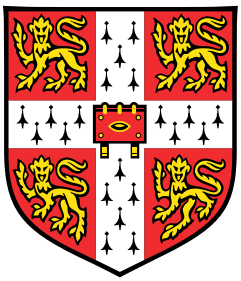
Dynamic Latent Spaces with Statistical Finite Elements



Andrew Wang

Supervisor: Prof. M.A. Girolami

Department of Engineering
Christ's College, University of Cambridge
1 June 2022

I hereby declare that, except where specifically indicated, the work submitted herin is my own original work.

Signed  *Date* 01/06/2022

Abstract

Dynamic Latent Spaces with Statistical Finite Elements
Andrew Wang, Christ's College
1 June 2022

Time-dependent physical phenomena in nature are often modelled by partial differential equations (PDEs), which are widely used in science and engineering, for example, for structural mechanics, tidal waves, heat transfer or chemical diffusion. It has become clear now that the best models of physical phenomena are constrained not only by these laws of physics, but also by observed data. However, it is still challenging to assimilate real-world data with underlying physics in a coherent, statistical way and with a quantifiable uncertainty, so that model estimates can be interpreted with risk, because of the variability and complexity of practical data.

A new data-driven approach that encapsulates inherent uncertainties in the physical world is pivotal in developing digital twins, which are making an increasing impact in a growing number of science and engineering fields, for example in structural health monitoring, climate modelling, precision medicine and agricultural monitoring. Digital twins are useful in estimating the true state of the physical twin, and forecasting their future. For this purpose, all of these digital twins applications are characterised by the need to assimilate large quantities of sensor data into a virtual, "digital twin" physics model, whether that model be of sea surface temperature, the heart, or a structural beam [21].

The statistical finite element method (statFEM) has been developed in [6] in order to tackle this problem of data assimilation. The method enhances the finite element method in solving spatiotemporal PDEs by providing a framework to assimilate noisy data in a Bayesian manner using stochastic filtering. However, with the widespread use of more advanced sensing technologies, it has become vital to achieve the same style of data assimilation for complex datasets, such as video and audio, where the mapping between data and underlying variables is non-linear, spatially dependent in 2D, and harder to estimate. Our work focuses on developing a statistical framework around statFEM for complex datasets, leveraging recent advances in machine learning and computer vision techniques. We approach this problem using representation learning; as in statFEM, the physics model is contained in a dynamic low-dimensional latent

space, but we learn the mapping to and from higher-dimensional data using deep unsupervised learning.

This project describes the use of the Variational Autoencoder (VAE) [16], a form of deep generative model, which is capable of being trained in order to probabilistically model the data likelihood and a meaningful low-dimensional latent space, using deep convolutional neural networks to model the mapping. We train and test the VAE neural networks on computer vision datasets to show that it can quickly learn a map from data to smooth latent spaces, where the coordinates of inferred points encode some information characteristic of the dataset, for example, the shape of a handwritten digit or the spatial position of an object in the image.

Building on this, we show that the Kalman VAE (KVAE) [9] allows us to model time dependency in the dataset in order to model linear dynamics. We show that by factorising the log-likelihood following [9], we can separate the problem of inferring a dynamic latent space into two parts: the standard time-independent VAE latent space inference to model a complex data-generating process, and a Kalman filter to infer sequences in the state-space in the context of a dynamical model. We note that there has been much interest in developing autoencoder-based models for dynamic latent space modelling; however, the specific use of the Kalman filter, which is a form of sequential Bayesian inference, will be of use to us below. Training on videos generated from a dynamical model, we show that the KVAE can infer a latent space similar to the generating physics, and also generate realistic observations forwards in time from the latent space.

However, for our purposes, and for digital twins applications, we do not actually want to learn the dynamical model, as was the focus in [9]. Instead, we wish to fix the dynamics to a dynamical model derived from a physical PDE. Since the dynamical model in the KVAE is ultimately a linear state-space transition model, we show that in the case of a linear PDE, we can use statFEM to solve the PDE and directly fix the parameters of the transition model, where the statFEM stochastic filter becomes the Kalman filter in the KVAE. We name our resulting *physics-informed* model the statFEM-KVAE for this reason. We train and test statFEM-KVAE using videos generated from simple linear PDEs, such as of waves following the wave equation. We show that the trained model can extract not only a meaningful, but the correct latent space, that represents a Bayesian combination of the underlying PDE variables (e.g. wave position) and the data (videos of waves moving). Our model provides a statistical framework for the motivating aims outlined above, for example for digital twins applications: it can continually estimate and calibrate a physical model given data, predict and interpolate in time and space, and return all predictions with an associated uncertainty.

Table of contents

Chapter 1

Introduction

1.1 Motivation

Natural physical phenomena are often described with physics models such as with partial differential equations (PDEs). However, in many fields of science and engineering, it is still an important and challenging problem to estimate these models from real-world observed data. For example, given fluid flow video data captured in a laboratory experiment, which is assumed to follow a certain spatiotemporal PDE, this could mean:

- Estimate the evolution of the variables of the underlying low-dimensional physical process with a statistical uncertainty quantification, given higher-dimensional data;
- Calibrate a misspecified underlying physical model by assimilating data;
- Generate new, reconstructed observations from the underlying model for data synthesis or imputation of sparse data.

This work has an importance in the research of data-driven digital twins, which are making an impact in a growing number of domains, such as medicine [21], civil engineering [8], agriculture [26] and climate science [3]. A digital twin is a mathematical model that must be "dynamically updated with data from its physical twin" [1]. In order to do this, we need a way to incorporate large quantities of data with the underlying physical model, and learn a mapping between the two.

The current method used to address the tasks above is the statistical finite element method (statFEM) introduced in [6, 11]. However, given an increase in the complexity of sensing options, it is now important to be able to apply this data assimilation method with increasingly complex datasets. For example, in [8], a digital twin of the main I-beams of a bridge is successfully developed, combining strain sensor measurements with the underlying structural

beam theory using statFEM. However, it is often only possible to have remotely-sensed data such as video or audio data, such as when monitoring tidal waves [13]. Such challenging data requires advances in artificial intelligence in order to learn the data mapping.

Furthermore, digital twins "must be able to extrapolate and issue predictions about yet-unseen conditions and future system states, and it must do so with quantified uncertainties and acceptable levels of risk" [21]. This means that any model we develop must be equipped with a coherent, interpretable statistical model for quantification of uncertainty in the model, data and the predictions.

1.2 Proposed approach

We approach this problem by embedding dynamics into a latent space that is *physics-informed*; that is, a physical model dictates how some latent variables should evolve spatiotemporally. We then probabilistically combine the latent space and data into one coherent generative model that is then trained using deep representation learning.

Our approach is built on two advances. First, we use the recent statistical finite element method [6] as a way to synthesise finite element models with observed data under uncertainty. Second, we use a deep generative model with a dynamic latent space to perform deep unsupervised representation learning. We choose the Kalman Variational Autoencoder introduced in [9] for reasons detailed in Section 1.3.1. There are two ways of looking at how our model solves the above problem. Firstly, we want to equip a deep generative model with dynamics determined by a physical model by combining the two in a Bayesian manner, instead of learning dynamics from scratch. Alternatively, we want to equip a physics model solver with deep feature extraction capabilities.

In Section 2 we introduce our proposed model applied to linear SPDEs. First, we solve the PDE using statFEM to obtain a linear *transition model*. Our model can then be split into two separate parts: a standard, time-independent Variational Autoencoder (VAE) which learns a latent representation, and a Kalman filter which infers hidden latent sequences from the VAE latent space samples according to the transition model. This turns out to be equivalent to setting a physics-informed prior in the VAE to the transition model itself, so that the data assimilation is an elegant Bayesian combination of the data likelihood and the physics prior. For this reason, we name our model statFEM-KVAE. This methodology is readily extended to the non-linear case, building upon theory in [6].

Recently, there has been little progress on physics-informed representation learning models for estimating dynamical models. Most work focuses on learning the dynamical model itself, for example, see [10], and Section 1.4 for a full review. However, for digital twins and many

other science and engineering applications, the problem is to learn a representation from data in the face of a preexisting and assumed underlying physical model. We therefore need a coherent statistical framework where we can define fully the dynamical model as a statistical prior. Our work is novel, since we combine the ability of statFEM to assimilate data with a physical model under uncertainty with a deep representation learning model.

1.3 Background

1.3.1 Deep Generative Models

Deep generative models emerged as a way of writing a joint probabilistic model of data and latent representations, which is learned using neural networks given large quantities of data. The Variational Autoencoder [16] is a popular model for learning meaningful latent space representations in a convenient Bayesian framework ("encoding"). It combines deep encoder and decoder models to create a non-linear "bottleneck", while placing a prior and introducing a sampling stage in the latent space. This has a regularising effect on the latent space, enforcing a smooth space where points that are close together are similar in their data reconstructions. The VAE also performs well as a generative model ("decoding"); samples can be taken from the regularised latent space to generate, for example, "new" images of human faces [23]. The VAE also improves on other non-linear representation-learning methods such as kernel variants of ICA or tICA [24], where it is hard to extract meaning from the implicit kernel-space coordinates. Furthermore, VAEs have been extensively studied and there are many improvements to the basic model [25].

There have been many models which take the base structure of the VAE and employ some kind of time-dependent model. [10] compares several of such models. From this model selection, we base our model off the Kalman VAE (KVAE) [9], as it is convenient to have the approximate VAE posterior split into a *disentangled* encoder posterior and sequence posterior, separating the VAE inference task from the dynamical inference task. As we see later in the report, this allows us to directly replace ("inform") the parameters of the dynamics model with the solution to a PDE. This property arises from the factorisation of the joint likelihood, and is not present in other models such as [14] or RNN-based models such as [4]. Note that although [10] doesn't report good training results for KVAE, we do not worry about its performance to *learn* dynamics, as that is not our main task.

By defining encoder and decoder functions within the VAE probabilistic framework, the VAE has access to a rich variety of neural network architectures to model highly non-linear and complex dimensionality reduction and expansion functions. For example, in order to model

observed image sequence data we can use convolutional neural networks (CNNs) to extract spatially dependent features from the images.

Finally, generative adverserial networks [12] are popular alternatives for deep generative modelling, and have been also used in physics-informed dynamic representation learning [27, 28] . However, the complexity and computational cost of training GANs is higher than in feedforward network models such as VAEs, and can furthermore suffer from training instability [28].

1.3.2 Spatiotemporal modelling in autoencoding models

Partial differential equations (PDE) are widely used to model physical systems in space and time, such as in fluid dynamics or electromagnetism. These systems are most often solved using numerical analysis, such as with the finite element method (FEM), since there is usually no analytical solution. To create a framework in which we can combine governing equations with observed data, [6] introduces a stochastic forcing term inside a PDE, creating a stochastic PDE (SPDE) [30], which casts the problem as a probabilistic one. The SPDE also emits noisy observations. Then, using their statistical FEM, they show that the task amounts to filtering in a Bayesian framework, where the state-space model (SSM) transitions are governed by the solved PDE.

There are many alternatives to modelling dynamical systems in autoencoder-based models. For example, [19] models dynamics with a finite dimensional Koopman operator, and [29] sets dynamics defined by an ordinary differential equation (ODE). We prefer the PDE approach, due to the existing prevailing usage and flexibility of PDEs in wider science and engineering. Furthermore, when we assume an underlying physical model, we do not want the burden of having to also accurately learn the dynamics, as is the focus of much of the literature such as [29]. The worry of failing to define sufficiently accurate dynamic models, as mentioned in [19], is accounted by statFEM which can deal with model misspecification by modelling uncertainty in the SPDE. We expect our model to perform much better than a model with a dynamical model not informed by physics.

1.4 Related work

Here we review some existing, latent space models where some part of the dynamics is physics-informed. [7] learns a transition matrix using a similar autoencoder construction. The only physics-informed aspect of the model is in regularising the learnt transition matrix to be Lyapunov-stable, that is, to keep inferred trajectories robust with respect to perturbations in

Model	Deep observation mapping	Fully PI space	Uncertainty in dynamics	Uncertainty in observations
[7]	✓	✗	✓	✓
[18]	✓	✓	✗	✓
[31]	✓	✓	✓	✗
statFEM [6]	✗	✓	✓	✓
statFEM-KVAE	✓	✓	✓	✓

Table 1.1 Comparison of physics-informed (PI) dynamic latent space models

the initial conditions. With this being the only condition, this still means that learnt physical dynamics could be completely different from the true dynamics. [18] simulates from a stipulated PDE solution and tries to minimise the error between the VAE reconstruction and this, instead of the original observations. This means that we cannot quantify uncertainty when assimilating data into our physics model: we are assuming our PDE solution is the absolute ground truth. This is of no use in, for example, digital twins applications. In [31], the decoder directly models the terms of a stipulated stochastic differential equation (SDE) problem, while the encoder takes observations as ground truth which it decodes through the SDE. As the authors mention, this provides no possibility to model uncertainty in the observations: for our purpose, we wish to model uncertainty as a combination of sensor observation uncertainty and model stochasticity. Finally, the original statFEM [6, 11], assumes that the data-generating process (DGP) is known, and is a noisy linear transformation onto the observation grid. Instead, we want to model non-linear mappings from complex data such as images, and learn these.

See Table 1.1 for a summary of this discussion.

Chapter 2

Methods

In our problem we wish to learn latent, or low-dimensional, representations of multivariate, time-sequence data. In the following, we discuss the Variational Autoencoder (VAE) methodology, which allows us to learn an encoder to achieve this task. We start with a detailed introduction of the original VAE from [16] which does not evolve with time, as this will help in deriving dynamic variants. Then we introduce the dynamical variant of the VAE which models continuous-time data.

We then discuss how we can achieve a physics-informed model by using a stochastic partial differential equation (SPDE) to place a prior on the latent space. This is solved by the statistical finite elements method (statFEM), which discretises the PDE from continuous time and space. This allows the incoming data to be incorporated with knowledge of the physical dynamical model, in the following way: if we know that the data should have originated from a physical law or equation, then we can set the parameters of the dynamical model to how we expect the latent variables to evolve spatiotemporally according to physics. Then, the actual filtered variables will combine time-evolving observations with presumed knowledge of the dynamics, in a Bayesian manner.

2.1 The Variational Autoencoder

The VAE model, first introduced independently by [16] and [23], is a simple model in which each data point or observation $\mathbf{y}$ is represented by a point in the latent space $\mathbf{x}$, whose dimension is typically smaller, giving a data "bottleneck". It builds a probabilistic model of the data and the latent space in order to learn the relationship between these. This relationship can be thought of a dimensionality reduction problem. We can represent the VAE model by writing out the joint probability density function:

$$p(\mathbf{y},\mathbf{x}) = p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})$$

We can now see the form of the generative, or decoder model: latent points are drawn from the prior $p(\mathbf{x})$, and then data is drawn from the likelihood $p(\mathbf{y} \mid \mathbf{x})$. This must be coupled with the inference, encoder model of the VAE in order to infer the latent space points from the data, requiring the posterior from Bayes rule:

$$p(\mathbf{x} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}$$

However, as in many Bayesian machine learning methods, the naive decomposition of the evidence $p(\mathbf{y}) = \int p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})\mathrm{d}\mathbf{x}$ is difficult as it requires integrating over all possible latent space configurations. Furthermore, in order to train the parameters, it is desirable to use and maximise the evidence $p(\mathbf{y})$ as the training criterion. The VAE solves this by introducing an approximate posterior $q_\phi(\mathbf{x} \mid \mathbf{y})$ from the variational family ϕ which is often taken to be Gaussian. Then the evidence can now be decomposed using variational inference (VI), and the parameters ϕ are to be learned. We bear this derivation in mind when we move onto the dynamical model in Section 2.2.2.

$$\begin{aligned}\log p(\mathbf{y}) &= \log \int p(\mathbf{y},\mathbf{x})\mathrm{d}\mathbf{x} = \log \int \frac{p(\mathbf{y},\mathbf{x})}{q_\phi(\mathbf{x} \mid \mathbf{y})} q_\phi(\mathbf{x} \mid \mathbf{y})\mathrm{d}\mathbf{x} \\ &\geq \int q_\phi(\mathbf{x} \mid \mathbf{y}) \log \frac{p(\mathbf{y},\mathbf{x})}{q_\phi(\mathbf{x} \mid \mathbf{y})}\mathrm{d}\mathbf{x} = ELBO(\phi) \end{aligned} \tag{2.1}$$

where the last line is due to Jensen's inequality. Equation (2.1) shows that we can maximise the log evidence by maximising the Evidence Lower Bound (ELBO) of the data; see [16] for a full discussion. We denote the parameters of the likelihood distribution as θ since these are unknown so are also to be learned. However, since we are still integrating over $\mathrm{d}\mathbf{x}$, why is this decomposition useful? Using the definition of conditional probability, we see that, in a vanilla VAE, we can further decompose the ELBO as

$$ELBO(\phi,\theta) = \mathbb{E}_{q_\phi(\mathbf{x}|\mathbf{y})}[\log p_\theta(\mathbf{y} \mid \mathbf{x})] - D_{\mathrm{KL}}(q_\phi(\mathbf{x} \mid \mathbf{y}) \parallel p_x(\mathbf{x})) \tag{2.2}$$

where $D_{\mathrm{KL}}(\cdot \parallel \cdot)$ is the Kullback-Leibler divergence of two distributions. The first term is called the negative "reconstruction loss" and the second is called the "regularisation" for reasons below. This decomposition has let us rewrite the integration as a sum of expectations which can be estimated easily as described in Section 2.1.1. In addition, this type of decomposition will be equally useful when we describe the dynamic variant of the VAE.

2.1.1 The VAE algorithm

We can now describe the VAE algorithm. First, the ith data point from the nth minibatch is passed through a randomly initialised encoder neural network $d_\phi(\cdot)$ with parameters ϕ to output the parameters of the approximate posterior. A sample $\tilde{\mathbf{x}}_i \sim q_\phi(\mathbf{x}_i \mid \mathbf{y}_i)$ is drawn from this approximate posterior, encoding the data point onto the latent space. Then $\tilde{\mathbf{x}}_i$ is passed through a similar decoder neural network $d_\theta(\cdot)$ with parameters θ to generate a reconstruction $\hat{\mathbf{y}}_i$ according to the likelihood $p_\theta(\mathbf{y}_i \mid \mathbf{x})$. We discuss the neural network designs in Section 2.4. Note that this also forces the learnt encoding function to be invertible. Next a Monte Carlo estimate of the ELBO is calculated; since each observation $\mathbf{y}_i$ is uniquely mapped to a point $\mathbf{x}_i$ in latent space, we can further decompose the ELBO as a sum of ELBOs for each i, allowing us to use stochastic minibatch gradient ascent on the ELBO:

$$ELBO(\phi,\theta) \approx \frac{1}{M}\sum_{i=1}^{M}\left[\log p_\theta(\mathbf{y}_i \mid \tilde{\mathbf{x}}_i)\right] - D_{\mathrm{KL}}(q_\phi(\mathbf{x} \mid \mathbf{y}) \parallel p_x(\mathbf{x})) \tag{2.3}$$

where M is the minibatch size. It is common to let the posterior, prior and likelihood to all take Gaussian forms:

$$q_\phi(\mathbf{x}_i \mid \mathbf{y}_i) = \mathcal{N}(\mathbf{x}_i; \mu_\phi(\mathbf{y}_i), \mathrm{diag}(\sigma^2_\phi(\mathbf{y}_i))) \tag{2.4}$$

$$p_x(\mathbf{x}_i) = \mathcal{N}(\mathbf{x}_i; \mathbf{0}, \mathbf{I}) \tag{2.5}$$

$$p_\theta(\mathbf{y}_i \mid \mathbf{x}_i) = \mathcal{N}(\mathbf{y}_i; \mu_\theta(\mathbf{x}_i), \mathrm{diag}(\sigma^2_\theta(\mathbf{x}_i))) \tag{2.6}$$

where the encoder neural network $d_\phi(\cdot) = (\mu_\phi(\cdot), \sigma^2_\phi(\cdot))$ and the decoder neural network $d_\theta(\cdot) = \mu_\theta(\cdot), \sigma^2_\theta(\cdot)$. Then the loss function per minibatch is the negative ELBO, which becomes:

$$\mathscr{L}(\phi,\theta;\mathbf{y}) \approx \frac{1}{M}\sum_{i=1}^{M}\left[\|\mathbf{y}_i-\hat{\mathbf{y}}_i\|^2 - \frac{1}{2}\sum_{d=1}^{D}\left(1+\log[\sigma_\phi^2(\mathbf{y}_i)]_d - [\sigma_\phi^2(\mathbf{y}_i)]_d + [\mu_\phi(\mathbf{y}_i)]_d\right)\right] \quad (2.7)$$

Here we can clearly see that minimising the loss equates to both minimising the reconstruction error between the original and reconstructed data vectors, and regularising the learnt posterior distribution towards the prior, creating a meaningful latent space; the encoded variables are forced to seem distributed as a Gaussian. We see in later models that we can choose the prior differently to encode more meaningful representations. Finally, the parameters (ϕ,θ) are backpropagated using the stochastic gradients. This is called end-to-end training of the neural network parameters.

2.1.2 Sampling from the posterior

Following [16], we write the sampling stage of the posterior $\tilde{\mathbf{x}}_i \sim q_\phi(\mathbf{x} \mid \mathbf{y}_i)$ in the stochastic form $\tilde{\mathbf{x}}_i = \mu + \sigma\varepsilon$ where $\varepsilon \sim \mathscr{N}(0,1)$. This lets the Monte Carlo estimate of the expectation with respect to the posterior $\mathbb{E}_{q_\phi(\mathbf{x}|\mathbf{y})}[f(\mathbf{x})]$ be differentiable, which is necessary for training. See [16] for a proof of this "reparameterization trick".

2.2 Dynamic Variational Autoencoder

To introduce this model, we first define the dynamic model in which the latent variables are evolving with time. Then we introduce the probabilistic model to learn the mapping from the data to the latent space which is analogous to the standard VAE. Similarly, we decompose the joint to show how the model is trained.

2.2.1 State space model

As mentioned in Section 1.3.2, we assume a continuous linear time-evolving model for the latent space. We let latent states be u that evolve linearly with parameters Λ and Gaussian noise and emit *pseudo-observations* x also linearly with parameters v and Gaussian noise. That is, we have a linear Gaussian state space model (LGSSM):

$$p_\Lambda(\mathbf{u}_t \mid \mathbf{u}_{t-1}) = \mathscr{N}(\mathbf{u}_t; \mathbf{F}_\Lambda \mathbf{u}_{t-1}, \mathbf{Q}) \quad (2.8)$$

$$p_v(\mathbf{x}_t \mid \mathbf{u}_{t-1}) = \mathscr{N}(\mathbf{x}_t; \mathbf{C}_v \mathbf{u}_t, \mathbf{R}) \quad (2.9)$$

where Equation (2.8) is the dynamics model and Equation (2.9) is the pseudo-observation model. If we do not fix the model parameters $\mathbf{F}_\Lambda, \mathbf{C}_\nu$, we can set these parameters to be jointly learned by the neural network training as in [9], and $\mathbf{Q}, \mathbf{R}$ are to be chosen. We consider this case in experiments on the KVAE. Eventually, as we will see in Section 2.3.3, in our model, the transition parameters $\mathbf{F}_\Lambda, \mathbf{Q}$ will be fixed by the *physics-informed* nature of our model, and the emission parameters $\mathbf{C}_\nu, \mathbf{R}$ are to be chosen.

2.2.2 Probabilistic model

The dynamic VAE follows from the standard VAE, except our latent states are now $\mathbf{u}$ and each data point is a time sequence over $\{1:T\}$, as presented in [9]. Recall our entire probabilistic model,

$$\begin{aligned} \mathbf{u}_t \mid \mathbf{u}_{t-1} &\sim p_\Lambda(\mathbf{u}_t \mid \mathbf{u}_{t-1}) && \text{(Transition)} \\ \mathbf{x}_t \mid \mathbf{u}_t &\sim p_\nu(\mathbf{x}_t \mid \mathbf{u}_t) && \text{(Pseudo-observation)} \\ \mathbf{y}_t \mid \mathbf{x}_t &\sim p_\theta(\mathbf{y}_t \mid \mathbf{x}_t) && \text{(Observation)} \end{aligned}$$

for $t \geq 1$ and $p_\Lambda(\mathbf{u}_1 \mid \mathbf{u}_0) := p(\mathbf{u}_1)$, where this initial condition depends on the dataset at hand. As above, we need the evidence of the data over all time steps, which follows the derivation in Equation (2.1) except we also need to marginalise over the latent spaces $\mathbf{u}_{1:T}$:

$$\begin{aligned} \log p(\mathbf{y}_{1:T}) &= \log \int p(\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{y}_{1:T}) \mathrm{d}\mathbf{x}_{1:T} \mathrm{d}\mathbf{u}_{1:T}, \\ &= \log \int \frac{p(\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{y}_{1:T})}{q(\mathbf{u}_{1:T}, \mathbf{x}_{1:T} | \mathbf{y}_{1:T})} q(\mathbf{u}_{1:T}, \mathbf{x}_{1:T} | \mathbf{y}_{1:T}) \mathrm{d}\mathbf{x}_{1:T} \mathrm{d}\mathbf{u}_{1:T}, \\ &\geq \int q(\mathbf{u}_{1:T}, \mathbf{x}_{1:T} | \mathbf{y}_{1:T}) \log \frac{p(\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{y}_{1:T})}{q(\mathbf{u}_{1:T}, \mathbf{x}_{1:T} | \mathbf{y}_{1:T})} \mathrm{d}\mathbf{x}_{1:T} \mathrm{d}\mathbf{u}_{1:T}, \\ &= \int q(\mathbf{u}_{1:T}, \mathbf{x}_{1:T} | \mathbf{y}_{1:T}) \log \frac{p_\theta(\mathbf{y}_{1:T} | \mathbf{x}_{1:T}) p_\nu(\mathbf{x}_{1:T} | \mathbf{u}_{1:T}) p_\Lambda(\mathbf{u}_{1:T})}{q(\mathbf{u}_{1:T}, \mathbf{x}_{1:T} | \mathbf{y}_{1:T})} \mathrm{d}\mathbf{x}_{1:T} \mathrm{d}\mathbf{u}_{1:T} \\ &= \int p_{\Lambda,\nu}(\mathbf{u}_{1:T} | \mathbf{x}_{1:T}) \prod_{t=1}^{T} q_\phi(\mathbf{x}_t | \mathbf{y}_t) \log \frac{p_\theta(\mathbf{y}_{1:T} | \mathbf{x}_{1:T}) p_\nu(\mathbf{x}_{1:T} | \mathbf{u}_{1:T}) p_\Lambda(\mathbf{u}_{1:T})}{p_{\Lambda,\nu}(\mathbf{u}_{1:T} | \mathbf{x}_{1:T}) \prod_{t=1}^{T} q_\phi(\mathbf{x}_t | \mathbf{y}_t)} \mathrm{d}\mathbf{x}_{1:T} \mathrm{d}\mathbf{u}_{1:T} \\ &= \mathbb{E}_{q_\phi(\mathbf{x}_{1:T} | \mathbf{y}_{1:T})} \left[\log \frac{p_\theta(\mathbf{y}_{1:T} | \mathbf{x}_{1:T})}{q_\phi(\mathbf{x}_{1:T} | \mathbf{y}_{1:T})} + \mathbb{E}_{p_{\Lambda,\nu}(\mathbf{u}_{1:T} | \mathbf{x}_{1:T})} \left[\log \frac{p_\nu(\mathbf{x}_{1:T} | \mathbf{u}_{1:T}) p_\Lambda(\mathbf{u}_{1:T})}{p_{\Lambda,\nu}(\mathbf{u}_{1:T} | \mathbf{x}_{1:T})} \right] \right] \\ &= ELBO(\theta, \phi, \Lambda, \nu) \end{aligned} \tag{2.10}$$

∎

where in Equation (2.10) we have substituted in our variational family for the posterior,

$$q(\mathbf{u}_{1:T}, \mathbf{x}_{1:T}|\mathbf{y}_{1:T}) := p_{\Lambda,\nu}(\mathbf{u}_{1:T}|\mathbf{x}_{1:T}) \prod_{t=1}^{T} q_\phi(\mathbf{x}_t|\mathbf{y}_t).$$

$p_{\Lambda,\nu}(\mathbf{u}_{1:T}|\mathbf{x}_{1:T})$ is called the filtering distribution and depends on the SSM dynamics, hence why we subscript by both the transition and emission parameters. We describe how to compute it in Section 2.2.4. $q_\phi(\mathbf{x}_t|\mathbf{y}_t)$ is the analogue to the approximate posterior in the original VAE in Section 2.1, where time steps are independent. As before, the expectations can be computed using a Monte Carlo estimate using samples drawn from the inference network approximate posterior $\tilde{\mathbf{x}}_{1:T} \sim q_\phi(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ and samples drawn from the exact filtering posterior $\tilde{\mathbf{u}}_{1:T} \sim p_{\Lambda,\nu}(\mathbf{u}_{1:T}|\tilde{\mathbf{x}}_{1:T})$. The stochastic ELBO is then

$$\begin{aligned} ELBO(\theta,\phi,\Lambda,\nu) \approx \frac{1}{M}\sum_{i=1}^{M} \big[&\log p_\theta(\mathbf{y}_{1:T,i}|\tilde{\mathbf{x}}_{1:T,i}) - \log q_\phi(\tilde{\mathbf{x}}_{1:T,i}|\mathbf{y}_{1:T,i}) + \\ &\log p_\nu(\tilde{\mathbf{x}}_{1:T,i}|\tilde{\mathbf{u}}_{1:T,i}) + \log p_\Lambda(\tilde{\mathbf{u}}_{1:T,i}) - \log p_{\Lambda,\nu}(\tilde{\mathbf{u}}_{1:T,i}|\tilde{\mathbf{x}}_{1:T,i})\big] \end{aligned} \tag{2.11}$$

2.2.3 Overall training algorithm

The DVAE algorithm is an extension of the original VAE described in Section 2.1. For each minibatch n:

- Pass each ith data point $\mathbf{y}_{1:T,i}$ in the minibatch through the encoder neural network ϕ to obtain posterior $q_\phi(\mathbf{x}_{1:T}|\mathbf{y}_{1:T,i})$.
- Draw sample $\tilde{\mathbf{x}}_{1:T,i} \sim q_\phi(\mathbf{x}_{1:T}|\mathbf{y}_{1:T,i})$
- Pass $\tilde{\mathbf{x}}_{1:T,i}$ through decoder neural network θ to obtain reconstructions $\hat{\mathbf{y}}_{1:T,i}$.
- For fixed Λ, ν, pass $\tilde{\mathbf{x}}_{1:T,i}$ through "stochastic filter" (as in Section 2.2.4) to obtain the filtering posterior distribution, and draw samples $\tilde{\mathbf{u}}_{1:T,i} \sim p_{\Lambda,\nu}(\mathbf{u}_{1:T}|\tilde{\mathbf{x}}_{1:T,i})$. Note that in our experiments we also use the mean of this distribution, $\mu_{1:T,i}$ to visualise the "latent space" variables corresponding to the observations.
- Construct loss used for stochastic gradient descent, which is the negative ELBO from Equation (2.11).
- Update parameters (ϕ, θ) using stochastic gradients. If learning the dynamics too, as we will do in Section 3.2, also update parameters (Λ, ν) using stochastic gradients.

2.2.4 Kalman Filter

Since, from Section 2.2.1, we have assumed a linear Gaussian SSM, our filtering posterior $p_{\Lambda,\nu}(\mathbf{u}_{1:T}|\tilde{\mathbf{x}}_{1:T})$ can be calculated in closed form as a Gaussian distribution using the Kalman Filter recursion equations. Furthermore, since, in our setup, we can do offline inference, we can use the Kalman Smoother [22] which propagates in a similar fashion backwards in time, to achieve smoother results. We do not repeat here the Kalman filtering or smoothing equations since they are considered classical; see [20] for an introduction.

Intuitively, the purpose of this stochastic filter is to combine, in a Bayesian sense, incoming observations with the prior knowledge of the latent SSM distributions to obtain the latent posteriors.

2.3 Towards physics-informed dynamic latent spaces

So far, we have set the spatiotemporal dynamics of the latent space to vary linearly with parameters Λ with Gaussian noise, from Equation (2.8), and we have inferred the latent variables in this SSM given data using a Kalman Filter available in closed form. In our problem, we are given the physical dynamics that governs the latent space variables, and we wish to set $\Lambda = \{\mathbf{F}_\Lambda\}$ which embodies the discrete dynamics which govern the latent space.

As a simple example, consider a situation in which the dynamics of the latent space $(\mathbf{u}_{t_n})_{t_n\geq 0, n\geq 0}$ should represent discrete noisy 2D circular motion, where the noise is Gaussian and uncorrelated. Then the discrete-time dynamics can be expressed as

$$\begin{bmatrix} u_{t_n}^{(1)} \\ u_{t_n}^{(2)} \end{bmatrix} = \mathrm{Rot}(\omega\Delta_t) \begin{bmatrix} u_{t_{n-1}}^{(1)} \\ u_{t_{n-1}}^{(2)} \end{bmatrix} + \mathbf{Q}\varepsilon, \quad \mathrm{Rot}(x) = \begin{bmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{bmatrix} \tag{2.12}$$

where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{2\times 2})$, ω is constant and $\Delta_t = t_n - t_{n-1}$ is the time discretisation constant, fixed for every n. Then we can set $\mathbf{F}_\Lambda = \mathrm{Rot}(\omega\Delta_t), \mathbf{Q} = q\mathbf{I}$ in Equation (2.8) which sets our prior knowledge of the dynamics.

In the following sections we consider similar scenarios in which physics can be represented with linear stochastic partial differential equations (SPDEs).

2.3.1 Stochastic Partial Differential Equations

We assume that the spatiotemporal latent space process $(u_t)_{t\geq 0}$ can be described by a continuous linear spatiotemporal SPDE of arbitrary order, that is, from [6, Section A.2]:

$$\partial_t u + L_\Lambda u = f + \xi, \quad \xi \sim \mathcal{GP}(0, k(x,x') \cdot \delta(t-t')) \tag{2.13}$$

where $u := u(x,t), f := f(x), x \in \Omega \subset \mathbb{R}^d, t \in [0,T], L_\Lambda$ is any order linear spatial differential operator, and ∂_t is any linear time differential operator. The stochastic forcing term ξ is modelled as a zero-mean Gaussian process (GP), which can be seen as an infinite-dimensional noise term, and induces a probability measure over the solution space. For simplicity, the covariance operator of the GP is chosen to be separable: white noise in time, and governed spatially by $k(\cdot,\cdot)$. For example, two forms of Equation (2.13) which are of relevance are given below where $d = 1$. We study the boundary and initial conditions in experiments in Section 3.3.1.

- The 1D wave equation, where c is the wave speed

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = f + \xi \tag{2.14}$$

  We study this equation in detail in our experiments.

- The 1D heat equation, where α is the thermal diffusivity

$$\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} = f + \xi \tag{2.15}$$

In this model, as in Equation (2.9), the latent space u emits pseudo-observations x linearly, which is the same data-generating process as studied in [6].

2.3.2 Solving SPDEs with statFEM

Now, given the above equations, we wish to solve the SPDEs to obtain a probabilistic forward transition model of the latent variables in discrete space and time. The statistical finite elements method [6] allows us to proceed. Firstly, following classical finite elements theory, we discretise the domain x. We consider solutions u in an infinite-dimensional Sobolev space with basis $\{\psi_i\}_i$ and multiply the SPDE by test functions v from the same space to obtain the weak form of the SPDE. We then project u and v onto finite-dimensional subsets of the basis ψ to obtain finite elements u_h, v_h. The construction of the finite element mesh on which these functions are defined is described in Section 3.3.1. This gives a stochastic ODE in terms of the finite-dimensional time-varying spatial vector $\mathbf{u}(t) = (u_1(t), u_2(t), \ldots)^\intercal = (u(x_1,t), u(x_2,t), \ldots)^\intercal$:

$$\mathbf{M}\mathrm{d}\mathbf{u} + \mathbf{A}_\Lambda \mathbf{u}\mathrm{d}t = \mathbf{b}\mathrm{d}t + \mathrm{d}\beta_t \tag{2.16}$$

where $\mathbf{M}, \mathbf{A}_\Lambda, \mathbf{b}$ are finite-dimensional matrices generated from finite elements, and can be computed using a numerical PDE solver program. The derivation follows [6]. β_t is a Brownian motion process with diffusion matrix $\mathbf{G}$, where $\mathbf{G}$ depends on $k(\cdot,\cdot)$. For simplicity, we take $k(\cdot,\cdot) = \beta^2 = 0.05^2$ and use the lumped approximation described in [2], which gives $\mathbf{G} = \mathrm{diag}(\{G_{ii}\}), G_{ii} = \beta^2 \sum_j M_{ij}$.

Next, we discretise time using any discretisation method to obtain $\mathbf{u}_n := \mathbf{u}_{t_n} = \mathbf{u}(n\Delta_t)$ in terms of the previous $\mathbf{u}_{n-1}$. For example, with explicit Euler discretisation, we obtain

$$\mathbf{M}(\mathbf{u}_n - \mathbf{u}_{n-1}) + \Delta_t \mathbf{A}\mathbf{u}_{n-1} = \Delta_t \mathbf{b} + \mathbf{e}_{n-1}, \quad \mathbf{e}_{n-1} \sim \mathcal{N}(\mathbf{0}, \Delta_t \mathbf{G}) \tag{2.17}$$

Then, since the discretised equation still contains a stochastic term, we can solve for the transition model

$$p_\Lambda(\mathbf{u}_n|\mathbf{u}_{n-1}) = \mathcal{N}(\mathbf{u}_n; \mathbf{F}_\Lambda \mathbf{u}_{n-1} + \mathbf{f}, \mathbf{Q}) \tag{2.18}$$

$$\mathbf{F}_\Lambda = \mathbf{I} - \Delta_t \mathbf{M}^{-1}\mathbf{A} \tag{2.19}$$

$$\mathbf{f} = \Delta_t \mathbf{M}^{-1}\mathbf{b}$$

$$\mathbf{Q} = \Delta_t \mathbf{M}^{-1}\mathbf{G}\mathbf{M}^{-\top} \tag{2.20}$$

Alternative discretisation methods are presented in Appendix A.1. Empirically we find that the Crank-Nicholson method provides the best stability.

2.3.3 The statFEM-KVAE model

From above, the solution to the SPDE governing the latent space with the given assumptions yields a Gaussian transition distribution. In this project we only consider scenarios where $\mathbf{f} = 0$, so the solution is the exact form of the LGSSM in Equation (2.8). We can therefore "drop in" the parameters $\mathbf{F}_\Lambda, \mathbf{Q}$ from this solution when we are performing inference in the dynamic VAE, and use the algorithm from Section 2.2.3 to train. The ELBO terms in Equation (2.11) are as follows, where we drop the subscript i for clarity:

- $\log p_\theta(\mathbf{y}_{1:T}|\tilde{\mathbf{x}}_{1:T}) = \|\mathbf{y}_{1:T} - \hat{\mathbf{y}}_{1:T}\|^2$

- $\log q_\phi(\tilde{\mathbf{x}}_{1:T}|\mathbf{y}_{1:T}) = \log \mathcal{N}(\tilde{\mathbf{x}}_{1:T}; \mu_\phi(\mathbf{y}_{1:T}), \text{diag}(\sigma^2_\phi(\mathbf{y}_{1:T})))$
- $\log p_\nu(\tilde{\mathbf{x}}_{1:T}|\tilde{\mathbf{u}}_{1:T}) = \log \mathcal{N}(\tilde{\mathbf{x}}_{1:T}; \mathbf{C}_\nu \tilde{\mathbf{u}}_{1:T}, \mathbf{R})$
- $\log p_\Lambda(\tilde{\mathbf{u}}_{1:T}) = \sum_{t=2}^{T} \log \mathcal{N}(\tilde{\mathbf{u}}_t; \mathbf{F}_\Lambda \tilde{\mathbf{u}}_{t-1}, \mathbf{Q}) + \log \mathcal{N}(\tilde{\mathbf{u}}_1; \mu_1, q_1 \mathbf{I})$
- $\log p_{\Lambda,\nu}(\tilde{\mathbf{u}}_{1:T}|\tilde{\mathbf{x}}_{1:T})$ following the Kalman posterior in Section 2.2.4.

2.3.4 Learning the wave speed

One further addition to the model is learning certain parameters of the PDE from data if they are not specified. For example, in the wave equation example, if the wave speed c is unknown, we can factor out c while solving the PDE and set it as a learnable parameter. First, since the differential operator in the wave equation example $L_\Lambda = -c^2 \frac{\partial^2}{\partial x^2}$, we can replace $\mathbf{A}_\Lambda$ with $c\tilde{\mathbf{A}}_\Lambda$ in the calculation of $\mathbf{F}_\Lambda$, where $\tilde{\mathbf{A}}_\Lambda$ is computed with $c = 1$. Next, we set c as a parameter to be learned during end-to-end training of the neural networks.

2.4 Encoder-decoder design

As mentioned in Section 2.1.1, the encoder and decoder functions define respectively the dimensionality reduction and dimensionality expansion functions $d_\phi(\cdot), d_\theta(\cdot)$. Specifically, the encoder function takes high-dimensional data samples and produces a lower rank latent representation, and the decoder function takes latent samples and produces original-dimension reconstructions of the data. As with all autoencoder models, we use neural networks to model these functions. Neural networks allow us to model highly non-linear transformations, train parameters using backpropagation and provide good generalisation properties. There is a huge variety of neural networks of varying complexity to model image data.

In this project, the multi-layer perceptron (MLP) and the convolutional neural network (CNN) are both studied. We only consider data that consists of grayscale images in $(0, 1]$ or sequences of such images. The networks must be designed with the following requirements: the encoder output $\mu_\phi, \log \sigma^2_\phi$ is required to take any value, while the decoder output $\hat{\mathbf{y}}$ is required to be in $(0, 1]$. In the below sections, we assume 32×32 images (i.e. dimension 1024) and pseudo-observations of dimension 32. For simplicity, we don't use additional modern techniques such as dropout or batch normalisation, since we note that generalisation results are already quite good.

2.4.1 Multilayer Perceptron

An MLP is the simplest form of modern feedforward artificial neural network and consists solely of fully-connected layers and non-linear activation functions. Example MLPs used for encoding and decoding are described in Tables 2.1 and 2.2. As a special case, if $f_{1,h}, f_{1,o}, f_{2,h}, f_{2,o}$ are chosen to be identity activation functions then the networks just become simple linear transformations. As an example MLP for use in the VAE, we take $f_{1,h}, f_{2,h}$ as ReLU, $f_{1,o}$ as identity, and $f_{2,o}$ as a Sigmoid.

Layer	Activation	dim in	dim out
FC	$f_{1,h}(\cdot)$	1024	1024
FC	$f_{1,o}(\cdot)$	1024	32

Table 2.1 MLP with a single hidden layer for dimensionality reduction. Two of these are needed: one that outputs the mean and one for the variance $\mu, \log \sigma^2$.

Layer	Activation	dim in	dim out
FC	$f_{2,h}(\cdot)$	32	32
FC	$f_{2,o}(\cdot)$	32	1024

Table 2.2 MLP with a single hidden layer for dimensionality expansion.

2.4.2 Convolutional Neural Network

A CNN contains convolutional layers to extract spatial dependencies from a 2D image, rather than from a flattened vector for a MLP. A CNN can be seen as a series of downsampling steps to reduce the dimensionality of the data. A fully-connected layer can be appended to push the extracted features into the correct size output vectors. For the decoder, we use an equivalent architecture but with transposed convolutional layers to upsample from the latent space to the reconstruction output. Examples are shown in Tables 2.3 and 2.4. Activation functions are same as in Section 2.4.1.

Layer	Activation	dim in	dim out	channels
Conv($k=3, s=2$)	ReLU	32×32	15×15	32
Conv($k=3, s=2$)	ReLU	15×15	7×7	32
Conv($k=3, s=2$)	ReLU	7×7	3×3	32
Flatten		3×3	288	1
FC	$f_{1,o}(\cdot)$	288	32	

Table 2.3 CNN with 3 hidden convolutional layers for dimensionality reduction, where k is the kernel size and s is the stride. The input data has one channel (grayscale). The final FC is repeated for outputting both the 32d vectors $\mu, \log \sigma^2$.

Layer	Activation	dim in	dim out	channels
FC	Identity	32	288	
Unflatten		288	3×3	32
ConvT($k=3, s=2$)	ReLU	3×3	7×7	32
ConvT($k=3, s=2$)	ReLU	7×7	15×15	32
ConvT($k=3, s=2, p=1$)	ReLU	15×15	32×32	32
ConvT($k=1, s=1$)	$f_{2,o}(\cdot)$	32×32	32×32	1

Table 2.4 CNN with 4 hidden transposed convolutional layers for upsampling, where p is output padding.

Chapter 3

Experiments

We report experiments that have been designed to evaluate the models discussed in Section 2. Results of the experiments are reported in Section 4. Each of the following experimental setups builds upon previous ones, unless specified. All experimental code is in the private repository https://github.com/Andrewwango/statfem-vae; please request access.

3.1 Experiment: the static VAE

This experiment tests the use of the most basic standard, non-dynamic VAE of Section 2.1. We want to show that the VAE gives a meaningful latent space. A meaningful latent space means that data points that are spatially close together in the observation space will be close together in the latent space. This means that if we sample a point in the latent space near given points, we should be able to generate new data that looks indistinguishable from the training data.

In this experimental setup, the encoder and decoder neural networks are CNNs as per Section 2.4.2. We test using different latent space dimensions to show the effect of the VAE "bottleneck".

3.1.1 Datasets

First, we evaluate the VAE on the MNIST dataset as was done by [16]. We use a subset of 5000 training and 1000 test images of handwritten digits from the MNIST dataset. Each image is 28×28 binary pixels $\in \{0, 1\}$ giving an observation dimension of 784. Since this is not the original 32×32 as assumed in Section 2.4.2, we adjust the CNN design slightly, by adding an extra output padding to the first transposed convolutional layer in the decoder in Table 2.4. We throw away the annotated labels of the digits. We expect the VAE to learn a representation

where the same digits are close together in a low-dimensional latent space, so that one can smoothly interpolate between digits.

We also test with a very simple toy synthetic *stationary ball* dataset, as we will use this in the next section where we move the ball around. This consists of a dataset of 5000 training and 1000 test 32×32 binary images of a small white ball on a black background. Further to above, we expect the VAE to learn how to detect the position of the ball, since similar images represent the ball in each image being close to each other.

3.1.2 Evaluation

We report ELBO during training which shows convergence of the reconstruction and the regularisation, and use this a metric to show success of the VAE training. The ELBO on the test set is also used to measure generalisation of the model. We use two main visualisations that show qualitatively the success of the trained VAE in modelling latent space representations, which we will also use for evaluating subsequent experiments.

- Reconstruction: test images are forward passed through the model's "bottleneck" and compared to the original image;
- Latent manifold: samples are taken from the latent space and their decoded images are compared based on their latent positions. We also study the effects of altering the latent space dimension "bottleneck" on the ability of the VAE to produce efficient representations and generate similar reconstructions.

3.1.3 Implementation details

The VAE is implemented in Python 3.9 using the PyTorch package, which allows easy implementation of neural network architectures and manipulation of vectors using PyTorch tensors.

3.1.4 Network training

The end-to-end neural network training is done using the Adam optimizer [15] using a learning rate of $\eta = 2\text{e}{-3}$ for 100 epochs. All training was done on a NVIDIA GeForce RTX 3090 GPU.

3.2 Experiment: the dynamic VAE

Next we test the dynamic VAE from Section 2.2, firstly without any knowledge of the physics. This is the situation in [9] - we do not presume any dynamics of the system. As discussed in 2.2.1, we set $\mathbf{Q} = 0.08\mathbf{I}, \mathbf{R} = 0.03\mathbf{I}$ and set $\mathbf{F}_\Lambda, \mathbf{C}_\nu$ to be learned at the same time as when ϕ, θ are learnt. In this experiment we are not interested in how well the model learns or resembles the real dynamics of the data; we test this in the next section. Rather, we test if, with some given dynamics, the model can infer some spatiotemporal structure in the data and have a meaningful latent space. A meaningful latent space further requires that means that data sequences in the observation space resembles an inferred trajectory in the latent space. We should also be able to sample a point in the latent space and, assuming correct dynamics, generate new sequences into the future.

Other hyperparameters are as follows: learning rate 8e$-$4, 60 epochs, and $\mu_1 = \mathbf{0}, q_1 = 20$ as the Kalman Filter initial state. We set the dimensions of the latent and pseudo-observation spaces to 4, to simulate capturing x and y positions and velocities.

3.2.1 Dataset

We use a toy synthetic *circular motion ball* dataset similar to the static ball dataset in Section 3.1.1, except now each data sample is a sequence of 40 frames showing the white ball travelling in circular motion about the centre at a fixed speed determined by the gravitational "force". The dataset is created using the Pymunk 2D rigid-body physics animation package.

3.2.2 Evaluation

Again, we report the training and test ELBO during training. Like in the static case, after training the model, we show the effectiveness of the VAE modelling by showing test image sequence reconstruction and inferred latent space sequence. Note the visualised latent space sequence is the mean of the Kalman posterior; furthermore, the covariance of the Kalman posterior can be used for uncertainty quantification. To test the model's dynamics and generation without filtering (i.e. without seeing data), we also show results where we sample from the latent space, forward propagate in time with the dynamics and decode the new sequence, which we call long-term generation.

3.3 Experiment: statFEM-KVAE with the wave equation

Next we use our full statFEM-KVAE model from Section 2.3.3, which is the same setup as the previous experiment, but we define the SSM parameters according to a physics-informed model, and use a synthetic dataset with the same dynamics. We train with 30 epochs, as since the model already has the dynamics defined, convergence is almost guaranteed.

3.3.1 Physics

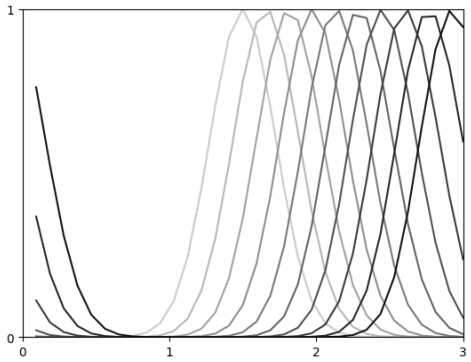
We would like to model data which we assume is generated from the 1D wave SPDE from Equation (2.14), where $f(x) = 0$, $c = 4$ and the spatial domain $\mathscr{D} = [0,3]$ in the same units as u. The PDE is solved numerically using the statFEM method following Section 2.3.2; spatial discretisation is performed using the FEniCS package [17], to obtain the matrices $\mathbf{M}, \mathbf{A}_\Lambda$. This is done as follows: first, a FEniCS function space object is defined on a uniform mesh with periodic boundary conditions. Then, trial and test function objects are taken from the function space. The matrices concerned are assembled using these objects according to their definitions in [6, Section A.2], where the the FEniCS assembling function takes care of the inner product.

We use the initial condition $u_0(x) = u(x,0) = \exp\left(-(x-m)^2/\ell\right), \ell = 0.1, m = 0.5$.

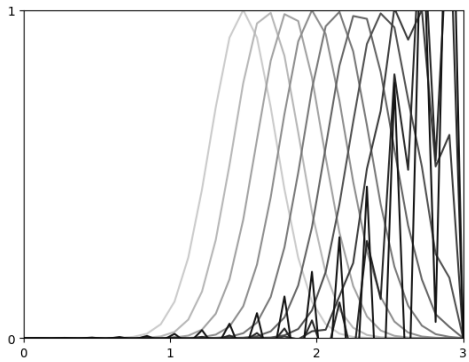
There are many choices to make regarding the PDE and its solution. We use and validate the following parameters and methods for the solution of the PDE, summarised in Figure 3.1a. We can validate them by forward simulating from the solution in the absence of noise, i.e. $\mathbf{u}_n = \mathbf{F}_\Lambda \mathbf{u}_{n-1}$, while varying these choices.

- We choose periodic boundary conditions on the domain boundary $\partial\mathscr{D}$, i.e. $u(0,t) = u(3,t), \frac{\partial u}{\partial x}(0,t) = \frac{\partial u}{\partial x}(3,t)$. This is preferable to the Dirichlet constant boundary conditions $u(0,t) = u(3,t) = 0$ or other fixed boundary conditions, as this leads to the integrator breaking down at the fixed boundaries; see Figure 3.1b.
- To perform time-discretisation, we use Crank-Nicholson discretisation, as it is an implicit method: it allows for larger time-steps while preserving stability than with explicit Euler [5]; see Figure 3.1c. See Appendix A.1 for the equations.
- For spatial discretisation, we choose $n_x = 32$ finite elements spaced evenly across the domain. Higher n_x would increase computation cost of model training, whereas a lower n_x results in a loss in accuracy; see Figure 3.1d.
- For time-discretisation, we choose $\Delta_t = 0.01$. Similar conclusion as above: a higher Δ_t gives worse results; see Figure 3.1e.

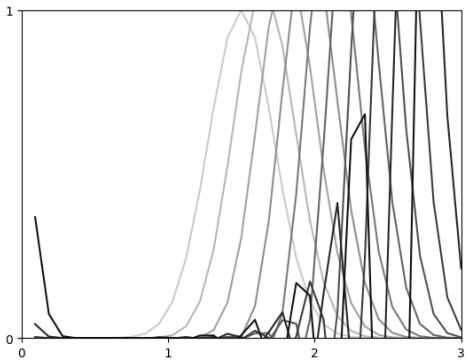
Finally we can calculate $\mathbf{F}_\Lambda, \mathbf{Q}$ using Equations (2.19), (2.20). We fix $\mathbf{C}_\nu$ to $\mathbf{I}_{32\times 32}$ for simplicity, so that both the pseudo-observation and latent spaces have dimension 32.



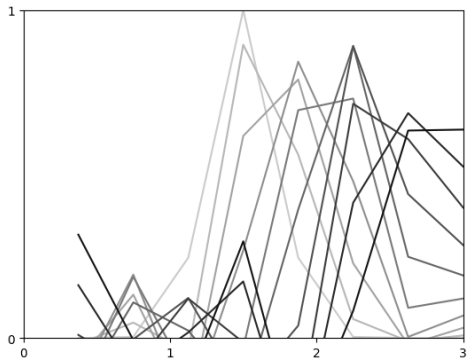
(a) Chosen scheme: periodic boundary conditions, Crank-Nicholson discretisation, $n_x = 32$, $\Delta_t = 0.01$



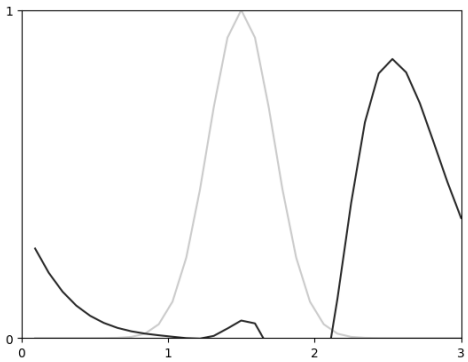
(b) Constant BCs



(c) Explicit Euler discretisation



(d) $n_x = 8$



(e) $\Delta_t = 0.08$

Fig. 3.1 Numerical solution to the wave PDE in the absence of noise for $\mathscr{D} = [0,3], t \in [0,\ldots]$. In the above, the y-axis is $u(x,t)$, the x-axis is x in metres, and light to dark shade represents forward time in seconds.

3.3.2 Dataset

The synthetic dataset used consists of image sequences generated from the solution of the wave PDE in the absence of noise. The dataset is constructed as follows, using the fact that the wave PDE fortunately already has an analytical solution as follows:

$$u(x,0) = u_0(x-ct), \quad u_0 = \exp\left(-(x-m)^2/\ell\right) \tag{3.1}$$

- Choose initial starting position $m \sim \mathscr{U}(0,1)$. Generate 1D wave solution for time $t = [0, 0.4)$ using Equation (3.1) giving a 32×40 matrix.
- Transform into 2D data sample $32 \times 32 \times 40$ using either of the following methods:
  - *wave_BEV*: copy 1D sample across 2nd dimension y, while shifting non-linearly according to a cosine, where i_x refers to the ith x element: $i_x \leftarrow i_x + A\cos((i_y + \phi)f), A = 3, f = 0.1, \phi = 0$. We note that the mapping from this dataset back to the PDE should be easily learnt by a single linear matrix: it simply needs to extract the 8th column. This dataset is therefore trivial.
  - *wave_side*: for every x, fill all y with $H(y - u(x,t))$, where H is the step function, to give a side-on view on the wave. We expect the inverse of this mapping should be harder to learn, since the wave shape has been truly mapped to a spatially dependent 2D image. Since real-world datasets are likely to be complex, we use only this dataset in our experiments.
- Save $u(x,0)$ and set $\mu_1 = u(x,0), q_1 = 0.0002$ for every sample, showing confidence in the Kalman Filter initial state, to aid training and identifiability of the solution. Despite this being a unsupervised learning problem, having this sort of "ground-truth" in real-life situations is acceptable, since it is often possible to measure an initial value directly.
- Repeat to obtain 5000 training and 1000 test samples.

See Figure 3.2 for an example data sample.

3.3.3 Evaluation

We will again use the reconstruction of test sequences to show training success. However, now since we have set the dynamics, we expect the inferred latent space to not only be meaningful but correct: the spatial shape and temporal evolution of the latent state sequence should match

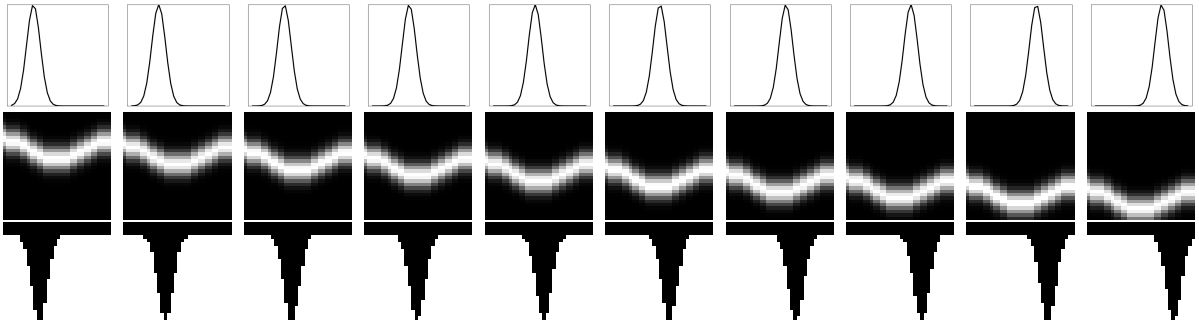


Fig. 3.2 Above: analytical solution $u(x,t)$ to wave PDE. Middle: *wave_BEV* dataset generated from the solution. Below: *wave_side* dataset.

up with the ground-truth generating the dataset, since both the dataset ("likelihood") and the dynamics ("prior") are already matched. Furthermore, by propagating forward in time with the PDE dynamics we expect to decode realistic, "new" generated reconstructions.

We will compare these results to the original KVAE, by applying the same dataset to the "uninformed" model, where we don't set the dynamical model and instead learn it. We still set the initial state of the Kalman Filter to be the wave equation initial condition, i.e. the ground truth, to help the model. Nevertheless, we expect the physics-informed model to perform better in the ways described above.

Finally, evaluating the capability of uncertainty quantification and calibration of misspecified models with data is beyond the scope of this project. However, since the statFEM-KVAE model is based on the data assimilation of the original statFEM, it follows naturally from the results shown in [6].

3.4 Experimental difficulties

Throughout the project work, all of the above implementations and experiments took several iterations to produce good results that matched what was expected from the theory. This was because there were a lot of small implementation details to be dealt with, for example:

- The dynamic VAE methodology has many hyperparameters to set, for example the emission noise, Kalman Filter initial state, transition noise to an extent, whose effect on the results was often difficult to interpret;
- Of course, network training hyperparameters such as learning rate and batch size which often had a big impact on training success;

- My original KVAE implementation was inspired by attached code from the KVAE model in [10], whose implementation was incorrect;
- The statFEM design choices, such as boundary conditions and time-discretisation methods (e.g. implicit Euler vs. Crank-Nicholson). This required a lot of experimenting, to get good dynamics from the given PDE.

These above details were complicated by the fact that often, incorrect setting of the hyperparameters would result in decent, but not perfect results anyway, for example due to the good capability of the CNN to model arbitrary non-linear transformations. See Section 4.3 for an example.

Chapter 4

Results and discussion

4.1 Experiment: the static VAE

Figure 4.3 shows that reconstruction images become clearer and more faithful as the dimension of the latent space increases. This is showing the effect of the VAE "bottleneck" - it becomes increasingly harder to model all digits on a smaller dimension latent space. However, the reconstructions are never perfect; artefacts can still be seen. The convergence of the ELBO in Figure 4.1 during training supports this. Intuitively, the VAE can converge to a better ELBO if it has more dimensions with which to model the data.

Figure 4.1 also shows the generalisation gap between training and test - checking that this gap does not become too big allows us to ensure that we still have good generalisation properties of the VAE.

Figure 4.2 shows the quality of the latent space learnt by the VAE in an unsupervised manner, as was done by [16]. We see that the latent space interpolates smoothly between digits and most digits are included in a tightly-packed neighbourhood. This regularising property of the VAE comes from the way we have decomposed the likelihood in Equation (2.1). Furthermore, Figure 4.4 shows that we have learnt a meaningful latent space, where images with balls close together are close in the latent space. We also note that the VAE has learnt an arbitrary orientation of the latent axes - the latent space seems to have primary axes on the diagonals. This shows non-identifiability of the solution.

4.2 Experiment: the dynamic VAE

Figures 4.5, 4.6 shows the success of the KVAE training, where both the dynamics and the encoder/decoder functions are to be learnt. However, the high quality reconstruction doesn't

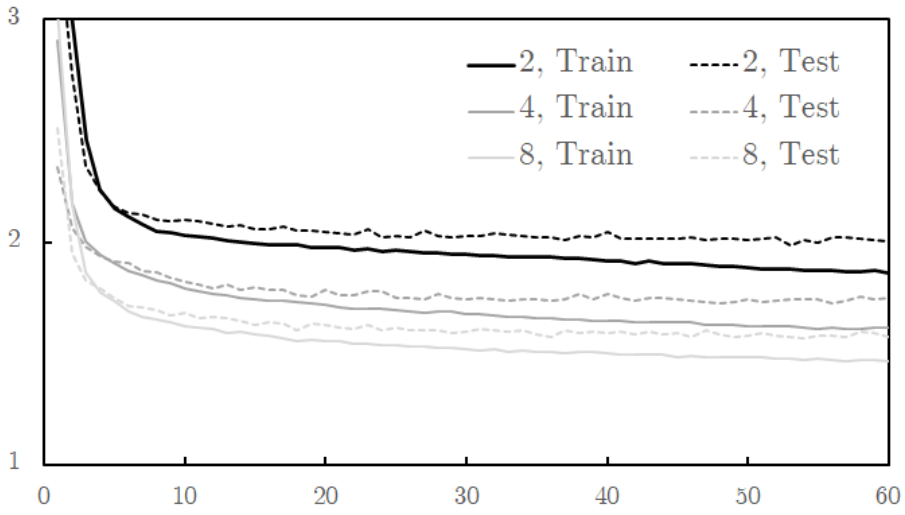


Fig. 4.1 Loss function vs epoch during training of VAE model on MNIST dataset, for different latent space dimensions: 2 (black), 4 (grey) and 8 (light grey).

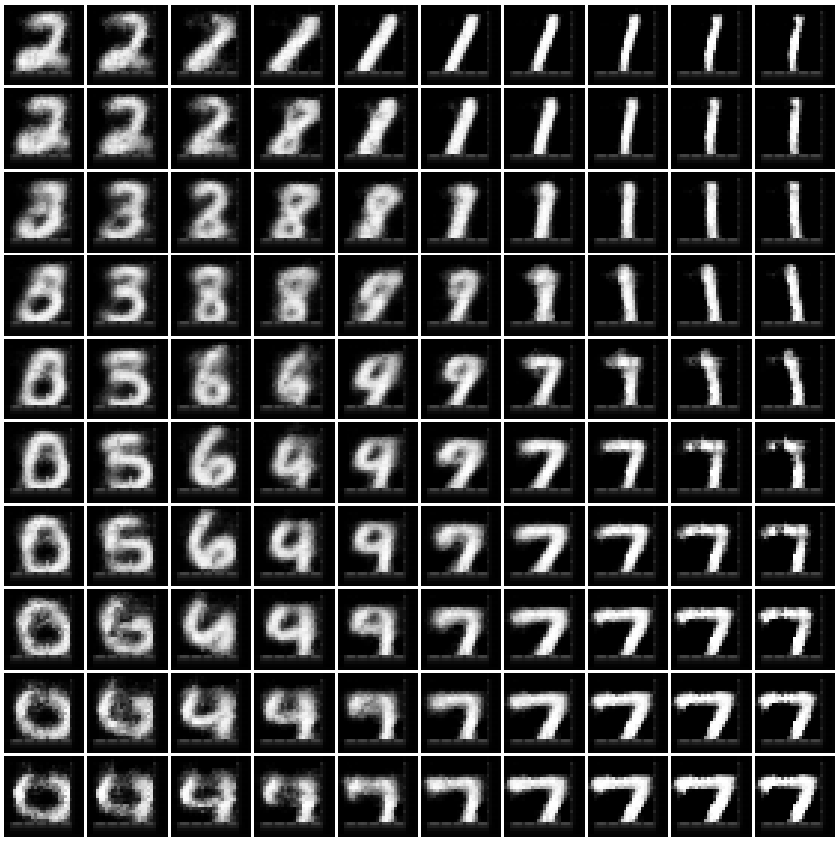
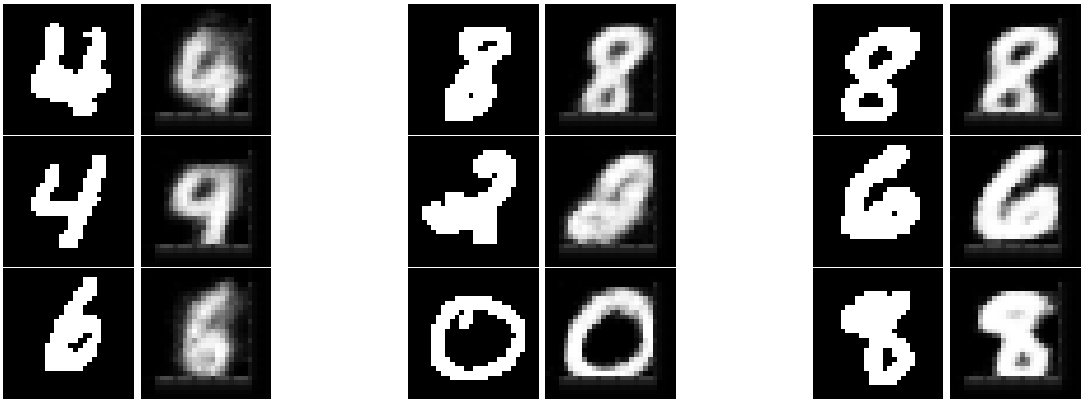


Fig. 4.2 2D latent space with latent samples taken uniformly over the space, labelled with the images generated from passing the samples through the decoder from the model trained on MNIST.



(a) Latent space dimension 2 (b) Latent space dimension 4 (c) Latent space dimension 8

Fig. 4.3 In each of 4.3a, 4.3b, 4.3c, left: random MNIST test images. Right: reconstruction of images after passing through model trained on MNIST.

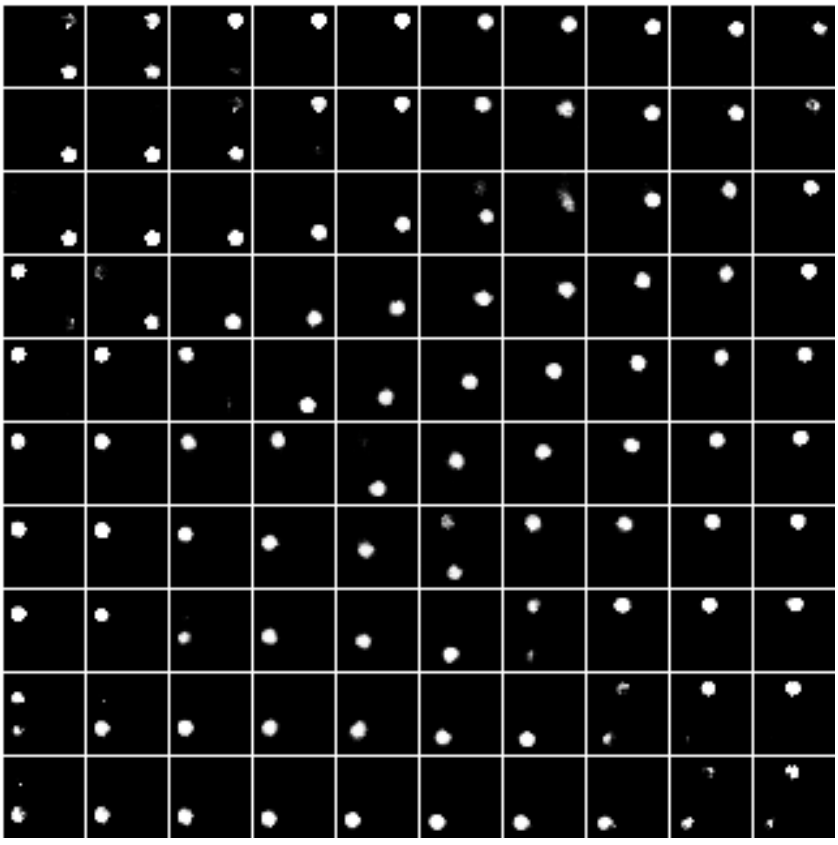


Fig. 4.4 2D latent space with latent samples taken uniformly over the space, labelled with the images generated from passing the samples through the decoder from the model trained on stationary ball dataset.

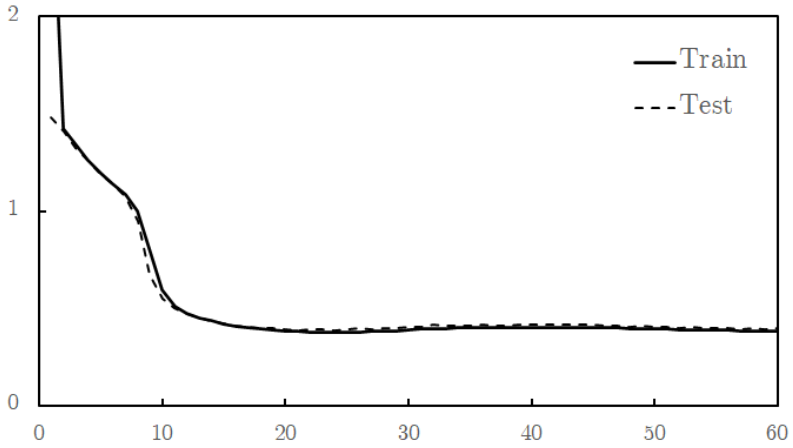


Fig. 4.5 Loss function vs epoch during training of dynamic KVAE model on *circular motion ball* dataset.

say anything about the quality of the latent space. We see from the middle 2 rows of Figure 4.6 that the VAE has learnt a meaningful latent space - consecutive images in circular motion result in the same pattern in the latent space. Less importantly, it seems like the circular motion dynamics have been learnt too. However, this is not the focus of our experiments, as in our statFEM-KVAE model, we eventually set the dynamics ourselves. We see that, although we can discern circular motion from the 4D latent space visualisation, we cannot say which dimensions represent the x- and y- positions and velocities, as we have only governed the latent space with an isotropic Gaussian. Indeed we cannot specify that we want certain dimensions to represent certain physical quantities without specifying the structure of the latent space.

The generation results in Figure 4.7 show that the circular motion dynamics have been correctly learnt. More importantly, it shows that the generative model functions correctly - the generated images show a ball following circular motion.

4.3 Experiment: statFEM-KVAE with the wave equation

In order to show the success of the statFEM-KVAE, we must show that the inferred latent space from the encoder is not only meaningful, as in the previous experiments, but also correct. This means that it should match the inherent dynamics that generated the data: the latent space (posterior) values should represent a combination of the data series and the prior dynamics. In Figure 4.9, we can clearly see that the inferred latent space evolves in a similar, but noisy fashion to the generating PDE - a 1D wave equation with Gaussian initial shape. This shows that we have assimilated the data likelihood and the physics prior; the latent space shows the

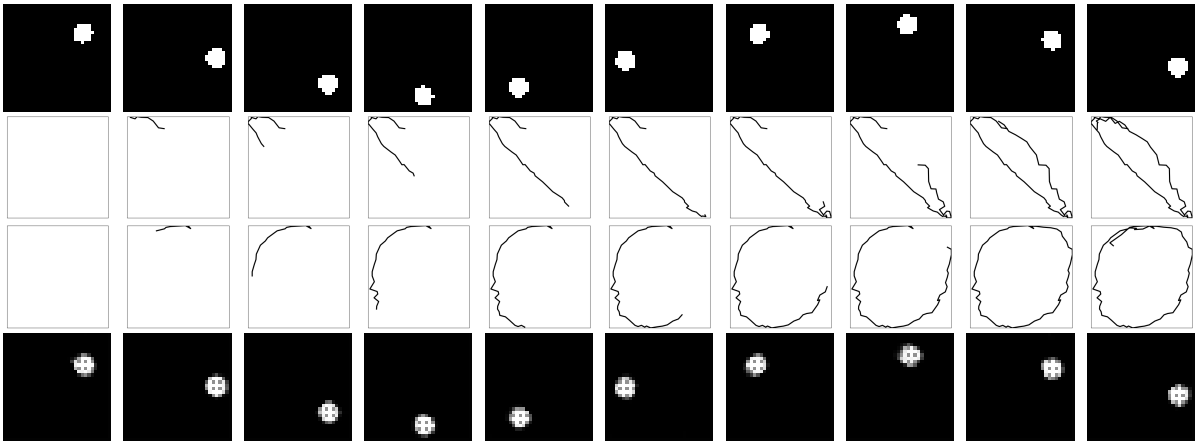


Fig. 4.6 Above: Test sample from the *circular motion ball* dataset (every 8th image in the series). Middle 2 rows: inferred 4D latent space using KVAE model trained on this dataset with no informed physics - row 1 shows dimensions 1 and 2, row 2 shows dimensions 3 and 4. Below: reconstruction through generative decoder.

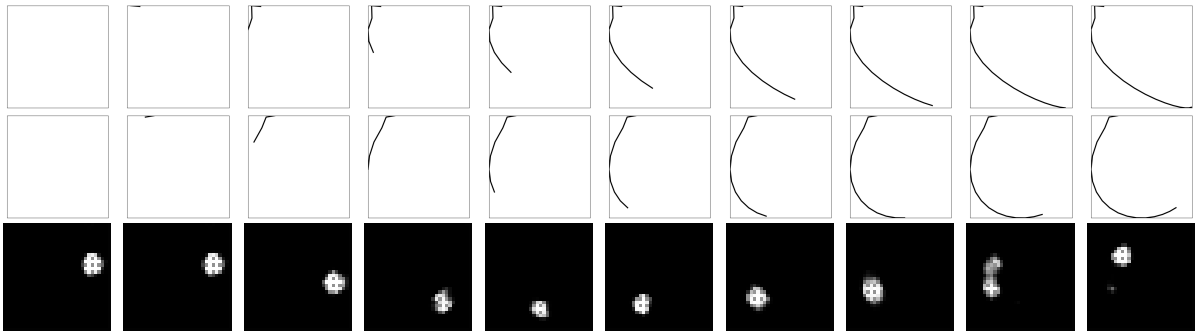


Fig. 4.7 Latent space forward generation. Above: first image: initial condition. Subsequently: forward generation - every 2nd timestep from learned circular motion dynamics. Below: after passing through generative decoder of the dynamic KVAE model.

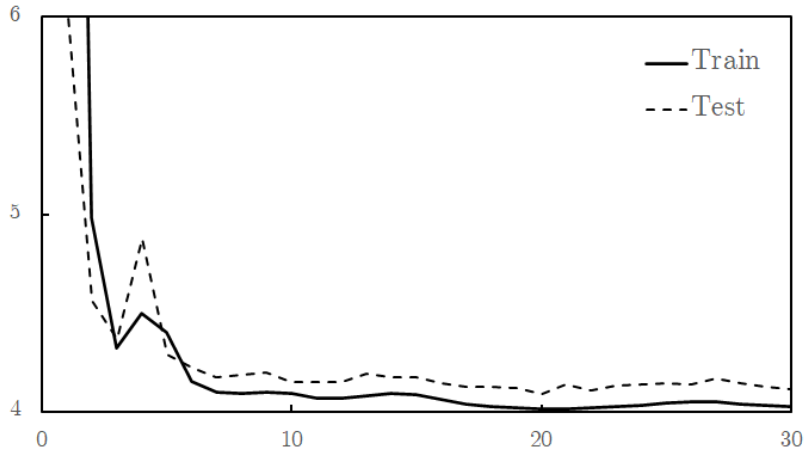


Fig. 4.8 Loss function vs epoch during training of statFEM-KVAE model on *wave_side* dataset.

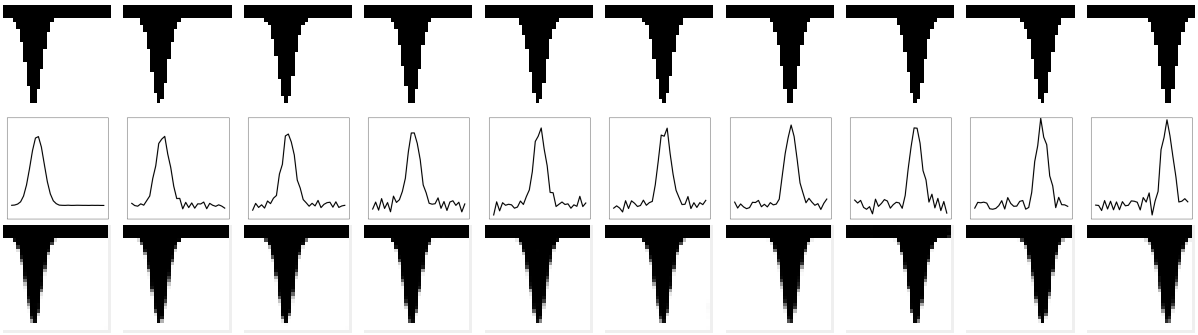


Fig. 4.9 Above: Test sample from the *wave_side* dataset (every 4th image in the series). Middle: inferred 32D latent space $\mathbf{u}(t)$ through time using statFEM-KVAE model trained on this dataset with latent space "informed" by wave PDE. Each point on the x-axis is the xth component of the 32D vector $\mathbf{u}(t)$. Below: reconstruction through generative decoder.

posterior mean. The latent space starts as the initial condition μ_1 that we have set, and gets progressively noisier through time, representing a build up of uncertainty. It is important to set μ_1 to the ground truth initial condition; otherwise, we lose identifiability of the solution. As before, the reconstruction in Figure 4.9 and the ELBO (Figure 4.8) merely show that the model has converged, not if the model is of any use.

The generation results in Figure 4.7 show that the generative model has learnt how to generate the whole space of admissible PDE solutions for the given PDE setup. That is, passing the propagated, new sequence through the generative model results in the expected wave images at each time step.

Furthermore we can compare these results to the training of the KVAE without the statFEM-informed latent space. Note that the data follows a linear DGP, so the uninformed KVAE should still be able to learn some linear dynamics. In the inferred latent space in Figure 4.11,

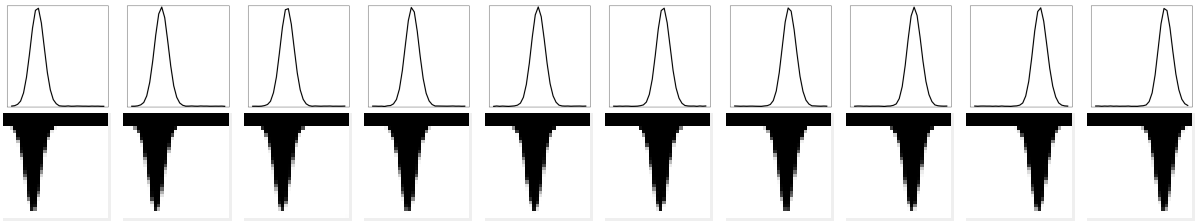


Fig. 4.10 Latent space forward generation. Above: first image: initial condition. Subsequently: forward generation - every 4th timestep from statFEM solution of wave PDE. Below: after passing through generative decoder of the statFEM-KVAE model.

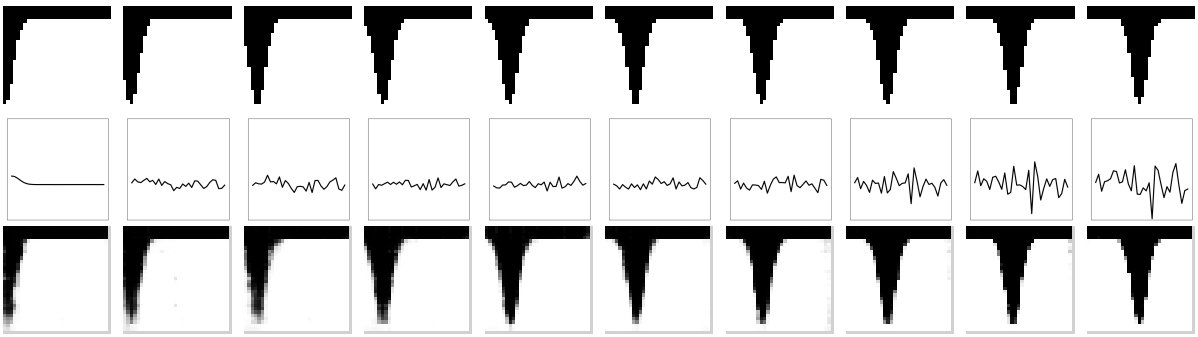


Fig. 4.11 Above: Test sample from the *wave_side* dataset (every 4th image in the series). Middle: inferred 32D latent space $\mathbf{u}(t)$ through time using uninformed KVAE model trained on this dataset. Each point on the x-axis is the xth component of the 32D vector $\mathbf{u}(t)$. Below: reconstruction through generative decoder.

the filtered variables bear no resemblance to the wave shape, even though the reconstruction images are very good. Because of the high performance of CNNs at approximating arbitrary non-linear transformations, the VAE seems to have learnt an excellent reconstruction even though the dynamics are not correct. This shows that having prior knowledge of the physics governing the dataset is beneficial in estimating models.

Chapter 5

Conclusions

Through this report, we have developed a model to that allows one to estimate physics models from real-world observed image data. We have achieved this using a generative probabilistic framework that models both the physics-informed nature of the dynamics as well as the spatial features of the image data. By approximating the posterior, we show that we can train the whole model using the evidence lower bound, guaranteeing a monotonic increase of the log-likelihood.

The dynamic variational autoencoding strategy is introduced as a way to learn low-dimensional representations of time sequence data. The Kalman VAE model is formulated such that the VAE encoding/decoding functions are separated from the posterior sequence inference. We also introduced the statistical finite element method to perform inference of a stochastic PDE in a coherent statistical framework. Then, we show that, in the linear case, the discrete dynamics of the KVAE latent space can be fixed to the discretised PDE solution given by statFEM.

Our model can therefore be seen as a way to provide statFEM with a way of doing synthesis of data and prior dynamics with deep encoder and decoders, in order to provide functionality for real-world image data. This can also be seen as providing a physics-informed capability in existing deep representational learning models.

In our experiments, we first show that the VAE and KVAE can estimate a meaningful latent space from data: the encoder and decoder models map the data to an intuitive low-dimensional space and vice versa. We show that each dimension of the inferred points in the latent space smoothly encodes some feature of the data - for example, the shape of a handwritten digit, or the position of the ball along a certain axis. However, since the latent space is only governed by an "uninformed" isotropic Gaussian, we cannot, for example, specify that we want the 1st and 2nd dimensions to respectively encode the moving ball's x and y coordinates. With the statFEM-KVAE model, the results show that we can learn the correct latent space mappings to the actual PDE solution. We show that a physics-informed latent space performs much

better than an uninformed one; the literature on learning governing dynamics is irrelevant if we already assume the dynamics of a system. This means that when we assume or know the underlying physics of a given dataset, as in a digital twins application, we can do the following tasks described in Section 1:

- Estimation: if we have complex observed data that is assumed to follow a certain spatiotemporal PDE, we can infer the PDE solution in the face of the data.
- Calibration: in the case of a filtering noisy, observed data in a misspecified model, the filtered latent space combines the two sources of information and two sources of uncertainty in a Bayesian manner. See [6, Figure 3] for a detailed example.
- Prediction: given the model trained with observed data, we can synthesise new image sequences by propagating with the statFEM solution and generating images with the decoder model.
- Uncertainty quantification: the data assimilation problem has been formulated in terms of a stochastic filter, which is a probability distribution. Then, naturally, estimates of the PDE solution have an associated variance (usually in a Gaussian distribution).

5.1 Future work

The basic statFEM-KVAE framework allows for plenty of future work, as follows:

- We have focussed on the linear case as this allows us to set the VAE's linear Gaussian state-space model parameters, and allows us to use the classical Kalman filter equations to do the sequence posterior inference. We can estimate non-linear PDEs using a non-linear state space and non-linear filtering techniques such as the extended Kalman filter, such as for the Korteweg-de Vries equation as studied in [6] for modelling tidal waves.
- We have used easily intepretable datasets such as the side-on view of a wave, which do not necessarily require particularly complex neural networks to extract the spatial features - as an example, the simplest wave dataset did not even require a neural network. However, real-world data may have a more complicated perspective or contain visual aberrations or other non-linearities. Modern deep CNNs are extremely capable and can learn highly non-linear and spatially complicated feature mappings.
- The general class of SPDEs that we have modelled means that we also have a high amount of flexibility in modelling the noise in the data. For simplicity, in our experiments, our

synthetic datasets have been noiseless. However, we can set prior belief in the type of noise by setting the covariance kernel of the stochastic forcing term in the SPDE appropriately.

- We have only studied 1D PDEs which have been discretised on a simple uniform finite element mesh. However, since the finite element method has been proven to be effective on complex multidimensional meshes, such as with triangular or hexagonal elements particularly in the field of structural modelling, our method based on statFEM readily extends to these cases.

5.2 Acknowledgements

This project was undertaken with the valuable help of colleagues at the Computational Statistics and Machine Learning laboratory, Professor Mark Girolami's group. **N.B.** in this project, I take the pronouns **we/us/our** to refer to work done solely and independently by the author, except for in the following cases. The author would like to thank:

- Deniz Akyildiz and Connor Duffin for their help with: explaining some of the advanced theory, such as variational inference, guidance with the derivation of the statFEM-KVAE model, checking my model results to what they'd expect would happen, suggesting hyperparameter changes to the model and pointing out mistakes that I'd made in my implementation;
- Connor Duffin for sharing his FEniCS code for solving linear PDEs from their statFEM paper [6], and suggesting design choices in the solver, such as changing the boundary conditions or changing the time-discretisation method;
- Alex Glyn-Davies for proof reading the report and suggesting improvements;
- Prof. Girolami for his direction, motivation and sanity checks in the project.

References

[1] AIAA Digital Engineering Integration Committee (2020). Digital Twin: Definition & Value. AIAA Position Paper.

[2] Akyildiz, O. D., Duffin, C., Sabanis, S., and Girolami, M. (2021). Statistical Finite Elements via Langevin Dynamics. *arXiv:2110.11131 [cs, math, stat]*. arXiv: 2110.11131.

[3] Bauer, P., Stevens, B., and Hazeleger, W. (2021). A digital twin of Earth for the green transition. *Nature Climate Change*, 11(2):80–83.

[4] Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., and Bengio, Y. (2016). A Recurrent Latent Variable Model for Sequential Data. *arXiv:1506.02216 [cs]*. arXiv: 1506.02216.

[5] Crank, J. and Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(1):50–67.

[6] Duffin, C., Cripps, E., Stemler, T., and Girolami, M. (2021). Statistical finite elements for misspecified models. *Proceedings of the National Academy of Sciences*, 118(2):e2015006118.

[7] Erichson, N. B., Muehlebach, M., and Mahoney, M. W. (2019). Physics-informed Autoencoders for Lyapunov-stable Fluid Flow Prediction. *arXiv:1905.10866 [physics]*. arXiv: 1905.10866.

[8] Febrianto, E., Butler, L., Girolami, M., and Cirak, F. (2021). Digital twinning of self-sensing structures using the statistical finite element method. *arXiv:2103.13729 [cs, math]*. arXiv: 2103.13729.

[9] Fraccaro, M., Kamronn, S., Paquet, U., and Winther, O. (2017). A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning. *arXiv:1710.05741 [cs, stat]*. arXiv: 1710.05741.

[10] Girin, L., Leglaive, S., Bie, X., Diard, J., Hueber, T., and Alameda-Pineda, X. (2021). Dynamical Variational Autoencoders: A Comprehensive Review. *Foundations and Trends® in Machine Learning*, 15(1-2):1–175. arXiv: 2008.12595.

[11] Girolami, M., Febrianto, E., Yin, G., and Cirak, F. (2021). The statistical finite element method (statFEM) for coherent synthesis of observation data and model predictions. *Computer Methods in Applied Mechanics and Engineering*, 375:113533.

[12] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*. arXiv: 1406.2661.

[13] Holloway, P. E., Pelinovsky, E., and Talipova, T. (1999). A generalized Korteweg-de Vries model of internal tide transformation in the coastal zone. *Journal of Geophysical Research: Oceans*, 104(C8):18333–18350.

[14] Karl, M., Soelch, M., Bayer, J., and van der Smagt, P. (2017). Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data. *arXiv:1605.06432 [cs, stat]*. arXiv: 1605.06432.

[15] Kingma, D. P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. arXiv: 1412.6980.

[16] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*. arXiv: 1312.6114.

[17] Logg, A., Mardal, K.-A., and Wells, G., editors (2012). *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[18] Lu, P. Y., Kim, S., and Soljačić, M. (2020). Extracting Interpretable Physical Parameters from Spatiotemporal Systems using Unsupervised Learning. *Physical Review X*, 10(3):031056. arXiv: 1907.06011 version: 3.

[19] Morton, J., Witherden, F. D., Jameson, A., and Kochenderfer, M. J. (2018). Deep Dynamical Modeling and Control of Unsteady Fluid Flows. *arXiv:1805.07472 [cs]*. arXiv: 1805.07472.

[20] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA.

[21] Niederer, S. A., Sacks, M. S., Girolami, M., and Willcox, K. (2021). Scaling digital twins from the artisanal to the industrial. *Nature Computational Science*, 1(5):313–320.

[22] Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450.

[23] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv:1401.4082 [cs, stat]*. arXiv: 1401.4082.

[24] Schwantes, C. R. and Pande, V. S. (2015). Modeling Molecular Kinetics with tICA and the Kernel Trick. *Journal of Chemical Theory and Computation*, 11(2):600–608.

[25] Tschannen, M., Bachem, O., and Lucic, M. (2018). Recent Advances in Autoencoder-Based Representation Learning. *arXiv:1812.05069 [cs, stat]*. arXiv: 1812.05069.

[26] Ward, R., Choudhary, R., Gregory, A., Jans-Singh, M., and Girolami, M. (2021). Continuous calibration of a digital twin: Comparison of particle filter and Bayesian calibration approaches. *Data-Centric Engineering*, 2:e15.

[27] Yang, L., Treichler, S., Kurth, T., Fischer, K., Barajas-Solano, D., Romero, J., Churavy, V., Tartakovsky, A., Houston, M., Prabhat, and Karniadakis, G. (2019). Highly-scalable, physics-informed GANs for learning solutions of stochastic PDEs. *arXiv:1910.13444 [physics, stat]*. arXiv: 1910.13444.

[28] Yang, L., Zhang, D., and Karniadakis, G. E. (2018). Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations. *arXiv:1811.02033 [cs, math, stat]*. arXiv: 1811.02033.

[29] Yildiz, C., Heinonen, M., and Lahdesmaki, H. (2019). ODE^{2}VAE: Deep generative second order ODEs with Bayesian neural networks. *arXiv:1905.10994 [cs, stat]*. arXiv: 1905.10994.

[30] Zhang, Z. and Karniadakis, G. E. (2017). *Numerical Methods for Stochastic Partial Differential Equations with White Noise*, volume 196 of *Applied Mathematical Sciences*. Springer International Publishing, Cham.

[31] Zhong, W. and Meidani, H. (2022). PI-VAE: Physics-Informed Variational Auto-Encoder for stochastic differential equations. *arXiv:2203.11363 [cs, stat]*. arXiv: 2203.11363.

Appendix A

A.1 PDE discretisation methods

In the following, take $\mathbf{e}_{n-1} \sim \mathscr{N}(\mathbf{0}, \Delta_t \mathbf{G})$.

The following time-discretisation methods result in the following equations,

- Explicit Euler: $\mathbf{M}(\mathbf{u}_n - \mathbf{u}_{n-1}) + \Delta_t \mathbf{A}\mathbf{u}_{n-1} = \Delta_t \mathbf{b} + \mathbf{e}_{n-1}$;
- Implicit Euler: $\mathbf{M}(\mathbf{u}_n - \mathbf{u}_{n-1}) + \Delta_t \mathbf{A}\mathbf{u}_n = \Delta_t \mathbf{b} + \mathbf{e}_{n-1}$;
- Crank-Nicholson: $\mathbf{M}(\mathbf{u}_n - \mathbf{u}_{n-1}) + \Delta_t \mathbf{A}(\mathbf{u}_n + \mathbf{u}_{n-1})/2 = \Delta_t \mathbf{b} + \mathbf{e}_{n-1}$. Note this is also an implicit method, which can also be solved by matrix inversion.

which can be solved in all cases to give the transition model, letting $\mathbf{b} = 0$,

$$p_\Lambda(\mathbf{u}_n|\mathbf{u}_{n-1}) = \mathscr{N}(\mathbf{u}_n; \mathbf{F}_\Lambda \mathbf{u}_{n-1}, \mathbf{Q})$$

where $\mathbf{Q} = \Delta_t \mathbf{B}^{-1}\mathbf{G}\mathbf{B}^{-\intercal}$, and $\mathbf{F}_\Lambda$, $\mathbf{B}$ are, in each case,

- Explicit Euler: $\mathbf{F}_\Lambda = \mathbf{I} - \Delta_t \mathbf{M}^{-1}\mathbf{A}$, $\mathbf{B} = \mathbf{M}$
- Implicit Euler: $\mathbf{F}_\Lambda = (\mathbf{M} + \Delta_t \mathbf{A})^{-1}\mathbf{M}$, $\mathbf{B} = \mathbf{M} + \Delta_t \mathbf{A}$
- Crank-Nicholson: $\mathbf{F}_\Lambda = (\mathbf{M} + \Delta_t \mathbf{A}/2)^{-1}(\mathbf{M} - \Delta_t \mathbf{A}/2)$, $\mathbf{B} = \mathbf{M} + \Delta_t \mathbf{A}/2$

A.2 Risk assessment retrospective

The risk assessment performed at the start of this project identified minimal risks to health, including risks associated with computer working - trip hazard and repetitive strain injury. The working environment was optimised to minimise these risks. If I were to retake this project, I would assess the same risks as before.