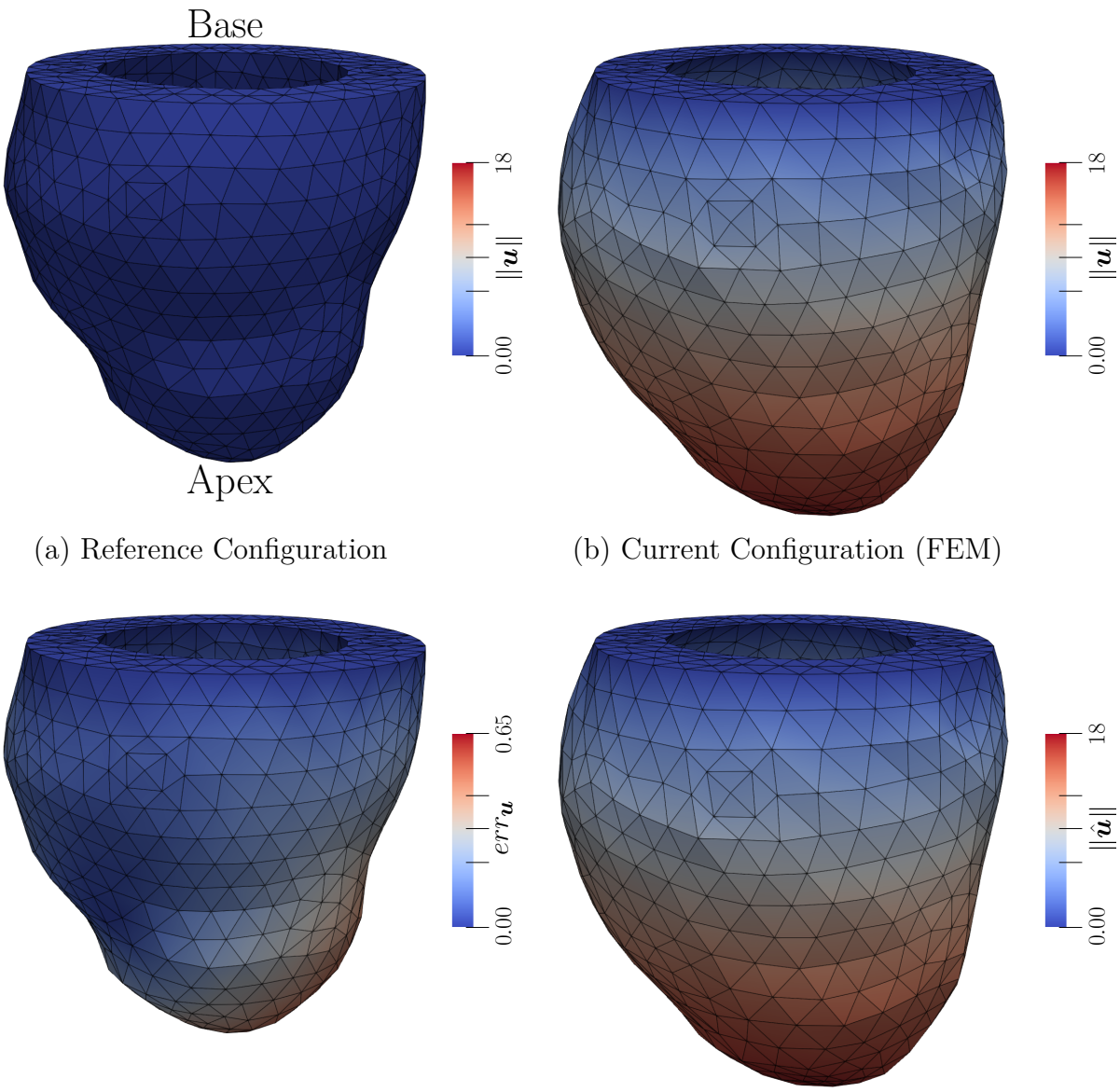


Figure 3.10: Error density plots for `LeftVentricle` model. The vertical lines indicate median values.

In addition to strong emulation accuracy, the emulator offers significant computational savings at prediction time over the FEM. A single forward evaluation for the `LeftVentricle` model takes 2.7×10^{-3} s, and once the message-passing stage is precomputed, only 2.3×10^{-4} s[10]. By contrast, simulations of the `LeftVentricle` model in FEniCS can take in excess of one hour[11]. Simulation times could however be reduced to the order of minutes by exploring parallel computing and solver optimisation in software such as ABAQUS. Nevertheless, even a one minute simulation time is prohibitively expensive for real-time applications where thousands of simulations need to be performed in sequence, as would be required for example with the use of a sampling-based Bayesian inference method for an inverse problem. As a consequence of the reduction in the computational costs, Bayesian sampling with our method becomes practically feasible. This

[10]Dual NVIDIA RTX A6000
[11]Dual Xeon Gold 6254



(a) Reference Configuration

(b) Current Configuration (FEM)

(c) $err_{\boldsymbol{u}} = \|\boldsymbol{u} - \hat{\boldsymbol{u}}\|_2$

(d) Current Configuration (PI-GNN)

Figure 3.11: Median out of sample emulation results for `LeftVentricle` model (mm). Note that the simulated inflation of the LV in the left column is slighly less than would be expected to be observed *in-vivo*.

indicates that our methodological improvements enable sound parameter inference with proper uncertainty quantification in real time, which would otherwise would not be feasible, thereby paving the path to genuine impact in the clinic. It is also trivial to parallelise the GNN to handle multiple input configurations simultaneously, using the `vmap` functionality in the JAX library.

3.4 Discussion

3.4.1 Data-Driven and Physics-Informed Training Comparison

The experiments from Section 3.3.1 reveal interesting differences between data-driven (DD) and physics-informed (PI) training. In particular, while both trainnig approaches yield similar errors in the prediction of the displacement field, the PI-GNN approache achieves much more accurate results as measured in terms of the potential energy, the deformation gradient $\boldsymbol{F}$ and first invariant I_1. This indicates that the PI-GNN is consistently capturing a more physically realistic displacement than the DD-GNN. In Section B.2 we re-perform these experiments using a DD-GNN where the loss function also includes a penalty term on $\boldsymbol{F}$. We find that this leads to improved accuracy, but nevertheless PI-GNN achieves lower values of err_{Π}, $err_{\boldsymbol{F}}$ and err_{I_1}.

The reason for the discrepancy in the accuracy of the deformation gradients recovered from the PI and DD predictions respectively relates to *bias*. In particular, we found that the prediction errors of the DD-GNN for the displacement field were relatively unbiased, whereas the errors for the PI-GNN were biased. Biased prediction errors in the displacement field cancel out in the evaluation of $\boldsymbol{F}$ (see Eq. (3.13), where we difference the predicted nodal positions), whereas unbiased errors propagate through this computation.

These results indicate that, while data-driven training may lead to accurate results in displacement space, these results are not guaranteed to respect the underlying physics. Therefore, if further quantities of interest are required beyond the displacements, for example stress and strain values, the use of physics-informed training will lead to more accurate results.

3.4.2 Computational Costs

The results from Table 3.2 indicate that an increase in mesh density leads to an increase in training and prediction times of a similar order of magnitude. We have not emphasised optimisation of training time in this work. Possible methods for doing so include mixed precision training, utilisation of multiple GPUs, or use of a second order optimisation approach such as conjugate gradients. Note that the particular advantage of a GNN

emulator is that training can be done offline - again, see [9] for details. Once the processor stage of the emulator is precomputed however, prediction times only increase by a factor of three when the mesh density increase by approximately a factor of sixty. This illustrates the advantage of our architecture design, which combines the modelling benefits of a GNN with the computational efficiency of an FCNN at prediction time.

3.4.3 `Liver` and `LeftVentricle` Emulation Results

The strong emulation accuracy for the `Liver` and `LeftVentricle` models highlights the ability of our PI-GNN approach to handle realistic models involving complex soft-tissue geometries[12]. The PI-GNN can also be applied to a soft-tissue geometry on which it has not seen during the initial training phase, which is demonstrated in Section B.5.

Comparing the results for the two real soft-tissue models, we see that higher accuracy was obtained for the `Liver` model, which assumed a Neo-Hookean material, compared to the `LeftVentricle`, where the more nonlinear H-O material model was used. This suggests that it may be easier to train in a physics-informed manner for more linear constitutive laws. A further comparison is explored in Section B.3, where the `LeftVentricle` emulation experiments are re-performed under the Neo-Hookean model. The results indicate that the emulator can consistently obtain a better approximation to the true Π under the more linear model, with lower error values in both $\boldsymbol{u}$ and $\boldsymbol{F}$ observed in turn. Nevertheless, predictions for the `LeftVentricle` model using the Holzapfel-Ogden material model were still highly accurate. For instance, the worst error observed in LV cavity volume of approximately half a percent is an order of magnitude lower than typical error measurements from manual segmentation of CMR scans [91]. In addition to strong predictive accuracy, the emulator is also several orders of magnitude less computationally expensive at prediction time when compared to the FEM. Furthermore, our PI-GNN implementation can be automatically differentiated using JAX, allowing $\partial\hat{\boldsymbol{U}}/\partial\boldsymbol{\theta}$ to be computed to machine precision at negligible additional computational costs. This combination of rapid, highly accurate predictions with end-to-end differentiability completely

[12]In Section B.4 we also consider emulation involving a biventricle cardiac geometry.

changes the range of applications for which soft-tissue models can be deployed in real time. For example, the parameter inference problem for passive cardiac mechanics considered in [60], which took over one week using the FEM, could be performed in seconds using a PI-GNN emulator.

3.4.4 Limitations and Future Work

The principle of minimum total potential energy is a fundamental concept in mechanics and engineering, and has been used in this work to handle soft-tissue mechanics problems using PI-GNNs, in a similar manner to some recent studies [185], [94]. An alternative approach can be derived from the weak formulation of PDE residuals based on the principle of virtual work, as done in [177]. With this approach, a physics-informed loss function may be defined as $L = \| (\int_\Omega \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{b}) \cdot \boldsymbol{\eta} dv \|_2$, with $\boldsymbol{\eta}$ the test function. There are a number of potential advantages to this approach, including the flexibility of handling time-dependent dynamic problems, discontinuous problems and non-conservative systems, i.e. shocks.

The strain energy density functions of the myocardium used here are considered to be slightly compressible, making use of the so-called F-bar method. To fully address incompressibility, future work can make use of the so-called multi-field variational principles [77, Chapter 8], where additional variables are introduced to take into account the incompressibility constraint.

In future directions of work we will extend the PI-GNN to consider higher-order finite elements with quadratic shape functions. This is to enhance the robustness of our method to the locking phenomenon which is a well-known issue when using linear tetrahedral elements for FE computations [200], in particular for incompressible or nearly-incompressible materials. A wider range of PDE systems with unstructured data could also be considered, for example material science [201], fluid dynamics (i.e. arterial blood flow [188], [202]), structural mechanics [177], and other large-scale engineering systems [203].

The results of the preliminary mesh convergence study (see Table 3.2) show that PI-GNN can match the FEM results well for different mesh densities, thus it can be expected that with increased mesh density, the emulator will obtain more accurate and reliable results. In real applications for clinical decision-making, the PI-GNN may also

need a fine discretisation as is the case for the FEM. However, the mesh density required by the surrogate model might be different from the classical FEM. There are some studies in the literature which explore this problem from a theoretical perspective, see for example He et al. [204] which draws an analogy between ReLU deep neural networks and linear finite elements.

Finally, the natural progression of the present work is to consider the inverse problem, that is, to infer material parameters for soft-tissue bodies from clinical data using a PI-GNN, and quantify the inference uncertainty. For the left ventricle, we have previously found that some parameters are weakly identifiable given observed strain values at 24 landmark points on the myocardium [164]. Emulation of the entire displacement field with a PI-GNN may provide important additional information that will reduce the posterior uncertainty. However, a verification of this hypothesis is beyond the remit of the present chapter and will be addressed in our future work.

3.5 Conclusion

This chapter has presented a PI-GNN emulation framework for application to soft-tissue mechanics. The GNN can operate directly on the unstructured mesh representation of a given soft-tissue geometry and is trained in a physics-informed manner by applying the principle of minimum total potential energy. Physics-informed training is enabled by the introduction of barrier transformation functions, which stabilise the objective function by explicitly incorporating known physical constraints such as the impenetrability of matter. A range of hyper-elastic models are considered, including realistic models of the human liver and left ventricle. Furthermore, significant computational savings at prediction time are made compared to the FEM. The authors believe this work is an important step in the development of real-time clinical applications of computational soft-tissue mechanics.

Chapter 4

Hard-constrained Gaussian processes for robust physics-informed learning of linear PDEs

Notation and Symbols

$\mathbb{N}$	set of all natural numbers
$\mathbb{R}$	set of all real numbers
$\mathbb{R}^D$	set of all D-tuples of real numbers
$\mathcal{X}$	space of input values
$C(\mathcal{X})$	space of continuous functions $u : \mathcal{X} \to \mathbb{R}$
$\|\cdot\|_\infty$	supremum norm
k	kernel/covariance function
$\boldsymbol{\xi}$	vector of kernel hyper-parameters
$\mathcal{H}$	Hilbert space
$\mathcal{H}_k$	reproducing kernel Hilbert space (RKHS)
$\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}$	linear partial differential equation (PDE) operator
$\boldsymbol{\theta}$	parameters of PDE operator
Ω	bounded subset of $\mathbb{R}^D$ on which PDE is defined
$\partial\Omega$	boundary of Ω
u	solution function to a boundary value problem
f	result of $\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}$ applied to u
b	known boundary function
ϕ	distance function
$\mathcal{GP}$	Gaussian process
$\tilde{k}$	hard-constrained covariance function
$(\tilde{m})$ m	(hard-constrained) mean function
$\mathcal{N}$	normal distribution
$\boldsymbol{y}_u/\boldsymbol{y}_f$	vectors of possibly noisy observations of u/f
$\mathbb{E}$/Cov	expectation/covariance operators
$\hat{u}$	prediction of u from machine learning model

4.1 Introduction

Physics-informed machine learning (PIML) has emerged in recent years as a new discipline which integrates data-based machine learning approaches with physics-based mathematical methods [59], [99]. A PIML model of a physical system leverages observational data with known physical principles, which can include for example boundary constraints, symmetries, partial differential equations (PDEs) and conservation laws. Physics informed approaches can offer more robust and interpretable predictions than data-based approaches, in addition to insights and inference about the system of interest that would not be possible without accounting for domain specific information. Consequently, PIML has rapidly become one the most topical research areas in computational physics and machine learning, with applications in a wide range of disciplines. Examples include quantum chemistry [205], solid mechanics [186] fluid dynamics [206], soft-tissue mechanics [10] and climate modelling [207].

Gaussian process regression (GPR) [130] is one machine learning framework which has found application in the context of PIML. GPR can be especially effective for data that is limited or expensive to obtain, and also offers well calibrated predictive uncertainty estimates that may be essential for scientific applications. In this work, we will consider the application of GPR to physical systems subject to boundary value and linear PDE constraints. GPR is particularly useful here, as the framework allows for seamless integration of observational data with linear PDEs. This in turn enables efficient joint inference of any unknown PDE parameters together with the unknown function itself. Our objective in this chapter is to expand upon existing work and design a GPR approach which also seamlessly integrates boundary information into the inference framework. We begin with a review of this related literature.

4.1.1 Motivation

The strategy proposed in this chapter was motivated by a stark difference we observed in the training behaviour of the PI-GNN from the previous chapter, depending on how the Dirichlet boundary conditions were handled. Recall from Section 3.2.4.4 that we applied a post-processing step to the output of the PI-GNN, which ensured that the Dirichlet conditions were exactly satisfied, irrespective of the value of the emulation parameters. As a result, the boundary conditions did not have to be explicitly handled when training the emulator. Alternatively, it is also possible to incorporate boundary conditions in a *soft* manner without using a post-processing step, by including an additional term to the physics-informed objective function (Eq. (3.12)) which penalises emulator prediction errors at the boundary. When performing experiments, however, we found that training using the penalty approach was very inefficient, as it was extremely difficult to tune the learning rate so that both the potential energy was minimised and boundary conditions satisfied. Note that this problem was not specific to the setting of 3D solid mechanics - we found that it arises even for a simple 1D Poisson equation (see Section 1.7.3). Learning rate scheduling approaches have been proposed to overcome the issue [208], however we found it more convenient to use explicit boundary condition enforcement. In this chapter, then, we delve more deeply into the idea of hard boundary condition enforcement, by developing and evaluating approaches for explicit enforcement of a wider range of boundary conditions typically encountered in PDE modelling, beyond simple Dirichlet conditions. We also make use of physics-informed *Gaussian processes* in this chapter, in place of neural networks, as were used in the previous two chapters. However, as we detail below, the two approaches are strongly related in this context (see Section 4.6).

4.1.2 Related Work

The closure of GPs under linear operators has been long understood **ADLER**, which follows from the closure of multivariate Gaussian random variables under linear/affine functions. Early work leveraged this property of GPs to build models to incorporate derivative information [209], learn latent forces [210], and model ordinary and partial differential equations [183], [211], [212]. Of specific relevance to the material presented here is the physics-informed Gaussian process (PIGP) inference framework introduced in the seminal work [102], where it was outlined how noise levels, any unknown PDE parameters and the function of interest itself could be jointly inferred using GPs.

A recent thread of research in the GP literature has focused on the design of models that exactly satisfy known boundary conditions in advance of training. In the context of data-driven surrogate modelling, approaches have been proposed which design non-stationary mean and covariance functions so that boundary conditions on the value of the unknown function itself are satisfied [213], [214]. This type of approach can also be used in the context of multi-output GPs, to enforce both boundary and PDE constraints [215]. Other approaches have also been proposed, including the design of a bespoke kernel in terms hyperbolic sine and cosine functions so that Dirichlet conditions are satisfied [216], and the use kernels derived from variational harmonic features which can be used to enforce a range of boundary conditions [217], [218].

Physics-informed neural networks (PINNs) are an alternative paradigm for the machine learning of PDE systems [101]. While PINNs are considered to be a more "data-hungry" method than PIGPs, they have the advantage of not being limited to the modelling of linear PDEs. Contemporaneous with the introduction of the above hard-constrained methods in the PIGP literature has been the development of methods for imposing the exact same type of constraints in PINNs. As we discuss in Section 4.6, these two threads of research are in fact highly analogous. Early work in PINNs used simple distance functions to enforce homogeneous Dirichlet conditions [186], while more recent work has introduced techniques to handle more complex domains and more versatile boundary conditions [219]–[221].

4.1.3 Contributions

In this chapter, we introduce a GP modelling framework for imposing hard enforcement of boundary conditions, which naturally accommodates time-dependent PDEs in the context of both inverse and forward problems. We do this by extending existing approaches [213], [214] beyond Dirichlet conditions to enforce more general Neumann, Cauchy and Robin conditions. We prove that the construction allows for universal approximation within the boundary-constrained function space under Dirichlet conditions, and also introduce a theoretical link between the hard-constrained PINN and PIGP literatures in the limit of an infinite width network. Furthermore, we then conduct extensive numerical experiments involving multiple PDE systems, the results of which demonstrate that explicitly enforcing boundary conditions allows for more *robust* inference, when compared to methods which ignore boundary conditions or which introduce boundary information using a penalty approach.

The chapter is laid out as follows; Sections 4.2 and 4.3 give background material on GPR and linear PDEs, respectively, while Section 4.4 describes how Dirichlet boundary conditions can be explicitly enforced in a GP model. Sections 4.5 and 4.6 present theoretical results which explore respectively the representation capacity of the kernel used to enforce Dirichlet conditions, and the connection between neural networks and GPs in this context. Section 4.7 then outlines how general boundary conditions can be modelled in addition to Dirichlet conditions, before Section 4.8 details the numerical experiments we conducted to evaluate these methods. Section 4.9 concludes. Supplementary material is provided in Appendix C.

4.2 Gaussian process regression

Regression refers to the process of using a finite set of possibly noise corrupted data to perform inference about an unknown function of interest, which we denote $u : \mathcal{X} \to \mathbb{R}$. Specifically, suppose a dataset of input-output pairs $(\boldsymbol{x}_u^{(i)}, y_u^{(i)})$ has been observed for $i = 1, \ldots, N_u$, with the following noise model assumed for each observation:

$$y_u^{(i)} = u(\boldsymbol{x}_u^{(i)}) + \epsilon_u^{(i)} \text{ with } \boldsymbol{x}_u^{(i)} \in \mathcal{X} \text{ and } \epsilon_u^{(i)} \sim \mathcal{N}(0, \sigma_u^2). \tag{4.1}$$

The goal of regression is to use these observations $\boldsymbol{y}_u = (y_u^{(1)}, \ldots, y_u^{(N_u)})^\top$ to learn the underlying function, allowing its output $\boldsymbol{u}_* = (\boldsymbol{u}_*^{(1)}, \ldots, \boldsymbol{u}_*^{(N_*)})^\top$ to be predicted at any test points of interest $\boldsymbol{x}_*^{(1)}, \ldots, \boldsymbol{x}_*^{(N_*)}$. In this section, we discuss the essential concepts of a non-parametric Bayesian method for performing regression based on *Gaussian processes*. For further details, we direct the reader to [130, Chapter 2] and [222, Chapter 18], while [223] offers a more technical introduction.

Central to the formulation of Gaussian processes are *Mercer kernels* [223, Definition 2.1], which we will henceforth refer to as simply kernels.

Definition 4.2.1 (Mercer kernel). *Let $\mathcal{X}$ be a nonempty set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a Mercer kernel, if for any $N \in \mathbb{N}, c_1, \ldots, c_N \subset \mathbb{R}$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \subset \mathcal{X}$, we have*

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \geq 0. \tag{4.2}$$

This property of kernels is referred to as positive-definiteness. The prototypical example of a kernel when the input space is Euclidean is the *squared-exponential*.

Example 4.2.1 (Squared-exponential kernel). *Let $\mathcal{X} \subset \mathbb{R}^D$. Given $\tau, \lambda > 0$, a squared-exponential kernel k_{SE} is defined as*

$$k_{SE}\left(\boldsymbol{x}, \boldsymbol{x}'; \tau, \lambda\right) \triangleq \tau^2 \exp\left(-\frac{\left\|\boldsymbol{x} - \boldsymbol{x}'\right\|^2}{2\lambda^2}\right), \quad \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}. \tag{4.3}$$

A kernel will in general depend on a set of hyperparameters, which we will denote as $\boldsymbol{\xi}$. With the squared exponential kernel, we have $\boldsymbol{\xi} = (\tau, \lambda)$, where τ is called the amplitude and λ the lengthscale. For ease of notation we usually leave this dependence implicit, i.e. writing $k(\boldsymbol{x}, \boldsymbol{x}')$ in place of $k(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi})$.

We are now ready to define a Gaussian process [223, Definition 2.2], where kernels allow for the covariance between random function outputs to be specified and are hence often called covariance functions in this context.

Definition 4.2.2 (Gaussian process). *Let $\mathcal{X}$ be a non-empty set, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a Mercer kernel and $m : \mathcal{X} \to \mathbb{R}$ any function. Then a random function $u : \mathcal{X} \to \mathbb{R}$ is called a Gaussian process with mean m and covariance k, which we denote*

$$u(\boldsymbol{x}) \sim \mathcal{GP}\left(m(\boldsymbol{x}), k\left(\boldsymbol{x}, \boldsymbol{x}'\right)\right), \tag{4.4}$$

if, for any finite collection of inputs $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \in \mathcal{X}$, the distribution of the corresponding outputs $\boldsymbol{u} = (u(\boldsymbol{x}^{(1)}), u(\boldsymbol{x}^{(2)}), ..., u(\boldsymbol{x}^{(N)}))^\top$ is Gaussian, that is $p(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{m}, \mathbf{K})$, where the mean vector $\boldsymbol{m}$ and covariance matrix $\mathbf{K}$ are found by evaluating the mean and covariance functions from Eq. (4.4) as follows:

$$\mathbb{E}\left(u(\boldsymbol{x}^{(i)})\right) = m^{(i)} = m(\boldsymbol{x}^{(i)}), \tag{4.5}$$

$$\mathrm{Cov}\left(u(\boldsymbol{x}^{(i)}), u(\boldsymbol{x}^{(j)})\right) = K^{(i,j)} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}). \tag{4.6}$$

Performing *Gaussian process regression* (GPR) begins with the assumption that the unknown function of interest u follows a Gaussian process with specified mean and covariance functions, which we denote m_u and k_{uu} respectively. This assumption, along with the independent and identically distributed Gaussian observation noise model (see Eq. (4.1)), implies that the training observations $\boldsymbol{y}_u \in \mathbb{R}^{N_u \times 1}$ and unknown test outputs $\boldsymbol{u}_* \in \mathbb{R}^{N_* \times 1}$ have the following joint Gaussian distribution:

$$\begin{bmatrix} \boldsymbol{y}_u \\ \boldsymbol{u}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m}_u \\ \boldsymbol{m}_* \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{uu} + \sigma_u^2 \mathbf{I}_{N_u} & \mathbf{K}_{u*} \\ \mathbf{K}_{*u} & \mathbf{K}_{**} \end{bmatrix}\right). \tag{4.7}$$

Here, $\boldsymbol{m}_u \in \mathbb{R}^{N_u \times 1}$ and $\boldsymbol{m}_* \in \mathbb{R}^{N_* \times 1}$ are found using the mean function m_u as in Eq. (4.5), while $\mathbf{K}_{uu} \in \mathbb{R}^{N_u \times N_u}$ and $\mathbf{K}_{**} \in \mathbb{R}^{N_* \times N_*}$ are found using the covariance function k_{uu} as in Eq. (4.6). $\boldsymbol{I}_{N_u} \in \mathbb{R}^{N_u \times N_u}$ is the identity matrix while the off diagonal terms are found as $K_{u*}^{(j,i)} = K_{*u}^{(i,j)} = k_{uu}(\boldsymbol{x}_*^{(i)}, \boldsymbol{x}_u^{(j)})$.

Given this joint distribution, the regression task reduces to evaluating $p(\boldsymbol{u}_* \mid \boldsymbol{y}_u)$, the conditional distribution of the unknown test function outputs given the observed data. The properties of the multivariate Gaussian are such that this distribution is again a Gaussian [222, Section 2.3.1.5].

Proposition 4.2.1 (Posterior Predictive Gaussian Distribution). *Let $\boldsymbol{u}_*$ and $\boldsymbol{y}_u$ be jointly Gaussian distributed as in Eq. (4.7). Then $p(\boldsymbol{u}_* \mid \boldsymbol{y}_u) = \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ with*

$$\begin{aligned} \boldsymbol{\mu}_* &= \boldsymbol{m}_* + \mathbf{K}_{u*}^\top \left(\mathbf{K}_{uu} + \sigma_u^2 \mathbf{I}_{N_u}\right)^{-1} (\boldsymbol{y}_u - \boldsymbol{m}_u), \\ \boldsymbol{\Sigma}_* &= \mathbf{K}_{**} - \mathbf{K}_{u*}^\top \left(\mathbf{K}_{uu} + \sigma_u^2 \mathbf{I}_{N_u}\right)^{-1} \mathbf{K}_{u*}. \end{aligned} \tag{4.8}$$

Carrying out GPR requires then the mean and covariance function to be specified. A common choice in practice is to use a zero mean function and squared exponential kernel. Various approaches can then be used to fit any tunable mean/kernel parameters, including Markov-chain Monte-Carlo, variational inference and empirical Bayesian methods respectively.

4.3 GPR under Linear PDE constraints

In this work, we study processes for which *linear* partial differential equation (PDE) information is available, which we define as follows [224, Definition 1].

Definition 4.3.1 (Linear PDE). *Let u be a real valued function on some subset of $\mathbb{R}^D$. A PDE $\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u](\boldsymbol{x}) = f(\boldsymbol{x})$ with possibly non-homogeneous term $f(\boldsymbol{x})$ is called a linear PDE if $\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u](\boldsymbol{x})$ is a linear differential operator:*

$$\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u](\boldsymbol{x}) \triangleq \sum_{i=1}^{L} c_i(\boldsymbol{x}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\alpha}_i} u(\boldsymbol{x}), \qquad \textit{with} \qquad \nabla_{\boldsymbol{\alpha}_i} u \triangleq \frac{\partial^{\alpha_{i,1}}}{\partial x_1^{\alpha_{i,1}}} \cdots \frac{\partial^{\alpha_{i,D}}}{\partial x_D^{\alpha_{i,D}}} u, \tag{4.9}$$

where $\boldsymbol{\theta}$ parameterises the PDE, L is the number of derivatives, $\boldsymbol{\alpha}_i = (\alpha_{i,1}, \cdots, \alpha_{i,D})$ indicates the order of the derivative for each input dimension, $\boldsymbol{x} = (x_1, \cdots, x_D)$ and $c_i(\boldsymbol{x}, \boldsymbol{\theta})$ the coefficient at $\boldsymbol{x}$.

Suppose that, in addition to observations $\boldsymbol{y}_u \in \mathbb{R}^{N_u \times 1}$ in u-space (see Eq. (4.1)), observations in $\boldsymbol{y}_f \in \mathbb{R}^{N_f \times 1}$ in f-space are available with observation model

$$y_f^{(i)} = f(\boldsymbol{x}_f^{(i)}) + \epsilon_f^{(i)} \text{ with } \epsilon_f^{(i)} \sim \mathcal{N}(0, \sigma_f^2), \tag{4.10}$$

for all $i = 1, \ldots, N_f$. We seek to incorporate $\boldsymbol{y}_u$ and $\boldsymbol{y}_f$ into a joint inference framework using Gaussian processes. To do so, we employ the GPR algorithm introduced in [102], which is generalised in this section to allow for non-zero mean function. As with usual GPR, the algorithm begins by assuming that $u(\boldsymbol{x})$ follows a Gaussian process,

$$u(\boldsymbol{x}) \sim \mathcal{GP}\left(m_u(\boldsymbol{x}), k_{uu}\left(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi}\right)\right). \tag{4.11}$$

For clarity, we use subscripts in this section to denote the spaces in which observations are made, so that $m_g(\boldsymbol{x}) = \mathbb{E}\left(g(\boldsymbol{x})\right)$ and $k_{gh}(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{Cov}\left(g(\boldsymbol{x}), h(\boldsymbol{x}')\right)$ for $g, h \in \{u, f\}$. Additionally, we make explicit the dependence of the kernel on any hyperparameters $\boldsymbol{\xi}$ (we assume the mean function has no trainable parameters).

They key insight required here is that Gaussian processes are closed under linear operations [130, Section 9.4]. This means that our assumption for the distribution of $u(\boldsymbol{x})$ in Eq. (4.11) implies that

$$\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u](\boldsymbol{x}) = f(\boldsymbol{x}) \sim \mathcal{GP}\left(m_f(\boldsymbol{x}; \boldsymbol{\theta}), k_{ff}\left(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi}, \boldsymbol{\theta}\right)\right). \tag{4.12}$$

Furthermore, we have the following fundamental relationship between the mean and covariance functions of the two processes [102]:

$$m_f(\boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}} m_u(\boldsymbol{x}), \tag{4.13}$$

$$k_{ff}\left(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi}, \boldsymbol{\theta}\right) = \mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{x}'}^{\boldsymbol{\theta}} k_{uu}\left(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi}\right). \tag{4.14}$$

Similarly, the cross-covariance between the observations of the two processes are found as

$$k_{uf}\left(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi}, \boldsymbol{\theta}\right) = \mathcal{L}_{\boldsymbol{x}'}^{\boldsymbol{\theta}} k_{uu}\left(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi}\right), \tag{4.15}$$

$$k_{fu}\left(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi}, \boldsymbol{\theta}\right) = \mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}} k_{uu}\left(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\xi}\right) \tag{4.16}$$

Note that we are assuming here that the chosen kernel k_{uu} is sufficiently smooth for the PDE operator to be applied, an issue which is discussed in detail in **ADLER**. The above properties of GPs along with our Gaussian noise assumptions for observations $\boldsymbol{y}_u$ in u-space and observations $\boldsymbol{y}_f$ in f-space imply the following joint distribution for the observed data.

Proposition 4.3.1 (Joint distribution of $\boldsymbol{y}_u$ and $\boldsymbol{y}_f$). *Let u follow a Gaussian process as in Eq. (4.11), with $\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u] = f$. Assume $\boldsymbol{y}_u \in \mathbb{R}^{N_u \times 1}$ has been observed with noise model given in Eq. (4.1), and $\boldsymbol{y}_f \in \mathbb{R}^{N_f \times 1}$ has been observed with noise model in Eq. (4.10). Then, $\boldsymbol{y}_u$ and $\boldsymbol{y}_f$ follow a joint normal distribution $p(\boldsymbol{y}_u, \boldsymbol{y}_f; \boldsymbol{\xi}, \boldsymbol{\theta}, \sigma_u^2, \sigma_f^2)$ of the form*

$$\begin{bmatrix} \boldsymbol{y}_u \\ \boldsymbol{y}_f \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m}_u \\ \boldsymbol{m}_f \end{bmatrix}, \boldsymbol{K}_{\boldsymbol{yy}} \right), \quad \textit{with} \quad \boldsymbol{K}_{\boldsymbol{yy}} = \begin{bmatrix} \boldsymbol{K}_{uu} + \sigma_u^2 \boldsymbol{I}_{N_u} & \boldsymbol{K}_{uf} \\ \boldsymbol{K}_{fu} & \boldsymbol{K}_{ff} + \sigma_f^2 \boldsymbol{I}_{N_f} \end{bmatrix}, \tag{4.17}$$

where $m_g^{(i)} = m(\boldsymbol{x}_g^{(i)})$ and $K_{gh}^{(i,j)} = k_{gh}(\boldsymbol{x}_g^{(i)}, \boldsymbol{x}_h^{(i)})$ for $g, h \in \{u, f\}$, while $\boldsymbol{I}_N$ is the identity matrix of dimension N, for $N \in \{N_u, N_f\}$.

A major contribution of [102] was to recognise that the parameters $\boldsymbol{\theta}$ of the linear PDE operator are turned into hyperparameters of m_f, k_{ff}, k_{uf} and k_{fu} (note how these functions depend on $\boldsymbol{\theta}$ in Eqs. (4.13)-(4.16)). Therefore, if these parameters are unknown, they can be inferred in the same manner as the original kernel parameters $\boldsymbol{\xi}$. As in [102], we perform inference by maximisation of $p(\boldsymbol{y}_u, \boldsymbol{y}_f; \boldsymbol{\xi}, \boldsymbol{\theta}, \sigma_u^2, \sigma_f^2)$:

$$\{\hat{\boldsymbol{\xi}}, \hat{\boldsymbol{\theta}}, \hat{\sigma}_u^2, \hat{\sigma}_f^2\} = \underset{\boldsymbol{\xi}, \boldsymbol{\theta}, \sigma_u^2, \sigma_f^2}{\operatorname{argmax}} \log p(\boldsymbol{y}_u, \boldsymbol{y}_f; \boldsymbol{\xi}, \boldsymbol{\theta}, \sigma_u^2, \sigma_f^2), \tag{4.18}$$

where the log is taken for reasons of numerical stability.

When viewed as a function of the tunable parameters given fixed observations as in Eq. (4.18) above, $p(\boldsymbol{y}_u, \boldsymbol{y}_f; \boldsymbol{\xi}, \boldsymbol{\theta}, \sigma_u^2, \sigma_f^2)$ is called the *marginal likelihood* or *evidence* of the observed data. Maximisation of the log marginal likelihood is a common method for training GPs, as this quantity balances a trade-off between model fit and complexity [222, Section 3.8.1]. This can be seen by writing the objective function out in explicit form [222, Section 18.3.5], where we suppress dependence on the parameters $\boldsymbol{\xi}, \boldsymbol{\theta}, \sigma_u^2, \sigma_f^2$ and ignore

any added constants for notational convenience:

$$\log p(\boldsymbol{y}_u, \boldsymbol{y}_f) = -\overbrace{\frac{1}{2}\begin{bmatrix} \boldsymbol{y}_u - \boldsymbol{m}_u \\ \boldsymbol{y}_f - \boldsymbol{m}_f \end{bmatrix}^\top \boldsymbol{K}_{\boldsymbol{yy}}^{-1} \begin{bmatrix} \boldsymbol{y}_u - \boldsymbol{m}_u \\ \boldsymbol{y}_f - \boldsymbol{m}_f \end{bmatrix}}^{\text{Data-Fit}} - \overbrace{\frac{1}{2}\log|\boldsymbol{K}_{\boldsymbol{yy}}|}^{\text{Regularisation}}. \tag{4.19}$$

The first term above is the squared distance between the observed and predicted values under the Mahalanobis metric [222, Section 2.3.1]. This is a *data-fit* term, as it favours models which better fit the observations. The second term is the log determinant of $\boldsymbol{K}_{\boldsymbol{yy}}$, which is a measure of model complexity, since smoother functions will yield smaller determinants. Therefore, this is a *regularisation* term, as it favours more simple models. By balancing fit and complexity, the marginal likelihood can enable effective model training even in the low data regime. This is discussed further in the context of a numerical experiment in Section 4.8.1.

Once the GP has been trained, prediction on any new test points of interest follows almost the exact same formulas as presented in Eq. (4.8). The only difference is that any terms involving the training data now have a block structure to account for the fact that observations are available in two different spaces. So for example, $\mathbf{K}_{u*}^\top$ in Eq. (4.8) is replaced with a matrix of the form $[\mathbf{K}_{u*}^\top \; \mathbf{K}_{f*}^\top]$.

4.4 Hard-Enforcement of Dirichlet Boundary Conditions

Solutions to PDEs are not uniquely defined. To make a problem well posed, a boundary value problem (BVP) is considered, by subjecting the PDE to boundary constraints. Commonly used boundary conditions include Dirichlet, Neumann, Cauchy and Robin conditions. When a PDE is time-dependent, an initial boundary value problem (IBVP) can be defined, whereby an additional constraint is placed on the initial condition of the system. For notational simplicity, however, in the following we will not make explicit any temporal components to a process. Instead, several numerical experiments involving IBVPs are considered in Section 4.8, where we illustrate that initial conditions can be handled in exactly the same manner as boundary conditions.

As discussed in Section 4.1.2, a recent thread of research in the application of GPs to BVPs has been in the specification of mean and covariance functions so that the prescribed boundary conditions are exactly satisfied *a-priori*. In this work, we call such models *hard-constrained* GPs (HCGPs). Consider first the case of Dirichlet boundary conditions. If we consider a bounded domain $\Omega \subset \mathbb{R}^D$ with $\partial\Omega$ its boundary, then a Dirichlet boundary condition takes the form

$$u(\boldsymbol{x}) = b(\boldsymbol{x}) \text{ for all } \boldsymbol{x} \in \partial\Omega, \tag{4.20}$$

where $u : \Omega \to \mathbb{R}$ is the unknown function of interest (which we assume is continuous) and $b : \partial\Omega \to \mathbb{R}$ is known. In this case, a HCGP can be defined as follows.

Definition 4.4.1 (Hard-Constrained Gaussian Process (HCGP) for Dirichlet Boundary Conditions). *Consider bounded domain $\Omega \subset \mathbb{R}^D$ with boundary $\partial\Omega$ and known boundary function $b : \partial\Omega \to \mathbb{R}$. Then let $\tilde{m} : \Omega \to \mathbb{R}$ be any continuous function with $\tilde{m}(\boldsymbol{x}) = b(\boldsymbol{x})$ if $\boldsymbol{x} \in \partial\Omega$, and $\phi : \Omega \to \mathbb{R}$ be a continuous distance function satisfying $\phi(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \partial\Omega$ with $\phi(\boldsymbol{x}) > 0$ otherwise. Finally, let $\tilde{k}(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})\phi(\boldsymbol{x}')k(\boldsymbol{x}, \boldsymbol{x}')$ for arbitrary covariance function $k : \Omega \times \Omega \to \mathbb{R}$. Then we call a hard-constrained Gaussian processes (HCGP) a model of the form*

$$u(\boldsymbol{x}) \sim \mathcal{GP}\left(\tilde{m}(\boldsymbol{x}), \tilde{k}(\boldsymbol{x}, \boldsymbol{x}')\right). \tag{4.21}$$

We remark that $\tilde{k}$ above defines a valid covariance function since any inner product $\phi(\boldsymbol{x})\phi(\boldsymbol{x}')$ with $\phi : \Omega \to \mathbb{R}$ defines a valid kernel [225, Definition 4.1] and the product of two kernels is a kernel [225, Lemma 4.6].

The design of the mean and covariance functions in the HCGP is motivated by the elementary result from probability theory that if c is a deterministic/known variable, then $\text{Var}(c) = 0$, and additionally for any random variable X, we have $\text{Cov}(c, X) = 0$. Therefore, under Dirichlet conditions where the output of the function is known exactly on $\partial\Omega$, the variance of the GP prior on u should be zero at the boundary, that is $\tilde{k}(\boldsymbol{x}, \boldsymbol{x}') = 0$ if $\boldsymbol{x} \in \partial\Omega$, with mean function equal to the known value, that is $\tilde{m}(\boldsymbol{x}) = b(\boldsymbol{x})$. Similarly, the covariance should equal zero if one of the input points lies on the boundary, i.e. $\tilde{k}(\boldsymbol{x}, \boldsymbol{x}') = 0$ if $\boldsymbol{x} \in \partial\Omega$ and/or $\boldsymbol{x}' \in \partial\Omega$. Clearly, the forms of $\tilde{m}$ and $\tilde{k}$ in Definition 4.4.1

ensure that these conditions are met. In this sense, then, the HCGP model imposes a hard enforcement of the Dirichlet conditions[13]. The explicit construction of mean function $\tilde{m}$ and distance function ϕ in the case of a unit-cube domain is described in Section 4.4.1 below.

The advantage of using a HCGP over a more general GP is that our prior knowledge of the Dirichlet boundary conditions is reflected directly in the function space on which we place our Gaussian process prior, and therefore does not have to be learned from data. An extension to the framework presented here is required if more general Cauchy, Neumann and Robin boundary conditions are to be considered. We are not aware of any work in the literature on GPs which discusses this issue, however, and so we present the extension in Section 4.7.

In most numerical experiments performed in Section 4.8, we consider Ω to be a grid-like subset of $\mathbb{R}^D$. As we discuss in Section 4.4.1 below, this allows for simple hand-derivations of the form of the required HCGP. However we stress that the methods presented here are not limited to such simple domains. Several approaches recently presented in the PINN literature such as [219], [220] could easily be applied to specify boundary constrained mean and covariance functions $\tilde{m}$ and $\tilde{k}$ respectively if the PDE domain Ω has a more complex form. We present an experiment illustrating this in Section 4.8.1.

4.4.1 Example - unit cube domain

We assume now that our domain of interest is the unit cube in $\mathbb{R}^D$, i.e. $\Omega = [0,1]^D$. In this case, the boundary $\partial\Omega$ can be decomposed as

$$\partial\Omega = \bigcup_{i=1}^{D}\bigcup_{j=0}^{1} \partial\Omega_{ij}, \text{ where } \partial\Omega_{ij} = \{\boldsymbol{x} = (x_1, \ldots, x_D) \in \Omega : x_i = j\}. \tag{4.22}$$

Typically in the literature on BVPs, no intersection between different boundaries is allowed. To simplify notation however, here we have relaxed this to allow for possibly non-null but zero-measure intersections. Note this makes no practical difference as we are assuming continuity in the underlying function of interest u.

[13]For a more technical discussion of this idea in a similar context, we direct the reader to [214] (specifically Proposition 1).

Algorithm 3 HCGP Generator

Input: Covariance function $k : \Omega \times \Omega \to \mathbb{R}$, boundary function $b : \partial\Omega \to \mathbb{R}$
Output: $\tilde{m}, \tilde{k}$

1: $\tilde{m}_1(\boldsymbol{x}) = b(\boldsymbol{x} : x_1 = 0)(1 - x_1) + b(\boldsymbol{x} : x_1 = 1)x_1$
2: **for** $i = 2 : D$
3: $\quad \tilde{b}_{i,j}(\boldsymbol{x}) = b(\boldsymbol{x} : x_i = j) - \tilde{m}_{i-1}(\boldsymbol{x} : x_i = j)$ for all $j \in \{0, 1\}$
4: $\quad \tilde{m}_i(\boldsymbol{x}) = \tilde{m}_{i-1}(\boldsymbol{x}) + \tilde{b}_{i,0}(\boldsymbol{x})(1 - x_i) + \tilde{b}_{i,1}(\boldsymbol{x})x_i$
5: **end for**
6: $\tilde{m}(\boldsymbol{x}) \triangleq \tilde{m}_D(\boldsymbol{x})$
7: $\phi(\boldsymbol{x}) \triangleq \Pi_{i=1}^{D} x_i(1 - x_i)$
8: $\tilde{k}(\boldsymbol{x}, \boldsymbol{x}') \triangleq \phi(\boldsymbol{x})\phi(\boldsymbol{x}')k(\boldsymbol{x}, \boldsymbol{x}')$

The boundary constrained mean and covariance functions for specifying a HCGP can then be generated using Algorithm 3. The mean function is found initially using an iterative procedure. For the first input dimension (x_1), the mean is initialised on line 1 by linearly interpolating between the specified boundary functions on $\partial\Omega_{10}$ and $\partial\Omega_{11}$. For each additional dimension $i = 2, \ldots, D$, the mean is augmented by adding a similar linear interpolation for the i^{th} input dimension on line 4. The difference is that the interpolation is done with respect to augmented boundary functions $\tilde{b}_{i,0}$ and $\tilde{b}_{i,1}$ found on line 3, which ensures that the boundary values for dimensions $l < i$ are not affected by the augmentation procedure. The construction of $\tilde{k}$ on line 8 is even simpler, by combining the original kernel k with a feature transformation ϕ defined on line 7 which satisfies $\phi(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \partial\Omega$ and $\phi(\boldsymbol{x}) > 0$ otherwise.

We use this algorithm to specify the HCGPs considered in Section 4.8, adjusted slightly to account for IBVPs and for alternative boundary conditions. We remark that, clearly $\tilde{k}(\boldsymbol{x}, \boldsymbol{x}') = 0$ if $\boldsymbol{x} \in \partial\Omega$ and/or $\boldsymbol{x}' \in \partial\Omega$. Furthermore, the function $\tilde{m}$ exactly satisfies the given Dirichlet boundary conditions.

Lemma 4.4.1. *Dirichlet boundary conditions of the form given in Eq. (4.20), let $\tilde{m}$ be the mean function generated by Algorithm 3. Then we have*

$$\tilde{m}(\boldsymbol{x}) = b(\boldsymbol{x}) \;\; \textit{for all } \boldsymbol{x} \in \partial\Omega. \tag{4.23}$$

This result is proved in Section C.1.

4.5 Reproducing-Kernel Hilbert Space Analysis

In Section 4.2, we introduced a kernel as a positive definite function appropriate for use as a covariance function in the specification of a Gaussian process. Kernels also have a deep and one-to-one correspondence with Hilbert spaces of functions whose evaluation functional is continuous. Such function spaces are called *reproducing kernel Hilbert spaces* (RKHSs). This property of kernels allows for theoretical analysis of their representational capacity in terms of an associated RKHS, which we will make use of to analyse the boundary constrained kernel used in the specification of a HCGP for Dirichlet boundary conditions (Definition 4.4.1). To do so, we begin with some foundational results from kernel and RKHS theory. For a comprehensive introduction to this topic, see [225, Chapter 4].

Definition 4.5.1 (Reproducing Kernel Hilbert Space (RKHS) [223]). *Let $\mathcal{X}$ be a non-empty set, k a kernel on $\mathcal{X}\times\mathcal{X}$ and $\mathcal{H}$ an $\mathbb{R}$-Hilbert space over $\mathcal{X}$. That is $\mathcal{H}$ is a vector space consisting of functions u that map $\mathcal{X}$ to $\mathbb{R}$, which is equipped with an inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ that induces a metric for which the space is complete. $\mathcal{H}$ is additionally called a reproducing kernel Hilbert space (RKHS) with reproducing kernel k, if the following are satisfied:*

1. *For all $\boldsymbol{x}\in\mathcal{X}$, we have $k(\cdot,\boldsymbol{x})\in\mathcal{H}$;*
2. *For all $\boldsymbol{x}\in\mathcal{X}$ and for all $u\in\mathcal{H}$, we have*

$$u(\boldsymbol{x}) = \langle u(\cdot), k(\cdot,\boldsymbol{x})\rangle_{\mathcal{H}}.$$

The second condition above is called the *reproducing property* of the kernel. We introduce the notation $\mathcal{H}_k$ to denote an RKHS with reproducing kernel k, given the result of the below theorem.

Theorem 4.5.1 (Moore–Aronszajn Theorem [226]). *Let $\mathcal{X}$ be a nonempty set. Then for every kernel k on $\mathcal{X}\times\mathcal{X}$, there exists a unique RKHS $\mathcal{H}$ for which it is the reproducing kernel, and vice-versa.*

For Theorems 4.5.2, 4.5.3 and 4.5.3 below, the concept of *density* in a metric space is required [227, Theorem 4.2.1].

Definition 4.5.2 (Dense subset of a metric space). *A subset $\mathcal{D}$ of a metric space $\mathcal{X}$ with metric d is called dense in $\mathcal{X}$ if, for all $\varepsilon > 0$ and all $\boldsymbol{x} \in \mathcal{X}$, there exists $\hat{\boldsymbol{x}} \in \mathcal{D}$ such that $d(\boldsymbol{x}, \hat{\boldsymbol{x}}) < \varepsilon$.*

The following result allows us to consider the elements of $\mathcal{H}_k$ in terms of finite weighted sums of k [225, Theorem 4.21].

Theorem 4.5.2 (Reproducing kernel map representation). *Let $\mathcal{X}$ be a non-empty set and k a kernel on $\mathcal{X} \times \mathcal{X}$, with $\mathcal{H}_k$ its associated RKHS. Consider the set of functions*

$$\mathcal{H}_k^{pre} \triangleq \left\{ u(\boldsymbol{x}) = \sum_{i=1}^{N} c_i k\left(\boldsymbol{x}_i, \boldsymbol{x}\right) : N \in \mathbb{N}, c_1, \ldots, c_N \in \mathbb{R}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{X} \right\}. \tag{4.24}$$

Then $\mathcal{H}_k^{pre} \subset \mathcal{H}_k$ and $\mathcal{H}_k^{pre}$ is dense in $\mathcal{H}_k$ with respect to the metric induced by $\langle \cdot, \cdot \rangle_{\mathcal{H}_k}$.

We remark that the density of $\mathcal{H}_k^{pre}$ in $\mathcal{H}_k$ is equivalent to stating that the topological completion of $\mathcal{H}_k^{pre}$ is equal to $\mathcal{H}_k$ [227, Theorem 4.2.1], with appropriate choice of inner product on $\mathcal{H}_k^{pre}$ [223, page 11].

The implication of Theorem 4.5.2 for GP regression is clear by noticing that, in the case of a zero mean function, the posterior mean from Eq. (4.8) has exactly the form of a weighted sum of evaluations from the chosen kernel as seen in from Eq. (4.24). Therefore the RKHS of a given kernel k can be seen intuitively as the space of all posterior means of the GP. In the general regression case, it would clearly be desirable if this posterior mean could model any function arbitrarily well. Kernels for which this property is true are called *universal kernels* [225, Definition 4.52].

Definition 4.5.3 (Universal Kernel). *A continuous kernel k defined on $\mathcal{X} \times \mathcal{X}$ with $\mathcal{X}$ a compact metric space is called universal if its associated RKHS $\mathcal{H}_k$ is dense in $C(\mathcal{X})$ with respect to the metric induced by the supremum norm. That is, for every $u \in C(\mathcal{X})$ and for all $\varepsilon > 0$, there exists $\hat{u} \in \mathcal{H}_k$ such that $\|u - \hat{u}\|_\infty \leq \varepsilon$.*

It can be shown that the squared exponential kernel k_{SE} from Eq. (4.3) is universal on any compact subset of $\mathbb{R}^D$ [228]. By contrast, any boundary constrained kernel $\tilde{k}$ as considered in Section 4.4 for exactly satisfying Dirichlet boundary constraints will not be universal, because all functions in its associated RKHS $\mathcal{H}_{\tilde{k}}$ will be constrained to equal

zero at the domain boundary. This behaviour of the $\tilde{k}$ is desirable when the value of the function on the boundary is prescribed. However, the function is unknown on the interior of its domain. Therefore it would also be desirable if $\tilde{k}$ maintains universal approximation within the interior, for all functions which satisfy the boundary conditions.

For the purposes of exposition, we study this problem in the one-dimensional case with homogeneous Dirichlet boundary conditions. That is, we consider the function space

$$\mathcal{H}_{bc} \triangleq \{u \in C([0,1]) : u(0) = u(1) = 0\}. \tag{4.25}$$

In this case, Algorithm 3 generates a zero mean function, and boundary constrained kernel given by

$$\tilde{k}(x,x') = \phi(x)\phi(x')k(x,x') \quad \text{with} \quad \phi(x) = x(1-x), \tag{4.26}$$

for any input kernel k. We then have the following result.

Theorem 4.5.3 (Universal Boundary Constrained Kernel). *Let k be any universal kernel on $[0,1] \times [0,1]$, and $\tilde{k}$ be given as in Eq. (4.26). Then, the RKHS $\mathcal{H}_{\tilde{k}}$ is dense in $\mathcal{H}_{bc}$ from Eq. (4.25) with respect to the metric induced by the supremum norm. That is, for every $u \in \mathcal{H}_{bc}$ and for all $\varepsilon > 0$, there exists $\hat{u} \in \mathcal{H}_{\tilde{k}}$ such that $\|u - \hat{u}\|_\infty \leq \varepsilon$.*

This result is proved in Section C.2, the outline of which we sketch here. Firstly, consider the following definition.

Definition 4.5.4 (Latent Function z_u). *For any $u \in \mathcal{H}_{bc}$, we define its corresponding latent function $z_u : (0,1) \to \mathbb{R}$ as*

$$z_u(x) \triangleq \frac{u(x)}{\phi(x)}, \tag{4.27}$$

where ϕ is given as in Eq. (4.26).

Intuitively, for any $u \in \mathcal{H}_{bc}$, if there exists $\hat{z} \in \mathcal{H}_k$ so that its corresponding latent function z_u is approximated "well", then we can find $\hat{u} \in \mathcal{H}_{\tilde{k}}$ so that u is approximated "well" by simply multiplying $\hat{z}$ by ϕ. Note however that while z_u is clearly continuous on its domain $(0,1)$, Theorem 4.5.3 does not hold because $(0,1)$ is an open set. Furthermore, we cannot necessarily continuously extend z_u to the closure of its domain (i.e. to the

set $[0,1]$) because z_u may diverge for $x \in \{0,1\}$. For example, if $u(x) = (1-x)x^{\frac{1}{2}}$ then $u \in \mathcal{H}_{bc}$ but its latent function $z_u(x) = x^{-\frac{1}{2}}$ diverges at $x = 0$. Since z_u is continuous on $(0,1)$ however, it will also be continuous on any closed subset of $(0,1)$. In particular, we have then that k will be a universal approximator of z_u on subdomains of the form $[\delta, 1-\delta]$ for arbitrary $\delta \in (0, 1/2)$. This type of approximation is called convergence in (Lebesgue) measure. Moreover, the region $(0,\delta) \cup (1-\delta, 1)$ where we cannot guarantee uniform approximation is exactly the region where u is converging to zero by assumption. This allows uniform approximation to be obtained for u, even if only convergence in measure is possible for z_u.

4.6 Connection to Neural Networks

As discussed in Section 4.1.2, the problem of hard enforcement of boundary conditions has been explored in both the Gaussian process and neural network literatures. To our knowledge, however, no one has yet introduced a formal link between these two bodies of work. In this section, we introduce such a link, by proving that a hard-constrained *Gaussian process* (HCGP) is the infinite width limit of a hard-constrained *neural network* (HCNN). We define a HCNN as follows.

Definition 4.6.1 (Hard-Constrained Neural Network (HCNN) for Dirichlet Boundary Conditions). *Consider bounded domain $\Omega \subset \mathbb{R}^D$ with boundary $\partial\Omega$ and known boundary function $b : \partial\Omega \to \mathbb{R}$. Let $\tilde{m} : \Omega \to \mathbb{R}$ and $\phi : \Omega \to \mathbb{R}$ be as given in Definition 4.4.1, and z_{nn} be a neural network. We then define a hard-constrained neural network (HCNN) to be a model $u_{nn} : \Omega \to \mathbb{R}$ of the form*

$$u_{nn}(\boldsymbol{x}) = \tilde{m}(\boldsymbol{x}) + \phi(\boldsymbol{x}) z_{nn}(\boldsymbol{x}). \tag{4.28}$$

It is well known that, in the infinite width limit and under certain regularity conditions (see below), a single layer neural network converges to a Gaussian process. Specifically, we have the following result from [229, Chapter 2], reviewed succinctly in [222, Section 18.7.1].

Theorem 4.6.1 (Convergence of neural network to Gaussian process in infinite width limit). *Consider a single layer neural network of width H, i.e. a model of the form*

$$z_{nn}(\boldsymbol{x}) = b^{(1)} + \sum_{j=1}^{H} w_j^{(1)} h_j(\boldsymbol{x}), \quad h_j(\boldsymbol{x}) = \varphi\left(b_j^{(0)} + \boldsymbol{x}^\top \boldsymbol{w}_j^{(0)}\right) \tag{4.29}$$

where H is the width of the hidden layer and φ is a non-linear activation function taken to be bounded (such as the tanh function). Assuming Gaussian priors on the parameters of the form

$$b^{(1)} \sim \mathcal{N}\left(0, \sigma_{b^{(1)}}\right), w_j^{(1)} \sim \mathcal{N}\left(0, \sqrt{\frac{\omega}{H}}\right), b_j^{(0)} \sim \mathcal{N}\left(0, \sigma_{b_j^{(0)}}\right), \boldsymbol{w}_j^{(0)} \sim \mathcal{N}(0, \boldsymbol{\Sigma}_{\boldsymbol{w}^{(0)}}), \tag{4.30}$$

then, in the limit as $H \to \infty$, we have

$$z_{nn}(\boldsymbol{x}) \to \mathcal{GP}(0, k_{nn}(\boldsymbol{x}, \boldsymbol{x}')), \tag{4.31}$$

where the form of the limiting kernel k_{nn} depends on the choice of activation function φ.

We now introduce a corresponding theorem for the limiting form of a *hard-constrained* single layer neural network.

Theorem 4.6.2 (Convergence of HCNN to HCGP in infinite width limit). *Let u_{nn} be a HCNN of the form of Eq. (4.28) and z_{nn} a single layer network of width H, with prior distributions of the form of Eq. (4.30) assumed on its the parameters. Then we have in the limit as $H \to \infty$,*

$$u_{nn}(\boldsymbol{x}) \to \mathcal{GP}(\tilde{m}(\boldsymbol{x}), \tilde{k}_{nn}(\boldsymbol{x}, \boldsymbol{x}')), \tag{4.32}$$

where $\tilde{k}_{nn}$ is the boundary constrained kernel found by using k_{nn} in the construction of a HCGP (Definition 4.4.1).

This result is proved in Section C.3.

In the present work we concentrate on HCGPs because this approach is particularly suited to efficient inference of any unknown linear differential parameters $\boldsymbol{\theta}$ via the analytically tractable log marginal likelihood (see Eq. (4.18)). The objective of this section was to emphasise however that the methods used to construct HCGPs are not restricted

to the setting of kernel design, and can easily be applied to neural networks to yield similar results. Specifically, analogous arguments to those presented in Sections 4.4 and 4.5 respectively can be applied to show both that the HCNN architecture exactly satisfies the specified Dirichlet boundary conditions and is a universal function approximator within the boundary constrained function space[14]. Finally, Theorem 4.6.2 shows that the two seemingly disparate approaches of HCGPs and HCNNs are in fact equivalent in the infinite width limit with appropriate choice of prior.

4.7 General Boundary Conditions

In this section, we show how the hard-constrained framework for Dirichlet boundary conditions described in Section 4.4 can be extended to additional types of boundary conditions. For the purposes of exposition, we initially let $\Omega = [0, 1]$ and consider boundary constraints at the left end point $x = 0$. In this case, recall that a *Dirichlet* boundary condition takes the form

$$\textbf{Dirichlet Boundary :} \quad u(0) = b_0. \tag{4.33}$$

As detailed in Algorithm 3, a HCGP can be specified as follows to exactly satisfy these conditions,

$$\tilde{m}(x) = b_0, \tag{4.34}$$

$$\tilde{k}(x, x') = \phi(x)\phi(x')k(x, x') \text{ with } \phi(x) = x. \tag{4.35}$$

The top row of Figure 4.1 displays five functions and corresponding derivatives sampled from a HCGP with the above mean and covariance functions in the case where $b_0 = 0$.

A *Cauchy* boundary condition takes a similar form

$$\textbf{Cauchy Boundary :} \quad u(0) = b_0 \text{ and } u_x(0) = c_0, \tag{4.36}$$

[14]This relies on the well known universal approximation theorem for neural networks [106].

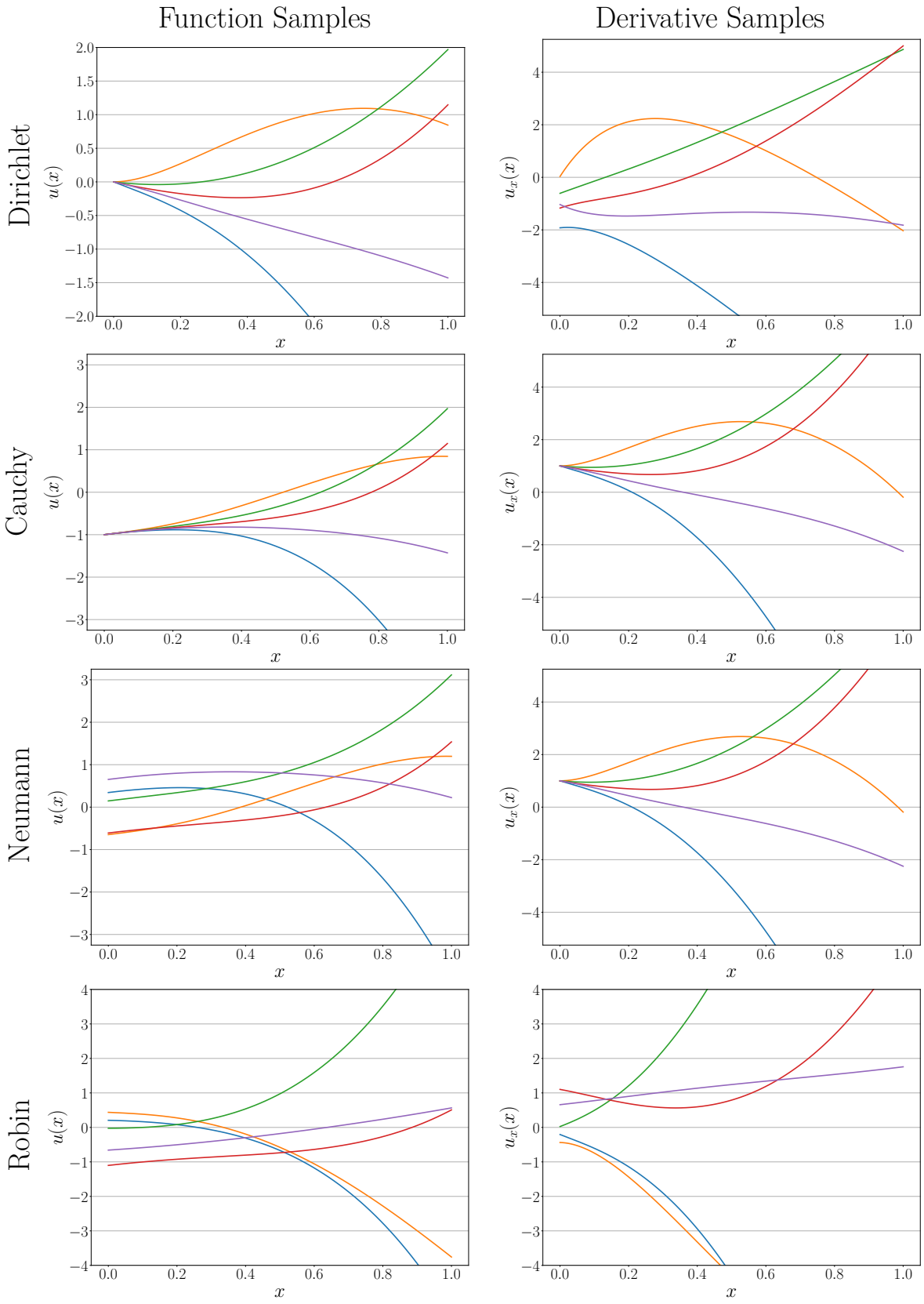


Figure 4.1: Function (left column) and derivative (right column) samples from four HCGPs corresponding to different boundary conditions at $x = 0$. The y-axis of each plot is clipped to make function and derivative samples easier to compare. For consistency, the same random seeds are used in drawing the samples from each row.

where in this case the value of the derivative u_x is exactly specified in addition to the function value itself u. It is typically used as an initial condition in time dependent PDEs, where it has the physical interpretation of describing the initial position and velocity of a system. A HCGP can be specified by slightly altering the mean and covariance functions of the HCGP for Dirichlet conditions as follows

$$\tilde{m}(x) = b_0 + c_0 x, \tag{4.37}$$

$$\tilde{k}(x, x') = \phi(x)\phi(x')k(x, x') \text{ with } \phi(x) = x^2. \tag{4.38}$$

By setting $\phi(x) = x^2$ so that $\phi(0) = \partial_x\phi(0) = 0$, we ensure $\tilde{k}(x, x')$, $\partial_x\tilde{k}(x, x')$, $\partial_{x'}\tilde{k}(x, x')$ and $\partial_x\partial_{x'}\tilde{k}(x, x')$ all equal zero if $x = 0$ and/or $x' = 0$. This in turn means that all functions sampled from $\mathcal{GP}(\tilde{m}, \tilde{k})$ will have value and derivative that revert to those of the mean function at $x = 0$, which we have specified so that it exactly satisfies the given boundary conditions, that is $\tilde{m}(0) = b_0$ and $\partial_x\tilde{m}(0) = c_0$.

The second row of Figure 4.1 displays five samples of a HCGP in the case where $b_0 = -1$ and $c_0 = 1$. At the left boundary for each sample, we see the function constraint is satisfied, as well as the derivative constraint.

A *Neumann* boundary condition at $x = 0$ has the form

$$\textbf{Neumann Boundary :} \quad u_x(0) = c_0, \tag{4.39}$$

where, unlike a Cauchy boundary, only the derivative value is specified and not the function itself. A HCGP in this case can be specified using the following mean and covariance functions

$$\tilde{m}(x) = \beta_0 + c_0 x, \tag{4.40}$$

$$\tilde{k}(x, x') = \phi(x)\phi(x')k(x, x') \text{ with } \phi(x) = x^2. \tag{4.41}$$

The specification of $\tilde{m}$ and $\tilde{k}$ are similar here as in the Cauchy boundary, with the exception that β_0 is a trainable parameter because the function value in this instance is not prescribed on the boundary and therefore must be learned. We remark that, when fitting a GP in practice, it may be more convenient to integrate this parameter out of the

mean function - for details, see Section 4.8.1. The third row of Figure 4.1 shows random samples from a HCGP of this form where we let $c_0 = 1$ and $\beta_0 \sim \mathcal{N}(0, 1)$, where we can see in the right column the Neumann condition is satisfied. Because we used consistent random seeds, the function samples here are almost exactly the same as for the Cauchy boundary, except they are shifted by the random value of β_0 in each case.

We finally consider *Robin* boundary conditions, which in the one dimensional case take the form

$$\textbf{Robin Boundary :} \quad a_0 u(0) + b_0 u_x(0) = c_0, \tag{4.42}$$

with $a_0, b_0 \neq 0$. Note the contrast here to a Cauchy condition - instead of specifying the value of both the function and the derivative, we instead specify a weighted combination of the two. A HCGP in this case can be specified by slightly modifying the mean and covariance functions used in the above examples as follows:

$$\tilde{m}(x) = \left(\frac{a_0 x - b_0}{b_0}\right)\beta_0 + \frac{c_0}{b_0}x, \tag{4.43}$$

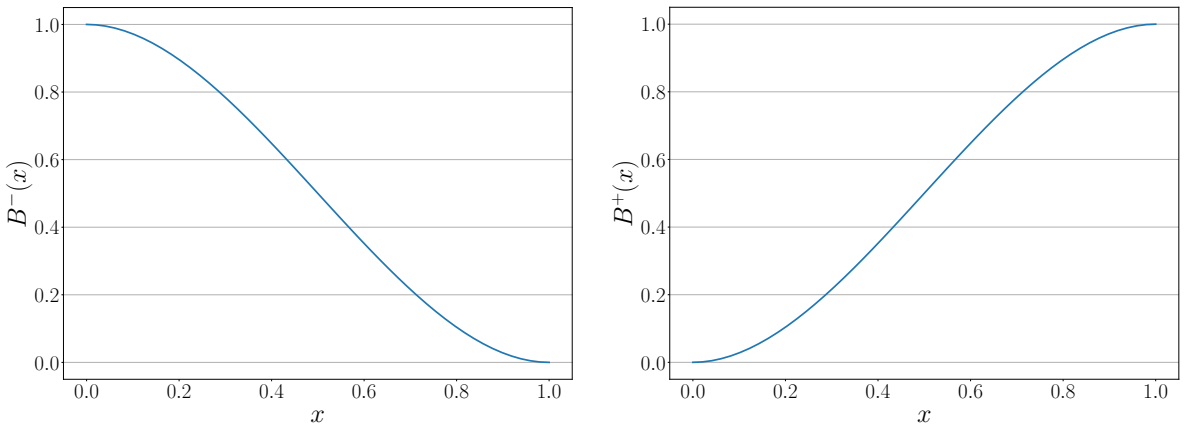
$$\tilde{k}(x, x') = \phi(x)\phi(x')k(x, x') \text{ with } \phi(x) = x^2. \tag{4.44}$$

The hard constrained kernel here is the same as used for a Cauchy boundary, which again means that all functions sampled from a GP with this kernel will have value and derivative which revert to the mean function at $x = 0$. Furthermore, we have by construction that $\tilde{m}(0) = -\beta_0$ while by differentiation $\partial_x \tilde{m}(0) = (a_0\beta_0 + c_0)/b_0$. Rearranging yields $\tilde{m}(0)a_0 + \partial_x \tilde{m}(0)b_0 = c_0$ as required. The final row of Figure 4.1 shows random samples from a HCGP of this form, where we let $a_0 = b_0 = c_0 = 1$ and once again $\beta_0 \sim \mathcal{N}(0, 1)$. For each sample, the values of both the function and its derivative at the left boundary are random. However, note we also have $u(0) = -u_x(0)$ for each sample, meaning the Robin boundary is satisfied.

4.7.1 Extension to higher dimensional domains

The above methods can be extended to higher dimensions. If the domain has a grid like structure, the mean and covariance functions of a HCGP can be derived in a similar manner as Algorithm 3. We show extensive examples of this in Section 4.8 below. In performing these derivations, we make use of two cubic polynomial functions $B^- : [a, b] \to [0, 1]$ and $B^+ : [a, b] \to [0, 1]$, whose coefficients are chosen such that the conditions given in Eq. (4.45) are satisfied. These functions are plotted for $[a, b] = [0, 1]$ in Figure 4.2.

$$\begin{aligned} B^-(a) = 1, \quad B^-(b) = 0, \quad \partial_x B^-(a) = 0, \quad \partial_x B^-(b) = 0, \\ B^+(a) = 0, \quad B^+(b) = 1, \quad \partial_x B^+(a) = 0, \quad \partial_x B^+(b) = 0. \end{aligned} \tag{4.45}$$



(a) $B^- : [0, 1] \to [0, 1]$ (b) $B^+ : [0, 1] \to [0, 1]$

Figure 4.2: Illustration of cubic polynomials $B^- : [a, b] \to [0, 1]$ and $B^+ : [a, b] \to [0, 1]$ for $[a, b] = [0, 1]$. See Eq. (4.45) for more details. These functions are useful for deriving boundary covariance functions $\tilde{k}$ for domains $\Omega \subset \mathbb{R}^D$ with $D > 1$.

For non grid-like domains, there exist several methods recently introduced in the PINN literature which could be adjusted for use in the case of PIGPs [219]–[221]. We illustrate this using a numerical example in Section 4.8.1.

4.8 Numerical Experiments

We now present numerical experiments which evaluate the effectiveness of the methods discussed above for hard enforcement of boundary conditions. The experiments include both forward and inverse problems for a range of different linear PDEs, including the three canonical linear PDEs, namely the Poisson, Heat and Wave Equations. The boundary conditions are chosen so that at least one example of Dirichlet, Cauchy, Neumann, Robin and periodic conditions respectively are considered.

To distinguish between different modelling approaches, we will make use of the following acronyms :

- **UCGP:** Unconstrained Gaussian Process - here no boundary information is accounted for in the modelling problem. This is the approach used in the seminal paper [102].

- **PCGP:** Penalty Constrained Gaussian Process - here a set of pseudo-observations on the boundaries are used to account for the boundary conditions. This is similar to a penalty or soft enforcement approach, as used in [230].

- **HCGP:** Hard Constrained Gaussian Process - here the mean and covariance functions of the GP are designed such that all boundary conditions are exactly satisfied, everywhere on the boundary.

- **HCNN:** Hard Constrained Neural Network - a model based on neural networks such that, as for the HCGP, all boundary conditions are exactly satisfied everywhere.

The above modelling approaches have different strengths and weaknesses, and the choice of which to deploy in practice will depend on the specifics of the problem of interest. PIGPs are particularly suited to inference in systems governed by linear PDEs. As detailed in Section 4.3, any unknown PDE parameters $\boldsymbol{\theta}$ become hyperparameters of the PIGP kernel, allowing for efficient inference of $\boldsymbol{\theta}$ from observational data. The number of parameters to be inferred when using a PIGP (that is the kernel, noise and PDE parameters) will be much lower than is the case with a PINN, where typically thousands of weights and biases will need to be trained. PIGPS also give full uncertainty quanti-

fication via the analytically tractable posterior predictive distribution (Eq. (4.8)). The advantage of PINNs is that they significantly more computationally efficient than PIGPs, and can naturally handle non-linear PDEs. As for the handling of boundary conditions, unconstrained approaches (i.e. UCGPs or UCNNs) are appropriate for applications where boundary conditions are unknown. When boundary conditions are known, they can either be softly enforced using a penalty approach, or explicitly enforced with a HCGP/HCNN. One problem with PCGPs is that computational resources are "wasted" accounting for the boundary conditions, whereas with a HCGP, the conditions are enforced without using observational data. However, implementing a HCGP/HCNN requires bespoke mean and kernel functions to be derived.

In all experiments, the base kernel k from which we construct a hard constrained kernel $\tilde{k}$ is the squared-exponential with seperate lengthscales for each dimension, i.e. a kernel of the form

$$k(\boldsymbol{x}, \boldsymbol{x}') = \tau^2 \exp\left(-\frac{1}{2}\sum_{i=1}^{D}\frac{(x_i - x_i')^2}{\lambda_i^2}\right). \tag{4.46}$$

Here, the kernel hyperparameters are $\boldsymbol{\xi} = (\tau, \lambda_1, \ldots, \lambda_D)$. This is referred to as an automatic relevance determination (ARD) kernel, as it allows for the most important input dimensions to be identified automatically during the training procedure [130, Section 5.1].

As discussed in Section 4.3, we train GPs by optimisation of the marginal likelihood (see Eq. (4.18)). Note that this objective function is not convex with respect to the hyperparameters of the kernel, and therefore we consider multiple restarts under different random initialisations when training.

For comparison, we also perform an experiment using a HCNN in Section 4.8.1. HCNNs can be trained to minimise prediction error against observations $\boldsymbol{y}_u \in \mathbb{R}^{N_u \times 1}$ in u-space (see Eq. (4.1)) and observations $\boldsymbol{y}_f \in \mathbb{R}^{N_f \times 1}$ in f-space (see Eq. (4.10)). We denote a HCNN as u_{nn}, with weights and biases denoted $\boldsymbol{\omega}$ and any trainable variables used to enforce the boundary conditions denoted $\boldsymbol{\beta}$. Then, assuming the parameters $\boldsymbol{\theta}$ of

the differential operator $\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}$ are known, loss terms for the HCNN can be defined as

$$L_u(\boldsymbol{\omega}, \boldsymbol{\beta}) = \frac{1}{N_u} \sum_{i=1}^{N_u} \left(y_u^{(i)} - u_{nn}(\boldsymbol{x}_u^{(i)}; \boldsymbol{\omega}, \boldsymbol{\beta}) \right)^2, \tag{4.47}$$

$$L_f(\boldsymbol{\omega}, \boldsymbol{\beta}) = \frac{1}{N_f} \sum_{i=1}^{N_f} \left(y_f^{(i)} - f_{nn}(\boldsymbol{x}_f^{(i)}; \boldsymbol{\omega}, \boldsymbol{\beta}) \right)^2, \tag{4.48}$$

where $f_{nn}(\boldsymbol{x}; \boldsymbol{\omega}, \boldsymbol{\beta}) = \mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u_{nn}](\boldsymbol{x}; \boldsymbol{\omega}, \boldsymbol{\beta})$. The parameters can then be trained to minimise the sum of these loss terms:

$$\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\beta}}\} = \underset{\boldsymbol{\omega}}{\operatorname{argmin}}\, L_{pinn}(\boldsymbol{\omega}, \boldsymbol{\beta}), \tag{4.49}$$

where $L_{pinn} = L_u + L_f$. Typically in PINNs, a boundary loss term is also included, but this is not necessary with a HCNN because the boundary conditions are satisfied automatically. Again we use multiple random restarts when solving this optimisation problem because the objective function is non-convex with respect to $\boldsymbol{\omega}$.

In the experiments performed in Section 4.8.1, solely the forward problem of learning u is considered, in which case only collocation points in f-space are used for training, where f is known. In the experiments performed in Sections 4.8.2-4.8.4, the inverse problem of identifying $\boldsymbol{\theta}$ is considered, given a finite set of noise corrupted observations of u. In each case, we assume $f = 0$ is a known conservation principle, which we incorporate into the inference once again via collocation points. Finally, the experiment in Section 4.8.5 is taken from [218], where the forward problem is considered given discrete observations of u and f, which are subject to observation noise in each case.

Accuracy of the ML models assessed using the following error metrics

$$err_{\boldsymbol{u}} = \text{Mean}(|u - \hat{u}|), \quad err_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_2, \tag{4.50}$$

where $\hat{\boldsymbol{\theta}}$ is the point estimate of the PDE parameters found during training and, for GPs, $\hat{u}$ is the posterior mean. The Mean$(\cdot)$ operator in $err_{\boldsymbol{u}}$ is taken over a large grid of test points in the specified domain.

Experiments are implemented in Python using the JAX [48] library. Adam [111] is used for training with an exponentially decaying learning rate. For each PDE considered, the differential operator $\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}$ is implemented in JAX code, which allows for m_f, k_{ff}, k_{uf} and k_{fu} (see Section 4.3 for details) as well as f_{nn} (see Eq. (4.48) above) to be found using the automatic differentiation system provided by JAX, i.e. no hand derivations are required.

4.8.1 Poisson Equation

Our first set of numerical experiments involve Poisson's equation, the prototypical elliptic PDE that has wide utility in physics. Given two spatial dimensions x and y, the equation takes the form

$$-(u_{xx} + u_{yy}) = f. \tag{4.51}$$

For the Poisson equation, we will only consider the forward problem of learning the function u, using two BVPs stated below. Note that for Poisson-BVP-1, only one spatial dimension is considered.

$$\textbf{Poisson-BVP-1:} \begin{cases} x \in [0, 2] \\ u(0) = 0, \\ u_x(0) = 0, \\ u_x(2) = u(2). \end{cases} \qquad \textbf{Poisson-BVP-2:} \begin{cases} (x, y) \in \Omega_p, \\ u(x, y) = 0,\ (x, y) \in \partial\Omega_p. \end{cases}$$

Poisson-BVP-1 is a slightly adjusted version of the BVP presented in DeepXDE documentation [231], which we follow by setting $f(x) = 2$ which yields $u(x) = x^2$. This simple preliminary example is used for exposition of the hard-constrained framework for modelling BVPs, both for neural networks (HCNNs) and Gaussian processes (HCGPs). We also highlight a subtle difference in the handling of the trainable parameters for a HCNN, which is a frequentist approach, and a HCGP, which is a Bayesian method.

The following HCNN is specified for the problem,

$$u_{nn}(x; \boldsymbol{\omega}, \beta) = \phi_2(x)\beta + \phi_1(x) z_1(x; \boldsymbol{\omega}), \tag{4.52}$$

where z_1 is an FCNN with tanh activation function with weights and biases $\boldsymbol{\omega}$, $\boldsymbol{\omega}$, β is a trainable parameter while

$$\phi_1(x) = x^2(2-x)^2 \text{ and } \phi_2(x) = (x-1)B^+(x). \tag{4.53}$$

$B^+ : [0,2] \to [0,1]$ is a cubic polynomial with coefficients chosen to satisfy the conditions given in Eq. (4.45). This HCNN exactly satisfies the boundary conditions of Poisson-BVP-1, which can be seen by noting that $u_{nn}(0;\boldsymbol{\omega},\beta) = 0$, $u_{nn}(2;\boldsymbol{\omega},\beta) = \beta$, then using the chain rule to show that $\partial_x u_{nn}(0;\boldsymbol{\omega},\beta) = 0$ and $\partial_x u_{nn}(2;\boldsymbol{\omega},\beta) = \beta$.

A HCGP could be specified for this BVP using a mean function of the form $\tilde{m}(x) = \beta\phi_2(x)$. However, care must be taken with the parameter β. Directly optimising its value as was done with the HCNN would not allow uncertainty at the right boundary to be accounted for. Instead, then of optimising this parameter, we use a Bayesian approach by assigning it an $\mathcal{N}(0,\sigma_\beta^2)$ prior and then integrating out over the implied distribution in function space to yield the following kernel:

$$\begin{aligned}
\operatorname{Cov}(\tilde{m}(x),\tilde{m}(x')) &= \mathbb{E}[\tilde{m}(x)\tilde{m}(x')] - \overbrace{\mathbb{E}[\tilde{m}(x)]\,\mathbb{E}[\tilde{m}(x')]}^{0} \\
&= \mathbb{E}[(\beta\phi_2(x))(\beta\phi_2(x'))] \\
&= \sigma_\beta^2\phi_2(x)\phi_2(x').
\end{aligned} \tag{4.54}$$

The final HCGP then has a zero mean function and a kernel of the form

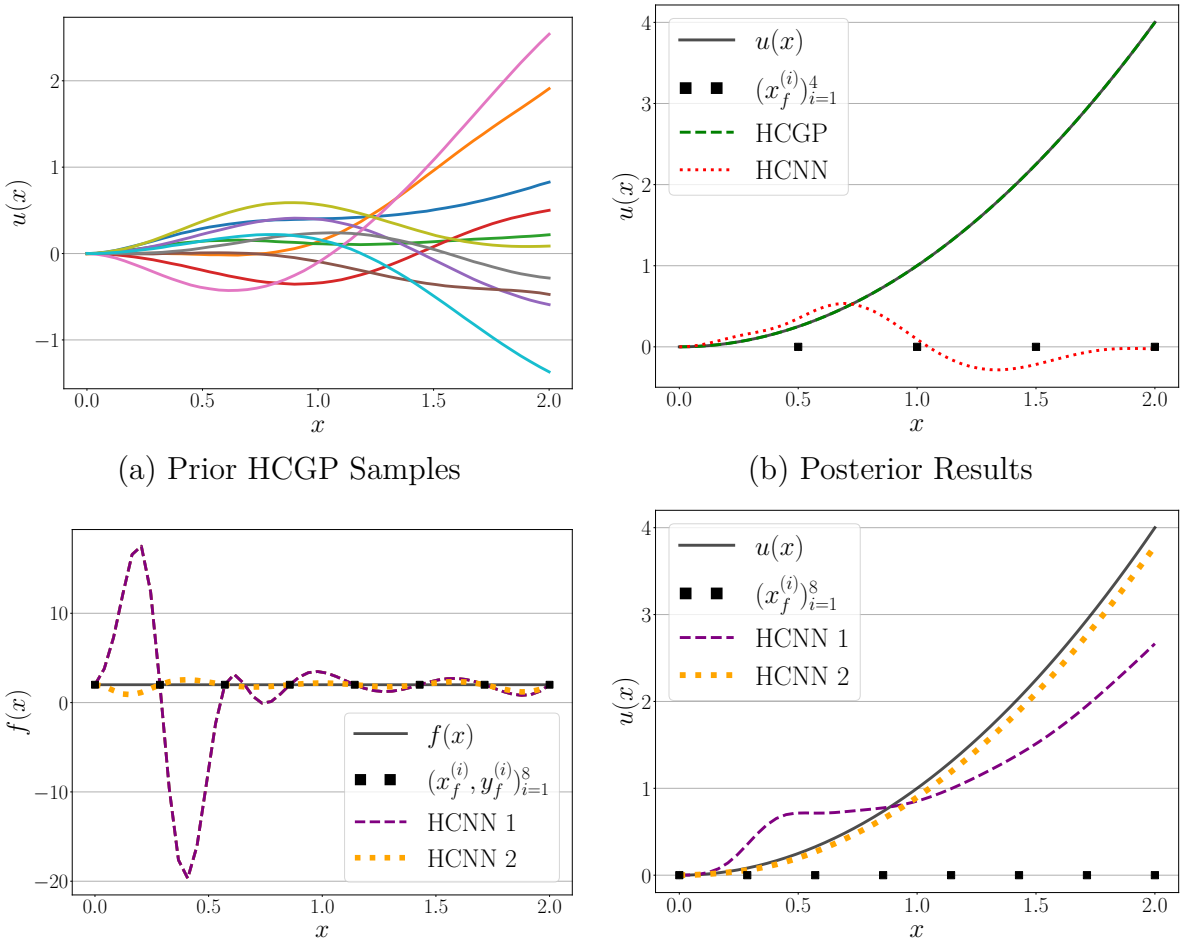
$$\tilde{k}(x,x') = \phi_1(x)\phi_1(x')k_1(x,x') + \phi_2(x)\phi_2(x')\sigma_\beta^2. \tag{4.55}$$

Ten prior samples from this HCGP are given in Figure 4.3 (a). On the left boundary, note how each sample is equal to zero with zero derivative, i.e. the Cauchy condition from the mixed BVP in Eq. (4.8.1) is satisfied. At the right boundary, the value of each sample scales exactly with the derivative, i.e. the Robin condition of the BVP is satisfied.

We consider the forward problem of learning u from a sparse data set of $N_f = 4$ collocation points at input locations $(x_f^{(1)}, x_f^{(2)}, x_f^{(3)}, x_f^{(4)}) = (0.5, 1.0, 1.5, 2.0)$. The HCGP was trained to maximise the marginal likelihood of this data (see Eq. (4.18)). The HCNN was trained to minimise the loss term L_{pinn} (see Eq. (4.49)), where $L_{pinn} = L_f$ in this case because no observations in u-space are considered here. For the latent FCNN z_1, we used one hidden layer of width 16, with tanh activation function.

Figure 4.3 (b) presents the predictions of both models after training. The HCNN completely fails to capture the true function, obtaining $err_{\boldsymbol{u}}$ value of 1.33. This is despite the fact that the HCNN learns to nearly perfectly interpolate the four collocation points, i.e. $L_f \approx 0$. By contrast, the HCGP recovers the true solution almost exactly, with $err_{\boldsymbol{u}} = 4.9 \times 10^{-5}$. This example illustrates the effectiveness of the HCGP in the low-data regime in comparison to the HCNN, which is a consequence of the different objective functions used to train the two models.

To elucidate further on this point, we re-performed the above experiment using $N_f = 8$ collocation points, the results of which are displayed in the bottom row of Figure 4.3. Two HCNNs are shown, which correspond to parameter estimates $\{\boldsymbol{\omega}^{(1)}, \beta^{(1)}\}$ and $\{\boldsymbol{\omega}^{(2)}, \beta^{(2)}\}$ which were obtained by solving the (nonconvex) optimisation problem given in Eq. (4.49) under two different random initialisations. From panel (d) of the figure, we see HCNN 2 is clearly a better approximation to the true function u. The reason for this is clear from panel (c), which indicates that for low values of x, HCNN 1 exhibits high oscillation around the true $f(x) = 2$. However, both models yield the same objective function value with $L_f(\boldsymbol{\omega}^{(1)}, \beta^{(1)}) = L_f(\boldsymbol{\omega}^{(2)}, \beta^{(2)}) = 1.7 \times 10^{-5}$. This illustrates the problem with the PINN training scheme from Eq. (4.49), as it is unable to distinguish between models which give the same data fit. A regularisation term could be added here, which would favour HCNN 2 because it is the simpler function. However, this requires both the form and the strength of the regularisation to be specified, which would likely require manual tuning. Note the contrast to the GP objective function, which *automatically* incorporates both a data fit and regularisation term (see Eq. (4.19)). This balance of model fit and complexity (called the Bayesian Occam's razor) is what allows the GP to learn the underlying function u even with only $N_f = 4$ collocation points.



(a) Prior HCGP Samples

(b) Posterior Results

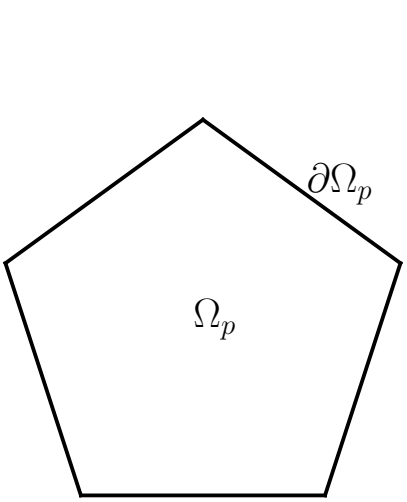
(c) $f_{nn}(x)$ under two random restarts

(d) $u_{nn}(x)$ under two random restarts

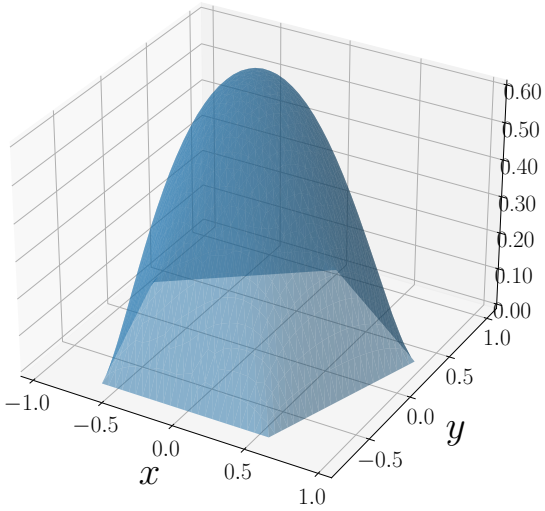
Figure 4.3: Illustration of results for Poisson-BVP-1, which compares the performance of a HCGP with a HCNN given a sparse number of collocation points. Both models are constrained to exactly satisfy the Cauchy condition on the left boundary and Robin condition on the right boundary - see the HCGP samples in panel (a) for instance.

With more training data, the performance of the HCNN improves considerably. In particular, we find that with 50 collocation points, the $err_{\boldsymbol{u}}$ value obtained with a HCNN is approximately equal to the $err_{\boldsymbol{u}}$ value for the HCGP in Figure 4.3 (b) with 4 collocation points.

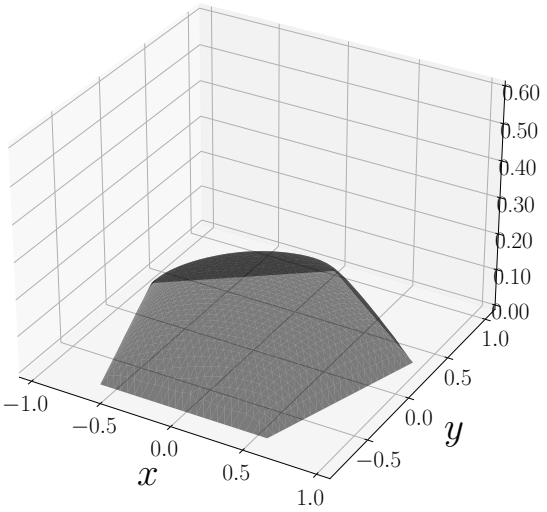
We move on now to the second BVP, which is used to illustrate how domains which do not have a grid-like structure can be modelled in the HCGP framework. Specifically, consider the pentagon shaped domain Ω_p embedded within $[-1, 1]^2$, which is displayed in Figure 4.4 (a). We set $f(x, y) = 4$ and solve for $u(x, y)$ using the finite-element analysis (FEA) software FEniCS, the results of which are displayed in Figure 4.4 (b).



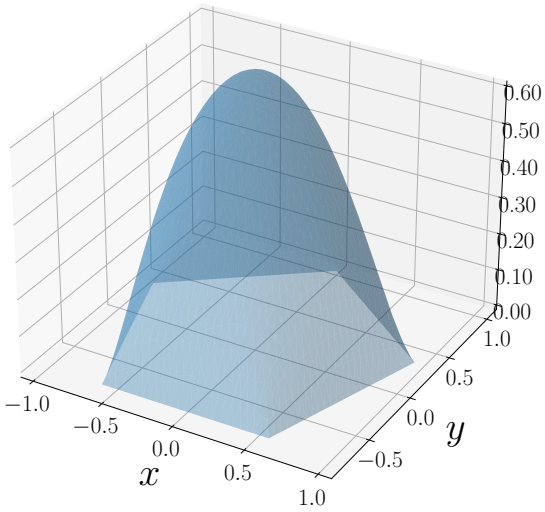
(a)



(b) $u(x, y)$



(c) $\phi_p(x, y)$



(d) $\hat{u}(x, y)$ ($err_{\boldsymbol{u}} = 1.1 \times 10^{-3}$)

Figure 4.4: Illustration of results for Poisson-BVP-2. Here, a HCGP is constructed on a non-rectangular domain using the distance function $\phi_p(x, y)$ - see panel (c) above and Eq. (4.56).

Since this BVP just involves homogeneous Dirichlet boundary conditions, clearly to specify a HCGP as described in Definition 4.4.1 we only need to find a smooth distance function $\phi_p : \Omega_p \to \mathbb{R}$ which satisfies $\phi_p(x, y) = 0$ if $(x, y) \in \partial\Omega_p$ and $\phi_p(x, y) > 0$ otherwise. Several recently developed approaches from the PINN literature could be applied here. One possibility is to define ϕ_p using radial basis interpolation [219]. For 2D domains, ϕ_p can be defined by making leveraging the theory of R-functions [220]. Since $\Omega_p \subset \mathbb{R}^2$, this is the approach we follow. Recall from Algorithm 3 that for a grid-

like domain, a distance function ϕ can be defined by first finding the distances to each individual boundary, and then using multiplication to combine these individual distances into a single scalar distance value. The approach used in [220] is conceptually similar, except generalised to domains whose boundary is comprised of an arbitrary finite number (n) of individual pieces. For Ω_p in Figure 4.4 (a), its boundary $\partial\Omega_p$ is made up of $n = 5$ pieces, each of which is a line segment. In this case, a distance function ϕ_p can be constructed as follows [232, Section 2.2.3], [220, Section 3.1]. Firstly, functions $\phi_i(x, y)$ for $i = 1, \ldots, n$ are constructed, each of which give the approximate distance from (x, y) to the i^{th} boundary segment. For specific details of this construction, see [232, Eq. (11)] and [220, Eq. (6)]. Then, the individual distance functions are combined into a global distance function using the R-equivalence formula [233]

$$\phi_p(x, y) \triangleq \frac{1}{\sqrt[m]{\frac{1}{\phi_1^m(x,y)} + \frac{1}{\phi_2^m(x,y)} + \ldots + \frac{1}{\phi_n^m(x,y)}}}, \tag{4.56}$$

which is both associative and normalised up to the m^{th} order [232]. Figure 4.4 (c) displays the distance function ϕ_p created using Eq. (4.56), with $m = 1$ as in [220].

This distance function allows us to specify a HCGP for the BVP on Ω_p, by using a zero mean function and boundary constrained kernel given by

$$\tilde{k}([x, y], [x', y']) = \phi_p(x, y)\phi_p(x', y')k([x, y], [x', y']). \tag{4.57}$$

We then trained the HCGP using 100 collocation points in f-space and plot the posterior mean $\hat{u}$ obtained after training in Figure 4.4 (d). There is strong agreement between the FEA and HCGP results, with $err_{\boldsymbol{u}} = 1.1 \times 10^{-3}$.

4.8.2 Heat Equation

We next analyse the heat equation, which describes the diffusion of heat in an isotropic medium. In Cartesian coordinates, the equation takes the form

$$u_t - \theta(u_{xx} + u_{yy}) = f = 0, \tag{4.58}$$

over two spatial dimensions x and y. Here, we consider the inverse problem of learning from observation data the parameter θ, which in this case is the thermal diffusivity of the material. Experiments are performed using two BVPs previously introduced in [230], which are stated below. In both cases, we set $\theta = 1$. Note that for Heat-BVP-1, only one spatial dimension is considered.

Heat-BVP-1:
$$\begin{cases} (x,t) \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \times [0,1] \\ u(x,0) = \sin(x), \\ u(-\frac{\pi}{2}, t) = -\exp(-t), \\ u(\frac{\pi}{2}, t) = \exp(-t). \end{cases}$$

Heat-BVP-2:
$$\begin{cases} (x,y,t) \in [0,\pi]^2 \times [0,.5] \\ u(x,y,0) = \sin(x)\sin(y), \\ u(0,y,t) = u(\pi,y,t) = 0, \\ u(x,0,t) = u(x,\pi,t) = 0. \end{cases}$$

The separation of variables technique can be used to show the following closed form solution to Heat-BVP-1

$$u(x,t) = \sin(x)\exp(-t), \tag{4.59}$$

which is plotted in Figure 4.5 (a). Because this BVP only involves Dirichlet boundary conditions, Algorithm 3 can be followed to yield the below mean and covariance functions so that the inhomogeneous Dirichlet conditions are satisfied exactly.

$$\tilde{m}(x,t) = 2\pi x \exp(-t) + (1-t)(\sin(x) - 2\pi x), \tag{4.60}$$

$$\tilde{k}([x,t],[x',t']) = \phi(x,t)\phi(x',t')k([x,t],[x',t']), \tag{4.61}$$

$$\text{with} \quad \phi(x,t) = \left(x + \frac{\pi}{2}\right)\left(\frac{\pi}{2} - x\right)t. \tag{4.62}$$

We use a HCGP to infer θ given $N_u = 25$ observations in function space corrupted with 2.5% Gaussian noise, and $N_f = 25$ collocation points observed noise free. The noise was calculated as a percentage of the variance of the underlying signal. This quantity can be considered an *inverse* signal-to-noise ration (SNR). Given this data, the HCGP learns an estimate $\hat{\theta} = 1.000$, while its posterior mean incurs a loss value of $err_{\boldsymbol{u}} = 4.9 \times 10^{-5}$, and is displayed in Figure 4.5 (c). Visually, the prediction provides a perfect match to the true function.

The following exact solution to Heat-BVP-2 can be also be found using separation of variables

$$u(x, y, t) = \sin(x) \sin(y) \exp(-2t). \tag{4.63}$$

This function is displayed in Figure 4.5 (b) with fixed $t = 0.5$. Algorithm 3 can once more be followed to find the following form of $\tilde{m}$ and $\tilde{k}$ so that the boundary conditions are satisfied:

$$\tilde{m}(x, y, t) = 2\pi x \exp(-t) + (1 - t)(\sin(x) - 2\pi x), \tag{4.64}$$

$$\tilde{k}([x, y, t], [x', y', t']) = \phi(x, y, t)\phi(x', y', t')k([x, y, t], [x', y', t']), \tag{4.65}$$

$$\text{with} \quad \phi(x, y, t) = x \left(\frac{\pi}{2} - x\right) y \left(\frac{\pi}{2} - y\right) t. \tag{4.66}$$

Again the inverse problem is considered, this time given $N_u = 50$ noise-corrupted function observations and $N_f = 50$ collocation points. More training points are considered here due to the increased dimensionality of the problem. From this data, the HCGP learns an estimate of $\hat{\theta} = 0.994$, with $err_{\boldsymbol{u}} = 5.8 \times 10^{-4}$. Figure 4.5 (d) shows the posterior mean prediction over both spatial dimensions at time $t = 0.5$, which accurately captures the true function from panel (b).

To benchmark the results of the HCGP, we re-performed the above experiments using a PCGP. Recall that with a PCGP, boundary condition information is introduced by conditioning on N_b pseudo-observations sampled from the boundary. A heteroskedastic noise model for function space observations is required here, because the pseudo observations are observed noised-free, in contrast to the noisy observations $\boldsymbol{y}_u$ from the interior of the domain. Therefore, we introduce a small noise variance σ_n^2 for the boundary points,

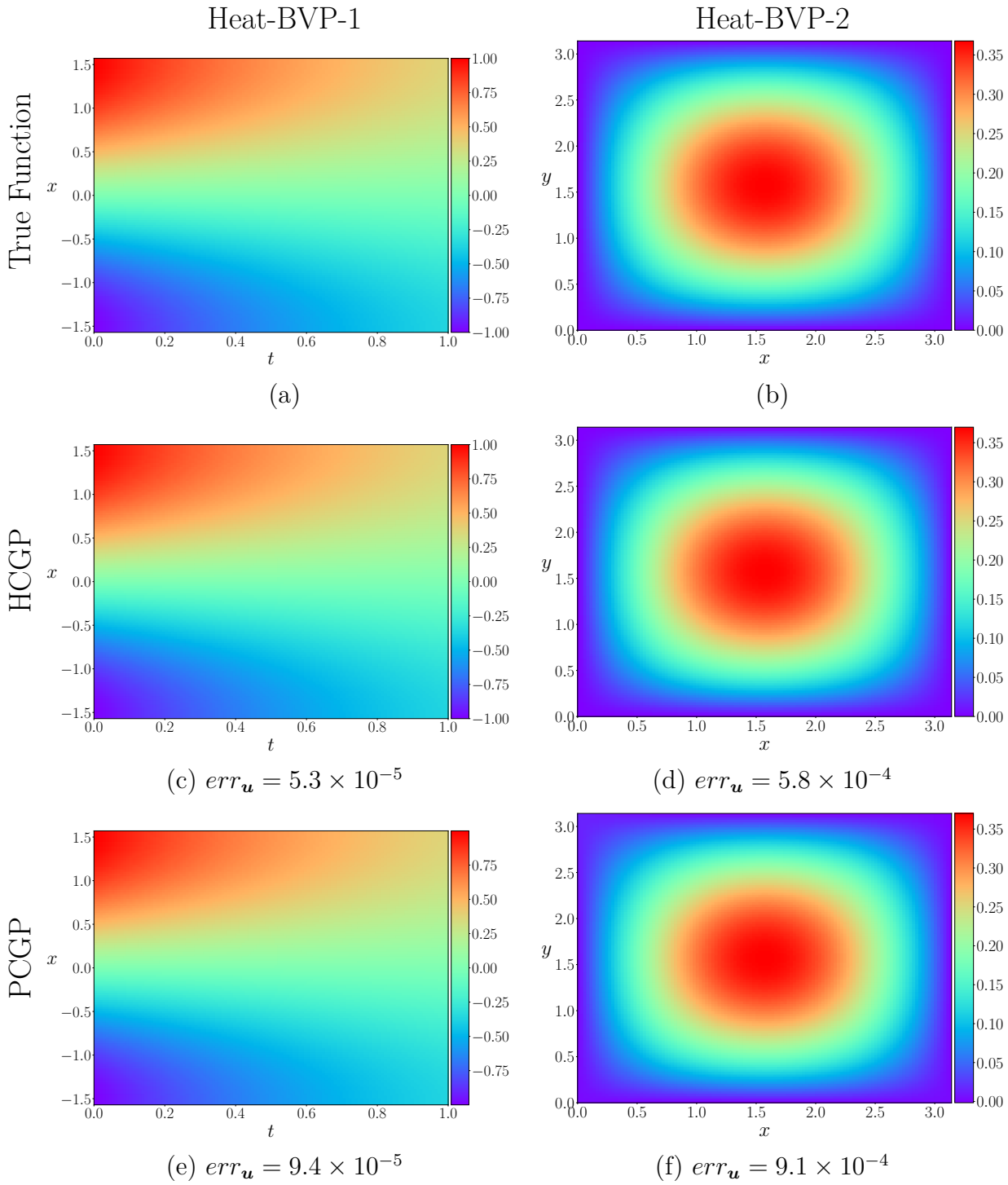
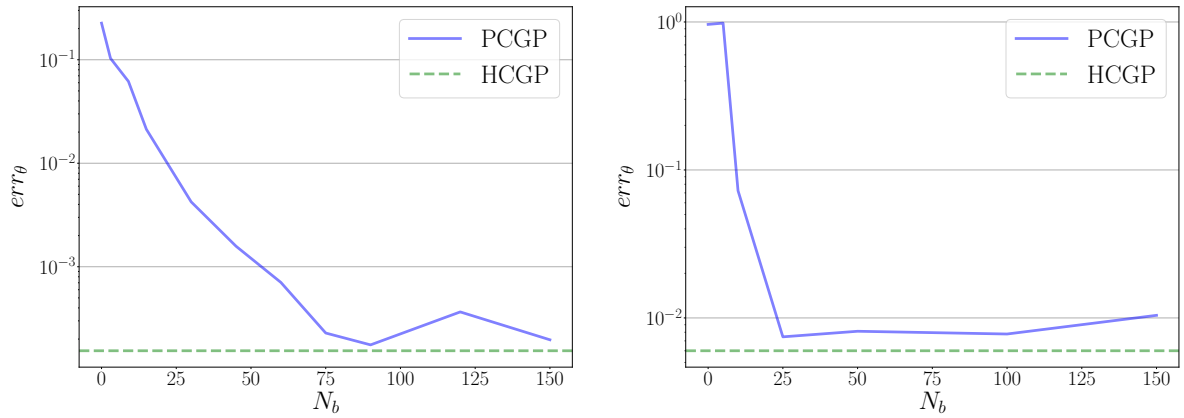


Figure 4.5: Plots of true solution (top row) and HCGP/PCGP predictions (second/third rows) for Heat-BVP-1 (left column) and Heat-BVP-2 (right column).

which ensures numerical stability in the matrix computations. We furthermore let σ_n^2 be a trainable hyperparameter, which we find improves performance. For both BVPs, we randomly sampled N_b points on the boundary and then used a PCGP to infer θ under 10 random restarts. We repeat this for a range of values of N_b, allowing the parameter inference results to be plotted against N_b in Figure 4.6. Panel (a) displays the results for



(a) Heat-BVP-1 (b) Heat-BVP-2

Figure 4.6: Effect on parameter inference accuracy of number of penalty observation points N_b for PCGP for Heat-BVP-1 and Heat-BVP-2.

Heat-BVP-1, which shows a reduction in $err_{\boldsymbol{\theta}}$ as more boundary enforcement points are included, and for $N_b > 75$, the PCGP closely approaches the performance of the HCGP, shown as a black dashed horizontal line for reference. For Heat-BVP-2 in panel (b), high $err_{\boldsymbol{\theta}}$ errors are incurred for $N_b < 25$, however results rapidly switch to yield significantly more accurate results for $N_b > 25$.

When training a PCGP for Wave-BVP-2 with a low number of boundary enforcement points N_b, we found that the optimiser would consistently converge to a result which yielded posterior mean $\hat{u}$ approximately equal to zero across the domain, in which case all variation in the data is attributed to noise with high value of σ_u^2 inferred. This is a well known problem in physics informed training in the presence of sparse data - see [234] for a detailed discussion. We call this the *trivial solution*, as the zero function trivially satisfies the Heat PDE in Eq. (4.58). As N_b was increased, a greater proportion of the random restarts began to (approximately) learn the true solution. However, even for values of N_b up to 1000, we continued to encounter the problem. Furthermore, this was not a local optima - in fact the value of the objective function (Eq. (4.18)) for restarts which learned $\hat{u} \approx 0$ was consistently higher than for restarts which learned $\hat{u} \approx u$. For the results presented in Figure 4.6 (b), we manually filtered all restarts and removed any optimisation results which yielded $\hat{u} \approx 0$. If this filtering was not performed, the performance of the PCGP would be significantly worse here. Note the contrast to the HCGP, which consistently found the correct solution under different random restarts without any hand-tuning being required.

The above experiments all applied 2.5% Gaussian observation noise to the function space observations $\boldsymbol{y}_u$. We next explore how results change as this noise level is varied. To this end, we re-performed the experiments with noise levels ranging from 0% to 10%. The results obtained using UCGPs, PCGPs and HCGPs respectively are displayed as line plots in Figure 4.7. For Heat-BVP-1, $N_b = 90$ boundary points were used when implementing the PCGP, while $N_b = 100$ points were used for Heat-BVP-2. First considering the UCGP results, we see that strong accuracy is obtained for Heat-BVP-1 when there is no observation noise. However, results rapidly deteriorate as noise levels increase. For Heat-BVP-2, even with low noise levels, the UCGP finds the trivial solution and therefore incurs high errors in prediction of u and θ. These results clearly indicate a lack of robustness of the UCGP to observation noise.

The PCGP and HCGP results more closely align, with a more gentle deterioration in accuracy observed with increasing observation noise. The HCGP results are almost always better, with the exception of $err_{\boldsymbol{\theta}}$ for Heat-BVP-1, the accuracy of the PCGP oscillates as observation noise rises. Also note the high errors incurred by the PCGP in Heat-BVP-2 for 10% observation noise. As discussed above, we manually filtered the PCGP optimisation results to remove those which found the trivial solution of $\hat{u} \approx 0$. However, with 10% noise, all random restarts found the trivial solution, leading to high errors in this case.

Another point to note here is that if boundary conditions involving derivatives are specified, implementing a PCGP becomes more complex. This is because derivative values will not lie in either u space or f space, which means that the training data covariance matrix $\boldsymbol{K}_{\boldsymbol{yy}}$ from Eq. (4.17) will need to be adjusted to a 3×3 block matrix.

4.8.3 Wave Equation

We next examine the wave equation, a hyperbolic, second-order PDE that is fundamental for describing wave phenomena in areas such as fluid dynamics, electromagnetics and acoustics. Over one spatial dimension, the wave equation takes the form

$$u_{tt} - \theta^2 u_{xx} = f = 0, \tag{4.67}$$

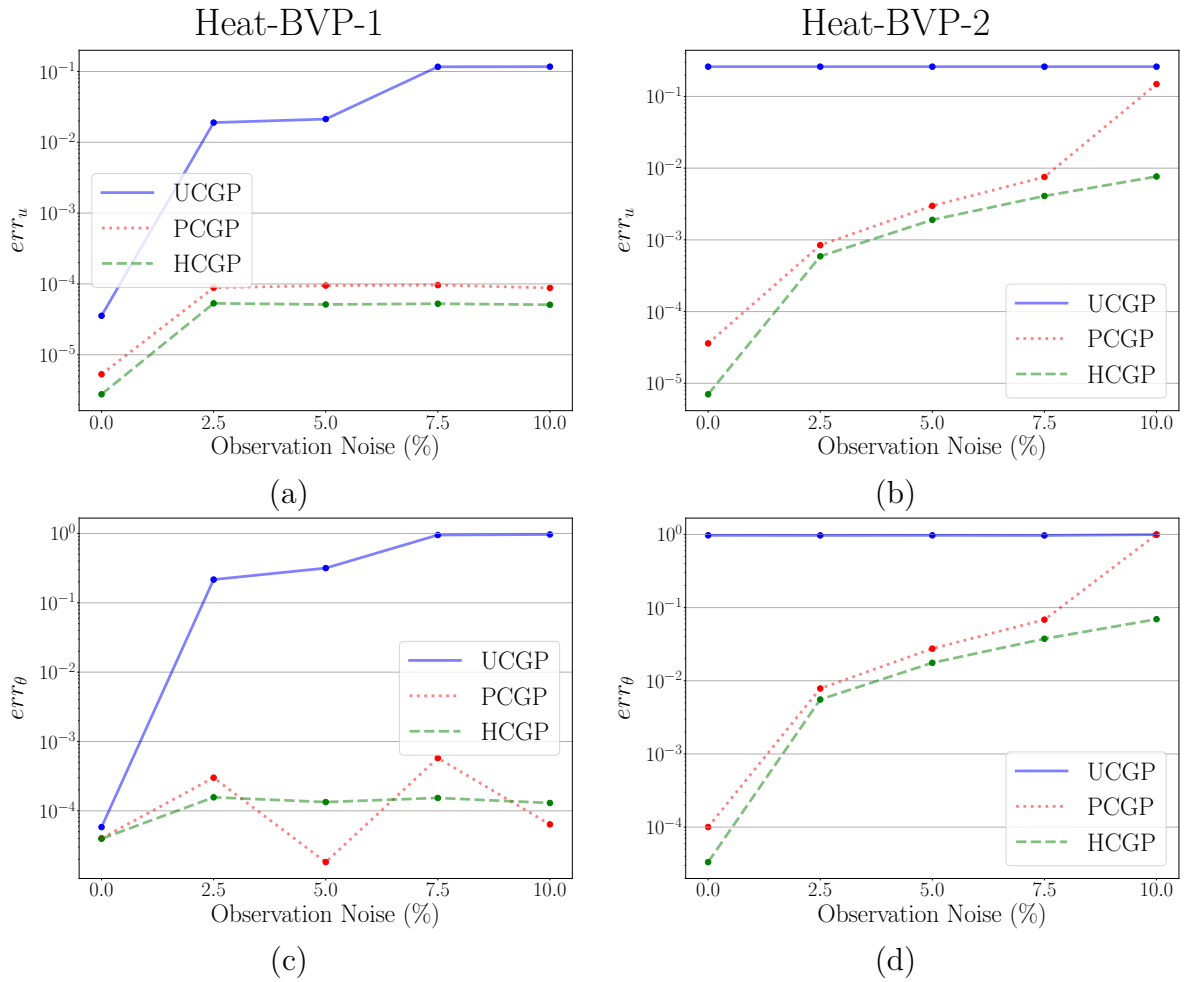


Figure 4.7: Effect of observation noise on the accuracy of solution-space (top row) and parameter-space (bottom row) results obtained using UCGP, PCGP and HCGP respectively, for Heat-BVP-1 (left column) and Heat-BVP-2 (right column).

with $\theta > 0$ a parameter controlling the wave propagation speed. We will consider the inverse problem of learning θ from observational data, for two different BVPs, which are stated below.

Wave-BVP-1:
$$\begin{cases} (x,t) \in [0,\pi] \times [0,1] \\ u(x,0) = \cos(x), \\ u_t(x,0) = \cos^2(x), \\ u_x(0,t) = u_x(\pi,t) = 0. \end{cases}$$

Wave-BVP-2:
$$\begin{cases} (x,t) \in [0,1] \times [.5,2.5] \\ u(x,0) = \sin(x), \\ u_t(x,0) = 0, \\ u(0,t) = u(\pi,t). \end{cases}$$

For Wave-BVP-1 we set $\theta = 1$, given which the separation of variables technique can be applied to find the following closed form solution for u

$$u(x,t) = \frac{1}{2}t + \cos(t)\cos(x) + \frac{1}{4}\sin(2t)\cos(2x). \tag{4.68}$$

This function is plotted as a heatmap in Figure 4.8 (a). We seek to learn θ given $N_u = 25$ noise corrupted observations of u, and $N_f = 25$ forcing point observations of $f = 0$, where in each case, the input locations are randomly sampled from within the spatio-temporal domain.

We first address the inverse problem using a UCGP, i.e. we follow the approach of [102] and do not make use of boundary condition information during the parameter inference. In this case, we achieve a best estimate of $\hat{\theta} = 0.53$. The posterior mean is plotted in Figure 4.8 (e) which indicates that it fails to capture the true function on the upper right of the domain, and it incurs prediction error of $err_{\boldsymbol{u}} = 1.5 \times 10^{-1}$. Further insight into the performance of the UCGP can be seen in Figure 4.9 (c), which plots $\hat{\theta}$ at each optimisation step for 10 random initialisations of the hyperparameters in blue. The traceplots indicate that the value of the parameter estimate $\hat{\theta}$ obtained with the UCGP is unstable with respect to the initialisation.

We then make use of a HCGP for this problem, specifying the below mean and covariance functions so that the boundary conditions are exactly satisfied. To see this, note that for $t/t' = 0$, $\tilde{k}([x,t],[x',t'])$, $\partial_t\tilde{k}([x,t],[x',t'])$, $\partial_{t'}\tilde{k}([x,t],[x',t'])$, $\partial_t\partial_{t'}\tilde{k}([x,t],[x',t'])$ all equal zero, while $\tilde{m}(x,0) = \cos(x)$ and $\partial_t\tilde{m}(x,0) = \cos^2(x)$, which ensures that the Cauchy initial condition is fulfilled by the HCGP. Similarly $\partial_x\tilde{k}([x,t],[x',t'])$, $\partial_{x'}\tilde{k}([x,t],[x',t'])$, $\partial_x\partial_{x'}\tilde{k}([x,t],[x',t'])$ are all equal to zero if either x or x' lie on the boundary by the chain rule and $\partial_x\tilde{m}(x,t) = 0$ for $x \in \{0,\pi\}$, ensuring that the Neumann spatial conditions are met (see Section 4.7.1 for details of the functions B^- and B^+).

$$\tilde{m}(x,t) = \cos(x) + t\cos^2(x), \tag{4.69}$$

$$\tilde{k}([x,t],[x',t']) = \phi_1(x,t),\phi_1(x',t')k_1([x,t],[x',t']) + \sum_{i=2}^{3}\phi_i(x,t)\phi_i(x',t')k_i(t,t'), \tag{4.70}$$

$$\text{with}\ \ \phi_1(x,t) = x^2(\pi - x)^2t^2,\ \ \phi_2(x,t) = B^-(x)t^2 \quad \text{and} \quad \phi_3(x,t) = B^+(x)t^2. \tag{4.71}$$

In this example, $\tilde{k}$ consists of three individual squared-exponential kernels. This is because of the Neumann conditions on the spatial boundaries, which means that form of the function itself is not prescribed at $x = 0$ or $x = \pi$. The additional kernels k_2 and k_3 are introduced then to represent priors on the function at the two boundaries. To see this, remark that $\tilde{k}([x,t],[x',t']) = k_2(t,t')$ when $x = x' = 0$, while similarly $\tilde{k}([x,t],[x',t']) = k_3(t,t')$ when $x = x' = \pi$. Note the contrast here from the kernel construction in Algorithm 3, where only Dirichlet boundary conditions were considered, in which case there is no uncertainty at the boundary and therefore the prescribed functional form can be "plugged in" instead of a prior.

Using the HCGP, we recover a parameter estimate of $\hat{\theta} = 1.01$, which gives a value of $err_{\boldsymbol{\theta}}$ several orders of magnitude lower than the UCGP. A massive gain performance is also seen in function space, where the posterior mean of the HCGP incurs an error value of $err_{\boldsymbol{u}} = 5.2 \times 10^{-4}$. This prediction is displayed in Figure 4.8 (b), and we see perfect agreement with the true function. Finally, the traceplots from Figure 4.9 (c) demonstrate that the HCGP estimates of θ are stable under different restarts, unlike the UCGP, and convergence is also noticeably faster.

For Wave-BVP-2, we set $\theta = 1$, given which d'Alembert's formula can be applied to yield the form of u given in Eq. (4.72), which we plot in Figure 4.8 (c). As in the above experiment, we again seek to learn θ given $N_u = N_f = 25$ observations using both a UCGP and a HCGP.

$$u(x,t) = \sin(x\pi)\cos((t-0.5)\pi). \tag{4.72}$$

For this inverse problem, the UCGP completely fails, with a best estimate of $\hat{\theta} = 0.06$. The posterior mean is plotted in Figure 4.8 (f) where we see it has collapsed to zero across the domain, yielding $err_{\boldsymbol{u}} = 4.0 \times 10^{-1}$. Traceplots of $\hat{\theta}$ for the UCGP under different restarts are given in Figure 4.9 (b), which show that the parameter estimate tends towards zero irrespective of the initialisation.

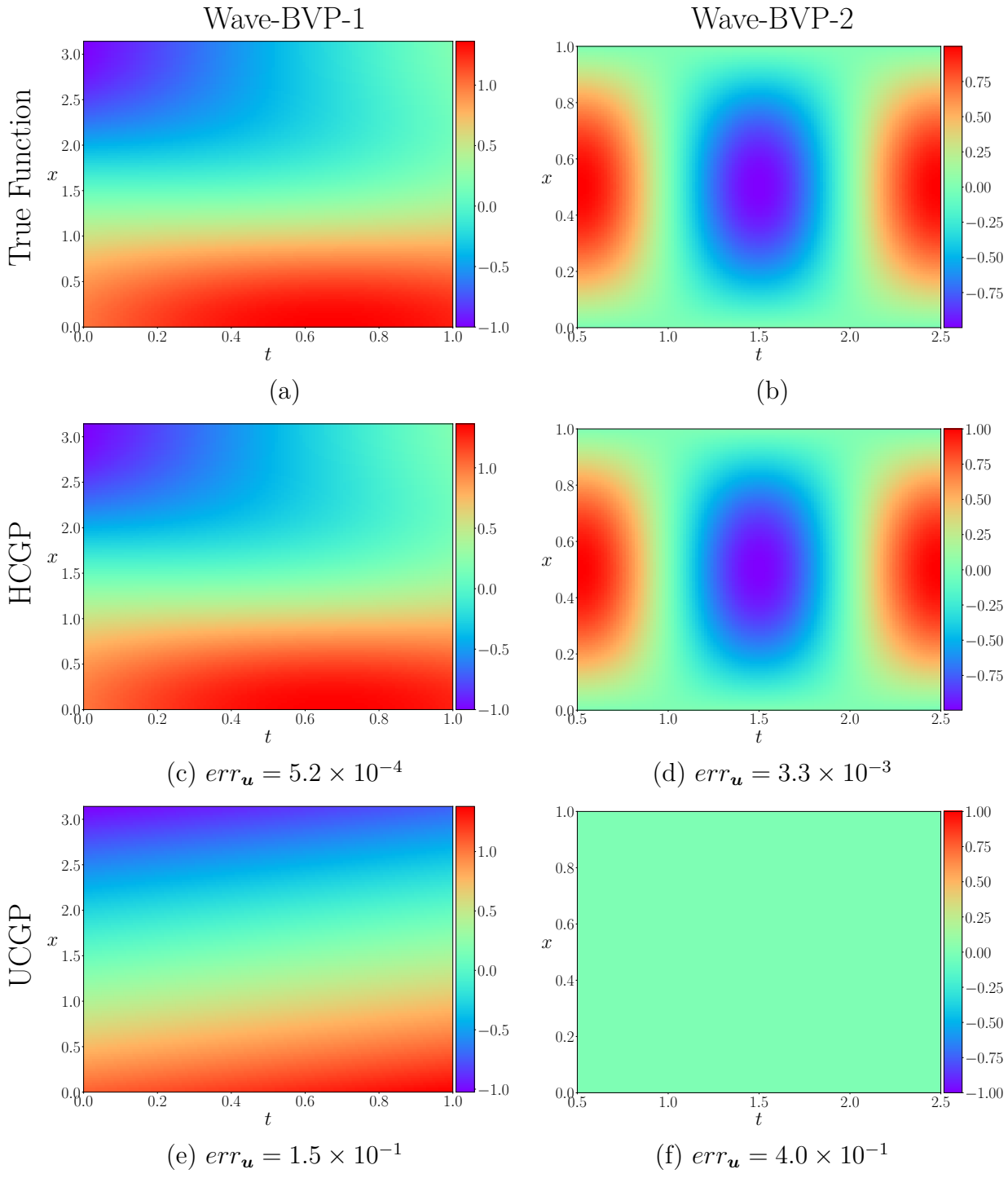


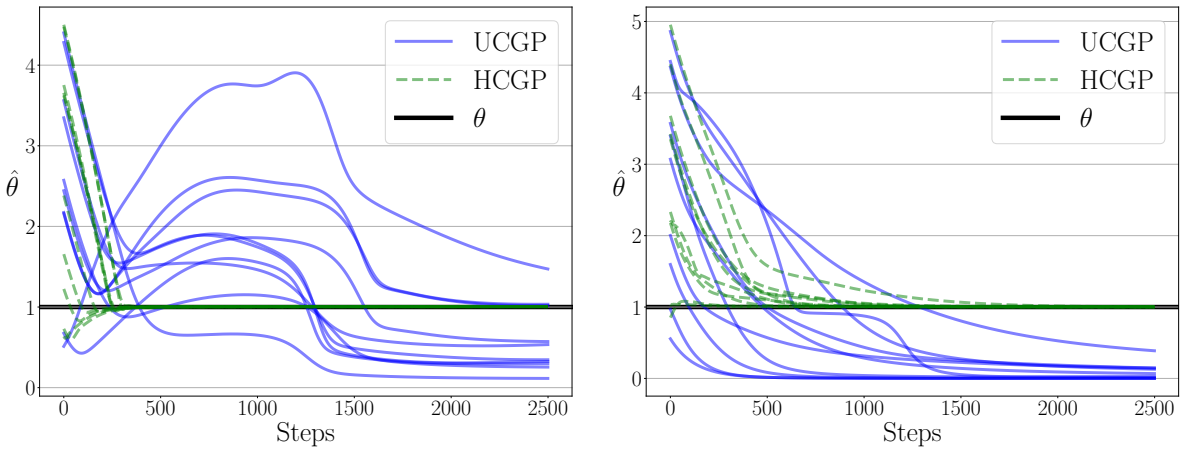
Figure 4.8: Plots of true solution (top row) and HCGP/PCGP predictions (second/third rows) for Wave-BVP-1 (left column) and Wave-BVP-2 (right column).

To use a HCGP for this problem, we specify the following mean and covariance functions such that the boundary conditions are satisfied.

$$\tilde{m}(x,t) = \sin(x\pi), \tag{4.73}$$

$$\tilde{k}([x,t],[x',t']) = \phi_1(x,t)\phi_1(x',t')k_1([x,t][x',t']) + \phi_2(t)\phi_2(t')k_2(t,t'), \tag{4.74}$$

$$\text{with} \quad \phi_1(x,t) = x(1-x)(t-0.5)^2 \quad \text{and} \quad \phi_2(t) = (t-0.5)^2. \tag{4.75}$$



(a) Wave-BVP-1 (b) Wave-BVP-2

Figure 4.9: Traceplots of $\hat{\theta}$ against number of optimisation steps for Wave-BVP-1 and Wave-BVP-2. The bold black horizontal lines indicate the true value of $\theta = 1$ in both plots.

The form of $\tilde{k}$ above is almost the same as that used for Wave-BVP-1, and satisfies the prescribed initial conditions for the same reason. The main difference is that only one additional squared exponential kernel k_2 is needed for Wave-BVP-2, because here periodic spatial boundary conditions are assumed and therefore only one spatial boundary prior is required. In contrast to the UCGP, the HCGP accurately captures both the unknown parameter θ and the underlying function. Specifically, the HCGP learns an estimate of $\hat{\theta} = 1.01$, while the posterior mean achieves $err_{\boldsymbol{u}}$ value of 3.3×10^{-3}. The posterior mean is plotted in Figure 4.8 (d), which illustrates it almost perfectly captures the true function. Finally, the traceplots of $\hat{\theta}$ for the HCGP show robustness of the estimated value to different restarts.

Note that the periodic spatial boundary conditions for Wave-BVP-2 could be handled through the use of a periodic kernel [130, Chapter 4], instead of introducing the additional kernel k_2 in Eq. (4.73). However, during experiments with this approach, we found that the accuracy of the parameter inference was poor, with best estimate of $\hat{\theta} = 2.40$.

4.8.4 Advection-Diffusion Equation

We now apply the HCGP approach to modelling an advection-diffusion equation, which is used to describe the evolution of some quantity of interest inside a physical system due to the processes of *advection* and *diffusion*, for instance the evolution of particles inside a fluid. We will assume a physical system over one spatial dimension x with no sources or sinks, in which case the equation takes the form

$$u_t + \theta_1 u_x - \theta_2 u_{xx} = f = 0, \tag{4.76}$$

where θ_1 controls the strength of the advection term, while θ_2 controls the level of diffusion. We denote the two parameters together as $\boldsymbol{\theta} = (\theta_1, \theta_2)$, and again the inverse problem of identifying $\boldsymbol{\theta}$ from observational data is considered. We set $\theta_1 = 0.05$ and $\theta_2 = 0.025$ and assume the following BVP:

$$\textbf{Advection-Diffusion-BVP:} \begin{cases} (x,t) \in [0,1]^2, \\ u(x,0) = \sin(2\pi x)\exp(x), \\ u(0,t) = u(1,t) = 0. \end{cases}$$

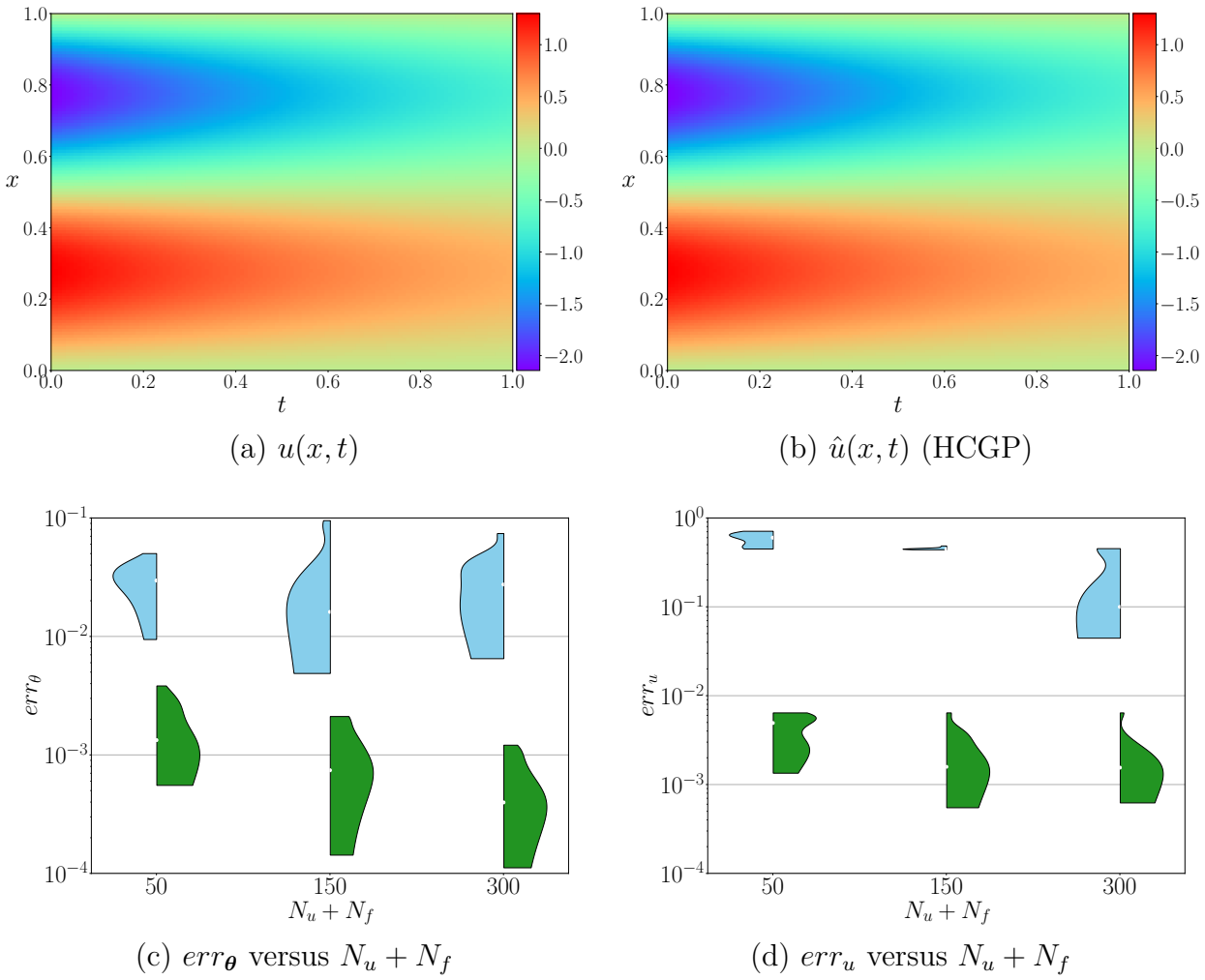
It can be seen that the function u which satisfies the PDE and BVP is given by Eq. (4.77) below, which we visualise using a heatmap in Figure 4.10 (a).

$$u(x,t) = \sin(2\pi x)\exp\left(-(0.1\pi^2 + 0.025)t + x\right). \tag{4.77}$$

Here we have a rectangular domain with Dirichlet boundaries, and therefore Algorithm 3 can be followed to yield the following hard constrained mean/covariance functions:

$$\tilde{m}(x,t) = \sin(2\pi x)\exp(x), \tag{4.78}$$

$$\tilde{k}([x,t],[x',t']) = \phi(x,t)\phi(x',t')k([x,t],[x',t']) \text{ with } \phi(x,t) = 4x(1-x)t. \tag{4.79}$$



(a) $u(x,t)$ (b) $\hat{u}(x,t)$ (HCGP)

(c) $err_{\boldsymbol{\theta}}$ versus $N_u + N_f$ (d) err_u versus $N_u + N_f$

Figure 4.10: Illustration of true solution and HCGP prediction for Advection-Diffusion-BVP (top row). Panel (c) shows parameter inference error as a function of the number of training points for both UCGP (blue) and HCGP (green) approaches, while panel (d) shows function space error.

We used this system to explore how changing the number of observed data points N_u and N_f affects the accuracy of the inference results. Specifically, we generated datasets of sizes $N_u + N_f = \{50, 150, 300\}$ with $N_u = N_f$, where the function space observations $\boldsymbol{y}_u$ were corrupted by 2.5% Gaussian noise. From these datasets, we used HCGPs and UCGPs to learn the parameter vector $\boldsymbol{\theta}$ and solution function u. To give uncertainty quantification for the results, we repeated this procedure under 15 different random regenerations of each dataset. Distributions plots of $err_{\boldsymbol{u}}$ and $err_{\boldsymbol{\theta}}$ against $N_u + N_f$ for both types of model are shown in the bottom row of Figure 4.10, where blue indicates the result of a HCGP, and green the HCGP results. Once again, here the HCGP approach outperforms the UCGP by several orders of magnitude by both error measures. The HCGP was significantly more efficient with respect to the number of training observations used - for instance, a HCGP

trained using only $N_u = N_f = 10$ data points attains similar error values to a UCGP trained with $N = 300$ data points. Figure 4.10 (b) shows the posterior mean of a HCGP trained on a dataset of this size, and we see strong agreement with the true function with $err_{\boldsymbol{u}}$ value of 9×10^{-3} and parameter estimate $\hat{\boldsymbol{\theta}} = (0.054, 0.252)$, compared to the true $\boldsymbol{\theta} = (0.05, 0.025)$.

4.8.5 Helmholtz Equation

We next consider solving the Helmholtz equation over two spatial dimensions x and y

$$-u_{xx} - u_{yy} + \theta^2 u = f, \tag{4.80}$$

with $\theta = 3$, subject to the boundary conditions:

$$\textbf{Helmholtz-BVP:} \begin{cases} (x, y) \in [0, 1]^2 \\ u_x(0, y) = u_y(x, 0) = 0, \\ u(1, y) = u(x, 1) = 0. \end{cases}$$

The following form of the underlying solution u and source term f is specified:

$$u(x, y) = \left(1 - x^2\right)\left(1 - y^2\right) + \cos\left(\frac{\pi x}{2}\right)(\exp(-y) + y - (1 + \exp(-1))), \tag{4.81}$$

$$\begin{aligned} f(x, y) =& 2\left(1 - x^2\right) + 2\left(1 - y^2\right) + \left(\frac{\pi}{2}\right)^2 \cos\left(\frac{\pi x}{2}\right)(\exp(-y) + y - 1 + \exp(-1)) \\ & - \cos\left(\frac{\pi x}{2}\right)\exp(-y) \\ & + \theta^2\left[\left(1 - x^2\right)\left(1 - y^2\right) + \cos\left(\frac{\pi x}{2}\right)(\exp(-y) + y - 1 + \exp(-1))\right]. \end{aligned} \tag{4.82}$$

We take this example from previous work [218], in which boundary-constrained Gaussian processes (BCGPs) are introduced. The BCGP method is based on the spectral theory of elliptic operators and allows for boundary conditions to be exactly enforced for inference in linear PDEs, the same objective as the HCGP framework presented in this chapter. The key difference in practice is that the BCGP method is not applicable to time dependent PDEs or to inverse problems, as it makes use of a spectral decomposition which assumes the parameters $\boldsymbol{\theta}$ of the linear PDE operator are known a-priori. While it

may be possible to extend BCGPs to overcome these limitations, the authors note that this would lead to increased complexity and computational costs [218]. Note the contrast here to the HCGP approach, which naturally accommodates both time-dependent PDEs and inverse problems.

We specify a HCGP to exactly satisfy the Helmholtz-BVP by using a zero mean function and kernel function given by

$$\begin{aligned}\tilde{k}([x,y],[x',y']) = &\phi_1(x,t)\phi_1(x',y')k_1([x,y],[x',y']) + \phi_2(x,y)\phi_2(x',y')k_2(x,x') \\ &+ \phi_3(x,y)\phi_3(x',y')k_3(y,y') + \phi_4(x,y)\phi_4(x',y')\sigma_b^2, \end{aligned} \tag{4.83}$$

where

$$\begin{aligned}&\phi_1(x,y) = x^2(1-x)^2y^2(1-y)^2, \quad \phi_2(x,y) = y^2(1-y)B^-(x), \\ &\phi_3(x,y) = x^2(1-x)^2B^+(y), \qquad \phi_4(x,y) = B^-(x)B^-(y). \end{aligned} \tag{4.84}$$

Again, we refer the reader to Section 4.7.1 for details of the functions B^- and B^+. The structure of this kernel is similar to the kernels used for the Wave-BVPs in Section 4.8.3. By construction, we have $\tilde{k}([x,y],[x',y']) = 0$ if any of x, y, x', y' is equal to one, to ensure the two Dirichlet spatial conditions are satisfied. Similarly, the use of the functions B^- and B^+ enforces the two Neumann conditions, which can be verified using the chain rule as shown for Wave-BVP-1 in Section 4.8.3. Finally, an additional hyperparameter σ_b^2 is included to capture the function behaviour at the point $(0,0)$ where the two Neumann boundaries intersect.

We perform an experiment following that presented in Section 4.2 of [218], whereby the forward problem of learning u is considered, given 10 observations each of u and f, in both cases corrupted with white noise with standard deviation $\sigma_u = \sigma_f = 0.01$. The posterior predictive results for both a UCGP and HCGP are presented in Figure 4.11. We see a clear difference in prediction error obtained with the two approaches, with the HCGP achieving $err_{\boldsymbol{u}}$ value of 1.2×10^{-3}, almost one-order of magnitude lower than the value of 1.0×10^{-2} obtained with the UCGP. For reference, in [218] the BCGP only improved errors by a factor of two over the UCGP. The density plots of prediction error

$|u(x) - \hat{u}(x)|$ and posterior predictive standard deviation $\sigma(x,t)$ further emphasise the advantage of the HCGP framework. From panels (c) and (d), we see that the highest level of prediction error and uncertainty for the UCGP is at the point $(1,1)$. However, this is exactly where the function value is prescribed, and so where uncertainty should in principle be lowest - this is precisely what is captured by the HCGP results in panels (d) and (f).

4.9 Conclusion

In this chapter, we have made use of hard-constrained Gaussian processes (HCGPs) for modelling systems for which linear PDE and boundary value information is available. A HCGP has bespoke mean and covariance functions that are designed so that the specified boundary conditions are satisfied exactly. We analysed the representational capacity of the HCGP framework in the case of Dirichlet boundary conditions, and found that our construction can satisfy universal approximation within the boundary constrained function space of interest. We also demonstrate an equivalence in the infinite width limit of a hard-constrained neural network (HCNN) with a HCGP. Finally, we showed how to extend existing methods to a wider range of commonly used boundary conditions. We then conducted extensive numerical experiments for different linear PDEs, where the primary focus was the inference of any unknown parameters $\boldsymbol{\theta}$ of the PDE operator. A consistent pattern we observed was the HCGP framework offered results that were robust to initialisation, observation noise and data sparsity.

We used the squared-exponential kernel for the experiments in Section 4.8 because it is infinitely differentiable [130, Chapter 4]. This means it is appropriate for modelling all linear PDEs, and is therefore the natural kernel to use in the specification of a PIGP. In subsequent work involving non-linear PDEs, we experimented with the rational-quadratic kernel. Experimental results showed little difference between the two choices. Note that because we used automatic differentiation to implement all the PDE operators, changing the base kernel is trivial - only one line of code needs to be updated.

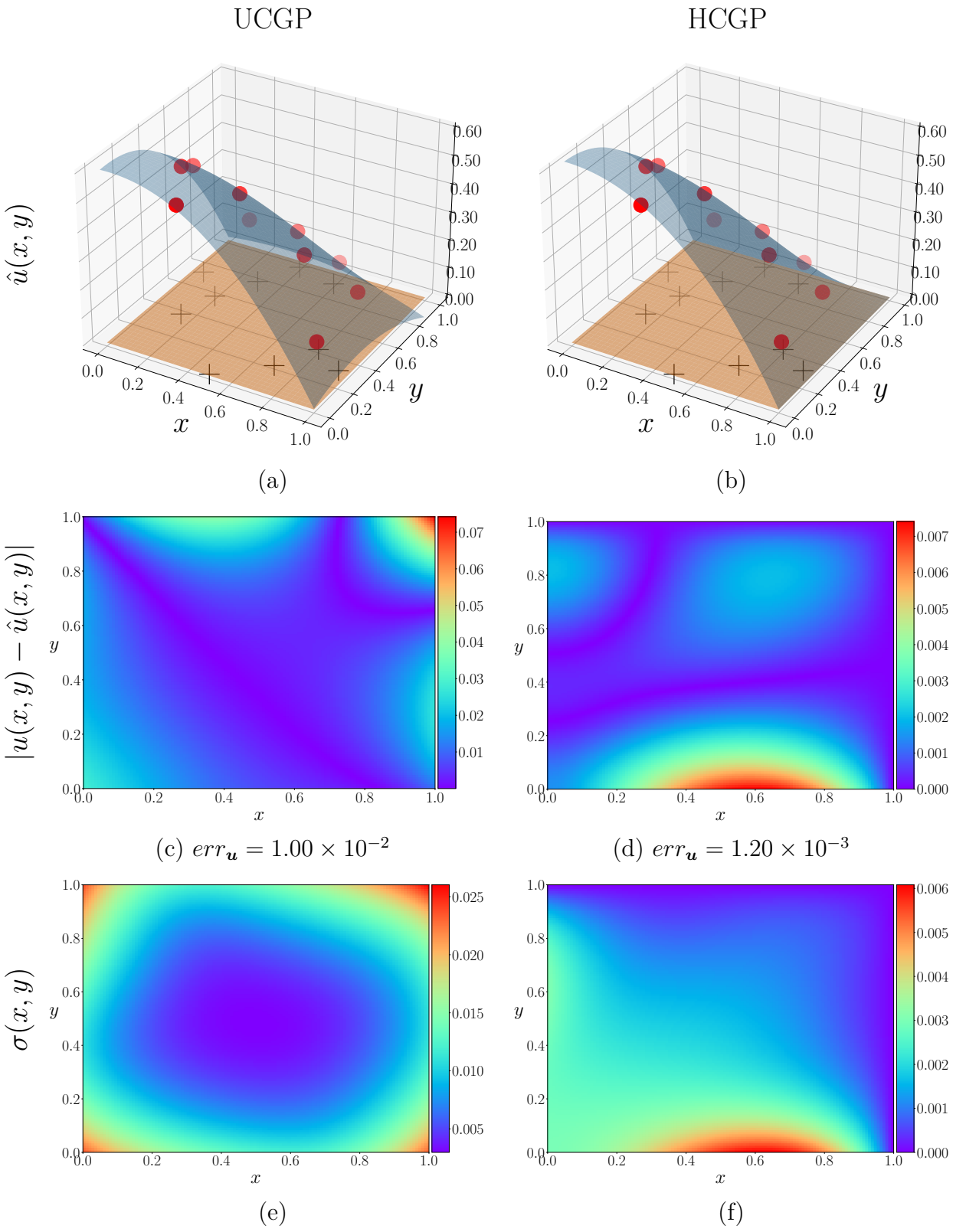


Figure 4.11: Posterior predictive results for Helmholtz equation using UCGP (left column) and HCGP (right column). The red points in the top row show the observations in u-space, while the black crosses show the input locations for the observations in f-space. The second row shows a heatmaps of prediction error, and the final row shows heatmaps of posterior predictive standard deviation.

There are a number of ways in which future work could expand on the material presented in this chapter. Firstly, the GPR approach we use here is limited to the modelling of linear PDEs, as Proposition 4.3.1 does not hold for non-linear differential operators. Several approaches have been proposed to extend GP inference to non-linear PDEs [235], [224], [236]. For more details on this issue, see the discussion in Section 5.1.

Furthermore, our approach is not appropriate for large datasets, as the computational cost of performing full GP inference would be prohibitive. This is because evaluating the marginal likelihood (Eq. (4.18) and posterior predictive distribution (Eq. (4.8)) requires inversion (or in practice Cholesky decomposition) of an $(N \times N)$ training data covariance matrix, where N is the total number of data points available both in u and f space. Since matrix inversion is an $\mathcal{O}(N^3)$ operation, this becomes computationally intractable for large datasets. For general GP regression, several sparse approximation methods have been proposed to alleviate this computational burden [237], [238]. We are not aware of any work in the literature, however, which has adapted these methods to PIGPs. One approach we believe has particular promise in the case of PIGPs is the Vecchia approximation, which allows for an accurate approximation of the matrix inversion to be found very efficiently [239]. The Vecchia approximation has recently shown to deliver significant computational savings with virtually no loss in predictive accuracy, both for shallow and deep GPs [240]. A key advantage of the Vecchia approximation is that it is *parameter free*, meaning that the number of parameters that need to be inferred remains low.

We have also only considered boundary information in the design of the mean and covariance functions $\tilde{m}$ and $\tilde{k}$ used in this work. Knowledge of the PDE itself could also be used however when designing these functions [241]. Finally, here we performed inference in an empirical Bayesian manner, which does not yield uncertainty bounds for the parameter estimates. This could be rectified in future work by taking a fully Bayesian approach to inference using Markov-chain Monte-Carlo, which would allow uncertainty to be more fully quantified.

Chapter 5
Summary

This thesis has focused on the development and application of methods for performing emulation of physical systems described using PDEs, with particular emphasis on the context of soft-tissue mechanics. Traditional, black-box emulators do not directly incorporate known physical principles; instead, these principles must be learned from data. For this reason, our focus in this work has been on *physics-informed emulation* methods, which do directly incorporate physical principles, through the design of the surrogate itself and/or the training routine.

The work of Chapter 2 involved the development of a Graph Neural Network (GNN) architecture for emulation of the passive filling process of the left ventricle (LV) in diastole. A GNN is a class of deep learning model that can be applied to graph-structured data, i.e. data consisting of a set of individual elements called *nodes*, and a set of pairwise relationship between the nodes called *edges*. A classic example of data with this structure is a social network, where the nodes represent different people, while the edges indicate the existence of some form of relationship between each pair of individuals. GNNs have recently been effectively deployed for emulation of physical systems - consider for example a particle based fluid simulation, which can be represented as a graph where the nodes are the individual fluid particles, while the edges denote an interaction relationship between neighbouring nodes. The objective of Chapter 2 was to deploy this type of emulation model in the context of cardiac mechanics, in which case the graph represents a discretised form of the LV geometry. This is a challenging emulation problem, due to the heterogeneity in the geometries of different subjects. A message passing GNN was introduced for this emulation task, which allowed for generalisation to different LVs geometries and material parameter configurations. To improve generalisation to different LVs, global geometry information was accounted for using a low order representation

found with PCA. To improve computational efficiency at prediction time, the material parameters were handled independently of the message passing stage of the emulator. A systematic evaluation of the proposed method was performed using a series of numerical experiments. Strong out of sample performance was observed - for instance, expected errors in LV volume extracted from the GNN results were on the order of 1%, lower than the errors that can be expected from manual extraction [91]. The GNN also significantly outperformed a benchmark emulation based on a fully connected neural network. Finally, massive computational savings were made at prediction time compared to the simulator.

In Chapter 3, a physics-informed GNN (PI-GNN) framework was introduced for emulation of soft-tissue mechanics. Physics-informed neural networks (PINNs) are a class of deep learning architecture where known physical laws or principles are directly incorporated into the training routine. To create a PI-GNN for soft tissue mechanics, we leveraged the Principle of Minimum Potential Energy, which states that in static equilibrium, the total potential energy of a system is minimised. Therefore, instead of training an emulator to against simulation data, training can be performed to minimise the potential energy (PE). The clear advantage of this approach is that it does not require any simulation data to be generated for emulator training. However, we found that the PE was an unstable objective function for emulator training, as unphysical predictions from the GNN would cause the PE to diverge. To alleviate this, we introduced internal transformations to stabilise the PE computations, enabling effective and efficient training. Extensive numerical experiments showed strong predictive performance of the PI training routine. Of particular note were the results of comparison between data-driven and physics-informed training. While both yielded similar accuracy in the prediction of forward displacements, the predictions from the data-driven emulator consistently failed to find the minimum energy state, in contrast to the physics-informed emulator.

While performing the numerical experiments presented in Chapter 3, we found that the way in which the boundary conditions (BCs) were handled had a significant impact on the training behaviour of the physics-informed emulator. If BCs were softly enforced by a penalty term in the loss function, then it was extremely difficult to tune the learning rate to yield good performance. However, if the BCs were explicitly enforced by a post-processing transformation of the emulator's predictions (see Section 3.2.4.4), then

training was far more efficient. In Chapter 4, we investigated more deeply the idea of hard boundary condition enforcement in the context of PIML modelling. Instead of considering neural networks, here we made use of Gaussian process (GP) models, and applied them in the context of linear PDEs. In contrast to the NN models from Chapters 2 and 3 respectively, a GP offers a probabilistic approach to learning the solution function to a PDE. GPs are particularly well suited to modelling systems where linear PDE information is available, as in this case a consistent prior distribution can be placed over both solution and PDE space. This can then be updated to a posterior distribution given (possibly noisy) observational data. The first aspect of Chapter 4 involved the theoretical analysis of what we label hard-constrained Gaussian processes (HCGPs), where the kernel is designed so that the GP explicitly satisfies the boundary conditions. We first established a universal approximation theorem for the HCGP kernel, and then introduced a theoretical link between HCGPs and hard-constrained neural networks (HCNNs). Finally, extension of HCGPs to more general boundary conditions (such as Robin conditions) was discussed. Extensive numerical experiments involving several different linear PDEs were then conducted, involving both forward and inverse problems. The results showed that explicitly controlling for boundary conditions allowed more accurate results to be obtained than ignoring boundary conditions or using a soft-enforcement of boundary conditions. We also observed that HCGPs were much less prone to the well known problem of PIML models learning the trivial solution of $\hat{u} \approx 0$ in the presence of noisy/sparse data.

5.1 Future Work

The design, testing and use of emulators which directly incorporate knowledge of physical laws and principles is still in its infancy. For this reason, there exist a range of potential future research directions in this area which could directly expand upon the work that has been presented in the preceding three chapters, examples of which we give below.

In Chapter 2, we used a GNN emulator that explicitly enforced the known *translation invariance* of the underlying simulator. However, we did not explicitly enforce the *rotational equivariance* of the simulator, and therefore this property had to be learned by the surrogate model. The design of neural networks which satisfy equivariance under

rotation (and more general group operations) is a topical research area [98] - future work could build on this literature to construct a new emulator for the LV model. Another possible extension would be to consider modelling the entire cardiac cycle, not just diastole, and the entire heart geometry, not just the LV. Finally, it is natural to consider the inverse problem in future work, where the objective is to infer the stiffness properties of the cardiac tissue using data obtained from cardiac imaging scans [89].

In Chapter 3, we performed physics-informed training by making use of an *energy* formulation of the mechanics problem. An alternative approach would be to train the emulator by making use of the so-called *weak* formulation. Here, training would be performed by application of the Principle of Virtual Work [77, Section 8.2], rather than the Principle of Minimum Potential Energy [77, Section 8.3]. While both formulations are equivalent mathematically, numerical differences between the two can arise in practice, and it would be interesting to see if this impacts the efficiency and accuracy of the emulator. Another extension of this PI-GNN framework would be to consider emulation of a particle-based fluid simulator, which could be used in the context of biofluid mechanics, for instance. The main complication here over the work presented in Chapter 3 is that in this case the particle topology changes over time and therefore needs to be recalculated at each time step [158].

Future work could also explore how the use of prior information could leverage the gains yielded by physics-informed approaches. For instance, consider the parameter inference study by Lazarus et al. [92], which sought to infer the material parameters of the cardiac muscle tissue from observational data, using a similar mechanical model to the one described in Chapter 2. The authors improved the parameter inference results by designing physically meaningful priors using the results of existing empirical studies. It would be interesting to see what effect imbuing this inference routine with physics informed knowledge (e.g. The Principle of Minimum Potential Energy) would have. There could be a synergistic effect in incorporating information from different sources (i.e. empirical studies and physical laws), which may allow for more accurate inference results in the presence of noisy or sparse data.

For the GP modelling work presented in Chapter 4, the obvious limitation of this framework is its restriction to linear PDEs. In subsequent work completed after the submission of this thesis [8], we have extended this approach to allow for non-linear PDEs to be accommodated, using an approximation method called variational inference [222, Chapter 10] (similar to the approach of [236]). To explain how this works, consider the posterior predictive distribution $p(\boldsymbol{u}_s \mid \boldsymbol{y}_u, \boldsymbol{y}_f)$, where $\boldsymbol{u}_s$, $\boldsymbol{y}_u$ and $\boldsymbol{y}_f$ are defined in Chapter 4. We can write the posterior in un-marginalised form as

$$p(\boldsymbol{u}_s \mid \boldsymbol{y}_u, \boldsymbol{y}_f) = \int p(\boldsymbol{u}_s \mid \boldsymbol{u}, \boldsymbol{f}) p(\boldsymbol{u}, \boldsymbol{f} \mid \boldsymbol{y}_u, \boldsymbol{y}_f) d\boldsymbol{u} d\boldsymbol{f}, \tag{5.1}$$

where $[\boldsymbol{u}]^{(i)} = u(\boldsymbol{x}_u^{(i)})$ (see Eq. (4.1)) and $[\boldsymbol{f}]^{(i)} = f(\boldsymbol{x}_f^{(i)})$ (see Eq. (4.10)). In Chapter 4, we assumed that $f(\boldsymbol{x}) = \mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u](\boldsymbol{x})$ with $\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{\theta}}$ a *linear* PDE operator. In this case, both probability distributions under the integral in Eq. (5.1) are Gaussian, allowing $\boldsymbol{u}$ and $\boldsymbol{f}$ to be marginalised out exactly, yielding a closed form expression for $p(\boldsymbol{u}_s \mid \boldsymbol{y}_u, \boldsymbol{y}_f)$, which is another Gaussian (see Eq. (4.8)).

If, however, we have $f(\boldsymbol{x}) = \mathcal{F}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u](\boldsymbol{x})$ with $\mathcal{F}_{\boldsymbol{x}}^{\boldsymbol{\theta}}$ a *non-linear* PDE operator, then neither distribution under the integral in Eq. (5.1) are Gaussian, because Gaussian random variables are not closed under non-linear operations. This means in turn that the latent vectors $\boldsymbol{u}$ and $\boldsymbol{f}$ cannot be marginalised out exactly. We overcome this issue using two stages. Firstly, we decompose the nonlinear PDE into its *sublinear* differential operators. Consider for instance the Allen-Cahn equation over one spatial dimension:

$$\mathcal{F}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u](\boldsymbol{x}) = \partial_t u(\boldsymbol{x}) - \theta \partial_{xx} u(\boldsymbol{x}) - (u(\boldsymbol{x}) - u(\boldsymbol{x})^3) = 0, \tag{5.2}$$

where $\boldsymbol{x} = (x, t)$. If we set $d_0(\boldsymbol{x}) = u(\boldsymbol{x})$ and $d_1(\boldsymbol{x}) = \partial_t u(\boldsymbol{x}) - \theta \partial_{xx} u(\boldsymbol{x})$, then we can write our PDE as an algebraic equation of the form:

$$\mathcal{F}_{\boldsymbol{x}}^{\boldsymbol{\theta}}[u](\boldsymbol{x}) = F(d_0(\boldsymbol{x}), d_1(\boldsymbol{x})) = d_1(\boldsymbol{x}) - (d_0(\boldsymbol{x}) - d_0(\boldsymbol{x})^3). \tag{5.3}$$

We then replace $\boldsymbol{f}$ with $\boldsymbol{d}_0$ and $\boldsymbol{d}_1$ in Eq. (5.1)

$$p(\boldsymbol{u}_s \mid \boldsymbol{y}_u, \boldsymbol{y}_f) = \int p(\boldsymbol{u}_s \mid \boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1) p(\boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1 \mid \boldsymbol{y}_u, \boldsymbol{y}_f) d\boldsymbol{u} d\boldsymbol{d}_0 d\boldsymbol{d}_1, \tag{5.4}$$

where $[\boldsymbol{d}_j^{(i)}] = d_j(\boldsymbol{x}_f^{(i)})$ for $j = 0, 1$. The advantage of this reformation is that $p(\boldsymbol{u}_s \mid \boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1)$ is now a Gaussian, because both d_0 and d_1 are found by applying a linear operator to u. The posterior distribution over the latent vectors $\boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1$ given the observational data is still not a Gaussian, however, due to the nonlinearity of F in Eq. (5.3). For this reason we proceed by variational inference, whereby a variational Gaussian approximation is introduced:

$$p(\boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1 \mid \boldsymbol{y}_u, \boldsymbol{y}_f) \approx q(\boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1) = \mathcal{N}(\boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1 \mid \boldsymbol{a}, \mathbf{S}). \tag{5.5}$$

The variational mean $\boldsymbol{a}$ and covariance $\mathbf{S}$ are jointly inferred together with the kernel/noise/PDE parameters by maximising a tractable lower bound to the log marginal likelihood called the *evidence lower bound* (ELBO). Once all parameters have been inferred, we can evaluate an approximate posterior predictive distribution $\hat{p}$ as follows:

$$\hat{p}(\boldsymbol{u}_s \mid \boldsymbol{y}_u, \boldsymbol{y}_f) = \int p(\boldsymbol{u}_s \mid \boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1) q(\boldsymbol{u}, \boldsymbol{d}_0, \boldsymbol{d}_1) d\boldsymbol{u} d\boldsymbol{d}_0 d\boldsymbol{d}_1, \tag{5.6}$$

We do not show the derivations here, but $\hat{p}$ is Gaussian.

Appendices

A Appendix for Chapter 2

A.1 Additional Beam Emulation Results

The emulation results for the non-augmented graph representations from Figure 2.8 are tabulated in Table A.1 below, while the results for the augmented graph representations are given in Table A.2. Table A.4 displays emulation results for different values of the node cardinality input vector to Algorithm 1, $\boldsymbol{\eta}$, from Figure 2.9 (a), while Table A.3 gives the results from Figure 2.9 (b), displaying emulation errors for different values of internal embedding dimension M. Note that for these tables the representative graph was chosen using the "averaging" method described in Section 2.2.3.

Table A.1: Emulation error (in mm) statistics for different values of message passing steps K, when using **non-augmented** graph representations with $M = 40$.

Error Percentile	**Number of Message Passing Steps**					
	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 6$
25^{th}	4.16×10^{-1}	5.33×10^{-2}	2.30×10^{-2}	1.78×10^{-2}	1.38×10^{-2}	1.39×10^{-2}
50^{th}	1.47	6.64×10^{-1}	8.21×10^{-2}	4.06×10^{-2}	2.72×10^{-2}	2.85×10^{-2}
75^{th}	2.69	1.74	9.74×10^{-1}	1.01×10^{-1}	4.82×10^{-2}	5.35×10^{-2}

Table A.2: Emulation error (in mm) statistics for different values of message passing steps K, when using **augmented** graph representations with $\boldsymbol{\eta} = [24, 6]$ and $M = 40$.

Error Percentile	**Number of Message Passing Steps**					
	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 6$
25^{th}	4.75×10^{-2}	1.36×10^{-2}	1.47×10^{-2}	1.33×10^{-2}	1.35×10^{-2}	1.78×10^{-2}
50^{th}	1.61×10^{-1}	2.79×10^{-2}	2.71×10^{-2}	2.73×10^{-2}	3.09×10^{-2}	3.47×10^{-2}
75^{th}	1.79	5.04×10^{-2}	5.42×10^{-2}	6.07×10^{-2}	6.26×10^{-2}	6.60×10^{-2}

Table A.3: Emulation error (in mm) statistics for different values for $\boldsymbol{\eta}$, the node cardinality input vector to Algorithm 1.

Error Percentile	**Node Cardinality Input Vector to Algorithm 1**					
	$[4]$	$[8]$	$[24, 6]$	$[48, 12]$	$[48, 12, 4]$	$[48, 24, 6]$
25^{th}	1.23×10^{-2}	1.58×10^{-2}	1.36×10^{-2}	1.68×10^{-2}	1.48×10^{-2}	1.35×10^{-2}
50^{th}	2.37×10^{-2}	2.84×10^{-2}	2.79×10^{-2}	3.45×10^{-2}	2.76×10^{-2}	2.49×10^{-2}
75^{th}	4.58×10^{-2}	4.64×10^{-2}	5.04×10^{-2}	6.34×10^{-2}	4.91×10^{-2}	4.70×10^{-2}

Table A.4: Emulation error (in mm) statistics for different values for the dimensionality M of the embedding vectors internal to the GNN, with $K = 2$ and $\boldsymbol{\eta} = [24, 6]$.

Error Percentile	**Dimensionality of Internal Embedding Vectors**					
	$M = 3$	$M = 6$	$M = 10$	$M = 20$	$M = 40$	$M = 60$
25^{th}	5.63×10^{-2}	2.00×10^{-2}	1.84×10^{-2}	1.76×10^{-2}	1.36×10^{-2}	1.22×10^{-2}
50^{th}	1.02×10^{-1}	3.80×10^{-2}	3.46×10^{-2}	3.21×10^{-2}	2.79×10^{-2}	2.49×10^{-2}
75^{th}	1.96×10^{-1}	6.72×10^{-2}	6.07×10^{-2}	5.53×10^{-2}	5.04×10^{-2}	4.97×10^{-2}

Tables A.5-A.7 display the analagous results to Tables A.2-A.4 except that a template beam generated using FEniCS was used for representative graph. All triangular elements of the template had height of 1 cm each, giving total height/width of 8 cm/12 cm. The variation in emulation performance for the two choices of reference graph (on the order of 1×10^{-2}mm) is very slight relative to the magnitude of the mean nodal forward displacement of 4.2mm from the test set.

Table A.5: Emulation error (in mm) statistics for different values of message passing steps K, when using **augmented** graph representations when using **FEniCS Template** as reference graph with $\boldsymbol{\eta} = [24, 6]$ and $M = 40$.

Error Percentile	**Number of Message Passing Steps**					
	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 6$
25^{th}	3.90×10^{-2}	1.34×10^{-2}	1.38×10^{-2}	1.18×10^{-2}	1.24×10^{-2}	1.21×10^{-2}
50^{th}	1.04×10^{-1}	2.35×10^{-2}	2.79×10^{-2}	2.53×10^{-2}	2.58×10^{-2}	2.57×10^{-2}
75^{th}	9.02×10^{-1}	4.23×10^{-2}	5.68×10^{-2}	5.20×10^{-2}	5.23×10^{-2}	5.33×10^{-2}

Table A.6: Emulation error (in mm) statistics for different values for $\boldsymbol{\eta}$, the node cardinality input vector to Algorithm 1 when using **FEniCS Template** as reference graph with $K = 2$ and $M = 40$.

Error Percentile	**Node Cardinality Input Vector η to Algorithm 1**					
	$[4]$	$[8]$	$[24, 6]$	$[48, 12]$	$[48, 12, 4]$	$[48, 24, 6]$
25^{th}	1.49×10^{-2}	1.26×10^{-2}	1.34×10^{-2}	1.21×10^{-2}	1.22×10^{-2}	1.60×10^{-2}
50^{th}	2.80×10^{-2}	2.39×10^{-2}	2.35×10^{-2}	2.38×10^{-2}	2.42×10^{-2}	3.08×10^{-2}
75^{th}	5.23×10^{-2}	4.34×10^{-2}	4.23×10^{-2}	4.48×10^{-2}	4.71×10^{-2}	5.93×10^{-2}

Table A.7: Emulation error (in mm) statistics for different values for the dimensionality M of the embedding vectors internal to the GNN when using **FEniCS Template** as reference graph with $\boldsymbol{\eta} = [24, 6]$ and $K = 2$.

Error Percentile	**Dimensionality of Internal Embedding Vectors**					
	$M = 3$	$M = 6$	$M = 10$	$M = 20$	$M = 40$	$M = 60$
25^{th}	2.94×10^{-2}	2.23×10^{-2}	1.69×10^{-2}	1.32×10^{-2}	1.34×10^{-2}	1.38×10^{-2}
50^{th}	5.71×10^{-2}	4.13×10^{-2}	3.25×10^{-2}	2.73×10^{-2}	2.35×10^{-2}	2.84×10^{-2}
75^{th}	1.08×10^{-1}	7.24×10^{-2}	5.83×10^{-2}	5.16×10^{-2}	4.23×10^{-2}	5.53×10^{-2}

A.2 Additional LV Emulation Results

Table A.8 gives emulation error statistics for different numbers of PCs retained in $\boldsymbol{z}^{\text{global}}$ from Figure 2.13 (a), while Figure 2.13 (b), which displays emulation error against number of message passing steps K, is tabulated in Table A.9. Table A.10 displays the comparison between GNN and MLP prediction errors for LVV from Figure 2.14. Finally, the comparison of prediction accuracy for the general and transfer-learned GNN emulators from Figure 2.15 is given in Table A.11. In addition, learning curves for the GNN and MLP emulators are displayed in Figure A.1, for both training and validation data.

Table A.8: LV emulation error statistics (mm) for different number of principal components (PCs) retained in $\boldsymbol{z}^{\text{global}}$, with $K = 5$ and $M = 40$.

Error Percentile	**Number of PCs Retained**					
	0 PCs	8 PCs	16 PCs	24 PCs	32 PCs	40 PCs
25^{th}	2.80×10^{-1}	2.43×10^{-1}	2.20×10^{-1}	1.76×10^{-1}	1.51×10^{-1}	1.40×10^{-1}
50^{th}	5.27×10^{-1}	4.46×10^{-1}	4.13×10^{-1}	3.18×10^{-1}	2.66×10^{-1}	2.41×10^{-1}
75^{th}	1.05	9.00×10^{-1}	8.23×10^{-1}	6.10×10^{-1}	4.99×10^{-1}	4.34×10^{-1}

Table A.9: LV emulation error statistics (mm) for different values of message passing steps K with $M = 40$.

Error Percentile	**Number of Message Passing Steps**					
	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 6$
25^{th}	1.72×10^{-1}	1.62×10^{-1}	1.59×10^{-1}	1.60×10^{-1}	1.51×10^{-1}	1.56×10^{-1}
50^{th}	2.98×10^{-1}	2.81×10^{-1}	2.78×10^{-1}	2.78×10^{-1}	2.66×10^{-1}	2.67×10^{-1}
75^{th}	5.42×10^{-1}	5.20×10^{-1}	5.25×10^{-1}	5.10×10^{-1}	4.99×10^{-1}	4.90×10^{-1}

Tables A.12 and A.13 display MLP emulation error statistics when early stopping using patience is applied during training. When using patience, either the parameters for which the patience level is exceeded can be returned, or back-tracking can be applied to return the network parameters for which validation loss is minimised. Table A.12 displays

Table A.10: LVV emulation error statistics (%) for two MLP emulators and two GNN emulators.

Error Percentile	**Emulator**			
	MLP 1	MLP 2	GNN 1	GNN 2
25^{th}	9.51%	1.95%	0.28%	0.24%
50^{th}	20.34%	4.35%	0.60%	0.49%
75^{th}	36.13%	7.78%	1.13%	0.89%

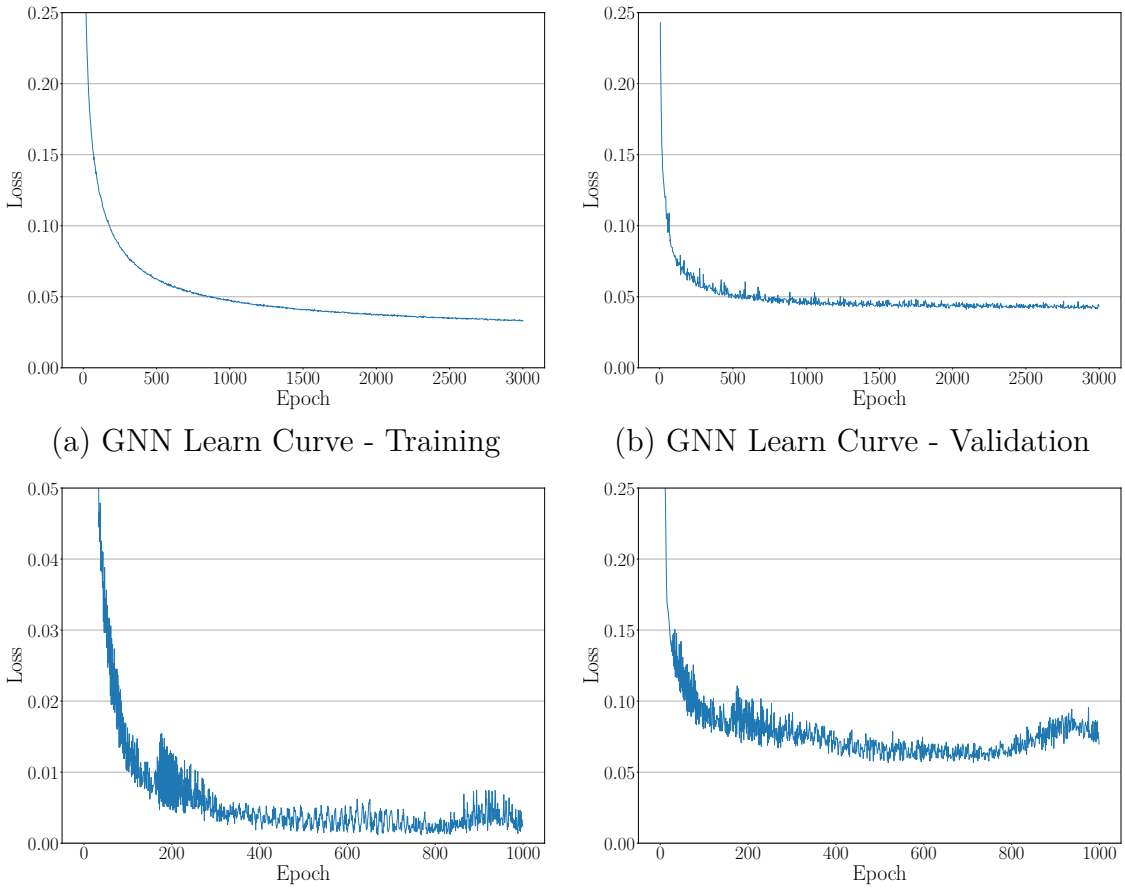
Table A.11: Emulation error statistics for **general** and **transfer-learned** emulators. The "Displacement" columns give prediction errors on the nodal displacement vectors (in mm), while the "LVV" columns give percentage errors in LVV predictions.

Error Percentile	**General**		**Transfer-Learned**	
	Displacement	LVV	Displacement	LVV
25^{th}	1.85×10^{-1}mm	0.18%	6.34×10^{-2}mm	0.16%
50^{th}	2.95×10^{-1}mm	0.43%	1.13×10^{-1}mm	0.34%
75^{th}	5.13×10^{-1}mm	0.76%	2.08×10^{-1}mm	0.64%

results when back-tracking is applied, and Table A.13 shows results when back-tracking is not applied. For higher patience levels, emulation results are slightly more accurate when back-tracking is used, while the converse is true for lower patience levels. In both cases, if the patience level is less than 100, the early stopping routine halts training before the plateau in validation loss is reached, leading to less accurate results. Note that training using back-tracking with a patience level of 100 leads to the same parameter values being chosen as training for 1000 epochs and selecting the parameters with optimal validation loss.

Table A.12: MLP LVV emulation error statistics (in %) for different choices of patience level, where **back-tracking is applied** to find the optimal network weights on the validation data.

	Patience Level (Stopping Epoch)				
Error	**10** (93)	**25** (142)	**50** (319)	**100** (620)	**None** (1000)
25^{th}	2.85%	2.69%	2.31%	1.95%	1.95%
50^{th}	6.31%	5.86%	5.05%	4.35%	4.35%
75^{th}	12.11%	10.73%	9.38%	7.78%	7.78%



(a) GNN Learn Curve - Training

(b) GNN Learn Curve - Validation

(a) MLP Learn Curve - Training

(b) MLP Learn Curve - Validation

Figure A.1: Emulator learning curves. Panel (a) shows training loss against epoch number and panel (b) shows validation loss, both for the **GNN Emulator**. The second row of panels shows learning curves for the **MLP Emulator** - panel (c) shows training loss against epoch number, and panel (d) validation loss.

Table A.13: MLP LVV emulation error statistics (in %) for different choices of patience level, where **back-tracking is not applied** to find the optimal network weights on the validation data.

	Patience Level (Stopping Epoch)				
Error	**10** (93)	**25** (142)	**50** (319)	**100** (620)	**None** (1000)
25^{th}	2.90%	2.60%	2.52%	2.52%	1.95%
50^{th}	6.51%	5.71%	5.55%	5.33%	4.35%
75^{th}	12.03%	10.73%	10.50%	9.28%	7.78%

A.3 Synthetic LV Geometry Generation

As discussed in Section 2.4.2, the LV emulation study was performed with synthetic LV geometries that were generated by sampling in a latent space found using Principal Components Analysis (PCA). PCA was performed on a dataset of $N = 65$ real LV geometries $\{\mathbf{G}_n\}_{n=1}^{N}$, which were extracted from in-vivo cardiac magnetic resonance (CMR) images

of healthy volunteers, obtained at early-diastole. Both long and short axial CMR cine images were used in the extraction procedure. Full details of the study design, imaging protocol and extraction procedure used are given in previous work [60]. Each geometry is represented by 1696 nodes in both the endocardial and epicardial surfaces respectively, or a total of $N_v = 3392$ nodes, each of dimension 3. Before applying PCA, each representation $\mathbf{G}_n$ was flattened to a row vector $\mathbf{g}_n$ of width $D = 3392 \times 3 = 11796$, where the coordinates of each vector were centred with respect to the mean geometry vector $\bar{\mathbf{g}}$ evaluated over the N samples. The objective of PCA is to find an orthonormal matrix $\mathbf{B} \in \mathbb{R}^{D \times P}$ from which an $P << D$ dimensional compression of the data is computed as follows:

$$\mathbf{z}_n^{\text{PCA}} = \mathbf{B}^\top \mathbf{g}_n. \tag{A.1}$$

$\mathbf{B}$ is found as the projection for which the reconstructions in the original space,

$$\hat{\boldsymbol{g}}_n = \mathbf{B}\mathbf{B}^\top \mathbf{g}_n, \tag{A.2}$$

minimise the following mean squared error loss over the training data:

$$\mathcal{L}_{\text{PCA}}(\mathbf{B}) = \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{g}_n - \hat{\boldsymbol{g}}_n||_2^2 = \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{g}_n - \mathbf{B}\mathbf{B}^\top \mathbf{g}_n||_2^2. \tag{A.3}$$

Consider the covariance matrix of the data, and its associated eigendecomposition

$$\mathbf{S} = \frac{1}{N-1}\sum_{n=1}^{N} \mathbf{g}_n \mathbf{g}_n^\top = \sum_{n=1}^{N} \lambda_n \mathbf{b}_n \mathbf{b}_n^\top, \tag{A.4}$$

where the eigenvalues $\lambda_1 > \lambda_2 > ..., \lambda_N$ give the variances of the data along their associated eigenvectors $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_N$. It can be shown that $\mathbf{B}$ can be selected to minimise Eq. (A.3) by setting its p^{th} column equal to the eigenvector $\mathbf{b}_p$, for $p = 1, 2, ..., P$. Because these eigenvectors are the components along which the data varies the most, they are referred to as the principal component vectors. Note that in our case, the covariance matrix $\mathbf{S}$ is rank deficient, as we have $D > N$. In addition, its rank is further reduced by one as we have mean centred the data. Therefore, $\mathbf{S}$ has only $N - 1$ non-zero eigenvalues, meaning that up to $N - 1$ principal components can be extracted.

Applying PCA simply requires the number of leading PCs retained P to be selected. This value is typically specified by considering the cumulative proportion of variance explained by the first P PCs, τ_P^2, which is defined as follows

$$\tau_P^2 = \frac{\sum_{p=1}^{P} \lambda_p}{\sum_{n=1}^{N-1} \lambda_n} \tag{A.5}$$

for $P = 1, 2, ...N-1$. The value of P can be chosen as the smallest number of components such that τ_P^2 exceeds some threshold value that is deemed sufficient to obtain an acceptable representation of the LV geometry. This is the approach taken for example in [86] to select the use of 2 principal components for representation of the LV anatomy. The value of τ_P^2 on our dataset is displayed in Figure A.2 (a) for $P = 1, 2, ...20$. As has been observed in previous analyses of LV geometry data [86], τ_P^2 quickly approaches 100% for increasing values of P. The proportion of variance explained measure is widely applied and is useful for the selection of P for arbitrary datasets. In the case of the specific dataset considered here however, it does not account for the fact that the latent vectors $\mathbf{z}_n^{\text{PCA}}$ represent the geometry of a LV embedded in 3 dimensional space. For this reason, we take a different approach here to select the number of leading PCs P to retain, by considering the mean node-wise Euclidean distance between the real LV geometries $\mathbf{G}_n$ and the corresponding geometry reconstructions $\hat{\mathbf{G}}_n$ from the P-dimensional PC space:

$$\delta_P = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{N_v} \sum_{j=1}^{N_v} ||\mathbf{G}_i[j] - \hat{\mathbf{G}}_i[j]||_2 \tag{A.6}$$

The notation $\mathbf{G}_i[j]$ means we extract the j^{th} node from the i^{th} geometry. Using this measure allows the performance of PCA for different values of P to be evaluated in the same geometry space from which simulations are run. High values of δ_P mean that the synthetic LV geometries are failing to capture the anatomical features present in the real data. This in turns means that simulations run using the synthetically generated geometries may exhibit bias with respect to simulations using the true LV geometries, indicating that more principal components should be retained. Figure A.2 (b) plots the value δ_P (in mm) on our dataset for the first 40 principal components. In this case, we observe that δ_P does not go to zero as rapidly as τ_P^2 goes to 1.

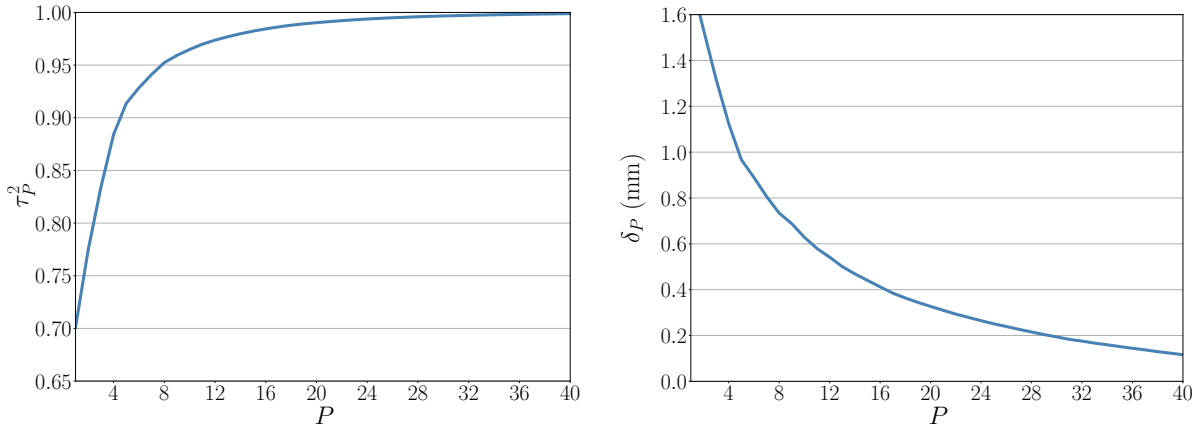
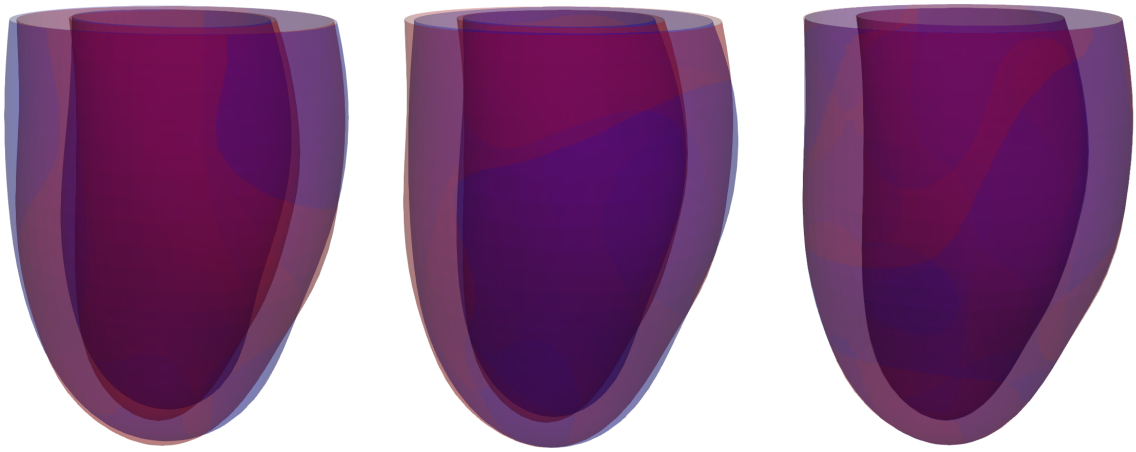


Figure A.2: Panel (a) plots τ_P^2, the cumulative proportion of variance explained by the first P principal components (PCs), for $P = 1, 2, ..., 20$. Panel (b) plots the mean reconstruction error δ_P (in mm) for the same values of P.

Consequently, we retained a large number $P = 40$ leading principal components when generating the synthetic LV geometries for this study, where the mean nodal reconstruction error δ_{40} was 0.12 mm. By sampling in a latent space of this dimensionality, the synthetic geometries that are generated exhibit greater realism than samples from a lower-dimensional space, in terms of asymmetry, skewness and wall thickness variation. This effect is illustrated in Figure A.3. Panel A.3 (a) displays a randomly selected, real LV geometry against its reconstruction from a 2-dimensional PC space, as used in [86]. Panels (b) and (c) display the analogous plots for reconstructions from a 5-dimensional space (as used in [89]) and a 40 dimensional space respectively. The reconstruction from the 40 dimensional latent space incurs an average nodal reconstruction error of 0.11 mm, significantly lower than that of the 2 dimensional reconstruction (1.60 mm) and the 5 dimensional reconstruction (0.81 mm). Higher values of P could be considered, but run the risk of overfitting to the dataset of only $N = 65$ real anatomies.

Once PCA has been performed, a synthetic LV geometry vector can be generated by first assigning a probability distribution over the P dimensional latent space, from which a random sample $\boldsymbol{z}^*$ is drawn. The sample is re-projected back to the full D dimensional space using $\mathbf{B}$, before adding the mean geometry vector from the training data $\bar{\mathbf{g}}$;

$$\mathbf{g}^* = \bar{\mathbf{g}} + \mathbf{B}\boldsymbol{z}^* \tag{A.7}$$



(a) $P = 2$ (b) $P = 5$ (c) $P = 40$

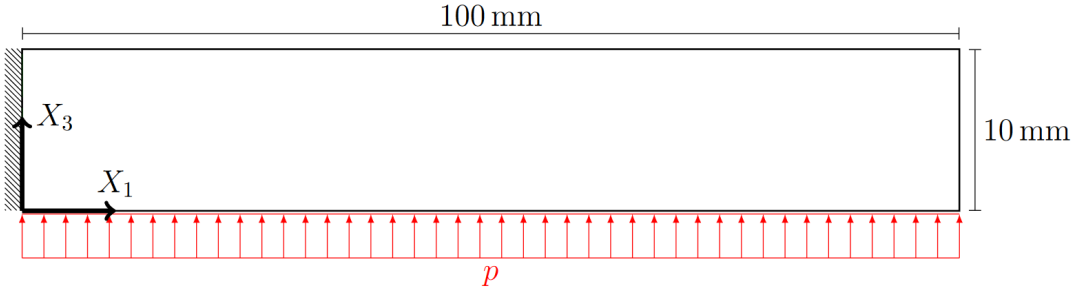
Figure A.3: An example LV geometry plotted against its PCA reconstruction. The red surface shows the ground truth, while the blue surfaces show the geometry reconstructed from an P dimensional latent PC space, for three different values of P.

creating a synthetic LV geometry $\mathbf{g}^*$. We used a $P = 40$-dimensional multivariate Gaussian distribution to generate the random samples. The variances of each dimension were found as the variances from the training data, increased by 10%, while the cross-covariance terms and mean values of the distribution were set to zero. The variances were increased by 10% to produce greater variation in the sampled geometries than was observed in the real geometries, while still ensuring that the majority of the sampled geometries were valid. Note that the sampling procedure outlined above is not guaranteed to produce a valid LV geometry, as for example the endocardial and epicardial surfaces of the reconstructed geometry may intersect for some values of $\boldsymbol{z}^*$ in Eq. (A.7). We therefore discarded any synthetic geometries for which wall thickness, length and width did not fall within ranges similar to those seen in the real LV data. The two outer surfaces of the synthetic geometries were further processed into a three-layer, hexahedral mesh with 6784 nodes and 4995 finite-elements. Finally, a rule based method (RBM) [165] was adopted to describe the myofiber structure of each geometry, where the fibre angle varied linearly from $\alpha_{\text{endo}} = 60°$ at endocardium to $\alpha_{\text{epi}} = -60°$ at epicardium. 3000 synthetic LV geometries were generated in this manner, a similar order to the number of real LV geometries available at UK Biobank [242].

B Appendix for Chapter 3

B.1 Effect of learning rate on emulator training

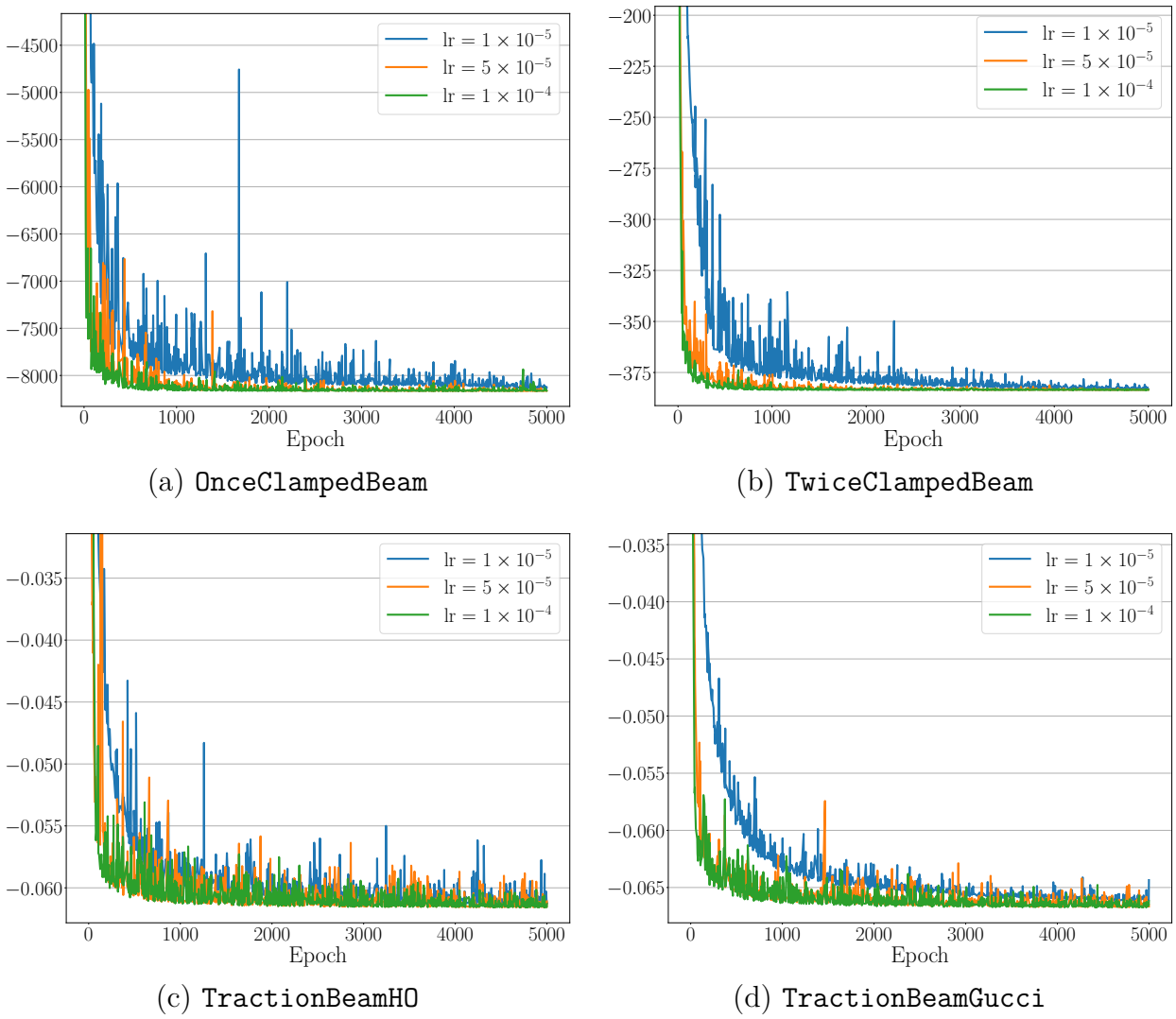
We use four beam models to explore the effect that learning rate has on emulator training. The first two models, `OnceClampedBeam` and `TwiceClampedBeam` were also considered in Section refsec:ddComparison for the data driven and physics informed training comparison. For the third model `TractionBeamHO`, we have $\Omega_0 = [0, 100] \times [0, 10]^2$ (mm), represented using 176 nodes and 440 elements. The beam is clamped at the left most end $\partial\Omega_0^d = \{\boldsymbol{X} \in \Omega_0 : X_1 = 0\}$, and subject to a pressure of 0.15 kPa on its bottom surface $\partial\Omega_0^\sigma = \{\boldsymbol{X} \in \Omega_0 : X_3 = 0\}$, with no body force applied. The H-O constitutive law is used (see Eq. (3.3)), with $\boldsymbol{\theta} \in [.75, 2.5]^4$ where a and a_f are in kPa, and a fixed myofibre field with $\boldsymbol{f}_0 = (1, 0, 0)^\top$ is assumed throughout Ω_0. The setup of the final beam model, `TractionBeamGucci` is identical with the exception that the Guccione material model is used (see Eq. (3.6)), with $\boldsymbol{\theta} \in [1.75, 3.5]^4$. The reference configuration and boundary conditions for both traction-force models is shown in Figure B.1.



`TractionBeamHO` and `TractionBeamGucci`

Figure B.1: Illustration of the `TractionBeamHO` and `TractionBeamGucci` models as 2-D slices in the (X_1, X_3) plane (not to scale). The dashed lines indicate a clamped Dirichlet boundary, and p represents a pressure applied to a Neumann boundary surface.

For each model, training was performed over 200 randomly sampled input material parameter values. An additional 100 independent simulations were used as a test set for evaluation of out of sample performance. Three different fixed learning rates were considered; 1×10^{-5}, 5×10^{-5} and 1×10^{-4}. 5000 epochs were used for training, which was a sufficient number to make comparisons between the different rates. Learning curves for the four beam models are displayed in Figure B.2, with the mean potential energy

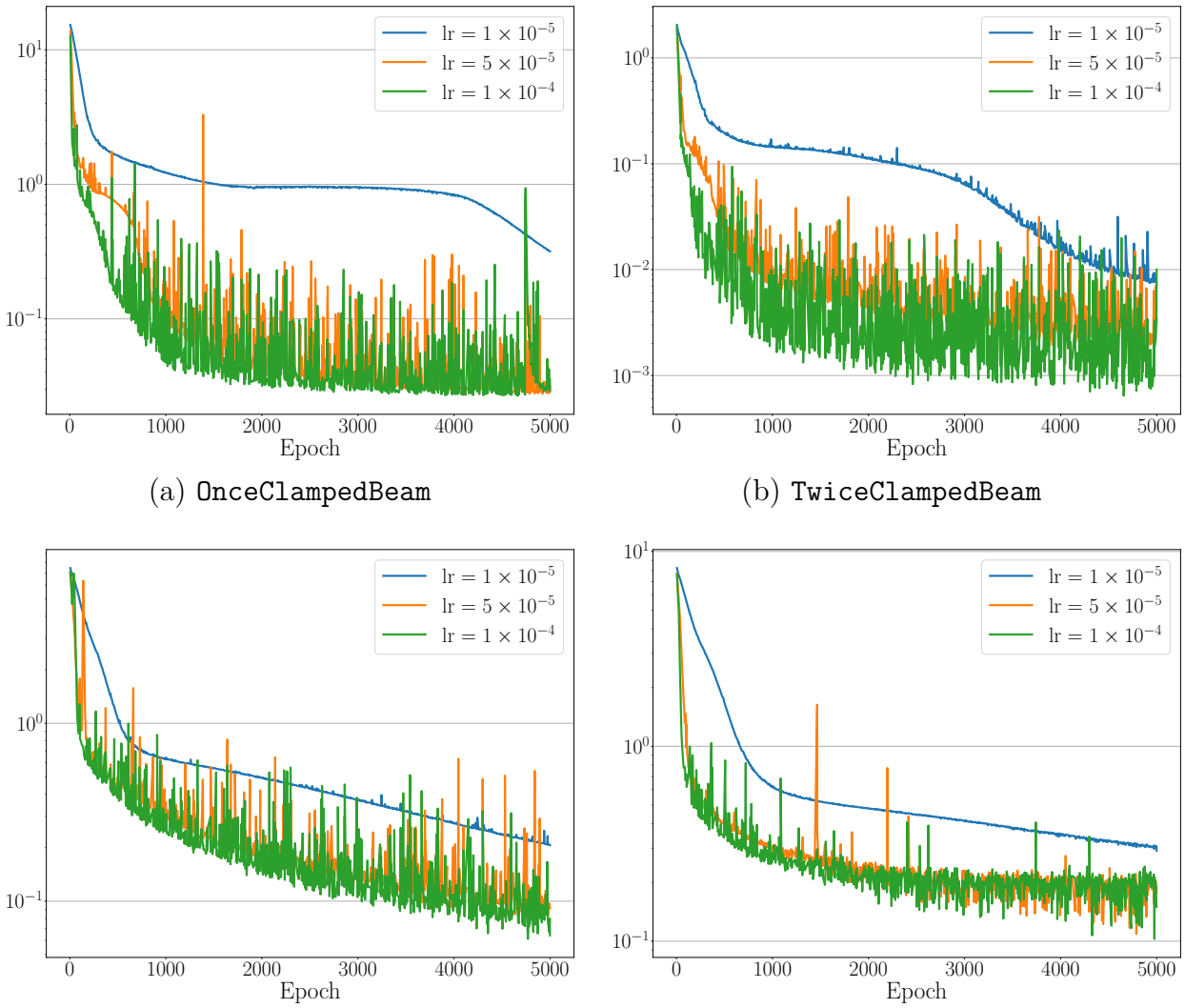


(a) `OnceClampedBeam` (b) `TwiceClampedBeam`

(c) `TractionBeamHO` (d) `TractionBeamGucci`

Figure B.2: Traceplots of Mean(Π) for four beam models using different learning rates.

value obtained at the 200 training input configurations plotted against number of training epochs performed. For all models, the higher learning rate of 1×10^{-4} allows the emulator to reach the minimum of the potential energy most rapidly. Training diverged in each case when a higher learning rate of 5×10^{-4} was used(not shown in figure to avoid clutter). Figure B.3 again displays learning curves, except here the mean $err_{\boldsymbol{u}}$ value across the 100 test simulations is plotted against number of epochs. A similar pattern is observed, whereby a high learning rate yields faster training. Significant variation in the $err_{\boldsymbol{u}}$ traceplots are present however when using a higher learning rate, suggesting that lowering its value at the end of training may be useful to settle on the optimal displacement values. For this reason we used a two-phase training approach for the emulation experiments in Sections 3.3.2-3.3.4 whereby a learning rate of 1×10^{-4} was used for the first half of training, before this was reduced by one order of magnitude for the second half of



(a) `OnceClampedBeam` (b) `TwiceClampedBeam`

(c) `TractionBeamHO` (d) `TractionBeamGucci`

Figure B.3: Traceplots of Mean($err_{\boldsymbol{u}}$) for four beam models using different learning rates.

training. We found that this simple strategy was highly effective at finding the minimum total potential energy state, and hence we did not pursue more complex approaches in this work, such as second order methods. Note that we also initially experimented with an automatic learning rate finder approach similar to [243] and different decay schedules, however we consistently found that using a fixed learning rate of 1×10^{-4} was optimal during the early stages of training.

B.2 Additional data-driven and physics-informed emulation experiments

The results from Section 3.3.1 illustrate that while data-driven can achieve strong accuracy in terms of displacements, the results obtained do not approximate the true potential energy state to the same accuracy as with physics-informed training, which is reflected in greater errors in the deformation gradient $\boldsymbol{F}$ and first invariant I_1. A possible way of alleviating this issue would be to train a data-driven emulator against both displacement and deformation gradient data. To test this approach, we re-performed the data-driven emulation experiments by making use of the following loss function

$$\boldsymbol{\omega}^* = \underset{\boldsymbol{\omega}}{\text{argmin}} \text{ Mean} \left[\sum_{j=1}^{N_{\text{sim}}} \sum_{i=1}^{N_{\text{node}}} \|\boldsymbol{u}_{ji} - \hat{\boldsymbol{u}}_{ji}(\boldsymbol{\omega})\|_2 + \sum_{j=1}^{N_{\text{sim}}} \sum_{k=1}^{N_{\text{elem}}} \|\boldsymbol{F}_{jk} - \hat{\boldsymbol{F}}_{jk}(\boldsymbol{\omega})\|_F \right], \quad \text{(B.1)}$$

which includes a penalty term for in terms of the Frobenius norm $\|cdot\|_F$. We refer to a GNN trained using this loss function as an "FDD-GNN". Figures B.4 and B.5 then replicate the corresponding figures from Section 3.3.1 for both the `OnceClampedBeam` and `TwiceClampedBeam` models, except this time the FDD-GNN results are also included. For each model, we see that including a penalty term on $\boldsymbol{F}$ in the data-driven loss function leads to improved emulation accuracy, particularly in terms of the err_{Π}, $err_{\boldsymbol{F}}$ and err_{I_1}, but nevertheless the PI-GNN still achieves better accuracy on these three error metrics.

B.3 Comparison of Neo-Hookean and Holzapfel-Ogden material models

The higher level of accuracy obtained for the `Liver` model in Section 3.3.3, which used the Neo-Hookean law, compared to the `LeftVentricle` emulation results in Section 3.3.4, where the Holzapfel-Ogden law was assumed, suggests that the PI-GNN is easier to train when the underlying constitutive relationship is linear or near-linear. For further insight here, in this section we re-perform the `LeftVentricle` emulation experiments using the Neo-Hookean material model. Specifically, was trained over material parameter space and LV pressure profiles between 4 and 10 mmHg, with Neo-Hookean constitutive law parameterised by Lame material parameters λ and μ, where both parameters varied in

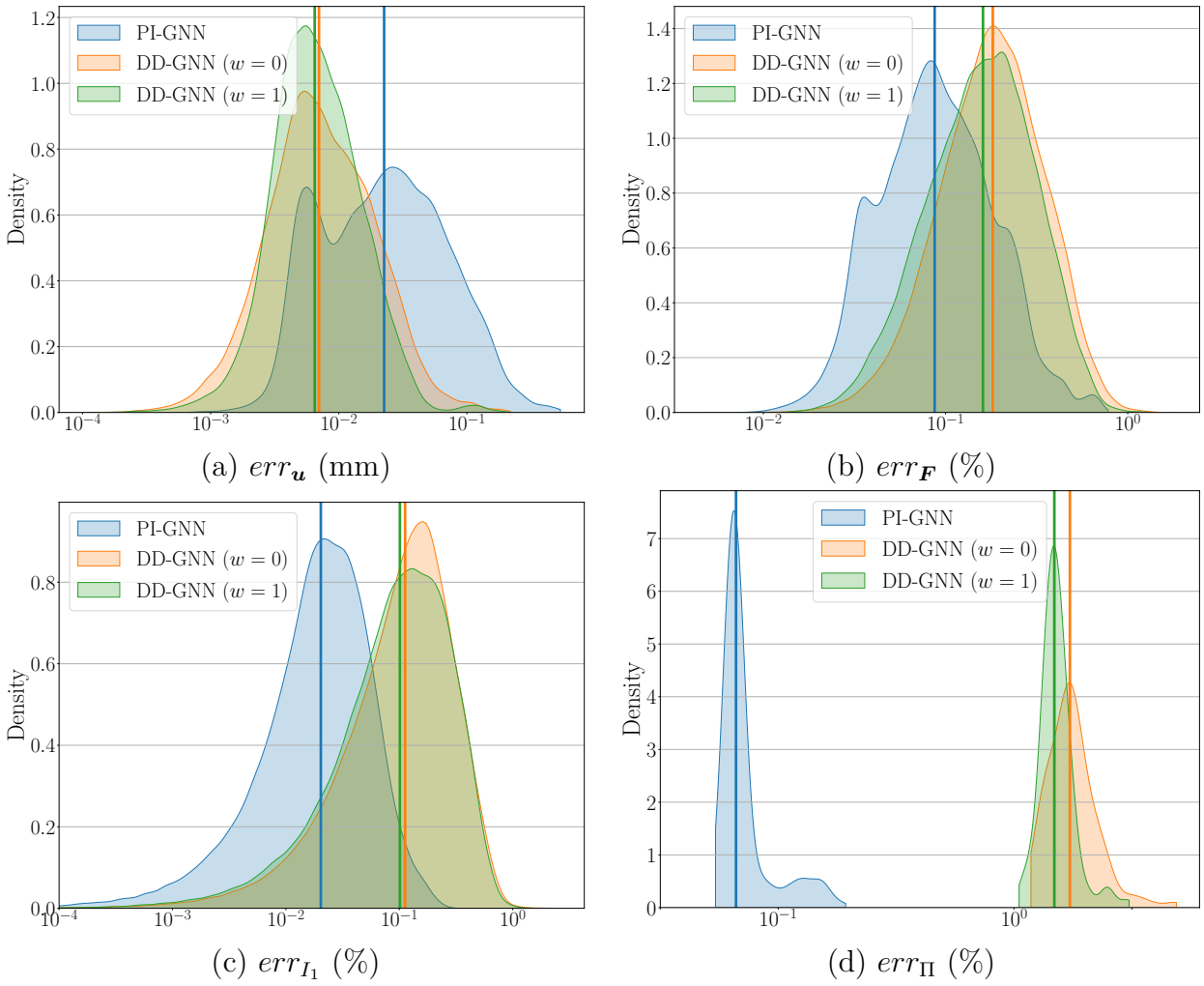


Figure B.4: Comparison of data-driven and physics-informed emulation results on `OnceClampedBeam` model. The PI-GNN is trained by optimisation of Eq. (3.12), the DD-GNN is trained purely on displacement data as in Eq. (3.11), while the FDD-GNN additionally incorporates a loss on the deformation gradient $\boldsymbol{F}$ following Eq. (B.1).

the range $[8, 10]$ kPa. Emulator performance of the surrogate was evaluated on a test set of 150 simulations obtained using the finite-element method (FEM). Figure B.6 displays error density plots for this emulator, compared with results from Section 3.3.4 for the original PI-GNN which made use of the H-O law. In all cases, we see noticeably better performance under the Neo-Hookean model. For relative errors in u seen in panel (a), median errors under the Neo-Hookean model are almost one order of magnitude lower than for the H-O model. For err_{I_1} displayed in panel (b), there is even greater disparity between the results, with very little overlap between the two densities. If we consider err_{Π} in panel (c), we see that the PI-GNN consistently achieves a better approximation to the true minimum potential energy value, indicating it is easier to train the emulator when the material model is near linear. Finally, for err_V in panel (d), the difference between the

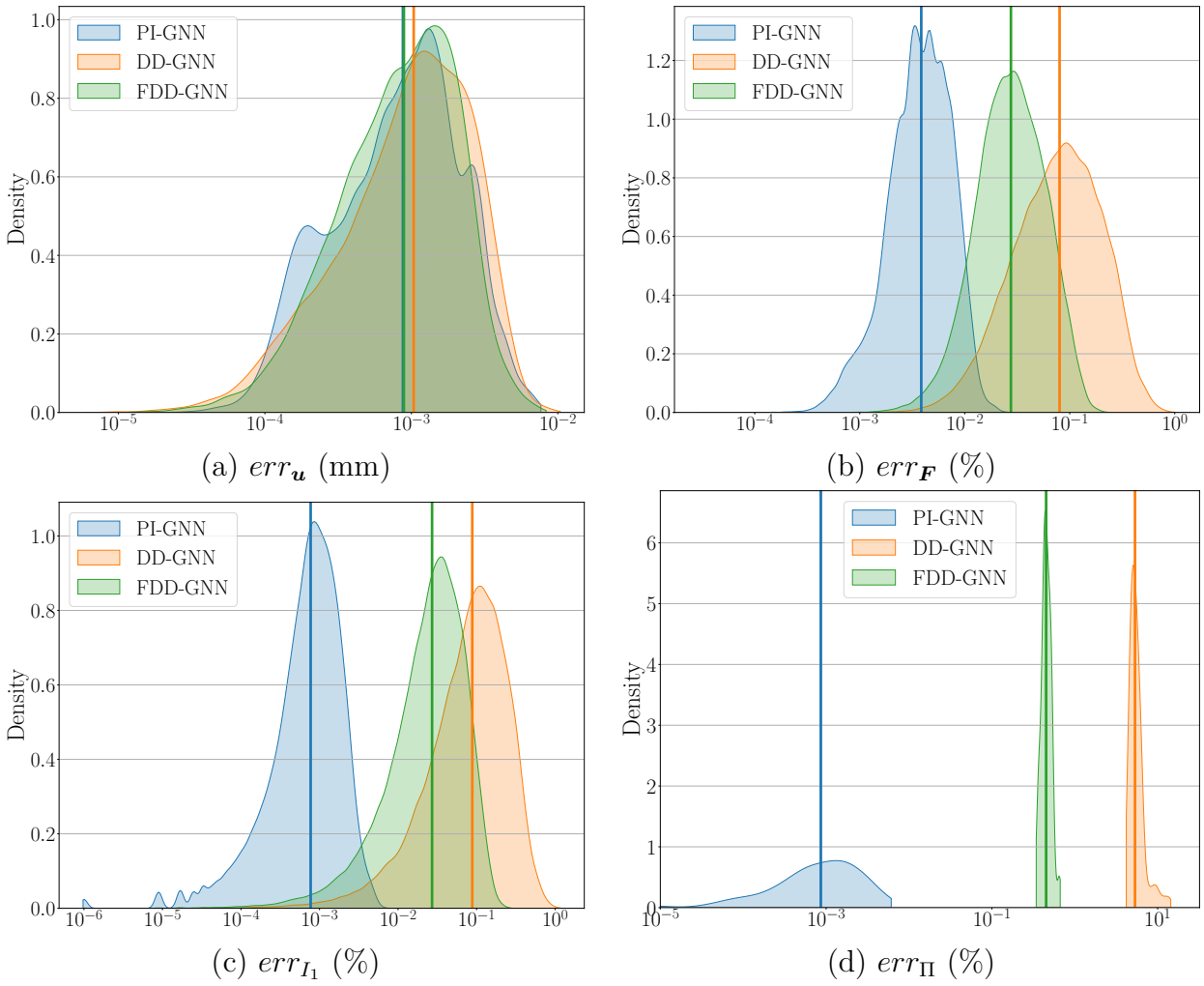


Figure B.5: Comparison of data-driven and physics-informed emulation results on `TwiceClampedBeam` model. The PI-GNN is trained by optimisation of Eq. (3.12), the DD-GNN is trained purely on displacement data as in Eq. (3.11), while the FDD-GNN additionally incorporates a loss on the deformation gradient $\boldsymbol{F}$ following Eq. (B.1).

two results is less pronounced. While lower errors are achieved under the Neo-Hookean model, there is a high degree of overlap between the two distributions. This results suggest that further experimentation could be required here to determine how PI-GNN accuracy can be increased for more non-linear material models. One possible approach could be the use of feature transformations of the material parameter vector $\boldsymbol{\theta}$.

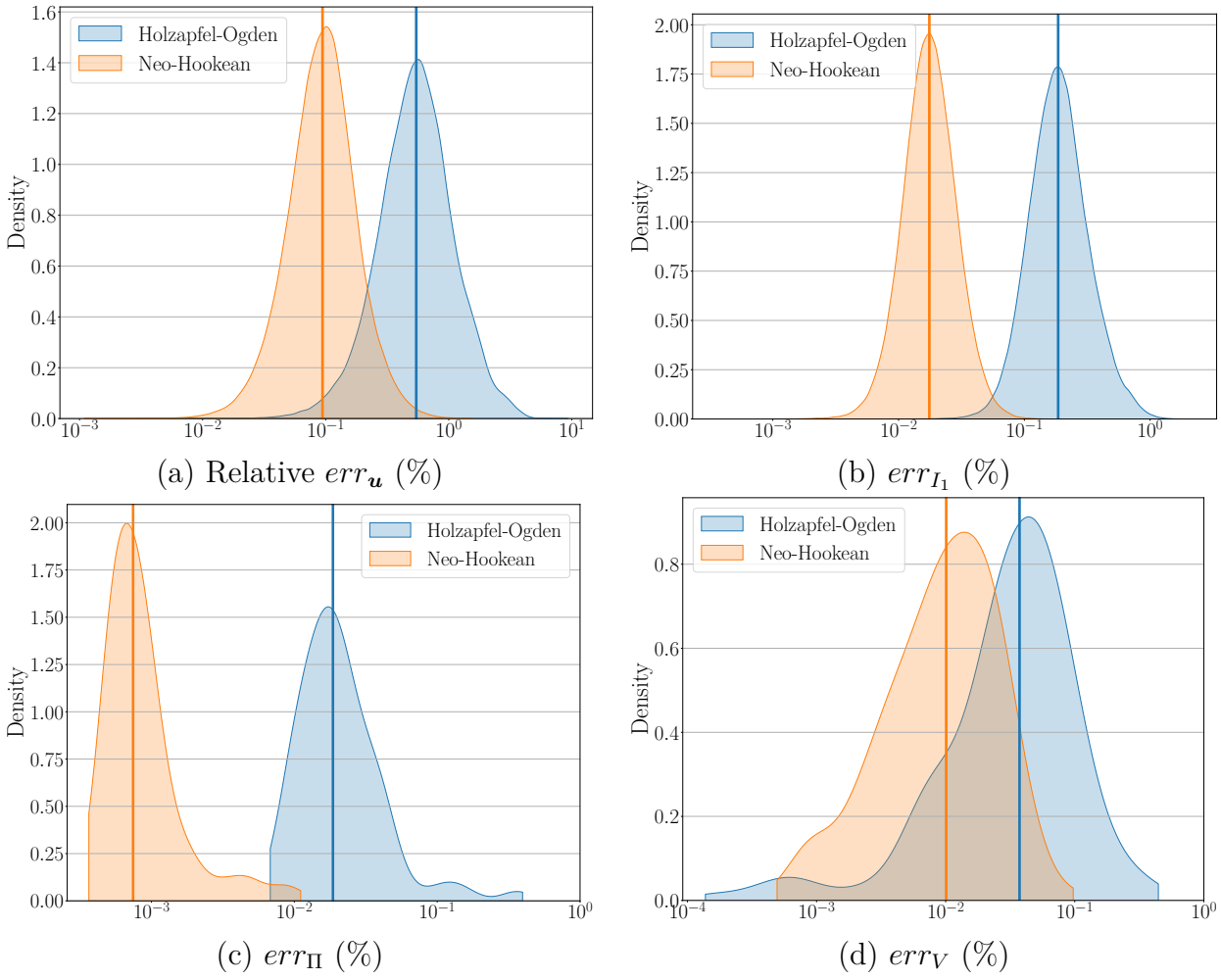
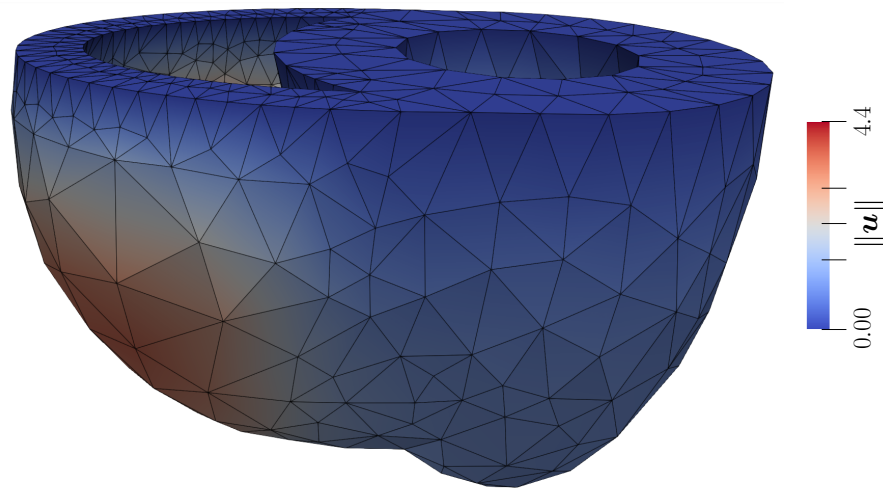


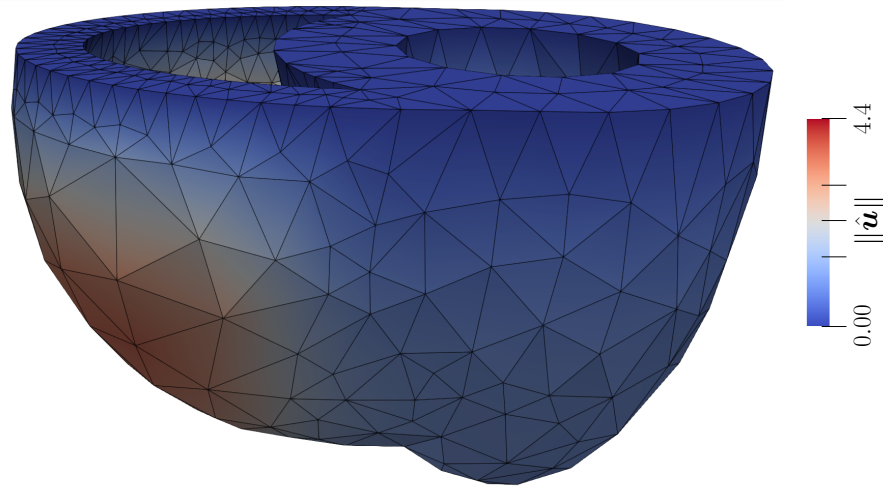
Figure B.6: Comparison of emulation results for `LeftVentricle` model under the Neo-Hookean and Holzapfel-Ogden material models.

B.4 Application to biventricle cardiac geometry

Here we demonstrate the application of a PI-GNN to the filling of a biventricular cardiac geometry. For this illustrative example, idealised symmetric ventricles are considered, the Neo-Hookean material model is assumed and the same cardiac pressure is used in each cavity. A PI-GNN emulator was trained for 1000 steps. Prediction results against an FEM simulation are shown in Figure B.7, where the mean $err_{\boldsymbol{u}}$ value is 4.8×10^{-2} mm.. Visually we see strong agreement, however relative errors are higher than seen in the `LeftVentricle` model, which may be alleviated by more training steps or a larger number of message passing steps.



(a) Current Configuration (FEM)



(b) Current Configuration (PI-GNN)

Figure B.7: Comparison of simulation and emulation results for biventricular cardiac geometry.

B.5 Emulation on new LV geometry

The geometry of a soft-tissue body will vary from patient to patient, therefore it is essential that the emulator can generalise to a new geometry not seen in the training phase. Note we have already addressed this in earlier work [9], where a GNN was trained on a dataset where each simulation was performed using a different left ventricle geometry. In this work, we take a simpler approach by considering the generalisation of the PI-GNN for the `LeftVentricle` model in the case of one new LV geometry, again extracted from cardiac imaging scans. Note that this geometry had a different mesh topology (1268 nodes and 4847 elements) to the original LV (1570 nodes and 6176 elements).

To test how the GNN can generalise to the new geometry, we took the pre-trained PI-GNN from the original LV, and transfer learned for 12 epochs or approximately 90 seconds on the new geometry (we refer to this as the "Transfer Learned" emulator). For baseline comparison, we consider a PI-GNN trained for the same computational budget (i.e. 12 steps / 90 seconds), but from randomly initialised network parameters (we refer to this as the "Baseline" emulator). Comparing the transfer learned results with the baseline allows the gain achieved by sharing information from the original geometry to be quantified. Finally, as reference we consider the results we the performance of a PI-GNN again trained from scratch, but this time for 15000 epochs (we refer to this as the "Reference" emulator). After training, emulation performance of the three emulators was then evaluated on a test set of 150 simulations run using the same material model (Holzapfel-Ogden) and material parameter domain as the original LV geometry, with fixed cardiac pressure of 6 mmHg.

Figure B.8 shows density plots of test set emulation errors for the three emulators considered. In each case, we see a clear gain in performance when re-using the network weights over the Baseline PI-GNN with randomly initialised weights. With $err_{\boldsymbol{u}}$ for instance, the increase in accuracy is approximately one order of magnitude. The accuracy of the full-trained Reference emulator is then one order of magnitude lower again. Performing over one thousand times as many training steps has allowed the Reference emulator to obtain clearly more accurate results.

Panel (c) shows the error in total potential energy Π incurred by the emulator, which illustrate allowed the fully-trained Reference PI-GNN to reach a better approximation of the true minimum potential energy state than the two PI-GNNs that were trained for 12 steps. However, the transfer learned emulator has clearly benefited from the initialisation at the weights found on the other LV geometry, as the worst err_{Π} is lower seen in the transfer-learning case is lower than the smallest error seen with the Baseline, randomly initialised PI-GNN. A similar pattern is observed in panel (d), which displays density plots of err_V. Here however the transfer-learned emulator more closely matches the performance of the fully-trained Reference emulator. While lower errors are attained in general with full training, there is significant overlap between the two error distributions, and for the transfer-learned PI-GNN, only a small fraction of errors exceed 1%.

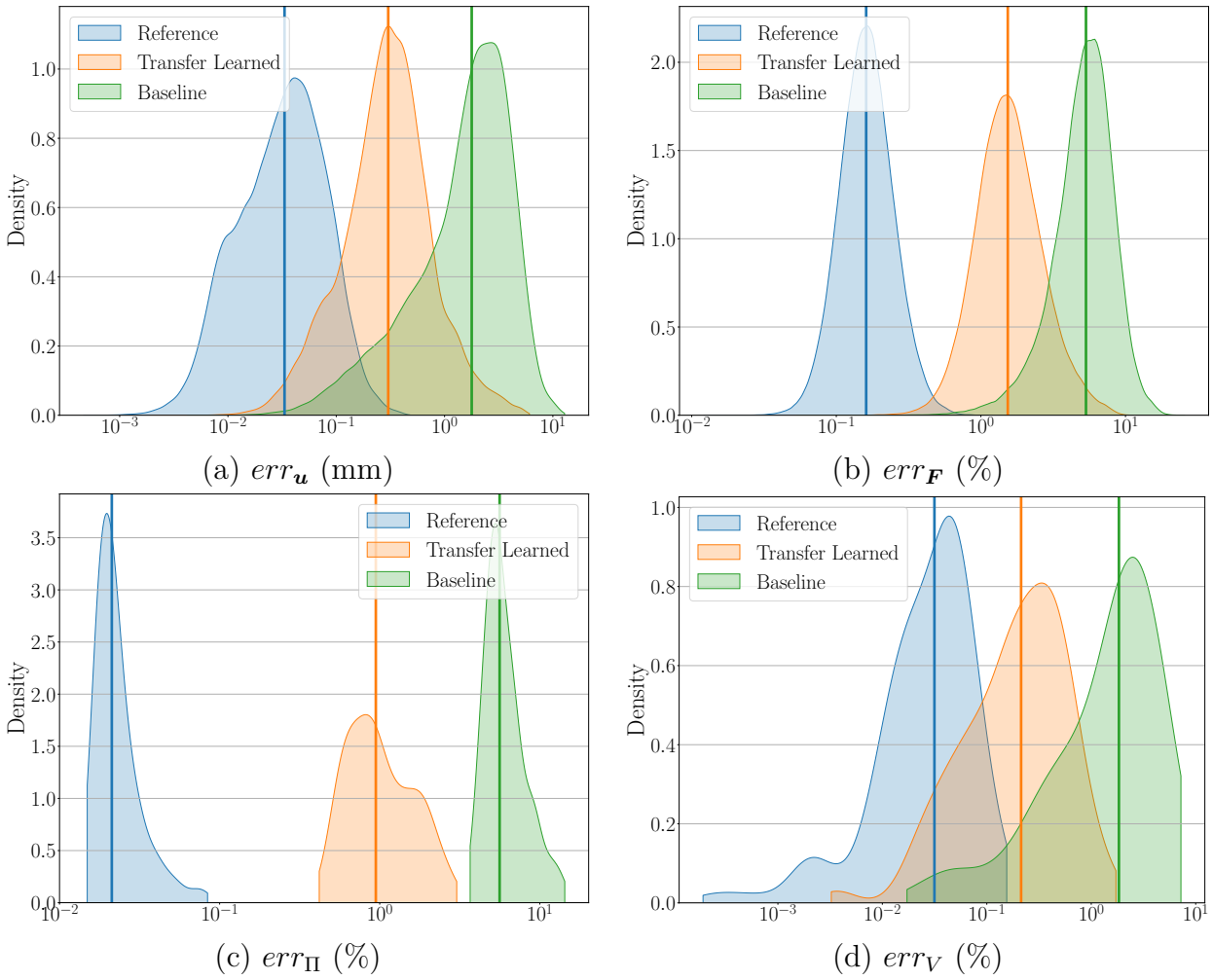
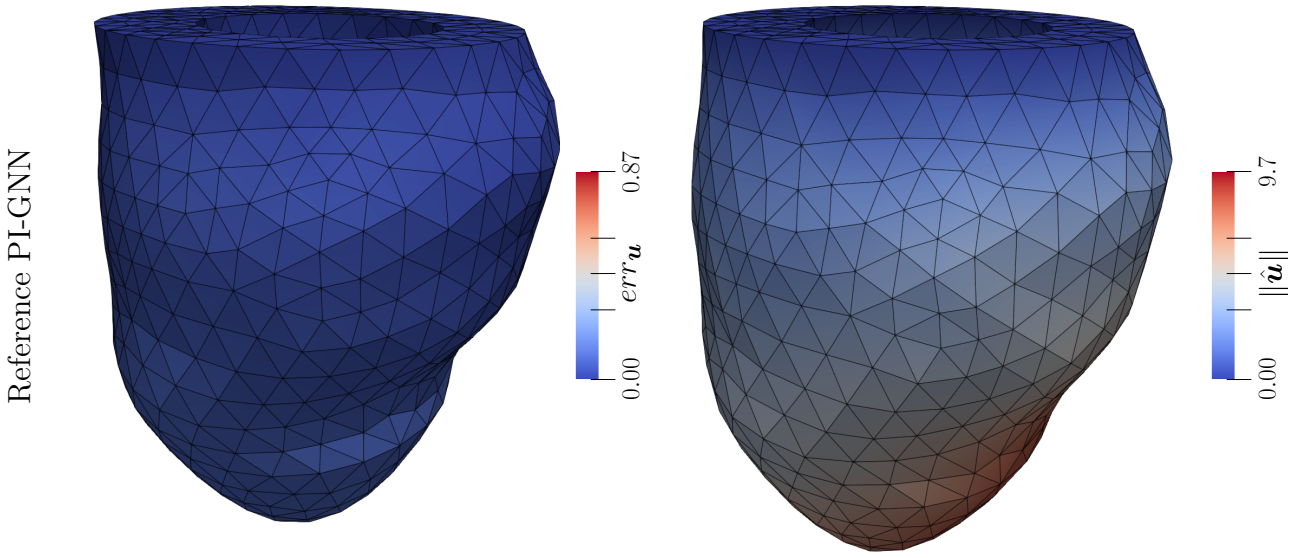
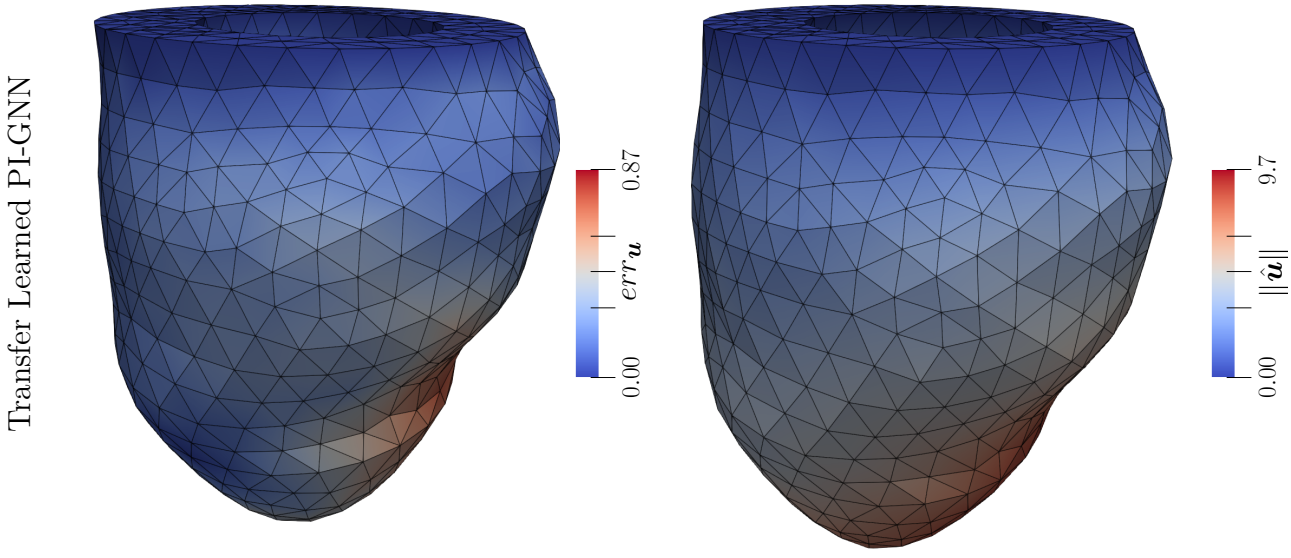


Figure B.8: Comparison of emulation results for new LV geometry for GNN trained from scratch for 15000 epochs (Reference), versus a transfer-learned GNN trained for 12 epochs (Transfer Learned) and a randomly initialised GNN trained for 12 epochs (Baseline).

Figure B.9 displays emulation plots for the new geometry for the simulation which gave median out of sample error. In the left column, we see that the errors incurred by the transfer learned PI-GNN are consistently higher across the surface of the geometry than for the Reference PI-GNN. In the right column, differences between the two are only only slightly visually apparent, and the err_V values were very similar for each surrogate - for the Reference PI-GNN, err_V error was 0.03%, while for the transfer-learned PI-GNN, the corresponding error was 0.04%.



(a) $err_{\boldsymbol{u}} = \|\boldsymbol{u} - \hat{\boldsymbol{u}}\|_2$ (d) Current Configuration (PI-GNN)



(c) $err_{\boldsymbol{u}} = \|\boldsymbol{u} - \hat{\boldsymbol{u}}\|_2$ (d) Current Configuration (PI-GNN)

Figure B.9: Median out of sample emulation results for `LeftVentricle` model (mm) using new geometry. Top row shows a fully trained PI-GNN, bottom row shows a PI-GNN transfer-learned for 90 seconds.

These results suggest that for macro level quantities of interest such as cardiac volume, where precise node-wise accuracy levels, the PI-GNN exhibits reasonable generalisation performance when applied to a new geometry. It is likely that more accurate results can be obtained by incorporating a larger number of different geometries into the training routine as done in [9], which is the direction we will pursue in future work.

C Appendix for Chapter 4

C.1 Proof of Lemma 4.4.1

Proof. We prove that for all $i = 1, \ldots, D$, $\tilde{m}_i(\boldsymbol{x})$ exactly satisfies boundary conditions on $\bigcup_{l=0}^{i} \bigcup_{j=0}^{1} \partial\Omega_{lj}$. Since $\tilde{m}(\boldsymbol{x}) \triangleq \tilde{m}_D(\boldsymbol{x})$, this is sufficient to prove the claim.

We proceed by induction. Firstly, note by construction of $\tilde{m}_1$ that $\tilde{m}_1(\boldsymbol{x} : x_1 = 0) = b(\boldsymbol{x} : x_1 = 0)$ and $\tilde{m}_1(\boldsymbol{x} : x_1 = 1) = b(\boldsymbol{x} : x_1 = 1)$, i.e $\tilde{m}_1(\boldsymbol{x})$ satisfies the specified boundary conditions on $\partial\Omega_{10}$ and $\partial\Omega_{11}$.

Next, assume that $\tilde{m}_{i-1}(\boldsymbol{x})$ exactly satisfies boundary conditions on $\bigcup_{l=0}^{i-1} \bigcup_{j=0}^{1} \partial\Omega_{lj}$ for $i = 2, \ldots, D$. Consider the output of $\tilde{m}_i(\boldsymbol{x})$ with $x_i = j$ for $j \in \{0, 1\}$, that is, $\tilde{m}_i(\boldsymbol{x} : x_i = j)$. We have

$$\tilde{m}_i(\boldsymbol{x} : x_i = j) = \tilde{m}_{i-1}(\boldsymbol{x} : x_i = j) + \tilde{b}_{i,0}(\boldsymbol{x} : x_i = j)(1 - j) + \tilde{b}_{i,1}(\boldsymbol{x} : x_i = j)j \tag{C.1}$$

$$= \tilde{m}_{i-1}(\boldsymbol{x} : x_i = j) + \tilde{b}_{i,j}(\boldsymbol{x} : x_i = j) \tag{C.2}$$

$$= \tilde{m}_{i-1}(\boldsymbol{x} : x_i = j) + b(\boldsymbol{x} : x_i = j) - \tilde{m}_{i-1}(\boldsymbol{x} : x_i = j) \tag{C.3}$$

$$= b(\boldsymbol{x} : x_i = j). \tag{C.4}$$

That is, $\tilde{m}_i$ satisfies the boundary conditions on $\partial\Omega_{i0}$ and $\partial\Omega_{i1}$.

Now, we show that $\tilde{m}_i$ satisfies the boundary conditions on $\partial\Omega_{ij}$ for $l < i$. Since $\tilde{m}_{i-1}$ satisfies these conditions by assumption, by construction of $\tilde{m}_i$ we simply need to prove that for all $l < i$ we have $\tilde{b}_{i,j}(\boldsymbol{x} : x_l = q) = 0$ for $j, q \in \{0, 1\}$. To see this, let $l < i$ be arbitrary. We then have

$$\tilde{b}_{i,j}(\boldsymbol{x} : x_l = q) = b(\boldsymbol{x} : x_i = j, x_l = q) - \tilde{m}_{i-1}(\boldsymbol{x} : x_i = j, x_l = q) \tag{C.5}$$

$$= 0. \tag{C.6}$$

The above equality holds because $\tilde{m}_{i-1}$ equals b where $x_l = q \in \{0,1\}$ by assumption. Therefore, $\tilde{m}_i(\boldsymbol{x}) = \tilde{m}_{i-1}(\boldsymbol{x})$ where $\boldsymbol{x} \in \partial\Omega_{l0} \cup \partial\Omega_{l1}$, Since l was arbitrary, this holds for all $l < i$, i.e. boundary conditions are satisfied for all $\bigcup_{l=0}^{i-1}\bigcup_{j=0}^{1}\partial\Omega_{lj}$. Combined with the above result for $\partial\Omega_{i0} \cup \partial\Omega_{i1}$, we conclude by induction that $\tilde{m}$ satisfies boundary conditions on all of $\partial\Omega$.

□

C.2 Proof of Theorem 4.5.3

We begin by introducing an alternative definition of a reproducing kernel in terms of a *feature map* [225, Definition 4.1].

Definition C.1. *Let $\mathcal{X}$ be a non-empty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if there exists an $\mathbb{R}$-Hilbert space $\mathcal{H}_0$ and a map $\Phi_0 : \mathcal{X} \to \mathcal{H}_0$ such that, for all $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$, we have*

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi_0(\boldsymbol{x}), \Phi_0(\boldsymbol{x}') \rangle_{\mathcal{H}_0} . \tag{C.7}$$

We call Φ_0 a feature map of k, and $\mathcal{H}_0$ a feature space.

There are a number of equivalent ways of representing an RKHS. One way that is particularly useful when considering the density of an RKHS is the feature map representation [225, Theorem 4.21].

Theorem C.1 (Feature map representation of RKHS). *Let $\mathcal{X}$ be a non-empty set and k a kernel on $\mathcal{X} \times \mathcal{X}$ with feature map $\Phi_0 : \mathcal{X} \to \mathcal{H}_0$ where $\mathcal{H}_0$ is a Hilbert space. Consider the normed space*

$$\mathcal{H}_{\Phi_0} \triangleq \left\{ u : \mathcal{X} \to \mathbb{R} : \exists\, w \in \mathcal{H}_0 \text{ with } u(\boldsymbol{x}) = \langle w, \Phi_0(x) \rangle_{\mathcal{H}_0} \text{ for all } \boldsymbol{x} \in \mathcal{X} \right\} \tag{C.8}$$

where

$$\|u\|_{\mathcal{H}_{\Phi_0}} \triangleq \inf \left\{ \|w\|_{\mathcal{H}_0} : w \in \mathcal{H}_0 \text{ with } u = \langle w, \Phi_0(\cdot) \rangle_{\mathcal{H}_0} \right\} . \tag{C.9}$$

Then, $\mathcal{H}_k = \mathcal{H}_{\Phi_0}$.

We remark that kernels do not have unique feature maps, however the above construction is independent of the specific choice of Φ_0.

Lemma C.2. *Let $k : [0,1]\times[0,1] \to \mathbb{R}$ be a kernel with feature map $\Phi_0 : [0,1] \to \mathcal{H}_0$ and let $\tilde{k}$ and ϕ be as defined in Eq. (4.26). Finally, define $\tilde{\Phi}_0 : [0,1] \to \mathcal{H}_0$ as $\tilde{\Phi}_0(x) \triangleq \phi(x)\Phi_0(x)$. Then $\tilde{\Phi}_0$ is a feature map for $\tilde{k}$.*

Proof. First remark that $\phi(x) \in \mathbb{R}$ for all $x \in [0,1]$ and $\mathcal{H}_0$ is an $\mathbb{R}$-Hilbert space, and therefore $\phi(x)\Phi_0(x) \in \mathcal{H}_0$ for all x.

Now, let $x, x^{'} \in [0,1]$ be arbitrary. Then we have

$$\left\langle \tilde{\Phi}_0(x), \tilde{\Phi}_0(x') \right\rangle_{\mathcal{H}_0} = \langle \phi(x)\Phi_0(x), \phi(x')\Phi_0(x') \rangle_{\mathcal{H}_0} \tag{C.10}$$

$$= \phi(x)\phi(x') \langle \Phi_0(x), \Phi_0(x') \rangle_{\mathcal{H}_0} \tag{C.11}$$

$$= \phi(x)\phi(x')k(x,x') \tag{C.12}$$

$$= \tilde{k}(x,x'). \tag{C.13}$$

□

Lemma C.3. *Let $k : [0,1]\times[0,1] \to \mathbb{R}$ be a kernel, and $\tilde{k}$ and ϕ be as defined in Eq. (4.26). Then $\hat{z} \in \mathcal{H}_k$ implies $\hat{u} \in \mathcal{H}_{\tilde{k}}$ where $\hat{u}(x) = \phi(x)\hat{z}(x)$.*

Proof. Let $\hat{z} \in \mathcal{H}_k$ be arbitrary. By Theorem C.1, there exists feature map $\Phi_0 : [0,1] \to \mathcal{H}_0$ with $\mathcal{H}_0$ a Hilbert space so that

$$\hat{z}(x) = \langle w, \Phi_0(x) \rangle_{\mathcal{H}_0} \tag{C.14}$$

for all $x \in [0,1]$ for some $w \in \mathcal{H}_0$. Therefore, we have

$$\hat{u}(x) = \phi(x)\hat{z}(x) \tag{C.15}$$

$$= \phi(x) \langle w, \Phi_0(x) \rangle_{\mathcal{H}_0} \tag{C.16}$$

$$= \langle w, \phi(x)\Phi_0(x) \rangle_{\mathcal{H}_0} \tag{C.17}$$

$$= \left\langle w, \tilde{\Phi}_0(x) \right\rangle_{\mathcal{H}_0}. \tag{C.18}$$

for all $x \in [0,1]$. Recall by Lemma C.2 that $\tilde{\Phi}_0(x) = \phi(x)\Phi_0(x)$ is a feature map for $\tilde{k}$, which means $\hat{u} \in \mathcal{H}_{\tilde{\Phi}_0} = \mathcal{H}_{\tilde{k}}$ by Theorem C.1. Since $\hat{z}$ was chosen arbitrarily, this holds for all $\hat{z} \in \mathcal{H}_k$. □

Lemma C.4. *Let $u \in \mathcal{H}_{bc}$ be arbitrary, where $\mathcal{H}_{bc}$ is defined as in Eq. (4.25). Given corresponding z_u as in Eq. (4.27) and ϕ as in Eq. (4.26), then*

$$\lim_{x\to 0^+} \phi(x)z_u(x) = \lim_{x\to 1^-} \phi(x)z_u(x) = 0. \tag{C.19}$$

Proof. Let $u \in \mathcal{H}_{bc}$ be arbitrary. Note by construction of its associate latent function z_u that we can express u on the interior of its domain as

$$u(x) = \phi(x)z_u(x) \quad \text{for } x \in (0,1). \tag{C.20}$$

Since $u \in \mathcal{H}_{bc}$, u is continuous and therefore

$$u(0) = \lim_{x\to 0^+} u(x) \tag{C.21}$$

$$= \lim_{x\to 0^+} \phi(x)z_u(x) \tag{C.22}$$

Since $u(0) = 0$ by assumption (see Eq. (4.25)), we have $\lim_{x\to 0^+} \phi(x)z_u(x) = 0$. By the exact same line of reasoning, we have $\lim_{x\to 1^-} \phi(x)z_u(x) = 0$. □

We can now consider Theorem 4.5.3.

Proof. Let both $u \in \mathcal{H}_{bc}$ and $\varepsilon > 0$ be arbitrary. We need to find some $\hat{u} \in \mathcal{H}_{\tilde{k}}$ such that $\|u - \hat{u}\|_\infty \leq \varepsilon$.

To begin, note that Lemma C.4 implies the existence of $\delta_0, \delta_1 \in (0, \frac{1}{2})$ such that

$$|\phi(x)z_u(x)| < \frac{\varepsilon}{3} \text{ for all } x \in (0, \delta_0] \text{ and } |\phi(x)z_u(x)| < \frac{\varepsilon}{3} \text{ for all } x \in [1-\delta_1, 1). \tag{C.23}$$

Let $\delta = \min(\delta_0, \delta_1)$. Now consider the function $\tilde{z}_u : [0,1] \to \mathbb{R}$ given by

$$\tilde{z}_u(x) = \begin{cases} z_u(\delta), & \text{if } x \in [0, \delta) \\ z_u(x), & \text{if } x \in [\delta, 1-\delta] \\ z_u(1-\delta), & \text{if } x \in (1-\delta, 0] \end{cases} \tag{C.24}$$

where z_u is the latent function of u, as defined as in Eq. (4.27). Clearly, $\tilde{z}_u \in C([0,1])$. Since k is a universal kernel by assumption, there exists $\hat{z} \in \mathcal{H}_k$ such that

$$\|\tilde{z}_u - \hat{z}\|_\infty \leq \frac{\varepsilon}{3}. \tag{C.25}$$

Consider now the function $\hat{u} : [0,1] \to \mathbb{R}$ defined as

$$\hat{u}(x) = \phi(x)\hat{z}(x). \tag{C.26}$$

Note that $\hat{u} \in \mathcal{H}_{\tilde{k}}$ by Lemma C.3. We claim that $\|u - \hat{u}\|_\infty \leq \varepsilon$. We prove this by partitioning the domain into four subdomains and considering each case in turn.

Case I: $x \in \{0,1\}$. Clearly $\hat{u}(x) = 0$ in this case since $\phi(0) = \phi(1) = 0$, which is exactly the value that $u(x)$ takes at the boundary points by assumption. Therefore, $\hat{u}(x) = u(x)$ for $x \in \{0,1\}$.

Case II: $x \in [\delta, 1-\delta]$. By Eq. (C.25) we know

$$|\tilde{z}_u(x) - \hat{z}(x)| < \frac{\varepsilon}{3}. \tag{C.27}$$

Since $x \in [\delta, 1-\delta]$, we have that $\tilde{z}_u(x) = z_u(x)$ by construction (see Eq. (C.24)). Therefore

$$|z_u(x) - \hat{z}(x)| < \frac{\varepsilon}{3}. \tag{C.28}$$

Since $\phi(x) \in (0, \frac{1}{2}]$ if $x \in (0,1)$, we furthermore have

$$\phi(x)\,|z_u(x) - \hat{z}(x)| \;<\; \phi(x)\frac{\varepsilon}{3} \tag{C.29}$$

$$\Rightarrow |\phi(x)z_u(x) - \phi(x)\hat{z}(x)| \;<\; \frac{\varepsilon}{3}. \tag{C.30}$$

Finally, note that $u(x) = \phi(x)z_u(x)$ for $x \in (0,1) \supset [\delta, 1-\delta]$ by construction of z_u (see Definition 4.5.4) and $\hat{u}(x) = \phi(x)\hat{z}(x)$ by definition (see Eq. (C.26)). Therefore

$$|u(x) - \hat{u}(x)| < \frac{\varepsilon}{3}. \tag{C.31}$$

Case III: $x \in (0, \delta)$. Then again by Eq. (C.25) we know

$$|\tilde{z}_u(x) - \hat{z}(x)| < \frac{\varepsilon}{3}. \tag{C.32}$$

Since $x \in (0, \delta)$, we have that $\tilde{z}_u(x) = z_u(\delta)$ by construction (see Eq. (C.24)). Therefore

$$|z_u(\delta) - \hat{z}(x)| < \frac{\varepsilon}{3}. \tag{C.33}$$

This inequality can be expressed equivalently as

$$-\frac{\varepsilon}{3} < z_u(\delta) - \hat{z}(x) < \frac{\varepsilon}{3} \tag{C.34}$$

$$\Rightarrow\ -z_u(\delta) - \frac{\varepsilon}{3} < -\hat{z}(x) < -z_u(\delta) + \frac{\varepsilon}{3}. \tag{C.35}$$

Since inequalities are not affected by the addition of constants, the above implies

$$z_u(x) - z_u(\delta) - \frac{\varepsilon}{3} < z_u(x) - \hat{z}(x) < z_u(x) - z_u(\delta) + \frac{\varepsilon}{3}. \tag{C.36}$$

Furthermore, because $\phi(x) > 0$ for $x \in (0, 1) \supset (0, \delta)$, the above expression is valid when multiplied by $\phi(x)$:

$$\phi(x)z_u(x) - \phi(x)z_u(\delta) - \phi(x)\frac{\varepsilon}{3} < u(x) - \hat{u}(x) < \phi(x)z_u(x) - \phi(x)z_u(\delta) + \phi(x)\frac{\varepsilon}{3}, \tag{C.37}$$

where, as in Case II, we have made use of the fact that $u(x) = \phi(x)z_u(x)$ for $x \in (0, 1) \supset (0, \delta)$ and $\hat{u}(x) = \phi(x)\hat{z}(x)$.

Finally, note that by construction we have $|\phi(x)| \leq \frac{1}{2}$ for all $x \in [0, 1]$ (see Eq. (4.26)). Additionally, we have $|\phi(x)z_u(x)| < \frac{\varepsilon}{3}$ for $x \in (0, \delta]$ since $\delta = \min(\delta_0, \delta_1)$ (see Eq. (C.23)). Together with the inequality in Eq. (C.37), this implies

$$-\frac{\varepsilon}{3} - \frac{\varepsilon}{3} - \frac{\varepsilon}{3} < u(x) - \hat{u}(x) < \frac{\varepsilon}{3} + \frac{\varepsilon}{3} + \frac{\varepsilon}{3} \tag{C.38}$$

$$\Rightarrow\ |u(x) - \hat{u}(x)| < \varepsilon. \tag{C.39}$$

Case IV: $x \in (1-\delta, 1)$. Follows by the exact same reasoning as $x \in (0, \delta)$.

Therefore putting all cases together we have

$$\sup\{|u(x) - \hat{u}(x)| : x \in [0,1]\} = \|u - \hat{u}\|_\infty \leq \varepsilon. \tag{C.40}$$

□

C.3 Proof of Theorem 4.6.2

Proof. The proof here follows the proof of Theorem 4.6.1 from [222, Section 18.7.1]. We denote the trainable parameters of the latent fully connected neural network z_{nn} (Eq. (4.29)) used in the definition of the HCNN (Eq. (4.28)) as

$$\boldsymbol{\omega} = \{b^{(1)}, w_1^{(1)}, \ldots, w_H^{(1)}, b_1^{(0)}, \ldots, b_H^{(0)}, \boldsymbol{w}_1^{(0)}, \ldots, \boldsymbol{w}_H^{(0)}\}. \tag{C.41}$$

Let $\boldsymbol{x} \in \Omega$ be arbitrary. Then, the expected value of $u_{nn}(\boldsymbol{x})$ (see Eq. (4.28)) under the prior distributions from Eq. (4.30) is given by:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\omega}}\left[u_{nn}(\boldsymbol{x})\right] &= \mathbb{E}_{\boldsymbol{\omega}}\left[\tilde{m}(\boldsymbol{x}) + \phi(\boldsymbol{x}) z_{nn}(\boldsymbol{x})\right] && \text{(C.42)}\\
&= \mathbb{E}_{\boldsymbol{\omega}}\left[\tilde{m}(\boldsymbol{x}) + \phi(\boldsymbol{x})\left(b^{(1)} + \sum_{j=1}^{H} w_j^{(1)} h_j(\boldsymbol{x})\right)\right] && \text{(C.43)}\\
&= \mathbb{E}_{\boldsymbol{\omega}}\left[\tilde{m}(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{\omega}}\left[\phi(\boldsymbol{x})\left(b^{(1)} + \sum_{j=1}^{H} w_j^{(1)} h_j(\boldsymbol{x})\right)\right] && \text{(C.44)}\\
&= \tilde{m}(\boldsymbol{x}) + \phi(\boldsymbol{x})\mathbb{E}_{\boldsymbol{\omega}}\left[b^{(1)} + \sum_{j=1}^{H} w_j^{(1)} h_j(\boldsymbol{x})\right] && \text{(C.45)}\\
&= \tilde{m}(\boldsymbol{x}) + \phi(\boldsymbol{x})\left(\mathbb{E}_{\boldsymbol{\omega}}\left[b^{(1)}\right] + \sum_{j=1}^{H} \mathbb{E}_{\boldsymbol{\omega}}\left[w_j^{(1)} h_j(\boldsymbol{x})\right]\right) && \text{(C.46)}\\
&= \tilde{m}(\boldsymbol{x}) + \phi(\boldsymbol{x})\left(\underbrace{\mathbb{E}_{\boldsymbol{\omega}}\left[b^{(1)}\right]}_{=0} + \sum_{j=1}^{H} \underbrace{\mathbb{E}_{\boldsymbol{\omega}}\left[w_j^{(1)}\right]}_{=0} \mathbb{E}_{\boldsymbol{\omega}}\left[h_j(\boldsymbol{x})\right]\right) && \text{(C.47)}\\
&= \tilde{m}(\boldsymbol{x}). && \text{(C.48)}
\end{aligned}$$

We remark that $\mathbb{E}_{\boldsymbol{\omega}}\left[w_j^{(1)} h_j(\boldsymbol{x})\right] = \mathbb{E}_{\boldsymbol{\omega}}\left[w_j^{(1)}\right] \mathbb{E}_{\boldsymbol{\omega}}\left[h_j(\boldsymbol{x})\right]$ holds because $w_j^{(1)}$ is independent of $h_j(\boldsymbol{x})$ for all $j = 1, \ldots, H$ under our prior assumption for $\boldsymbol{\omega}$ (see Eq. (4.30)).

Now let $\boldsymbol{x}, \boldsymbol{x}' \in \Omega$ be arbitrary. Then by Eq. (4.28) and Eq. (4.30), the covariance between the corresponding outputs $u_{nn}(\boldsymbol{x})$ and $u_{nn}(\boldsymbol{x}')$ is given by

$$\begin{aligned}
\operatorname{Cov}\left(u_{nn}(\boldsymbol{x})u_{nn}(\boldsymbol{x}')\right) &= \mathbb{E}_{\boldsymbol{\omega}}\left[\left(u_{nn}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{\omega}}\left[u_{nn}(\boldsymbol{x})\right]\right)\left(u_{nn}(\boldsymbol{x}') - \mathbb{E}_{\boldsymbol{\omega}}\left[u_{nn}(\boldsymbol{x}')\right]\right)\right] && \text{(C.49)}\\
&= \mathbb{E}_{\boldsymbol{\omega}}\left[\phi(\boldsymbol{x})z_{nn}(\boldsymbol{x})\phi(\boldsymbol{x}')z_{nn}(\boldsymbol{x}')\right] && \text{(C.50)}\\
&= \phi(\boldsymbol{x})\phi(\boldsymbol{x}')\mathbb{E}_{\boldsymbol{\omega}}\left[z_{nn}(\boldsymbol{x})z_{nn}(\boldsymbol{x}')\right]. && \text{(C.51)}
\end{aligned}$$

We furthermore have that

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\omega}}\left[z_{nn}(\boldsymbol{x})z_{nn}(\boldsymbol{x}')\right] &= \mathbb{E}_{\boldsymbol{\omega}}\left[\left(b^{(1)} + \sum_{j=1}^{H} w_j^{(1)} h_j(\boldsymbol{x})\right)\left(b^{(1)} + \sum_{j=1}^{H} w_j^{(1)} h_j(\boldsymbol{x}')\right)\right] && \text{(C.52)}\\
&= \sigma_{b^{(1)}}^2 + \sum_{j=1}^{H} \frac{\omega}{H}\mathbb{E}_{\boldsymbol{\omega}}\left[h_j(\boldsymbol{x})h_j\left(\boldsymbol{x}'\right)\right], && \text{(C.53)}
\end{aligned}$$

where the final expression holds because $\mathbb{E}_{\boldsymbol{\omega}}\left[b^{(1)}\right] = 0$ and $\operatorname{Cov}\left(w_i^{(1)}, w_j^{(1)}\right) = 0$ for $i \neq j$. Note that all $b_j^{(0)}$ $\left(\boldsymbol{w}_j^{(0)}\right)$ share a common prior. Therefore, $\mathbb{E}_{\boldsymbol{\omega}}\left[h_j(\boldsymbol{x})h_j\left(\boldsymbol{x}'\right)\right]$ is the same for all $j = 1, \ldots, H$. This means we have

$$\mathbb{E}_{\boldsymbol{\omega}}\left[z_{nn}(\boldsymbol{x})z_{nn}(\boldsymbol{x}')\right] = \sigma_{b^{(1)}}^2 + \omega\mathbb{E}_{\boldsymbol{\omega}}\left[h_1(\boldsymbol{x})h_1\left(\boldsymbol{x}'\right)\right] \triangleq k_{nn}\left(\boldsymbol{x}, \boldsymbol{x}'\right). \tag{C.54}$$

We remark that k_{nn} is exactly the form of the kernel for the infinite width limit of the unconstrained neural network z_{nn} given in Eq. (4.31) [222, Section 18.7.1]. We therefore have

$$\begin{aligned}
\operatorname{Cov}\left(u_{nn}(\boldsymbol{x})u_{nn}(\boldsymbol{x}')\right) &= \phi(\boldsymbol{x})\phi(\boldsymbol{x}')k_{nn}\left(\boldsymbol{x}, \boldsymbol{x}'\right) && \text{(C.55)}\\
&= \tilde{k}_{nn}\left(\boldsymbol{x}, \boldsymbol{x}'\right), && \text{(C.56)}
\end{aligned}$$

i.e. the covariance between any two function outputs is given by the boundary constrained kernel $\tilde{k}_{nn}$ generated by using k_{nn} in the construction of a HCGP (see Definition 4.4.1).

Now, consider the limit as $H \to \infty$. Note that the contribution of the hidden units (the h_j's) is an infinite sum of random variables with shared mean and variance, respectively. Furthermore, the variance is bounded because we are assuming φ is bounded. By the Central Limit Theorem, we conclude then that the contribution of the hidden

units converges in distribution to a Gaussian. Therefore the output of the HCNN itself at any point $\boldsymbol{x} \in \Omega$ converges to a Gaussian, with mean $\tilde{m}(\boldsymbol{x})$ and covariance $\tilde{k}_{nn}(\boldsymbol{x}, \boldsymbol{x})$. Furthermore, given any set of inputs $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \in \Omega$ with $N \geq 2$ the joint distribution of the corresponding outputs $(u_{nn}(\boldsymbol{x}^{(1)}), u_{nn}(\boldsymbol{x}^{(2)}), ..., u_{nn}(\boldsymbol{x}^{(N)}))^\top$ converges to a multivariate Gaussian, where the cross covariance terms are found as $\text{Cov}(u_{nn}(\boldsymbol{x}^{(i)}), u_{nn}(\boldsymbol{x}^{(j)}) = \tilde{k}_{nn}\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right)$

Therefore in the limit, the function u_{nn} exactly satisfies the definition of a Gaussian process (Definition 4.2.2) over Ω with mean function $\tilde{m}$ and covariance function $\tilde{k}_{nn}$ so we conclude

$$u_{nn}(\boldsymbol{x}) \to \mathcal{GP}(\tilde{m}(\boldsymbol{x}), \tilde{k}_{nn}(\boldsymbol{x}, \boldsymbol{x}')). \tag{C.57}$$

□

Bibliography

[1] D. Dalton and D. Husmeier, 'Improved statistical emulation for a soft-tissue cardiac mechanical model', in *Proceedings of the 35th International Workshop on Statistical Modelling*, 2020, pp. 55–60.

[2] D. Dalton, A. Lazarus and D. Husmeier, 'Comparative evaluation of different emulators for cardiac mechanics', in *Proceedings of the 2nd International Conference on Statistics: Theory and Applications*, Virtual Conference, 2020.

[3] D. Dalton, A. Lazarus, A. Rabbani, H. Gao and D. Husmeier, 'Graph neural network emulation of cardiac mechanics', in *Proceedings of the 3rd International Conference on Statistics: Theory and Applications*, Virtual Conference, 2021.

[4] L. Romaszko, A. Borowska, A. Lazarus, D. Dalton, C. Berry, X. Luo, D. Husmeier and H. Gao, 'Neural network-based left ventricle geometry prediction from cmr images with application in biomechanics', *Artificial Intelligence in Medicine*, vol. 119, p. 102 140, 2021.

[5] A. Lazarus, D. Dalton, D. Husmeier and H. Gao, 'Sensitivity analysis and inverse uncertainty quantification for the left ventricular passive mechanics', *Biomechanics and Modeling in Mechanobiology*, vol. 21, no. 3, pp. 953–982, 2022.

[6] D. Husmeier, D. Dalton, A. Lazarus and H. Gao, 'Forward and inverse uncertainty quantification in cardiac mechanics', in *Proceedings of the 2nd International Conference on Statistics: Theory and Applications*, Prague, Czech Republic, 2022.

[7] A. Rabbani, H. Gao, A. Lazarus, D. Dalton, Y. Ge, K. Mangion, C. Berry and D. Husmeier, 'Image-based estimation of the left ventricular cavity volume using deep learning and gaussian process with cardio-mechanical applications', *Computerized Medical Imaging and Graphics*, vol. 106, p. 102 203, 2023.

[8] D. Dalton, D. Husmeier and H. Gao, 'Physics and lie symmetry informed gaussian processes', in *International Conference on Machine Learning*, PMLR, 2024.

[9] D. Dalton, H. Gao and D. Husmeier, 'Emulation of cardiac mechanics using graph neural networks', *Computer Methods in Applied Mechanics and Engineering*, 2022.

[10] D. Dalton, D. Husmeier and H. Gao, 'Physics-informed graph neural network emulation of soft-tissue mechanics', *Computer Methods in Applied Mechanics and Engineering*, vol. 417, p. 116 351, 2023.

[11] L. Zamecnik, 'Mathematical models as abstractions', *Organon F*, vol. 25, pp. 244–264, 2018.

[12] A. Fowler, *Mathematical Models in the Applied Sciences*. Cambridge University Press, 1997.

[13] M. Awasthi, M. Verma and M. Ram, *Advances in Mathematical and Computational Modeling of Engineering Systems*. CRC Press, 2023.

[14] G. Box, 'Robustness in the strategy of scientific model building', in *Robustness in Statistics*, Academic Press, 1979, pp. 201–236.

[15] G. Smith, 'Newton's Philosophiae Naturalis Principia Mathematica', in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., Winter 2008, Metaphysics Research Lab, Stanford University, 2008.

[16] J. Marchant, *The Human Cosmos: A Secret History of the Stars*. Canongate Books, 2020.

[17] J. L. Russell, 'Kepler's laws of planetary motion: 1609–1666', *The British journal for the history of science*, vol. 2, no. 1, pp. 1–24, 1964.

[18] R. W. Smith, 'The cambridge network in action: The discovery of neptune', *Isis*, vol. 80, no. 3, pp. 395–422, 1989.

[19] J. Chambers and J. Mitton, *From Dust to Life: The Origin and Evolution of Our Solar System*. Princeton University Press, 2014.

[20] W. Leontief, *Input-output Economics*. Oxford University Press, 1986.

[21] G. Owen, *Game theory*. Emerald Group Publishing, 2013.

[22] S. De Marchi and S. E. Page, 'Agent-based models', *Annual Review of political science*, vol. 17, pp. 1–20, 2014.

[23] M. Renardy and R. C. Rogers, *An introduction to partial differential equations*. Springer Science & Business Media, 2006, vol. 13.

[24] M. Braun and M. Golubitsky, *Differential equations and their applications*. Springer, 1983, vol. 2.

[25] H. Goldstein, C. Poole and J. Safko, *Classical mechanics*. Pearson, 2013.

[26] A. Lotka, 'Elements of Physical Biology', *Nature*, vol. 116, no. 2917, pp. 461–461, 1925.

[27] E. Schrödinger, 'An undulatory theory of the mechanics of atoms and molecules', *Phys. Rev.*, vol. 28, pp. 1049–1070, 6 1926.

[28] J. O. Campos, J. Sundnes, R. W. dos Santos and B. M. Rocha, 'Effects of left ventricle wall thickness uncertainties on cardiac mechanics', *Biomechanics and Modeling in Mechanobiology*, vol. 18, no. 5, pp. 1415–1427, 2019.

[29] H. Gould, J. Tobochnik and W. Christian, *An Introduction to Computer Simulation Methods: Applications To Physical Systems*. CreateSpace Independent Publishing Platform, 2017.

[30] R. Pukl, M. Jansta, J. Červenka, M. Vořechovskỳ, D. Novák and R. Rusina, 'Spatial variability of material properties in nonlinear computer simulation', in *Computational Modelling of Concrete Structures*, CRC Press, 2020, pp. 891–896.

[31] G. Enkavi, M. Javanainen, W. Kulig, T. Róg and I. Vattulainen, 'Multiscale simulations of biological membranes: The challenge to understand biological phenomena in a living substance', *Chemical reviews*, vol. 119, no. 9, pp. 5607–5774, 2019.

[32] A. Atangana and I. Koca, 'Chaos in a simple nonlinear system with atangana–baleanu derivatives with fractional order', *Chaos, Solitons & Fractals*, vol. 89, pp. 447–454, 2016.

[33] L. Wright and S. Davidson, 'How to tell the difference between a model and a digital twin', *Advanced Modeling and Simulation in Engineering Sciences*, vol. 7, no. 1, pp. 1–13, 2020.

[34] T. Haigh, P. M. Priestley and C. Rope, *ENIAC in action: Making and remaking the modern computer*. MIT press, 2016.

[35] G. O'Regan, 'The first commercial computers', in *A Brief History of Computing*, Springer, 2021, pp. 71–79.

[36] R. Rhodes, *Dark Sun: The Making of the Hydrogen Bomb*. Simon Schuster, 1995.

[37] L. De Mol, ''A Pretence of What is Not'? A Study of Simulation(s) from the ENIAC Perspective', *NTM Zeitschrift für Geschichte der Wissenschaften, Technik und Medizin*, vol. 27, no. 4, pp. 443–478, 2019.

[38] F. H. Harlow, 'Fluid dynamics in group t-3 los alamos national laboratory: (la-ur-03-3852)', *Journal of Computational Physics*, vol. 195, no. 2, pp. 414–433, 2004.

[39] J. Reimer, 'A history of the gui', *Ars Technica*, vol. 5, pp. 1–17, 2005.

[40] D. Goldsman, R. E. Nance and J. R. Wilson, 'A brief history of simulation revisited', in *Proceedings of the 2010 Winter Simulation Conference*, IEEE, 2010, pp. 567–574.

[41] H. Kitano, 'Computational systems biology', *Nature*, vol. 420, no. 6912, pp. 206–210, 2002.

[42] R. Scarpa and A. Alberini, *Applications of simulation methods in environmental and resource economics*. Springer Science & Business Media, 2005, vol. 6.

[43] E. Bisong and E. Bisong, 'An overview of google cloud platform services', *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pp. 7–10, 2019.

[44] G. O'Regan and O'Regan, *A brief history of computing*. Springer, 2008.

[45] J. Backus, 'The history of fortran i, ii, and iii', *ACM Sigplan Notices*, vol. 13, no. 8, pp. 165–180, 1978.

[46] B. W. Kernighan and D. M. Ritchie, 'The C programming language', 2002.

[47] B. Stroustrup, 'A history of c++ 1979–1991', in *History of programming languages—II*, 1996, pp. 699–769.

[48] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne and Q. Zhang, *JAX: Composable transformations of Python+NumPy programs*, version 0.2.5, 2018.

[49] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, 'Equation of state calculations by fast computing machines', *The Journal of Chemical Physics*, 1953.

[50] C. Robert and G. Casella, 'A short history of markov chain monte carlo: Subjective recollections from incomplete data', *Statistical Science*, vol. 26, no. 1, 2011.

[51] D. Gamerman, *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman & Hall/CRC, 2006.

[52] J. N. Reddy, *Introduction to the finite element method*. McGraw-Hill Education, 2019.

[53] G. D. Smith, *Numerical solution of partial differential equations: finite difference methods*. Oxford University Press, 1985.

[54] R. Eymard, T. Gallouët and R. Herbin, 'Finite volume methods', *Handbook of numerical analysis*, vol. 7, pp. 713–1018, 2000.

[55] S. Bardenhagen, J. Brackbill and D. Sulsky, 'The material-point method for granular materials', *Computer methods in applied mechanics and engineering*, vol. 187, no. 3-4, pp. 529–541, 2000.

[56] A. K. Aziz, *The mathematical foundations of the finite element method with applications to partial differential equations*. Academic Press, 2014.

[57] L. Sabat and C. K. Kundu, 'History of finite element method: A review', *Recent Developments in Sustainable Infrastructure: Select Proceedings of ICRDSI 2019*, pp. 395–404, 2020.

[58] M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes and G. N. Wells, 'The FEniCS project version 1.5', *Archive of Numerical Software*, vol. 3, 2015.

[59] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang and L. Yang, 'Physics-informed machine learning', *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021.

[60] H. Gao, A. Aderhold, K. Mangion, X. Luo, D. Husmeier and C. Berry, 'Changes and classification in myocardial contractile function in the left ventricle following acute myocardial infarction', *Journal of The Royal Society Interface*, vol. 14, no. 132, p. 20 170 203, 2017.

[61] A. Grow and J. Hilton, 'Statistical emulation', in *Wiley StatsRef: Statistics Reference Online*. John Wiley & Sons, Ltd, 2018, pp. 1–8.

[62] M. C. Kennedy and A. O'Hagan, 'Bayesian calibration of computer models', *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 3, pp. 425–464, 2001.

[63] R. B. Gramacy, *Surrogates: Gaussian Process Modeling, Design and Optimization for the Applied Sciences*. Boca Raton, Florida: Chapman Hall/CRC, 2020.

[64] A. Lucic, M. ter Hoeve, G. Tolomei, M. de Rijke and F. Silvestri, *Cf-gnnexplainer: Counterfactual explanations for graph neural networks*, 2022. arXiv: 2102.03322.

[65] X. Wang and H.-W. Shen, *Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks*, 2024. arXiv: 2209.07924.

[66] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen and X. Zhang, *Parameterized explainer for graph neural network*, 2020. arXiv: 2011.04573.

[67] J. Kakkad, J. Jannu, K. Sharma, C. Aggarwal and S. Medya, *A survey on explainability of graph neural networks*, 2023. arXiv: 2306.01958.

[68] T. L. Paez, 'Introduction to model validation.', Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2008.

[69] R. P. Baum and W. Sheehan, *In Search of Planet Vulcan: The Ghost in Newton's Clockwork*. Springer, 2013.

[70] O. J. Eggen, 'Vulcan', *Astronomical Society of the Pacific Leaflets, Vol. 6, No. 287, p. 291*, vol. 6, p. 291, 1953.

[71] T. Levenson, *The Hunt for Vulcan*. Random House, 2015.

[72] N. Straumann, *General relativity*. Springer Science & Business Media, 2012.

[73] E. Merzbacher, *Quantum mechanics*. John Wiley & Sons, 1998.

[74] A. Zee, *Einstein gravity in a nutshell*. Princeton University Press, 2013, vol. 14.

[75] G. A. Holzapfel and R. W. Ogden, 'Constitutive modelling of passive myocardium: A structurally based framework for material characterization', *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1902, pp. 3445–3475, 2009.

[76] A. Al-Mayah, *Biomechanics of Soft Tissues: Principles and Applications.* CRC, 2018.

[77] G. A. Holzapfel, *Nonlinear solid mechanics: a continuum approach for engineering science*. Kluwer Academic Publishers Dordrecht, 2002.

[78] J. J. Shim, S. A. Maas, J. A. Weiss and G. A. Ateshian, 'A formulation for fluid-structure interactions in febio using mixture theory', *Journal of Biomechanical Engineering*, 2019.

[79] C. Zhang, H. Gao and X. Hu, 'A multi-order smoothed particle hydrodynamics method for cardiac electromechanics with the purkinje network', *Computer Methods in Applied Mechanics and Engineering*, vol. 407, p. 115 885, 2023.

[80] W. Li, H. Gao, K. Mangion, C. Berry and X. Luo, 'Apparent growth tensor of left ventricular post myocardial infarction–in human first natural history study', *Computers in Biology and Medicine*, vol. 129, p. 104 168, 2020.

[81] J. Corral-Acero, F. Margara, M. Marciniak *et al.*, 'The 'digital twin' to enable the vision of precision cardiology', *European Heart Journal*, vol. 41(48), pp. 4556–4564, 2020.

[82] L. Paun, A. Schmidt, S. Mcginty and D. Husmeier, 'Statistical inference for optimisation of drug delivery from stents', in *Proceedings of the 4th International Conference on Statistics: Theory and Applications*, 2022.

[83] S. Qian, D. Ugurlu, E. Fairweather, M. Strocchi, L. D. Toso, Y. Deng, G. Plank, E. Vigmond, R. Razavi, A. Young, P. Lamata, M. Bishop and S. Niederer, 'Developing cardiac digital twins at scale: Insights from personalised myocardial conduction velocity', *medRxiv*, 2024.

[84] R. Laubenbacher, B. Mehrad, I. Shmulevich and N. Trayanova, 'Digital twins in medicine', *Nature Computational Science*, vol. 4, pp. 184–191, 2024.

[85] R. Rodriguez-Cantano, J. Sundnes and M. E. Rognes, 'Uncertainty in cardiac myofibre orientation and stiffnesses dominate the variability of left ventricle deformation response', *International Journal for Numerical Methods in Biomedical Engineering*, no. 35, e3178, 2018.

[86] G. D. Maso Talou, T. P. Babarenda Gamage, M. Sagar and M. P. Nash, 'Deep learning over reduced intrinsic domains for efficient mechanics of the left ventricle', *Frontiers in Physics*, vol. 8, 2020.

[87] J. O. Campos, J. Sundnes, R. W. dos Santos and B. M. Rocha, 'Uncertainty quantification and sensitivity analysis of left ventricular function during the full cardiac cycle', *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2173, p. 20 190 381, 2020.

[88] B. Milićević, M. Milošević, V. Simić, A. Preveden, L. Velicki, Đorđe Jakovljević, Z. Bosnić, M. Pičulin, B. Žunkovič, M. Kojić and N. Filipović, 'Machine learning and physical based modeling for cardiac hypertrophy', *Heliyon*, vol. 9, no. 6, e16724, 2023.

[89] A. Lazarus, H. Gao, X. Luo and D. Husmeier, 'Improving cardio-mechanic inference by combining in vivo strain data with ex vivo volume-pressure data', *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 2022.

[90] F. Caforio, F. Regazzoni, S. Pagani, E. Karabelas, C. Augustin, G. Haase, G. Plank and A. Quarteroni, *Physics-informed neural network estimation of material properties in soft tissue nonlinear biomechanical models*, 2023. arXiv: 2312.09787 [cs.LG].

[91] A. Rabbani, H. Gao, A. Lazarus, D. Dalton, Y. Ge, K. Mangion, C. Berry and D. Husmeier, 'Image-based estimation of the left ventricular cavity volume using deep learning and gaussian process with cardio-mechanical applications', *Computerized Medical Imaging and Graphics*, vol. 106, p. 102 203, 2023.

[92] A. Lazarus, H. Gao, X. Luo and D. Husmeier, 'Improving Cardio-Mechanic Inference by Combining in Vivo Strain Data with Ex Vivo Volume–Pressure Data', *Journal of the Royal Statistical Society Series C: Applied Statistics*, vol. 71, no. 4, pp. 906–931, 2022.

[93] D. Dalton, A. Lazarus and D. Husmeier, 'Comparative evaluation of different emulators for cardiac mechanics', 2020.

[94] S. Buoso, T. Joyce and S. Kozerke, 'Personalising left-ventricular biophysical models of the heart using parametric physics-informed neural networks', *Medical Image Analysis*, vol. 71, p. 102 066, 2021.

[95] A. Lazarus, 'Surrogate modelling of a patient-specific mathematical model of the left ventricle in diastole', PhD thesis, University of Glasgow, 2022.

[96] D. Bonomi, A. Manzoni and A. Quarteroni, 'A matrix deim technique for model reduction of nonlinear parametrized problems in cardiac mechanics', *Computer Methods in Applied Mechanics and Engineering*, vol. 324, pp. 300–326, 2017.

[97] L. Cicci, S. Fresca, A. Manzoni and A. Quarteroni, 'Efficient approximation of cardiac mechanics through reduced-order modeling with deep learning-based operator approximation', *International Journal for Numerical Methods in Biomedical Engineering*, vol. 40, no. 1, e3783, 2024.

[98] V. G. Satorras, E. Hoogeboom and M. Welling, *E(n) equivariant graph neural networks*, 2021. arXiv: 2102.09844.

[99] Z. Hao, S. Liu, Y. Zhang, C. Ying, Y. Feng, H. Su and J. Zhu, *Physics-informed machine learning: A survey on problems, methods and applications*, 2022. arXiv: 2211.08064.

[100] P. W. Battaglia, J. B. Hamrick, V. Bapst *et al.*, *Relational inductive biases, deep learning, and graph networks*, 2018. arXiv: 1806.01261.

[101] M. Raissi, P. Perdikaris and G. E. Karniadakis, 'Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations', *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[102] M. Raissi, P. Perdikaris and G. E. Karniadakis, 'Machine learning of linear differential equations using Gaussian processes', *Journal of Computational Physics*, vol. 348, pp. 683–693, 2017.

[103] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[104] W. McCulloch and W. Pitts, 'A logical calculus of ideas immanent in nervous activity', *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[105] F. Rosenblatt, 'The perceptron: A probabilistic model for information storage and organization in the brain', *Psychological Review*, vol. 65, pp. 386–408, 1958.

[106] G. Cybenko, 'Approximation by superpositions of a sigmoidal function', *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[107] W. Buntine, 'Machine learning after the deep learning revolution', *Frontiers of Computer Science*, vol. 14, pp. 1–3, 2020.

[108] S. R. Dubey, S. K. Singh and B. B. Chaudhuri, 'Activation functions in deep learning: A comprehensive survey and benchmark', *Neurocomputing*, vol. 503, pp. 92–108, 2022.

[109] I. Goodfellow, Y. Bengio, A. Courville, and F. Bach, *Deep Learning*. MIT Press, 2017.

[110] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli and Y. Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*, 2014. arXiv: `1406.2572 [cs.LG]`.

[111] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: `1412.6980`.

[112] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals and G. E. Dahl, 'Neural message passing for quantum chemistry', in *Proceedings of the 34th International Conference on Machine Learning*, vol. PMLR 70, 2017.

[113] C. Stark, B. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers, 'Biogrid: A general repository for interaction datasets', *Nucleic Acids Research*, vol. 34, pp. D535–D539, 2006.

[114] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende and K. Kavukcuoglu, 'Interaction networks for learning about objects, relations and physics', in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016.

[115] H. Ohtsuki, C. Hauert, E. Lieberman, and M. Nowak, 'A simple rule for the evolution of cooperation on graphs and social networks', *Nature*, vol. 25, pp. 441–502–5, 2006.

[116] I. Konstas, V. Stathopoulos and J. M. Jose, 'On social networks and collaborative recommendation', in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '09, Boston, MA, USA: Association for Computing Machinery, 2009, pp. 195–202.

[117] J. Bruna, W. Zaremba, A. Szlam and Y. LeCun, *Spectral networks and locally connected networks on graphs*, 2014. arXiv: 1312.6203.

[118] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, 2017. arXiv: 1609.02907.

[119] W. L. Hamilton, R. Ying and J. Leskovec, *Inductive representation learning on large graphs*, 2018. arXiv: 1706.02216.

[120] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, *Graph attention networks*, 2018. arXiv: 1710.10903.

[121] M. Gori, G. Monfardini and F. Scarselli, 'A new model for learning in graph domains', in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, 2005, 729–734 vol. 2.

[122] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi and F. Piccialli, 'Scientific machine learning through physics–informed neural networks: Where we are and what's next', *Journal of Scientific Computing*, vol. 92, pp. 1573–7691, 2022.

[123] L. C. Evans, *Partial Differential Equations*. American Mathematical Society, 1993.

[124] L. Yang, X. Meng and G. E. Karniadakis, 'B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data', *Journal of Computational Physics*, vol. 425, p. 109 913, 2021.

[125] L. Yang, D. Zhang and G. E. Karniadakis, 'Physics-informed generative adversarial networks for stochastic differential equations', *SIAM Journal on Scientific Computing*, vol. 42, no. 1, A292–A317, 2020.

[126] Z. Xiang, W. Peng, W. Zhou and W. Yao, *Hybrid finite difference with the physics-informed neural network for solving pde in complex geometries*, 2022. arXiv: 2202.07926.

[127] J. Sirignano and K. Spiliopoulos, 'Dgm: A deep learning algorithm for solving partial differential equations', *Journal of Computational Physics*, vol. 375, pp. 1339–1364, 2018, ISSN: 0021-9991.

[128] A. D. Jagtap, E. Kharazmi and G. E. Karniadakis, 'Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems', *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113 028, 2020.

[129] E. Kharazmi, Z. Zhang and G. E. Karniadakis, 'Hp-vpinns: Variational physics-informed neural networks with domain decomposition', *Computer Methods in Applied Mechanics and Engineering*, vol. 374, p. 113 547, 2021.

[130] C. E. Rasmussen and K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA.: MIT Press, 2006.

[131] R. J. Adler, *The Geometry of Random Fields*. Society for Industrial and Applied Mathematics, 2010.

[132] A. Lavin, H. Zenil, B. Paige, D. Krakauer, J. Gottschlich, T. Mattson, A. Anandkumar, S. Choudry, K. Rocki, A. G. Baydin, C. Prunkl, B. Paige, O. Isayev, E. Peterson, P. L. McMahon, J. Macke, K. Cranmer, J. Zhang, H. Wainwright, A. Hanuka, M. Veloso, S. Assefa, S. Zheng and A. Pfeffer, *Simulation intelligence: Towards a new generation of scientific methods*, 2021.

[133] C. Soize and R. Ghanem, 'Physical Systems with Random Uncertainties: Chaos Representations with Arbitrary Probability Measure', *SIAM Journal on Scientific Computing*, vol. 26, no. 2, pp. 395–410, 2004.

[134] M. Shahriari, D. Pardo, B. Moser and F. Sobieczky, 'A Deep Neural Network as Surrogate Model for Forward Simulation of Borehole Resistivity Measurements', *Procedia Manufacturing*, International Conference on Industry 4.0 and Smart Manufacturing (ISM 2019), vol. 42, pp. 235–238, 2020.

[135] V. Davies, U. Noè, A. Lazarus, H. Gao, B. Macdonald, C. Berry, X. Luo and D. Husmeier, 'Fast parameter inference in a biomechanical model of the left ventricle by using statistical emulation', *Journal of the Royal Statistical Society. Series C, Applied Statistics*, vol. 68, no. 5, pp. 1555–1576, 2019.

[136] G. Sun and S. Wang, 'A review of the artificial neural network surrogate modeling in aerodynamic design', *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 233, no. 16, pp. 5863–5872, 2019.

[137] I. Pan, M. Babaei, A. Korre and S. Durucan, 'Artificial Neural Network based surrogate modelling for multi- objective optimisation of geological CO2 storage operations', *Energy Procedia*, 12th International Conference on Greenhouse Gas Control Technologies, GHGT-12, vol. 63, pp. 3483–3491, 2014.

[138] R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, A. Merose, S. Hoyer, G. Holland, O. Vinyals, J. Stott, A. Pritzel, S. Mohamed and P. Battaglia, 'Learning skillful medium-range global weather forecasting', *Science*, vol. 382, no. 6677, pp. 1416–1421, 2023.

[139] F. e Avila Belbute-Peres, T. D. Economon and J. Z. Kolter, 'Combining differentiable pde solvers and graph neural networks for fluid flow prediction', in *International Conference on Machine Learning*, vol. 37, 2020.

[140] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum and D. L. K. Yamins, *Flexible neural representation for physics prediction*, 2018. arXiv: 1806.08047.

[141] M. Zheng, Y. Zhou, D. Ceylan and J. Barbič, *A deep emulator for secondary motion of 3d characters*, 2021. arXiv: 2103.01261.

[142] N. M. E. Ayad, S. Kaushik and V. M. Weaver, 'Tissue mechanics, an important regulator of development and disease', *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 374, no. 1779, p. 20 180 215, 2019.

[143] A. Quarteroni, T. Lassila, S. Rossi and R. Ruiz-Baier, 'Integrated heart—coupling multiscale and multiphysics models for the simulation of the cardiac function', *Computer Methods in Applied Mechanics and Engineering*, vol. 314, pp. 345–407, 2017.

[144] K. Mangion, H. Gao, D. Husmeier, X. Luo and C. Berry, 'Advances in computational modelling for personalised medicine after myocardial infarction', *Heart*, vol. 104, no. 7, pp. 550–557, 2018.

[145] M Peirlinck, F. S. Costabal, J Yao, J. Guccione, S Tripathy, Y Wang, D Ozturk, P Segars, T. Morrison, S Levine *et al.*, 'Precision medicine in human heart modeling', *Biomechanics and modeling in mechanobiology*, vol. 20, no. 3, pp. 803–831, 2021.

[146] S. A. Niederer, M. S. Sacks, M. Girolami and K. Willcox, 'Scaling digital twins from the artisanal to the industrial', *Nature Computational Science*, vol. 1, no. 5, pp. 313–320, 2021.

[147] S. Marchesseau, S. Chatelin and H. Delingette, 'Nonlinear biomechanical model of the liver', in *Biomechanics of Living Organs*, Elsevier, 2017, pp. 243–265.

[148] H. Gao, N. Qi, L. Feng, X. Ma, M. Danton, C. Berry and X. Luo, 'Modelling mitral valvular dynamics–current trend and future directions', *International Journal for Numerical Methods in Biomedical Engineering*, vol. 33, no. 10, e2858, 2017.

[149] G. A. Holzapfel and R. W. Ogden, 'An arterial constitutive model accounting for collagen content and cross-linking', *Journal of the Mechanics and Physics of Solids*, vol. 136, p. 103 682, 2020.

[150] S. Budday, T. C. Ovaert, G. A. Holzapfel, P. Steinmann and E. Kuhl, 'Fifty shades of brain: A review on the mechanical testing and modeling of brain tissue', *Archives of Computational Methods in Engineering*, vol. 27, no. 4, pp. 1187–1230, 2020.

[151] P. Wriggers, *Nonlinear finite element methods*. Springer Science & Business Media, 2008.

[152] M. Alber, A. Buganza Tepole, W. R. Cannon, S. De, S. Dura-Bernal, K. Garikipati, G. Karniadakis, W. W. Lytton, P. Perdikaris, L. Petzold and E. Kuhl, 'Integrating machine learning and multiscale modeling—perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences', *npj Digital Medicine*, vol. 2, no. 1, p. 115, 2019.

[153] A Olgac and B. Karlik, 'Performance analysis of various activation functions in generalized mlp architectures of neural networks', *International Journal of Artificial Intelligence And Expert Systems*, vol. 1, pp. 111–122, 2011.

[154] Q. Tan, N. Liu and X. Hu, 'Deep representation learning for social network analysis', *Frontiers in Big Data*, vol. 2, 2019.

[155] K. Han, B. Lakshminarayanan and J. Liu, *Reliable graph neural networks for drug discovery under distributional shift*, 2021. arXiv: 2111.12951.

[156] Z. Cui, K. Henrickson, R. Ke and Y. Wang, 'Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting', *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[157] A. Gruber, M. Gunzburger, L. Ju and Z. Wang, 'A comparison of neural network architectures for data-driven reduced-order modeling', *Computer Methods in Applied Mechanics and Engineering*, vol. 393, p. 114 764, 2022.

[158] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec and P. W. Battaglia, 'Learning to simulate complex physics with graph networks', in *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[159] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum and A. Torralba, *Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids*, 2018.

[160] C. Yang, W. Gao, D. Wu and C. Wang, *Learning to simulate unseen physical systems with graph neural networks*, 2022. arXiv: 2201.11976.

[161] J. L. Ba, J. R. Kiros and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450.

[162] H. P. Langtangen and A. Logg, *Solving PDEs in Python*. Springer International Publishing, 2016.

[163] M. Casdagli, S. Eubank, J. Farmer and J. Gibson, 'State space reconstruction in the presence of noise', *Physica D: Nonlinear Phenomena*, vol. 51, no. 1, pp. 52–98, 1991.

[164] A. Lazarus, D. Dalton, D. Husmeier and H. Gao, 'Sensitivity analysis and inverse uncertainty quantification for the left ventricular passive mechanics', *Biomechanics and Modeling in Mechanobiology*, pp. 1–30, 2022.

[165] H. M. Wang, H. Gao, X. Y. Luo, C. Berry, B. E. Griffith, R. W. Ogden and T. J. Wang, 'Structure-based finite strain modelling of the human left ventricle in diastole', *International Journal for Numerical Methods in Biomedical Engineering*, vol. 29, no. 1, pp. 83–103, 2013.

[166] E. Naghavi, H. Wang, L. Fan, J. S. Choy, G. Kassab, S. Baek and L.-C. Lee, *Rapid estimation of left ventricular contractility with a physics-informed neural network inverse modeling approach*, 2024. arXiv: `2401.07331 [cs.CE]`.

[167] K.-T. Fang, R. Li and A. Sudjianto, *Design and Modeling for Computer Experiments (Computer Science & Data Analysis)*. Chapman & Hall/CRC, 2005.

[168] N. Kawel-Boehm, A. Maceira, E. R. Valsangiacomo-Buechel, J. Vogel-Claussen, E. B. Turkbey, R. Williams, S. Plein, M. Tee, J. Eng and D. A. Bluemke, 'Normal values for cardiovascular magnetic resonance in adults and children', *Journal of Cardiovascular Magnetic Resonance*, vol. 17, no. 1, p. 29, 2015.

[169] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong and Q. He, *A comprehensive survey on transfer learning*, 2019. arXiv: `1911.02685`.

[170] Y. Dabiri, A. Van der Velden, K. L. Sack, J. S. Choy and G. S. Kassab, 'Prediction of Left Ventricular Mechanics Using Machine Learning', *Frontiers in Physics*, vol. 7, 2019.

[171] H. Osnes and J. Sundnes, 'Uncertainty analysis of ventricular mechanics using the probabilistic collocation method', *IEEE transactions on bio-medical engineering*, vol. 59, no. 8, pp. 2171–2179, 2012.

[172] U. Noè, A. Lazarus, H. Gao, V. Davies, B. Macdonald, K. Mangion, C. Berry, X. Luo and D. Husmeier, 'Gaussian process emulation to accelerate parameter estimation in a mechanical model of the left ventricle: A critical step towards clinical end-user relevance', *Journal of The Royal Society Interface*, vol. 16, no. 156, p. 20 190 114, 2019.

[173] L. Romaszko, A. Lazarus, H. Gao, A. Borowska, X. Luo and D. Husmeier, 'Massive dimensionality reduction for the left ventricular mesh', *International Conference on Statistics: Theory and Applications (ICSTA)*, 2019.

[174] R. M. Zur, Y. Jiang, L. L. Pesce and K Drukker, 'Noise injection for training artificial neural networks: A comparison with weight decay and early stopping', *Medical Physics*, vol. 36, no. 10, pp. 4810–4818, 2009.

[175] C. M. Bishop, 'Training with Noise is Equivalent to Tikhonov Regularization', *Neural Computation*, vol. 7, no. 1, pp. 108–116, 1995.

[176] W. Zhang, D. S. Li, T. Bui-Thanh and M. S. Sacks, 'Simulation of the 3D hyperelastic behavior of ventricular myocardium using a finite-element based neural-network approach', *Computer Methods in Applied Mechanics and Engineering*, vol. 394, p. 114 871, 2022.

[177] H. Gao, M. J. Zahr and J.-X. Wang, 'Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems', *Computer Methods in Applied Mechanics and Engineering*, vol. 390, p. 114 502, 2022.

[178] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner and M. van Zee, *Flax: A neural network library and ecosystem for JAX*, version 0.4.0, 2020.

[179] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[180] H. Lee and I. S. Kang, 'Neural algorithm for solving differential equations', *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[181] A. Meade and A. Fernandez, 'The numerical solution of linear ordinary differential equations by feedforward neural networks', *Mathematical and Computer Modelling*, vol. 19, no. 12, pp. 1–25, 1994.

[182] I. Lagaris, A. Likas and D. Fotiadis, 'Artificial neural networks for solving ordinary and partial differential equations', *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[183] T. Graepel, 'Solving Noisy Linear Operator Equations by Gaussian Processes: Application to Ordinary and Partial Differential Equations', in *Proceedings of The 20th International Conference on Machine Learning*, vol. 1, 2003, pp. 234–241.

[184] S. Särkkä, 'Linear operators and stochastic partial differential equations in gaussian process regression', in *Artificial Neural Networks and Machine Learning – ICANN 2011*, T. Honkela, W. Duch, M. Girolami and S. Kaski, Eds., Springer Berlin Heidelberg, 2011, pp. 151–158.

[185] E Weinan and B. Yu, 'The deep ritz method: A deep learning-based numerical algorithm for solving variational problems', *Communications in Mathematics and Statistics*, vol. 1, no. 6, pp. 1–12, 2018.

[186] V. M. Nguyen-Thanh, X. Zhuang and T. Rabczuk, 'A deep energy method for finite deformation hyperelasticity', *European Journal of Mechanics - A/Solids*, vol. 80, p. 103 874, 2020.

[187] J. He, D. Abueidda, S. Koric and I. Jasiuk, 'On the use of graph neural networks and shape-function-based gradient computation in the deep energy method', *International Journal for Numerical Methods in Engineering*, vol. 124, no. 4, pp. 864–879, 2023.

[188] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre and P. Perdikaris, 'Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks', *Computer Methods in Applied Mechanics and Engineering*, vol. 358, p. 112 623, 2020.

[189] A. Kovacs, L. Exl, A. Kornell, J. Fischbacher, M. Hovorka, M. Gusenbauer, L. Breth, H. Oezelt, M. Yano, N. Sakuma, A. Kinoshita, T. Shoji, A. Kato and T. Schrefl, 'Conditional physics informed neural networks', *Communications in Nonlinear Science and Numerical Simulation*, vol. 104, p. 106 041, 2022.

[190] M. Islam, M. S. H. Thakur, S. Mojumder and M. N. Hasan, 'Extraction of material properties through multi-fidelity deep learning from molecular dynamics simulation', *Computational Materials Science*, vol. 188, p. 110 187, 2021.

[191] M. Destrade, L. Dorfmann and G. Saccomandi, 'The ogden model of rubber mechanics: 50 years of impact on nonlinear elasticity', *Philosophical Transactions of the Royal Society A*, vol. 380, no. 2234, p. 20 210 332, 2022.

[192] J. Guccione, K. Costa and A. McCulloch, 'Finite element stress analysis of left ventricular mechanics in the beating dog heart', *Journal of biomechanics*, vol. 28, no. 10, pp. 1167–1177, 1995.

[193] E. D. S. Neto, F. A. Pires and D. Owen, 'F-bar-based linear triangles and tetrahedra for finite strain analysis of nearly incompressible solids. part i: Formulation and benchmarking', *International Journal for Numerical Methods in Engineering*, vol. 62, no. 3, pp. 353–383, 2005.

[194] J. T. Barron, *Continuously differentiable exponential linear units*, 2017. arXiv: 1704.07483.

[195] I. Babuschkin, K. Baumli, A. Bell *et al.*, *The DeepMind JAX Ecosystem*, 2020.

[196] J. Zhang and S. Chauhan, 'Fast computation of soft tissue thermal response under deformation based on fast explicit dynamics finite element algorithm for surgical simulation', *Computer Methods and Programs in Biomedicine*, vol. 187, p. 105 244, 2020.

[197] W. Kratzer, V. Fritz, R. A. Mason, M. M. Haenle, V. Kaechele and R. S. Group, 'Factors affecting liver size', *Journal of Ultrasound in Medicine*, vol. 22, no. 11, pp. 1155–1161, 2003.

[198] H. Liu, J. S. Soares, J. Walmsley, D. S. Li, S. Raut, R. Avazmohammadi, P. Iaizzo, M. Palmer, J. H. Gorman, R. C. Gorman *et al.*, 'The impact of myocardial compressibility on organ-level simulations of the normal and infarcted heart', *Scientific reports*, vol. 11, no. 1, pp. 1–15, 2021.

[199] O. Bernard, A. Lalande, C. Zotti, F. Cervenansky, X. Yang, P.-A. Heng, I. Cetin, K. Lekadir, O. Camara, M. A. G. Ballester *et al.*, 'Deep learning techniques for automatic mri cardiac multi-structures segmentation and diagnosis: Is the problem solved?', *IEEE transactions on medical imaging*, vol. 37, no. 11, pp. 2514–2525, 2018.

[200] S. H. Sheen, E. Larionov and D. K. Pai, 'Volume preserving simulation of soft tissue with skin', *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 4, no. 3, pp. 1–23, 2021.

[201] P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. van Hoesel, H. Schopmans, T. Sommer *et al.*, 'Graph neural networks for materials science and chemistry', *Communications Materials*, vol. 3, no. 1, p. 93, 2022.

[202] L. Pegolotti, M. R. Pfaller, N. L. Rubio, K. Ding, R. B. Brufau, E. F. Darve and A. L. Marsden, *Learning reduced-order models for cardiovascular simulations with graph neural networks*, 2023. ArXiv: `abs/2303.07310`.

[203] E. Haghighat, D. Amini and R. Juanes, 'Physics-informed neural network simulation of multiphase poroelasticity using stress-split sequential training', *Computer Methods in Applied Mechanics and Engineering*, vol. 397, p. 115 141, 2022.

[204] J. He, L. Li, J. Xu and C. Zheng, 'ReLU Deep Neural Networks and Linear Finite Elements', *Journal of Computational Mathematics*, vol. 38, no. 3, pp. 502–527, 2020.

[205] G. P. P. Pun, R. Batra, R. Ramprasad and Y. Mishin, 'Physically informed artificial neural networks for atomistic modeling of materials', *Nature Communications*, vol. 10, no. 1, p. 2339, 2019.

[206] S. Cai, Z. Mao, Z. Wang, M. Yin and G. E. Karniadakis, *Physics-informed neural networks (pinns) for fluid mechanics: A review*, 2021. arXiv: 2105.09506.

[207] B. Lütjens, C. H. Crawford, M. Veillette and D. Newman, *Pce-pinns: Physics-informed neural networks for uncertainty propagation in ocean modeling*, 2021. arXiv: 2105.02939.

[208] S. Wang, X. Yu and P. Perdikaris, *When and why pinns fail to train: A neural tangent kernel perspective*, 2020. arXiv: 2007.14527.

[209] E. Solak, R. Murray-Smith, W. Leithead, D. Leith and C. Rasmussen, 'Derivative observations in gaussian process models of dynamic systems', in *Advances in Neural Information Processing Systems*, vol. 15, MIT Press, 2002.

[210] M. Alvarez, D. Luengo and N. Lawrence, 'Linear latent force models using gaussian processes', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2693–2705, 2013.

[211] A. Melkumyan, 'Operator induced multi-task gaussian processes for solving differential equations', in *Advances in Neural Information Processing Systems Workshop: New Directions in Multiple Kernel Learning*, MIT Press, 2012.

[212] F Dondelinger, M Filippone, S Rogers and D Husmeier, 'ODE parameter inference using adaptive gradient matching with Gaussian processes', in *Proceedings of The 16th International Conference on Artificial Intelligence and Statistics*, 2013, pp. 216–228.

[213] M. Tan, 'Gaussian process modeling with boundary information', *Statistica Sinica*, 2016.

[214] Z. Li and M. H. Y. Tan, 'Improving Gaussian Process Emulators with Boundary Information', in *Artificial Intelligence, Big Data and Data Science in Statistics: Challenges and Solutions in Environmetrics, the Natural Sciences and Technology*, Cham: Springer International Publishing, 2022, pp. 171–192.

[215] M. Lange-Hegermann, 'Linearly Constrained Gaussian Processes with Boundary Conditions', in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, PMLR, 2021, pp. 1090–1098.

[216] L. Ding, S. Mak and C. F. J. Wu, *Bdrygp: A new gaussian process model for incorporating boundary information*, 2019. arXiv: 1908.08868.

[217] A. Solin and M. Kok, 'Know your boundaries: Constraining Gaussian processes by variational harmonic features', in *Proceedings of Machine Learning Research*, vol. 89, 2019, pp. 2193–2202.

[218] M. Gulian, A. Frankel and L. Swiler, 'Gaussian process regression constrained by boundary value problems', *Computer Methods in Applied Mechanics and Engineering*, vol. 388, p. 114 117, 2022.

[219] H. Sheng and C. Yang, 'PFNN: A Penalty-Free Neural Network Method for Solving a Class of Second-Order Boundary-Value Problems on Complex Geometries', *Journal of Computational Physics*, vol. 428, p. 110 085, 2021.

[220] N. Sukumar and A. Srivastava, 'Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks', *Computer Methods in Applied Mechanics and Engineering*, vol. 389, p. 114 333, 2022.

[221] S. Liu, Z. Hao, C. Ying, H. Su, J. Zhu and Z. Cheng, *A Unified Hard-Constraint Framework for Solving Geometrically Complex PDEs*, 2022.

[222] K. P. Murphy, *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[223] M. Kanagawa, P. Hennig, D. Sejdinovic and B. K. Sriperumbudur, *Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences*, 2018.

[224] J. Chen, Z. Chen, C. Zhang and C. F. Jeff Wu, 'Apik: Active physics-informed kriging model with partial differential equations', *SIAM/ASA Journal on Uncertainty Quantification*, vol. 10, no. 1, pp. 481–506, 2022.

[225] I. Steinward and A. Christmann, *Support Vector Machines* (Information Science and Statistics). New York, NY: Springer New York, 2008.

[226] N. Aronszajn, 'Theory of Reproducing Kernels', *Transactions of the American Mathematical Society*, vol. 68, no. 3, pp. 337–404, 1950.

[227] M. . Searcóid, *Metric Spaces*. London: Springer London, 2007.

[228] C. A. Micchelli, Y. Xu and H. Zhang, 'Universal Kernels', *Journal of Machine Learning Research*, vol. 7, no. 95, pp. 2651–2667, 2006.

[229] R. M. Neal, *Bayesian Learning for Neural Networks*. Springer, 1996.

[230] J. Zhang, S. Zhang and G. Lin, *PAGP: A physics-assisted Gaussian process framework with active learning for forward and inverse problems of partial differential equations*, 2022. arXiv: 2204.02583.

[231] L. Lu, X. Meng, Z. Mao and G. E. Karniadakis, 'DeepXDE: A deep learning library for solving differential equations', *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[232] D. Millán, N. Sukumar and M. Arroyo, 'Cell-based maximum-entropy approximants', *Computer Methods in Applied Mechanics and Engineering*, vol. 284, pp. 712–731, 2015.

[233] A. Biswas and V. Shapiro, 'Approximate distance fields with non-vanishing gradients', *Graphical Models*, vol. 66, no. 3, pp. 133–159, 2004.

[234] R. Leiteritz and D. Pflüger, *How to Avoid Trivial Solutions in Physics-Informed Neural Networks*, 2021.

[235] M. Raissi, P. Perdikaris and G. E. Karniadakis, 'Numerical Gaussian Processes for Time-Dependent and Nonlinear Partial Differential Equations', *SIAM Journal on Scientific Computing*, vol. 40, no. 1, A172–A198, 2018.

[236] D. Long, Z. Wang, A. Krishnapriyan, R. Kirby, S. Zhe and M. Mahoney, *AutoIP: A United Framework to Integrate Physics into Gaussian Processes*, 2022.

[237] M. Titsias, 'Variational learning of inducing variables in sparse gaussian processes', in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 5, PMLR, 2009, pp. 567–574.

[238] J. Hensman, N. Fusi and N. D. Lawrence, *Gaussian processes for big data*, 2013. arXiv: 1309.6835.

[239] A. V. Vecchia, 'Estimation and model identification for continuous spatial processes', *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 50, no. 2, pp. 297–312, 1988.

[240] A. C. Annie Sauer and R. B. Gramacy, 'Vecchia-approximated deep gaussian processes for computer experiments', *Journal of Computational and Graphical Statistics*, vol. 32, no. 3, pp. 824–837, 2023.

[241] S. Sarkka, A. Solin and J. Hartikainen, 'Spatiotemporal learning via infinite-dimensional bayesian filtering and smoothing: A look at gaussian process regression through kalman filtering', *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 51–61, 2013.

[242] K. Gilbert, W. Bai, C. Mauger, P. Medrano-Gracia, A. Suinesiaputra, A. M. Lee, M. M. Sanghvi, N. Aung, S. K. Piechnik, S. Neubauer, S. E. Petersen, D. Rueckert and A. A. Young, 'Independent Left Ventricular Morphometric Atlases Show Consistent Relationships with Cardiovascular Risk Factors: A UK Biobank Study', *Scientific Reports*, vol. 9, no. 1, p. 1130, 2019.

[243] L. N. Smith, *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*, 2018. arXiv: 1803.09820.