Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Martin Bjerke

A Spline-based Latent Variable Model for Neural State-space Discovery

Master's thesis in Applied Physics and Mathematics
Supervisor: Benjamin Adric Dunn
February 2021

NTNU
Norwegian University of
Science and Technology

Martin Bjerke

A Spline-based Latent Variable Model for Neural State-space Discovery

Master's thesis in Applied Physics and Mathematics
Supervisor: Benjamin Adric Dunn
February 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Abstract

Recent advances in neural data recording have given researchers the opportunity to harness the power of large neural populations, motivating the use of dimensionality reduction methods as a way to uncover latent variables that govern the activity of the neurons. Expanding upon the latent manifold tuning model devised by Wu et al. (2017), we propose replacing the non-parametric Gaussian process to model the tuning curves with a parametric B-spline function. Using an iterative maximum a posteriori procedure, we compare the performance of the two models with respect to scaling in data size, initialisation and choice of hyperparameters. Ultimately, we extend our model further to account for feature sharing among tuning curves of neurons, and utilise this to infer the head direction of a mouse from neural data gathered by Peyrache et al. (2015).

Sammendrag

Nylige fremskritt innen opptak av nevrale data har gitt forskere muligheten til å betrakte store ansamlinger av nevroner, noe som motiverer bruken av dimensjonsreduksjonsmetoder som et verktøy for å oppdage skjulte variabler som styrer oppførselen til nevronene. Wu et al. (2017) utarbeidet modellen ”latent manifold tuning”, og vi bygger videre på denne ved å introdusere parametriserbare B-Spline-funksjoner, som et alterativ til ikke-parametriserbare Gaussiske prosesser, for å modellere nevroners tuningkurver (sammenhengen mellom en ekstern påvirkning og et nevrons aktitivitet). Ved å ta i bruk en iterativ maksimum a posteriori-metode sammenligner vi de to modellene, og evalurerer hvor godt de presterer i forhold til ulik datamengde, initialisering og valg av hyperparametere. Avslutningsvis utvider vi modellen vår til å ta høyde for fellestrekk blant tuningkurver hos nevronene, og bruker denne modellen til å avdekke hodebevegelsene til en mus basert på data innsamlet av Peyrache et al. (2015).

Preface

This thesis is submitted as a requirement in TMA4900 Industrial Mathematics Master Thesis at the Department of Mathematical Sciences, and completes my Master of Science degree in Applied Physics and Mathematics, which is part of my Integrated Ph.D.-program in Mathematical Sciences at the Norwegian University of Science and Technology (NTNU).

I wish to thank my supervisor Benjamin Adric Dunn for the help and guidance along the way, his positive demeanour in the face of both more and less sensible questions related to neuroscience, and for sticking with me for a few more years while I pursue my Ph.D. I would also like to thank him for the great environment he has created within his research group, and extend a thank you to all its members for useful discussions and friendly banter. In particular, thanks to Claudia for both critical feedback and valuable conversations at the office.

Thanks to Grandma, Mom and Sis for three generations of female support throughout my childhood and during my studies. It has been said that ”Behind every great man is a great woman”, which unquestionably means I have thrice the greatness to live up to.
Thanks to Mina for taking that chance, and for spending much of her recent time off making sure I also got mine.
Finally, thanks to Grandpa for teaching me the art of counting cars and for encouraging a young ”Professor Tanke”.

Martin Bjerke
Trondheim, Norway
February 2021

Table of Contents

Chapter 1

Introduction

In this chapter, we give a brief introduction to the behaviour of neurons, how to record and interpret it, and the state of affairs in computational neuroscience. Section 1.1 covers the neural code, while we in Section 1.2 discuss trend in neural data recording. In Section 1.3, we introduce the concept of dimensionality reduction and its place in neuroscience, before finishing the chapter by stating the motivation for our work and its scientific contribution.

1.1 The neural code

The brain is our most complex organ. As with many complex structures, we try to understand it by breaking it down into smaller components. The most studied component of the brain is the nerve cell, or neuron, due to its ability to produce recordable electrical signals and pass these on to other neurons. A single neuron has an electrical potential difference compared to its surroundings. When a neuron is presented with certain stimuli, this potential can either increase or decrease in response. If it passes a certain threshold, it will be brought to a sharp peak, known as a spike. This event is also referred to as a neuron firing. When a neuron fires, it emits a signal which can further increase or decrease the probability of other interconnected neurons firing. Figure 1.1 shows how a neuron may react to received stimuli, and possibly be brought to a spike.

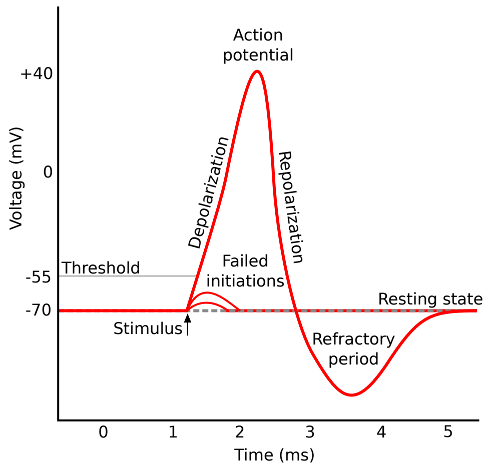

Figure 1.1: The red line shows the evolution in time of the potential of a neuron after it receives stimulus. If a neuron is brought to a spike, its voltage rapidly increases, then peaks and falls below the initial potential before stabilising where it started. Source: https://commons.wikimedia.org/w/index.php?curid=2241513. License: CC BY-SA 3.0.

As a more concrete and everyday example of how a neuron may react to a stimulus it receives, consider a neuron in the brain of a cat receiving information about what the cat observes. Imagine the neuron firing very rapidly when the cat is observing one thing, for example a mouse, but hardly fires at all when the cat is observing something else, for instance another cat. If the firing rate of a neuron is higher for a certain external stimulus, the neuron is said to be tuned to this stimulus. Said tuning can be visualised by a function that maps from the space of stimuli to the intensity of firing, commonly referred to as a tuning curve.

This hypothetical setting is in fact very similar to that of the famous experiment by Hubel and Wiesel (1979). By inserting an electrode into the primary visual cortex (V1) of an anaesthetised cat, they managed to record how neurons in V1 responded to images shown on a screen, particularly how the firing rate suddenly increased when a line was moved across the screen at a particular angle. These neurons, who fired more intensely when the line was positioned at a certain angle, were accordingly said to be tuned to this angle.

However, while our own hypothetical cat is observing a mouse through its eyes, it is also feeling the fabric it sits on, detecting the temperature of the room, and might also be considering whether it is hungry enough to go chasing after the mouse or not. All these are stimuli to which a neuron could also be tuned to, either in addition to or instead of the visual input. It is therefore essential that as many external variables as possible are controlled when doing neural recording, to avoid potential interference.

1.2 Advances and the state of neural data recording

For the past century, the concept of the lone neuron as the functional unit of neural computation has been somewhat set in stone. Recent advances have however led to the belief that we should look more towards populations of neurons, and not necessarily singular entities (Yuste, 2015). One possible driver for this shift in paradigm is the ability to record from hundreds of neurons simultaneously, a feat that is now more commonplace thanks to e.g. advanced silicon probes (Steinmetz et al., 2018) and light-sheet microscopy (Ahrens et al., 2013).

Stevenson and Kording (2011) discovered that the growth in the number of simultaneously recordable neurons exhibited an exponential trend the past decades, a trend that provides both exciting opportunities and challenges with respect to how one handles large amounts of data. Given that the exponentional trend continues, we should (theoretically) be able to, at one point, record from the whole neural population of the brain (although we are still approx. 200 years away from that, according to Stevenson and Kording, 2011).

One of the challenges with respect to such large amounts of data is how to account for variables that we are unable to record, be they either unobserved neurons or external input which might influence the activity of neurons in various ways (Brody, 1999). In particular, these variables are interesting because they might relate to the task performed by the animal, without being immediately visible in the neural population (Ahrens et al., 2012). As such, these variables elicit the use of a particular type of models to capture possibly important features in the data.

1.3 Dimensionality reduction and latent variables

Harking back to the case of Hubel and Wiesel (1979), where neurons were found to be tuned to a particular angle of an observed line. It is then reasonable to believe that other neurons may be tuned to different angles, and together give the cat the ability to observe lines of all angles. In this case, the angle directly influences the neural activity, and without knowledge of this connection, the angle might instead be referred to as a latent variable, governing the activity of the neurons.

A product of its time, the equipment used by Hubel and Wiesel (1979) naturally limited their recording capacity, which meant they had less neurons to consider simultaneously. However, as noted in the previous section, this is no longer an issue. While evaluating the activity of a single neuron by inspection might be feasible, it is a completely different story when the number of neurons approach hundreds, if not thousands. Hence, realising we are actually looking for an angle might be slightly more complicated.

Naturally, due to the increasingly large number of neurons that are possible to record from simultaneously, there has been an increasing demand in the neuroscientific community for models able to accommodate larger data sets (Paninski and Cunningham, 2018). Although the massive increase in sheer volume of data is a more recent phenomenon, researchers have advocated for methods that handle large and high dimensional neural data sets for a while (Fetz, 1992; Hatsopoulos et al., 1998; Cunningham and Byron, 2014), more specifically dimensionality reduction techniques. One of many points in favor of dimensionality reduction methods is that they allow us to visualise a condensed representation of high dimensional data in a lower dimensional space. This is quite a useful trait, as most humans have trouble envisioning anything that has more than three dimensions.

Perhaps the most known dimensionality reduction method is principal component analysis (PCA). PCA works by projecting the observed data onto a lower dimensional space, while keeping the maximal amount of variance intact. It is simple and intuitive, and hence also a widely used method (Briggman et al., 2005; Mazor and Laurent, 2005). Taking a step up the complexity ladder, we find e.g. ISOMAP (Tenenbaum et al., 2000), which creates a lower dimensional embedding based on the neighbourhood structure of the original data. While ISOMAP solves one of the limitations of PCA (it can only recover linear relationships), neither of them are probabilistic methods, limiting their viability in settings where one is interested in performing statistical inference. This lead to the hunt for more general methods (both with respect to manifold interactions and the statistical inference procedure), one of the results being the Gaussian process latent variable model (GPLVM), a customisable model used for latent variable discovery.

The term GPLVM was first used by Lawrence (2003), who proposed it as a generalisation of PCA to circumvent its inherent linear restrictions. The use of Gaussian processes, which are highly non-linear, allows the model to capture non-linear mappings between the data and the latent space. One possible interpretation is to consider it as ”a non-linear probabilistic version of PCA” (Lawrence, 2005, p. 1790). Gaussian processes, being as versatile as they are, are a common sight in the supervised setting, while also proving useful in this latent discovery setting. Kulkarni and Paninski (2007), for example, used Gaussian processes to model the effect of hidden neurons on visible neurons, while Ecker et al. (2014) managed to recover correlation structures in recordings from an anaesthetised macaque, using Gaussian process factor analysis (GPFA). Returning to the line of research that builds upon the advances of the GPLVM, Titsias (2009) introduced a variational approximation for selecting hyperparameters, an inference procedure which was further improved upon by Titsias and Lawrence (2010).

While the early work on the GPLVM assumed Gaussian observations, the structure of neural data makes it perhaps more suited for a point-process model (Smith and Brown, 2003). Employing a Poisson spiking model, Wu et al. (2017) devised the model known as the Latent Manifold Tuning (LMT, previously called the Poisson Gaussian process latent variable model (P-GPLVM)) for spike train data. They later extended the model to be applicable to calcium imaging data as well (Wu et al., 2018), and applied the model to olfactory data to infer a lower dimensional latent manifold to explain the relationship between an odorant and the neural population activity it induces.

More recent work building upon the family of GPLVMs include e.g. extending the latent space to include various non-Euclidean manifolds (Jensen et al., 2020), and generalising the likelihood to other distributions by incorporating random Fourier features (Gundersen et al., 2020).

1.4 Motivation and contribution

As mentioned by Wu et al. (2017), and also confirmed by Myklebust (2020), the LMT model can be prone to solutions of local optimality, which can make inference difficult due to the heavy importance of initialisation. As the model is also defined by a double Gaussian process, it may scale poorly with respect to large data sets, compared to other models. Due to Gaussian processes being non-parametric, one also has fewer ways of accounting for known structure in either tuning curves or the latent variable, as a means of helping the model take more informed steps during the exploration of the latent space.

To address these potential issues, we inspect the structure of the LMT model, which can be separated into three parts: a Gaussian process modeling the latent variable, a Gaussian process modeling the tuning curves and a Poisson point process modeling the neural spiking. We propose to exchange the non-parametric Gaussian process modeling the tuning curve with a parametric part, more explicitly a B-Spline function (Piegl and Tiller, 1996).

We present the framework for our model, derive expressions necessary for performing inference and define an iterative maximum a posteriori (MAP) procedure for inference of the latent variable and the tuning curves. Capitalising on the works of Myklebust (2020), we compare their free-standing implementation of the LMT model with our Spline-based variation, discuss strengths, weaknesses, and evaluate the performance of the models with respect to variation in hyperparameters, initialisation and scaling in data size.

Finally, we advocate for sharing features among tuning curves of the neurons where the situation allows it, and provide an extension to our Spline-based latent variable model for more efficient inference. We compare the performance against the non-feature sharing variant on simulated data, before applying the model to head direction data recorded by Peyrache et al. (2015), comparing it to PCA and the LMT model.

Chapter 2

Data

In this chapter, we introduce the data set recorded by Peyrache et al. (2015). We discuss the head direction data set on a general level in Section 2.1, while addressing the choice of bin width and consistency of tuning in Subsections 2.1.1 and 2.1.2. In the final subsection, we present our motivations behind selecting this particular data set for use in our analysis.

2.1 Head direction dataset

Peyrache et al. (2015) studied the brain's mechanisms to monitor head direction by making simultaneous recordings from the antero-dorsal thalamic nucleus and the post-subiculum of the brains of seven mice. The recordings were done using multi-site silicon probes, both while the mice were asleep and awake. We limit our interest to the recordings performed on awake mice where, while exploring and foraging in an open environment, the head direction of the animals were also tracked using stationary cameras. The head directions were derived based on the relative position of diodes mounted to the head of the mice, at intervals of 25.6 ms, over an awake period of approximately 36-and-three-quarters minutes, and we will focus on the recordings from a trial labeled Mouse12-120806. The approximately four-and-a-half first minutes of the recorded head direction in this trial can be seen in Figure 2.1, after having removed missing observations (which accounted for approx. 3% of the entire data set).

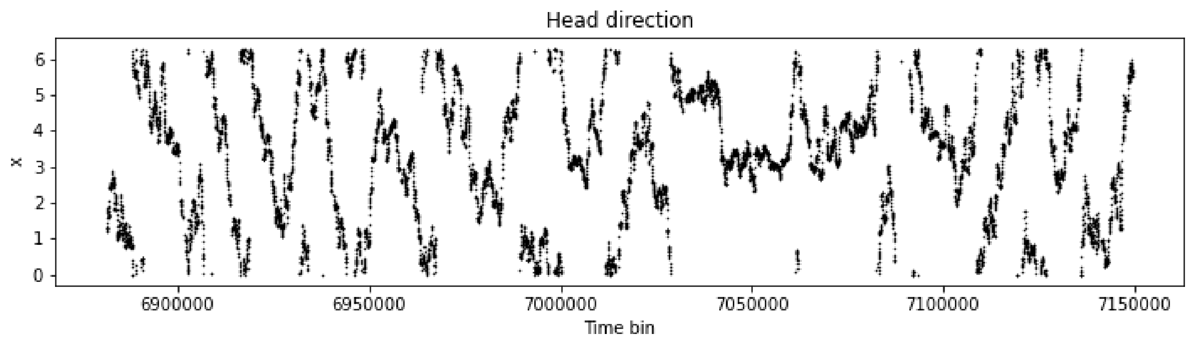

Figure 2.1: Observed head direction for the first 10.000 bins starting at time 6881305.6 in the dataset Mouse12-120806, after missing data has been removed.

Measured as an angle, the head direction can then be modelled as a one-dimensional variable on the interval $[0, 2\pi]$, with periodic boundary conditions. While we in this case have observations for the head direction, and evidence that certain neurons do seem to respond to it, we do in general not necessarily have this knowledge. The hunt for such an underlying, latent variable is of uttermost interest, and we will in this thesis consider the head direction as one such latent variable, with the added benefit of having knowledge of the true variable.

As for the neural activity, the time stamp for whenever a neuron produced a spike was recorded. A customary way of representing the spike activity is to partition the entire recording interval into bins of equal width, then count the number of neural spikes present in each bin, for each neuron. The activity can then be presented in the form of this spike count, or alternatively as a binary variable, indicating whether there is at least one spike in each bin or not. Such a spike presence representation can be seen in Figure 2.2, which covers the activity of the 73 recorded neurons in the trial Mouse12-120806. The interval is set to the same as for the tracked head direction in Figure 2.1.

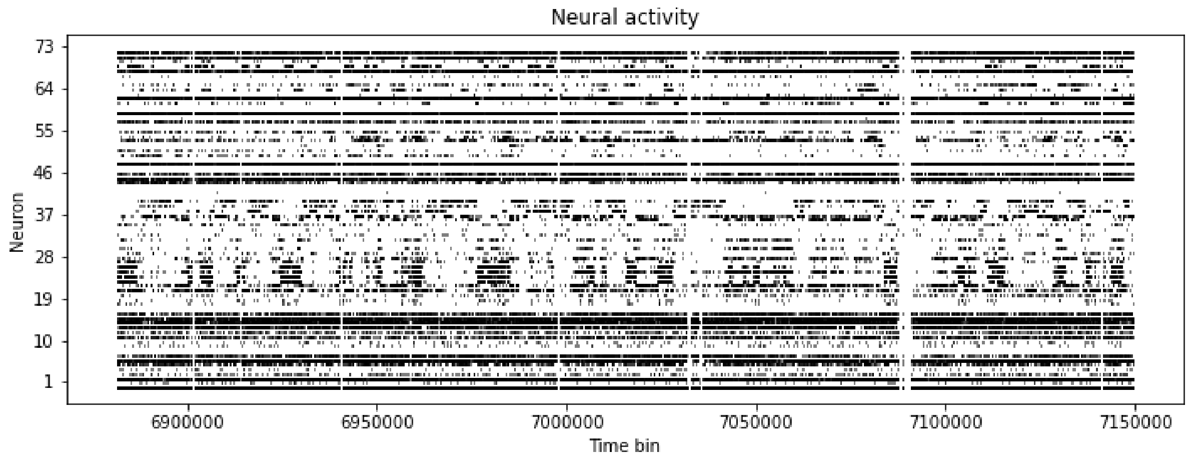

Figure 2.2: Binned and binarised spike data for the first 10.000 bins starting at time point 6881305.6. The 73 neurons are placed on the y-axis, and a black dot means that at least one spike was observed in that particular time bin (of width 25.6 ms).

2.1.1 Choice of bin width

The binned spike data shown in Figure 2.2 has a bin width of 25.6 ms, similarly to that of the recorded head direction. However, since the neural spike recording was originally done on a much finer time scale, the bin width can be set to a much shorter interval than 25.6 ms. Alternatively, one can also choose a bin width wider than 25.6 ms. In Figure 2.3 we showcase the distribution of the spike counts in the bins, for four different choices of bin widths.

As is evident from the spike distribution plots, increasing the bin width results in a much wider spread in the number of spikes in each bin, as one would expect. Although it might be tempting to select a very small bin width to construct a spike representation as close as possible to the ”true” recording (a binary variable that either spikes or not), one must account for the fact that this requires partitioning the total recording length into a staggering number of bins. In many cases, the computational time for an algorithm scales poorly with increasing data length, necessitating a more generous bin width. On the other hand, wider time bins means less partitioning, but we might lose important information by summing over spikes that occur at wildly different head directions. Some neural models also assume a binary or Bernoulli behaviour of the spike representation, in which case a larger bin width results in a loss of information whenever a bin includes more than one spike. This makes for an interesting decision where one has to consider computational complexity versus loss of neural information, as well as the inherent time scale of the dynamics of the latent variable itself (i.e. picking a bin width of for example 1 min would completely fail to capture the dynamics, as they happen on a much smaller time scale).

In our case, we chose a bin width of 25.6 ms when doing analysis, corresponding to the time between recordings of the head direction. Picking a wider time bin could present issues when interpolating the periodic head direction variable, and shorter bins were dismissed based on early indications of running time.

2.1.2 Neural tuning

While evaluating the tuning of the neurons, highly active neurons can allow us to more easily infer the head direction they are tuned to. However, a very active neuron does not necessarily equal a tuned neuron. To visualise this, we partition the domain of the head direction variable $[0, 2\pi]$ into 30 equally sized intervals, then calculate the average number of spikes a neuron produces while the head direction resides in each of these intervals (here we consider the same 10000 time bins as in Figures 2.1 and 2.2). We showcase the activity of two different neurons, and although the neuron in Figure 2.4 has a higher spiking frequency than the one in Figure 2.5, it seems to have no

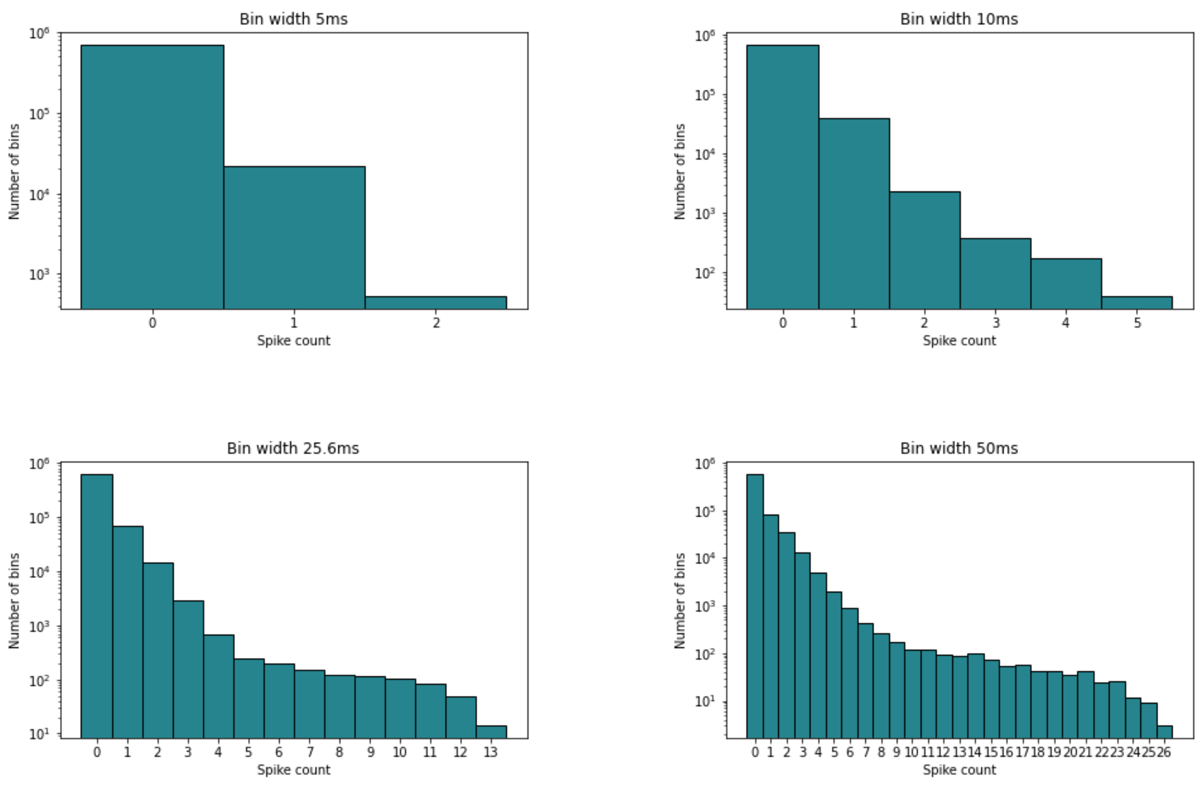

Figure 2.3: Log scale distribution of spikes in a bin, for all neurons, for 10.000 bins starting at time 6881305.6. The four different bin widths are, from upper left to bottom right: 5 ms, 10 ms, 25.6 ms and 50 ms.

preferred head direction for which it produces more spikes. Meanwhile, the neuron in Figure 2.5 seems to clearly prefer head directions values in the interval $[3, 4]$.

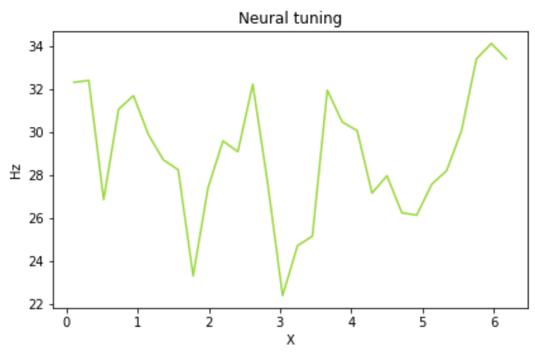

Figure 2.4: Example of a tuning curve with no particular tuning. Average firing rate of 28.99 Hz.

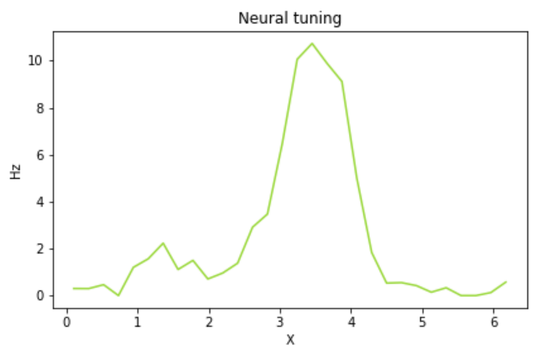

Figure 2.5: Example of a tuning curve with evident tuning to head direction. Average firing rate of 2.46 Hz.

For neurons with clear head direction tuning, we should expect the tuning to be consistent over time. We compare the spike rate over the first 10000 bins, starting at time 6881305.6, to the spike rate over an interval of equal size, starting where the first ends. Given consistent tuning, we would expect the rates to look approximately the same. A showcase of firing rates for four different neurons, all tuned to head direction, can be seen in Figure 2.6. The tuning seems to be more or less consistent across these two time intervals.

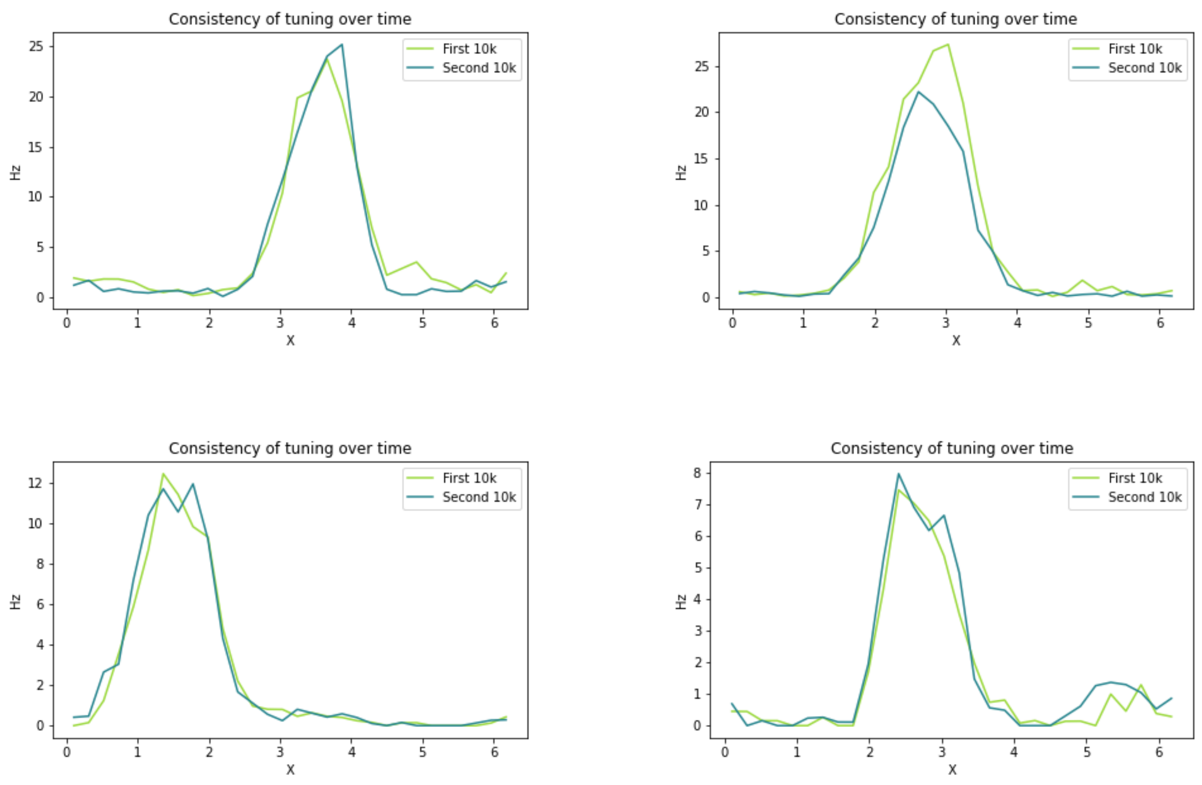

Figure 2.6: A comparison of observed firing rates (in Hz) between two non-overlapping time intervals for four selected neurons. The first interval start at time 6881305.6, the second 10000 time bins after the first one. Bin width of 25.6ms.

2.1.3 Motivation

The decision to use the data set recorded by Peyrache et al. (2015) was motivated by a couple of factors: first of all, it can be hard to judge whether a model has recovered the latent variable properly, since one usually does not know the true behaviour (ref. the term ”latent” variable). In this case, however, it is fair to assume that the head direction does drive the activity of the neurons (although there might be other factors as well), making it a prime candidate for what we can consider the ”true” latent variable. Having access to a ground truth during evaluation is a very useful feature, as it lets us compare the results between various models.

Another useful property of the data set is the distinct shape of the tuning curves for neurons that exhibit tuning with respect to the head direction. The bump shape is quite pronounced, and will hopefully make it a less demanding task for the models to reconstruct them. The fact that the tuning curves all share an overall similar form is also an important prerequisite for one of the models which we will introduce later.

Chapter 3

Background

In this chapter we introduce relevant background material for deriving and understanding the statistical models that will be presented in Chapter 4. We touch upon the Generalized Linear Model (GLM) in Section 3.1, before moving on to detailing the theory of Gaussian Processes and Splines in Sections 3.2 and 3.3.

3.1 Generalized Linear Models

According to Agresti (2015, p. 2), a GLM can be summarised by its three core components; the random component representing the response variable and its distribution, the systematic component represented by the linear predictor, and the link function which relates the random component to the systematic one. An overview of these three components will be presented here.

Assume that we have n independent observations of a response-covariate pair, i.e. $(Y_i, \mathbf{X}_i), i = 1, \ldots, n$. The Generalized Linear Model framework provides an efficient way of modeling the relationship between such pairs in situations where the response is not necessarily normally distributed.

In the classical linear regression setting, the linear model is

$$Y_i = \mathbf{X}_i\boldsymbol{\beta} + \epsilon_i, \tag{3.1}$$

with the errors ϵ_i being independent and identically distributed according to a normal distribution $\mathcal{N}(\epsilon_i; 0, \sigma_\epsilon^2)$. Thus, the response is also normally distributed $Y_i \sim \mathcal{N}(y_i; \mathbf{X}_i\boldsymbol{\beta}, \sigma_\epsilon^2)$.

Suppose instead that the response comes from a (not necessarily normal) distribution f_{Y_i}, with mean $\mathrm{E}[Y_i] = \mu_i$ and variance $\mathrm{Var}[Y_i]$. According to the GLM framework (Fahrmeir et al., 2013, pp. 301-304), the relationship between the response y_i and the linear predictor $\eta_i = \mathbf{X}_i\boldsymbol{\beta}$ can be modelled with a link function

$$g(\mu_i) = \eta_i, \tag{3.2}$$

when the distribution of the response belongs to the univariate exponential family. Given that the GLM describes the relation between the (conditional) mean and the linear predictor, one often uses the so-called response function instead

$$\mu_i = h(\eta_i), \tag{3.3}$$

with the requirement that h is twice differentiable and one-to-one. It then follows that $g(\cdot) = h^{-1}(\cdot)$.

When it comes to the distribution, f_{Y_i} is said to belong to the exponential family if it can be expressed in the following way

$$f_{Y_i}(y_i; \theta_i, \phi, w_i) = \exp\Big(\frac{y_i\theta_i - b(\theta_i)}{\phi} w_i + c(y_i, \phi, w_i)\Big). \tag{3.4}$$

Here, θ_i is the natural parameter, ϕ the dispersion parameter and w_i the weight parameter, taking the value 1 as long as the data is ungrouped.

Given that f_{Y_i} can be described as in Equation 3.4, we have the following relations

$$\begin{aligned} \mathrm{E}[Y_i] &= \mu_i = b'(\theta_i), \\ \mathrm{Var}[Y_i] &= \frac{\phi b''(\theta_i)}{w_i}. \end{aligned} \tag{3.5}$$

Finally, if the relation from Equation 3.2 is such that $\theta_i = g(\mu_i)$, the link function is said to be the canonical link. This leads to the log-likelihood function being concave, which makes gradient-based methods easy to use for optimization (see e.g. Haberman, 1977).

3.2 Gaussian Processes

To define a Gaussian process, we will follow Rasmussen and Williams (2006), whose comprehensive introduction we highly recommend for the interested. Here, we give a brief description of the properties of a Gaussian process.

Formally, let us consider a random process $\{X_t\}_{t\in T}$, where T is a continuous domain, for instance time or space. The process is then a Gaussian process if, for any collection of n time points or locations $t_1, \dots, t_n$, the collection $\mathbf{X} = (X_1, \dots, X_n)$ follows a multivariate normal distribution

$$\mathbf{X} \sim \mathcal{N}_n(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{3.6}$$

with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

As Gaussian processes are commonly used for modelling the distribution of functions directly, like in the setting of regression, we will, in accordance with Rasmussen and Williams (2006), denote our random process $f(\mathbf{x})$, the function value at $\mathbf{x}$. The Gaussian process can then equivalently be written as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x'})), \tag{3.7}$$

where $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x'})$ denote the mean function and covariance function (sometimes called kernel) of the Gaussian process respectively

$$\begin{aligned} m(\mathbf{x}) &= \mathrm{E}[f(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x'}) &= \mathrm{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x'}) - m(\mathbf{x'}))]. \end{aligned} \tag{3.8}$$

As stated by Rasmussen and Williams (2006, p. 13), the Gaussian process is completely defined by its mean and covariance function, and while the mean is commonly set to the zero-function, there exists a wide range of different covariance functions that alter how the corresponding Gaussian process behaves. Assuming that the variance stays the same for all $\mathbf{x}$, we can formulate three popular choices of covariance functions, the Gaussian, Matérn and Exponential covariance function, in the following way

$$\begin{aligned} k_{\text{Gauss}}(x, x') &= \sigma^2 \exp\left(\frac{-||x - x'||_2^2}{2\delta^2}\right), \\ k_{\text{Matérn}}(x, x') &= \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{|x - x'|}{\delta}\right)^{\nu} K_\nu \left(\sqrt{2\nu}\frac{|x - x'|}{\delta}\right), \\ k_{\text{Exponential}}(x, x') &= \sigma^2 \exp\left(\frac{-|x - x'|}{\delta}\right). \end{aligned} \tag{3.9}$$

Here, δ represents the length scale parameter and σ^2 the marginal variance. As for the Matérn, Γ is the gamma function, K_ν the modified Bessel function, and ν some extra parameter, usually a half-integer for simplification purposes. The behaviour of the three different covariance functions (with $\nu = \frac{3}{2}$ for the Matérn), as well as a realisation of a Gaussian process, with zero mean and an instance of each kernel, is shown in Figure 3.1 and 3.2.

It is clear from Figure 3.1 that the Gaussian kernel results in the stronger correlation over a longer distance, while the Exponential kernel has the weaker correlation (at least for values smaller than $\sim$ 1.2). Intuitively, the Gaussian kernel should then produce a smoother Gaussian process, since subsequent values are more closely correlated, while the Exponential kernel should yield a more jagged path. This is precisely what can be seen in Figure 3.2.

Respectively, the δ and σ^2 are used to control the smoothness and the variation of the Gaussian process. A large δ means that a point x can influence other x' that are farther away, while a large σ^2 means the Gaussian process is allowed to deviate farther away from the mean. For similar parameter values, the covariance functions also decay at a different rate, resulting in different behaviour for the Gaussian processes, as can be seen in Figure 3.3 and 3.4.

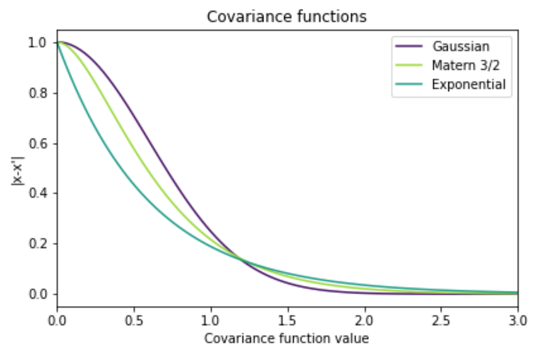

Figure 3.1: Behaviour of the 3 different covariance functions, with $\delta = 0.6$ and $\sigma^2 = 1^2$

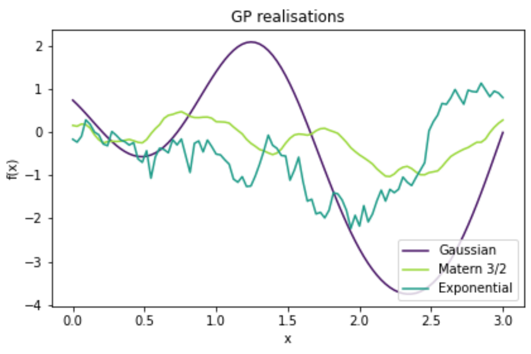

Figure 3.2: 3 realisations from a Gaussian process, using different kernels, with $\delta = 0.6$ and $\sigma^2 = 1^2$

Observe that the processes largely behave in the manner one would expect, based on the different values for δ and σ^2.

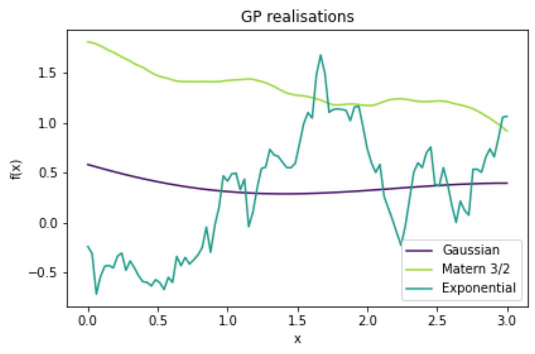

Figure 3.3: Gaussian process realisations with $\delta = 2.6$ and $\sigma^2 = 1^2$

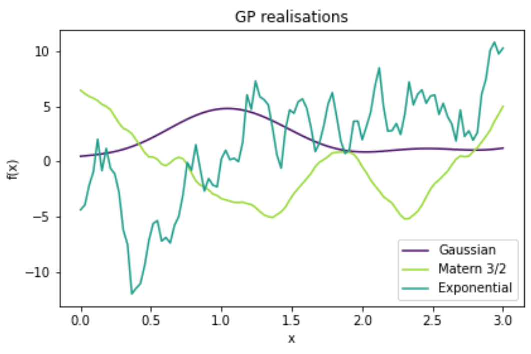

Figure 3.4: Gaussian process realisations with $\delta = 0.6$ and $\sigma^2 = 5^2$

3.3 Splines

A spline can be thought of as an extension of the linear regression model, providing more flexibility compared to simple polynomial regression. In many real-life cases, data does not necessarily share the same linear, or a specific polynomial, relationship on the whole interval of interest. Instead of modeling the connection between $(Y, \mathbf{X})$ as e.g. strictly linear, splines use a set of basis functions to define $Y = f(\mathbf{X})$ as a piece-wise function of polynomials, giving more flexibility and freedom in specified intervals. The general formula can be written as

$$f(\mathbf{X}) = \sum_{m=1}^{M} \beta_m h_m(\mathbf{X}), \tag{3.10}$$

where M is the number of basis functions used, $\{h_m\}$ the set of M basis function and the β_m's being the coefficients, or weights, of the basis functions. The β_m's are sometimes referred to as control points in spline literature.

The piece-wise part of the spline function comes from the fact that each basis function is defined on a set interval, thus only contributing to $f(\mathbf{X})$ if it is evaluated at an $\mathbf{X}$ which falls inside this interval. The result is that $f(\mathbf{X})$ will be defined by different polynomials, depending on which interval we look at. Assume now, for notational purposes, that $\mathbf{X}$ is one-dimensional. This separation is then defined by what is called a knot vector $\mathbf{t} = [t_1, \cdots, t_K]$, a vector of non-decreasing values, which divides the domain where the spline function is defined into $K + 1$ separate intervals, each with its own polynomial. The spline function can then be either continuous or discontinuous at the knots, depending on the order of the polynomials.

We will focus on a basis function set known as B-Splines, which defines a basis for any spline function of order p, and allow for great flexibility and control. For a classical introduction to both general splines and B-Splines, see De Boor (1978).

3.3.1 B-Splines

B-Spline functions of order p are piece-wise polynomials of degree $(p-1)$, joined together at K interior knots $[t_1, \cdots, t_K]$, using the B-Spline basis (hereby referred to as B-Splines). They are defined on a given interval specified by boundary knots $t_0 \leq t_1$ and $t_{K+1} \geq t_K$. B-Splines are uniquely defined by their knots, and can be expressed in the following manner, known as the Cox–de Boor recursion formula,

$$\begin{aligned} B_{i,0}(x) &= \begin{cases} 0, & \text{if} \quad t_i \leq x \leq t_{i+1} \\ 1, & \text{otherwise} \end{cases}, \\ B_{i,j+1}(x) &= \alpha_{i,j+1}(x) B_{i,j}(x) + [1-\alpha_{i+1,j+1}(x)] B_{i+1,j}(x), \end{aligned} \tag{3.11}$$

where

$$\alpha_{i,j}(x) = \begin{cases} \dfrac{x-t_i}{t_{i+j}-t_i}, & \text{if} \quad t_{i+j} \neq t_i \\ 0, & \text{otherwise} \end{cases}. \tag{3.12}$$

Since each unique B-Spline $B_{i,p-1}(x)$ of degree $(p-1)$ is defined over an interval of $(p+1)$ interior knots (as is evident by considering Equation 3.11 and 3.12), it is necessary to add an additional $(p-1)$ knots (excluding the boundary knot) at the ends of the knot vector for the B-Splines to be properly defined. It is customary to simply repeat the boundary knot $(p-1)$ times, which also has the added benefit of degenerating the B-spline at the edges. Due to the way the B-Spline is constructed, if the knots defining the B-Spline are all unique, its derivative is continuous up to the order $(p-2)$. For each repeating knot, the order of continuity is reduced by one, so the $(p-1)$ repeats of the boundary knots then results in the desired undefined property outside the boundary knots.

Due to the necessary augmentation of the knot vector, a B-Spline function is usually defined over what is called the augmented knot vector, in contrast to simply the interior knots. The augmented knot vector $\boldsymbol{\tau} = [\tau_0, \cdots, \tau_{K+2p-1}]$ consists of $K+2p$ knots, the first and last p being repeats ($\tau_0 = \tau_1 = \cdots = \tau_{p-1} = t_0$, and $\tau_{K+p} = \tau_{K+p+1} = \cdots = \tau_{K+2p} = t_{K+1}$). Together with the order p and $M = K+p$ weights, they fully define the the B-Spline function

$$f_{BS}(x) = \sum_{m=0}^{K+p-1} \beta_m B_{m,(p-1)}(x). \tag{3.13}$$

An example of how a B-Spline function is made is shown in Figures 3.5 - 3.7. In this case, the interval $[0, 5]$ is covered by $K = 5$ interior knots, excluding the repeated boundary knots at the edges of the interval. With a degree of three, the resulting $M = 9$ B-Splines of degree three are shown in Figure 3.5. The weights β_m allow us to control the shape of the B-Spline function, and with the weighting shown in Figure 3.6, they together form the B-Spline function in Figure 3.7.

Periodic B-Splines

At times, it might be desirable to have a periodic boundary condition for the B-Spline function, in contrast to the degeneration at the edges of the interval that is described above. This way, the spline will be continuous across the boundary knots t_0 and t_{K+1}. To achieve this, instead of the first and last $(p-1)$ elements of $\boldsymbol{\tau}$ being repeats of t_0 and t_{K+1} respectively, they are extended over the boundaries, with similar spacing as between the other elements of $\boldsymbol{\tau}$. Thus we have $\tau_0 < \tau_1 < \cdots < \tau_{p-1} < t_0$ and $t_{K+1} < \tau_{K+p} < \tau_{K+p+1} < \cdots < \tau_{K+2p}$. Another requirement is that the first $(p-1)$ β_m's must be equal the last $(p-1)$ β_m's, i.e. $\beta_0 = \beta_{K+1}, \cdots, \beta_{p-2} = \beta_{K+p-1}$. This ensures that the first and last basis functions are similarly weighted and overlapping across the boundary, this way ensuring continuity.

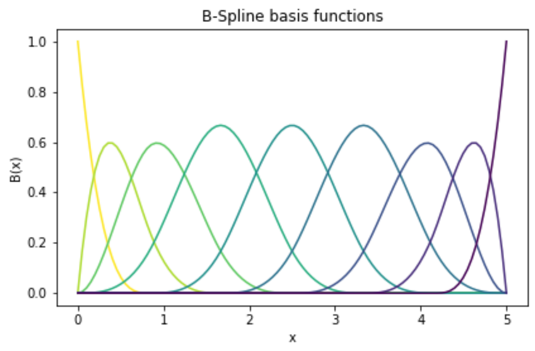

Figure 3.5: Nine B-Spline basis functions for a B-Spline of degree three

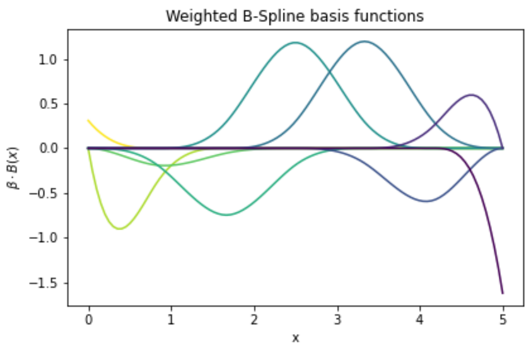

Figure 3.6: The same nine basis functions, now weighted randomly with $\beta_m \in [-2, 2]$

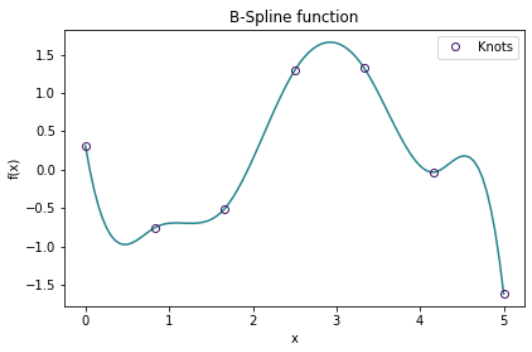

Figure 3.7: The weighted B-Splines summed together, yielding a piece-wise connected B-Spline function f_{BS}

Chapter 4

Latent Variable Models

In this chapter we introduce our models of choice, and highlight the differences between the LMT-model of Wu et al. (2017) and our own spline-based LVM. Section 4.1 addresses the setup of the LMT-model, while Section 4.2 describes our spline-based approach. The inference procedure of the models will be showcased in Section 4.3 and 4.4 respectively, where we derive the necessary posterior distributions for performing an iterative MAP estimation of the latent variable and the tuning curves. We describe and showcase the iterative MAP procedure in Section 4.5, before finishing off the chapter by highlighting an extension to the Spline-based LVM using a technique called feature sharing.

4.1 The Latent Manifold Tuning model

We wish to model a latent variable which governs the behaviour of the spike counts of N neurons indexed by $i = 1, \dots, N$, and discretise the time into bins indexed by $t = 1, \dots, T$ over the period we are interested in. Let the number of spikes of neuron i in bin t be denoted by $y_{i,t}$. Furthermore, let $\mathbf{y}_t \in \mathbb{R}^N$ denote the vector of spike counts for all neurons at time t, let $\mathbf{y}_j \in \mathbb{R}^T$ denote the vector of spike counts in all time bins for neuron j, and let $\mathbf{Y} \in \mathbb{R}^{N \times T}$ denote the matrix of spike counts for all neurons for all time bins, with rows equal to $\mathbf{y}_j$ and columns equal to $\mathbf{y}_t$.

4.1.1 The latent process

The latent process consists of a P-dimensional latent variable $\mathbf{x}(t) \in \mathbb{R}^P$ and tuning curves $\{h_i(\mathbf{x})\}, i = 1, \dots, N$ that map the latent variable to the firing rate of each neuron at a given time. Each component $x_j(t), j = 1, \dots, P$ of the latent variable is modeled as an independent Gaussian process in the time domain

$$x_j(t) \sim \mathcal{GP}(0, k_t), \tag{4.1}$$

with zero mean and temporal covariance function $k_t(t, t')$. We will follow Wu et al. (2017) and use an exponential covariance function, as defined in Section 3.2

$$k_t(t, t') = \sigma_x \exp\left(\frac{-|t - t'|}{\delta_x}\right), \tag{4.2}$$

which enforces smoothness in time for the latent variable. This is reasonable for several variables, for example head direction. Denote by $\mathbf{x}_j$ the vector of length T containing the values $x_j(t)$ evaluated at all time bins. Since it is a Gaussian process it will then follow a normal distribution,

$$\mathbf{x}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_t), \tag{4.3}$$

where $\mathbf{K}_t \in \mathbb{R}^{T \times T}$ is the covariance matrix containing the covariance function evaluated at every combination of time points. In a similar manner to $\mathbf{y}_t$ and $\mathbf{Y}$, we let the vector $\mathbf{x}_t = \mathbf{x}(t)$ denote the value of the latent variable at time t, and let matrix $\mathbf{X} \in \mathbb{R}^{P \times T}$ contain the values of the P-dimensional latent variable for all time bins, such that the rows of $\mathbf{X}$ are equal to $\mathbf{x}_j$.

4.1.2 The tuning curves and spiking model

Let the function h_i: $\mathbb{R}^P \longmapsto \mathbb{R}$ describe a mapping from the latent variable at time t, to the firing rate $\lambda_{i,t}$ of neuron i, at time t. The function $h_i(\mathbf{x})$ will then be referred to as a tuning curve

$$\lambda_{i,t} = h_i(\mathbf{x}_t). \tag{4.4}$$

Now, instead of modeling $h_i(\mathbf{x}), i = 1, \dots, N$ directly, we use the GLM framework from Section 3.1 to model the log tuning curves $f_i(\mathbf{x}), i = 1, \dots, N$. Exponentiating ensures positive values (as $\lambda_{i,t}$ is always positive), and allows us to harness the useful properties that the canonical link brings, resulting in the following relation

$$h_i(\mathbf{x}) = \exp(f_i(\mathbf{x})). \tag{4.5}$$

Following Wu et al. (2017), the log tuning curve $f_i(\mathbf{x})$ of neuron i are modeled as a Gaussian process over the P-dimensional space of the latent variable

$$f_i(\mathbf{x}) \sim \mathcal{GP}(0, k_x), \tag{4.6}$$

using the same Gaussian covariance function, which we assume to be shared across all N neurons,

$$k_x(\mathbf{x}, \mathbf{x}') = \sigma_f \exp\left(\frac{-||\mathbf{x} - \mathbf{x}'||_2^2}{2\delta_f^2}\right). \tag{4.7}$$

This enforces smoothness in the latent variable space for the tuning curves, where the degree can be adjusted by the choice of the parameter δ_x.

Let the vector $\mathbf{f}_i \in \mathbb{R}^T$ contain the value of the log tuning curve $f_i(\mathbf{x}_t)$ for all times t. Then, according to the properties of a Gaussian process, $\mathbf{f}_i$ has a multivariate normal distribution, given the value of the latent vector at all time bins. With an additional term handling the possibly noisy observations, the conditional distribution of $\mathbf{f}_i$ takes the following form

$$\mathbf{f}_i | \mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_x + \sigma_\epsilon^2 \mathbf{I}), \tag{4.8}$$

where $\mathbf{K}_x \in \mathbb{R}^{T \times T}$ is the covariance matrix of $\mathbf{f}_i$ containing elements $\mathbf{K}_{x\{t,t'\}} = k_x(\mathbf{x}_t, \mathbf{x}_{t'})$ for every pair of latent states $(\mathbf{x}, \mathbf{x}')$ that $\mathbf{x}$ attains between $t = 1$ and $t = T$. The noise term consists of the noise parameter σ_ϵ^2, and an identity matrix $\mathbf{I}$ of size $T \times T$.

To obtain a similar notation to $\mathbf{Y}$ and $\mathbf{X}$, we gather the $\mathbf{f}_i$ vectors as rows in the matrix $\mathbf{F} \in \mathbb{R}^{N \times T}$. Then the rows of $\mathbf{F}$ contain the values of the log tuning curves of a single neuron evaluated at every time bin, and a column $\mathbf{f}_t$ of $\mathbf{F}$ describes the values of the log tuning curves at time t for the entire neuron population $i = 1, \dots, N$.

Theoretically, it is possible for the latent variable $\mathbf{x}(t)$ to visit the exact same state multiple times, and by defining the conditional distribution of $\mathbf{f}_i$ as a multivariate normal distribution, there is no guarantee that the same $\mathbf{x}_t$ will result in the same value for $f_i(\mathbf{x}_t)$ on multiple occasions. Although this invalidates the assumption that the tuning curve maps any $\mathbf{x}$ to a single firing rate, it is mostly a theoretical curiosity, and has no practical implications in our case.

Finally, we introduce the Poisson spiking model by assuming that the number of spikes for neuron i in bin t, conditioned on the log tuning curve and the latent variable, follows a Poisson distribution with rate equal to $\lambda_{i,t}$

$$y_{i,t} | f_i, \mathbf{x}_t \sim \text{Poiss}(\exp(f_i(\mathbf{x}_t))). \tag{4.9}$$

4.2 The Spline-based Latent Variable Model

Our goal is still the same, in that we wish to model a latent variable $\mathbf{x}(t)$ that determines the behaviour of the spiking activity of N neurons. We therefore keep the notation of $\mathbf{y}_t$, $\mathbf{y}_j$ and $\mathbf{Y}$ from the LMT-model, denoting the vector of spike counts for all neurons, the vector for all time bins and the matrix of spike counts for all neurons and time bins, respectively.

4.2.1 The latent process

In similar fashion to the spike count notation, we also keep the setup of the latent process from the LMT-model, as the (partial) goal of both models is to recover this latent $\mathbf{x}(t)$. That means each component of the P-dimensional $\mathbf{x}(t)$ is modeled as a Gaussian processes, with the same covariance function described in Equation 4.2. Notation for $\mathbf{x}_t$, $\mathbf{x}_j$ and $\mathbf{X}$ will also remain the same.

4.2.2 The tuning curves and spiking model

When it comes to modeling the tuning curves, the spline-based approach deviates from the setup shown in Subsection 4.1.2. Still, we wish to model the function $h_i(\mathbf{x})$, the tuning curve of neuron i. Drawing again from the GLM framework, we keep the decision to consider the log of the tuning curve, but denote in this case the log tuning curve with $g_i(\mathbf{x}) = \log(h_i(\mathbf{x})), i = 1, \dots, N$, to differentiate from the log tuning curves of the LMT-model.

Instead of modeling the log tuning curve with a Gaussian process, we draw on the theory from Subsection 3.3.1 to model $g_i(\mathbf{x})$ with a B-Spline function of degree 3. Note that $g_i(\mathbf{x})$ here denotes an explicit real-valued function, and thus has no distribution

$$g_i(\mathbf{x}(t)) = \sum_{n=1}^{N_{\text{cp}}} \beta_{i,n} B_{n,3}(\mathbf{x}(t)). \tag{4.10}$$

Here B_n denotes the set of functions that make up the B-Spline basis of degree 3, specified in Equation 3.11, which is shared across all N g_i's. This basis is dependent on the total number of control points, which we denote N_{cp}, as well as the augmented knot vector $\boldsymbol{\tau}$, which is defined over a specific interval that we are interested in. This interval changes depending on the latent variable we are modeling, and thus the basis will look different for the various cases we will explore in Chapter 5. Keep in mind that the components of the latent $\mathbf{x}(t)$ are modeled with Gaussian processes, which technically means they can take values along the whole real line. This may conflict with the splines being defined only on the interval specified by the knot vector, notably in the inference part and when using gradient based searches for MAP estimation. To combat this problem, we define the splines periodically, repeating them along $\mathbb{R}$ in intervals of multiples of the length of the knot vector (note that this necessitates making an assumption of the domain of $\mathbf{X}$).

To keep the Bayesian setting intact, we give each $\beta_{i,n}$ a Gaussian prior distribution centred around zero

$$\beta_{i,n} \sim \mathcal{N}(0, \sigma^2_{\beta,[i,n]}), \tag{4.11}$$

with hyperparameter $\sigma^2_{\beta,[i,n]}$. The $\sigma^2_{\beta,[i,n]}$'s may then be given distributions of their own, but will in our case be considered to have a constant, specified value.

Since our target for the tuning curve modeling will be the spline coefficients, and not the spline function, we denote the vector of coefficients for the spline function for neuron i by $\boldsymbol{\beta}_i \in \mathbb{R}^{N_{\text{knots}}}$, and gather all $\boldsymbol{\beta}_i$ into the matrix $\boldsymbol{\beta} \in \mathbb{R}^{N \times N_{\text{cp}}}$.

Now that we have modeled the behaviour of the latent variable and the tuning curves, we can calculate the spike rate of neuron i at time t as $\lambda_{i,t} = \exp(g_i(\mathbf{x_t}))$. Then we assume that the spike count $y_{i,t}$ of neuron i in bin t, just like earlier, is Poisson distributed with firing rate equal to $\lambda_{i,t}$

$$y_{i,t}|\boldsymbol{\beta}_i, \mathbf{x}_t \sim \text{Poiss}(\exp(g_i(\mathbf{x}_t))). \tag{4.12}$$

4.3 Inference of the LMT model

Our goal is to find the point estimators $\hat{\mathbf{X}}^{\text{LMT}}_{\text{MAP}}$ and $\hat{\mathbf{F}}^{\text{LMT}}_{\text{MAP}}$ that maximize the posterior distribution of our parameters

$$\begin{aligned}\hat{\mathbf{X}}^{\text{LMT}}_{\text{MAP}}, \hat{\mathbf{F}}^{\text{LMT}}_{\text{MAP}} &= \text{argmax}_{\mathbf{X},\mathbf{F}} p(\mathbf{F}, \mathbf{X}, \boldsymbol{\xi}|\mathbf{Y}) \\ &= \text{argmax}_{\mathbf{X},\mathbf{F}} p(\mathbf{Y}, \mathbf{F}, \mathbf{X}, \boldsymbol{\xi}) \\ &= \text{argmax}_{\mathbf{X},\mathbf{F}} p(\mathbf{Y}|\mathbf{F}) p(\mathbf{F}|\mathbf{X}, \sigma_f, \delta_f, \sigma_\epsilon) p(\mathbf{X}|\sigma_x, \delta_x).\end{aligned} \tag{4.13}$$

Here we have omitted the term $[p(\mathbf{Y})]^{-1}$ in the second line, as it does not depend on neither $\mathbf{X}$ nor $\mathbf{F}$. Note that $\boldsymbol{\xi} = \{\sigma_f, \delta_f, \sigma_x, \delta_x\, \sigma_\epsilon\}$ denotes the set of hyperparameters, which we here assume to be known. The joint distribution becomes

$$
\begin{aligned}
p(\mathbf{Y},\mathbf{F},\mathbf{X},\boldsymbol{\xi}) =& p(\mathbf{Y}|\mathbf{F})p(\mathbf{F}|\mathbf{X},\sigma_f,\delta_f,\sigma_\epsilon)p(\mathbf{X}|\sigma_x,\delta_x) \\
=& \prod_{i=1}^{N}\prod_{t=1}^{T} p(y_{i,t}|f_{i,t}) \prod_{i=1}^{N} p(\mathbf{f}_i|\mathbf{X},\sigma_f,\delta_f,\sigma_\epsilon) \prod_{j=1}^{P} p(\mathbf{x}_j|\sigma_x,\delta_x) \\
=& \prod_{i=1}^{N}\prod_{t=1}^{T} \text{Poiss}(\exp(f_{i,t})) \prod_{i=1}^{N} \phi(\mathbf{f}_i;0,\mathbf{K}_x+\sigma_\epsilon^2\mathbf{I}) \prod_{j=1}^{P} \phi(\mathbf{x}_j;0,\mathbf{K}_t) \\
=& \prod_{i=1}^{N}\prod_{t=1}^{T} \frac{(\exp(f_{i,t}))^{y_{i,t}}}{y_{i,t}!} \exp(-\exp(f_{i,t})) \\
& \times \frac{1}{(2\pi)^{\frac{N}{2}}|\mathbf{K}_x+\sigma_\epsilon^2\mathbf{I}|^{\frac{N}{2}}} \exp\left(-\frac{1}{2}\sum_{i=1}^{N}\mathbf{f}_i^T\left[\mathbf{K}_x+\sigma_\epsilon^2\mathbf{I}\right]^{-1}\mathbf{f}_i\right) \\
& \times \frac{1}{(2\pi)^{\frac{P}{2}}|\mathbf{K}_t|^{\frac{P}{2}}} \exp\left(-\frac{1}{2}\sum_{j=1}^{P}\mathbf{x}_j^T\mathbf{K}_t^{-1}\mathbf{x}_j\right),
\end{aligned}
\tag{4.14}
$$

where we have assumed conditional independence between $y_{i,t}$, $\mathbf{f}_i$ and $\mathbf{x}_j$ respectively, and $\phi(\cdot;0,\mathbf{K})$ refers to the pdf of a multivariate normal distribution with 0 mean vector and covariance matrix $\mathbf{K}$. From here on we will also not refer to the hyperparameters, in order to simplify the notation.

Wu et al. (2017) use a method introduced as the decoupled Laplace approximation to perform the estimation, where a Laplace approximation of the tuning curve posterior is used as a tool to reduce the implicit dependency between $\mathbf{F}$ and $\mathbf{X}$. In contrast, we seek to iteratively compute the MAP estimate of $\mathbf{F}$ at iteration k, $\hat{\mathbf{F}}_{\text{MAP}}^{\text{LMT}^{(k)}}$, conditioned on the previous estimate of $\mathbf{X}$, $\hat{\mathbf{X}}_{\text{MAP}}^{\text{LMT}^{(k-1)}}$, then use this result to compute $\hat{\mathbf{X}}_{\text{MAP}}^{\text{LMT}^{(k)}}$, now conditioned on $\hat{\mathbf{F}}_{\text{MAP}}^{\text{LMT}^{(k)}}$. Iterations are then run until convergence.

To perform this iterative maximization, we first show how the posterior distributions can be calculated, before showcasing the corresponding algorithm.

4.3.1 MAP estimation of tuning curves

Given the assumed conditional independence, we can estimate the $\mathbf{f}_i$'s of $\mathbf{F}$ separately, before gathering them into $\mathbf{F}$. Using the standard relation from Bayes' rule, the posterior $p(\mathbf{f}_i|\mathbf{y}_i,\mathbf{X})$ is proportional to the joint distribution from Equation 4.14

$$
p(\mathbf{f}_i|\mathbf{y}_i,\mathbf{X}) \propto p(\mathbf{y}_i|\mathbf{f}_i)p(\mathbf{f}_i|\mathbf{X}). \tag{4.15}
$$

Since our goal is maximization, we consider the logarithm of the posterior, as this simplifies the expression substantially. Taking the logarithm, we see that

$$
\begin{aligned}
\log p(\mathbf{f}_i|\mathbf{y}_i,\mathbf{X}) &\propto \log p(\mathbf{y}_i|\mathbf{f}_i) + \log p(\mathbf{f}_i|\mathbf{X}) \\
&= \sum_{t=1}^{T}\Big[y_{i,t}f_{i,t}-\exp(f_{i,t})\Big] - \frac{1}{2}\mathbf{f}_i^T\left[\mathbf{K}_x+\sigma_\epsilon^2\mathbf{I}\right]^{-1}\mathbf{f}_i,
\end{aligned}
\tag{4.16}
$$

which we denote $\Psi_{\mathbf{F}}^{\text{LMT}}(\mathbf{f}_i)$, our objective function that we wish to maximize

$$
\Psi_{\mathbf{F}}^{\text{LMT}}(\mathbf{f}_i) := \sum_{t=1}^{T}\Big[y_{i,t}f_{i,t}-\exp(f_{i,t})\Big] - \frac{1}{2}\mathbf{f}_i^T\left[\mathbf{K}_x+\sigma_\epsilon^2\mathbf{I}\right]^{-1}\mathbf{f}_i. \tag{4.17}
$$

The elements of $\hat{\mathbf{F}}_{\text{MAP}}^{\text{LMT}}$ can then be found by

$$
\begin{aligned}
\hat{\mathbf{f}}_{i,\text{MAP}}^{\text{LMT}} &= \text{argmax}_{\mathbf{f}_i}\,\Psi_{\mathbf{F}}^{\text{LMT}}(\mathbf{f}_i) \\
&= \text{argmax}_{\mathbf{f}_i}\sum_{t=1}^{T}\Big[y_{i,t}f_{i,t}-\exp(f_{i,t})\Big] - \frac{1}{2}\mathbf{f}_i^T\left[\mathbf{K}_x+\sigma_\epsilon^2\mathbf{I}\right]^{-1}\mathbf{f}_i,
\end{aligned}
\tag{4.18}
$$

which can be solved using gradient-based optimization techniques. Since the gradient is simple to express analytically, and an analytical expression will improve the result of the optimization, we also compute this by differentiating the objective function from Equation 4.17. First, for clarity, note that here $\nabla = [\frac{\partial}{\partial f_{i,1}} \dots \frac{\partial}{\partial f_{i,T}}]^T$. To find $\nabla\Psi_{\mathbf{F}}^{\text{LMT}}(\mathbf{f}_i)$, we calculate the derivative of $\Psi_{\mathbf{F}}^{\text{LMT}}(\mathbf{f}_i)$

$$\begin{aligned}\frac{\partial}{\partial f_{i,t}}\Psi_{\mathbf{F}}^{\text{LMT}}(\mathbf{f}_i) &= y_{i,t} - \exp(f_{i,t}) - \sum_{j=1}^{T} f_{i,j}\left[\mathbf{K}_{x\{t,j\}} + \sigma_\epsilon^2\mathbf{I}_{\{t,j\}}\right]^{-1} \\ \iff \nabla\Psi_{\mathbf{F}}^{\text{LMT}}(\mathbf{f}_i) &= \mathbf{y}_i - \exp(\mathbf{f}_i) - \left[\mathbf{K}_x + \sigma_\epsilon^2\mathbf{I}\right]^{-1}\mathbf{f}_i,\end{aligned} \tag{4.19}$$

where we have used the fact that $\left[\mathbf{K}_x + \sigma_\epsilon^2\mathbf{I}\right]$ is symmetric. Here the vector $\exp(\mathbf{f}_i)$ has elements $\exp(f_{i,t})$, $t = 1, \dots, T$.

4.3.2 MAP estimation of the latent variable

Moving on to the inference of $\mathbf{X}$, due to how the spiking model is defined, conditioned on $\mathbf{F}$, the spiking model does not depend on $\mathbf{X}$, resulting in it not contributing to the posterior $p(\mathbf{X}|\mathbf{Y},\mathbf{F})$. Using Equation 4.14 again, we have the following relation

$$p(\mathbf{X}|\mathbf{Y},\mathbf{F}) \propto p(\mathbf{F}|\mathbf{X})p(\mathbf{X}). \tag{4.20}$$

Taking the logarithm and inserting expressions, we get

$$\begin{aligned}\log p(\mathbf{X}|\mathbf{Y},\mathbf{F}) &\propto \log p(\mathbf{F}|\mathbf{X}) + \log p(\mathbf{X}) \\ &= -\frac{N}{2}\log|\mathbf{K}_x + \sigma_\epsilon^2\mathbf{I}| - \frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{f}_i^T\left[\mathbf{K}_x + \sigma_\epsilon^2\mathbf{I}\right]^{-1}\mathbf{f}_i\right) - \frac{1}{2}\sum_{j=1}^{P}\left(\mathbf{x}_j^T\mathbf{K}_t^{-1}\mathbf{x}_j\right),\end{aligned} \tag{4.21}$$

which we will denote with $\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X})$.

Observe that since $\mathbf{K}_x$ is a function of $\mathbf{X}$, it must be calculated multiple times when performing gradient based optimization, a process that gets progressively more expensive as $\mathbf{X}$ increases in size. To combat the computational challenge this presents, we will draw on the theory of approximate Gaussian process techniques, namely the use of inducing points (see Quinonero-Candela and Rasmussen, 2005) to reduce the size of the matrix $\mathbf{K}_x$ that needs to be computed and inverted. Since this derivation is quite long and somewhat cumbersome, we will refer to Myklebust (2020) for a thorough derivation, and simply state the resulting expression when using inducing points.

First, let $\mathbf{u}$ denote the vector of function values at N_{ind} inducing points, uniformly spaced in the range of $\mathbf{x}$, and let the covariance matrices $\mathbf{K}_{\mathbf{u},\mathbf{u}} \in \mathbb{R}^{N_{\text{ind}}\times N_{\text{ind}}}$, $\mathbf{K}_{\mathbf{f},\mathbf{u}} \in \mathbb{R}^{T\times N_{\text{ind}}}$ and $\mathbf{K}_{\mathbf{u},\mathbf{f}} = \mathbf{K}^T$ be composed of elements

$$\begin{aligned}\mathbf{K}_{\mathbf{u},\mathbf{u}\{i,j\}} &= k_x(\mathbf{x}_{u_i}, \mathbf{x}_{u_j}) \\ \mathbf{K}_{\mathbf{f},\mathbf{u}\{i,j\}} &= k_x(\mathbf{x}_{t_i}, \mathbf{x}_{u_j}).\end{aligned} \tag{4.22}$$

Then, according to Myklebust (2020), we can define our objective function as

$$\begin{aligned}\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X}) :=& -\frac{N}{2}\log|\mathbf{K}_x + \sigma_\epsilon^2\mathbf{I}| - \frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{f}_i^T\left[\mathbf{K}_x + \sigma_\epsilon^2\mathbf{I}\right]^{-1}\mathbf{f}_i\right) - \frac{1}{2}\sum_{j=1}^{P}\left(\mathbf{x}_j^T\mathbf{K}_t^{-1}\mathbf{x}_j\right) \\ =& -\frac{1}{2}\sum_{i=1}^{N}\frac{N}{2}\log|\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{K}_{\mathbf{f},\mathbf{u}} + \sigma_\epsilon^2\mathbf{K}_{\mathbf{u},\mathbf{u}}| - \frac{N}{2}\log|\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}| - \frac{N(T - N_{\text{ind}})}{2}\log|\sigma_\epsilon^2| \\ & - \frac{1}{2\sigma_\epsilon^2}\sum_{i=1}^{N}\mathbf{f}_i^T\mathbf{f}_i + \frac{1}{2\sigma_\epsilon^2}\sum_{i=1}^{N}\left[\mathbf{f}_i^T\left(\mathbf{K}_{\mathbf{f},\mathbf{u}}\left(\sigma_\epsilon^2\mathbf{K}_{\mathbf{u},\mathbf{u}} + \mathbf{K}_{\mathbf{f},\mathbf{u}}^T\mathbf{K}_{\mathbf{f},\mathbf{u}}\right)^{-1}\mathbf{K}_{\mathbf{f},\mathbf{u}}^T\right)\mathbf{f}_i\right] \\ & - \frac{1}{2}\sum_{j=1}^{P}\left(\mathbf{x}_j^T\mathbf{K}_t^{-1}\mathbf{x}_j\right),\end{aligned} \tag{4.23}$$

whereby the MAP estimate can then be found, once again, using optimization techniques

$$\hat{\mathbf{X}}_{\text{MAP}}^{\text{LMT}} = \text{argmax}_{\mathbf{X}}\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X}). \tag{4.24}$$

An analytical expression for the gradient $\nabla\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X})$, which for clarity takes the following form (assuming a one-dimensional latent variable)

$$\nabla\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X}) = \left[\frac{\partial}{\partial x_1}\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X})\dots\frac{\partial}{\partial x_T}\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X})\right]^T \tag{4.25}$$

has also been derived by Myklebust (2020), with elements

$$\begin{aligned}\frac{\partial}{\partial x_t}\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X}) = &-\frac{N}{2}\text{trace}\left(B^{-1}\left[\left(\mathbf{K}_{\mathbf{u},\mathbf{f}}\left[\frac{\partial}{\partial x_t}\mathbf{K}_{\mathbf{f},\mathbf{u}}\right]\right)^T + \mathbf{K}_{\mathbf{u},\mathbf{f}}\left[\frac{\partial}{\partial x_t}\mathbf{K}_{\mathbf{f},\mathbf{u}}\right]\right]\right)\\ &+\frac{1}{2\sigma_\epsilon^2}\sum_{i=1}^{N}\mathbf{f}_i^T\left[\left(\mathbf{I}-\mathbf{K}_{\mathbf{f},\mathbf{u}}B^{-1}\mathbf{K}_{\mathbf{u},\mathbf{f}}\right)\left(\frac{\partial}{\partial x_t}\mathbf{K}_{\mathbf{f},\mathbf{u}}\right)B^{-1}\mathbf{K}_{\mathbf{u},\mathbf{f}}\right.\\ &\left.+\mathbf{K}_{\mathbf{f},\mathbf{u}}B^{-1}\left(\frac{\partial}{\partial x_t}\mathbf{K}_{\mathbf{u},\mathbf{f}}\right)\left(\mathbf{I}-\mathbf{K}_{\mathbf{f},\mathbf{u}}B^{-1}\mathbf{K}_{\mathbf{u},\mathbf{f}}\right)\right]\mathbf{f}_i\\ &-\mathbf{x}^T\mathbf{K}_{t\{.,t\}}^{-1},\end{aligned} \tag{4.26}$$

where $B = \mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{K}_{\mathbf{f},\mathbf{u}} + \sigma_\epsilon^2\mathbf{K}_{\mathbf{u},\mathbf{u}}$ and

$$\frac{\partial}{\partial x_t}\mathbf{K}_{\mathbf{f},\mathbf{u}} = \begin{bmatrix}\mathbf{0}_{t-1,N_{\text{ind}}}\\ \mathbf{k}_{\mathbf{f},\mathbf{u}}(x_t,\mathbf{x}_{\text{grid}})\\ \mathbf{0}_{T-t,N_{\text{ind}}}\end{bmatrix}, \tag{4.27}$$

with $\mathbf{k}_{\mathbf{f},\mathbf{u}}(x_t,\mathbf{x}_{\text{grid}}) = \left[k_{\mathbf{f},\mathbf{u}}(x_t,x_{u_1})\dots k_{\mathbf{f},\mathbf{u}}(x_t,x_{u_{N_{\text{ind}}}})\right]$, $k_{\mathbf{f},\mathbf{u}}(x_t,x_{u_j}) = -(x_t-x_{u_j})\frac{\sigma_f}{\delta_f^2}\exp\left(\frac{-(x_t-x_{u_j})^2}{(2\delta_f^2)}\right)$ and appropriately sized zero matrices.

4.4 Inference of the spline-based LVM

In a similar fashion to the inference of the LMT model, we look for point estimators for our latent variable $\hat{\mathbf{X}}_{\text{MAP}}^{\text{SPL}}$ and spline coefficients $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{SPL}}$

$$\begin{aligned}\hat{\mathbf{X}}_{\text{MAP}}^{\text{SPL}},\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{SPL}} &= \text{argmax}_{\mathbf{X},\boldsymbol{\beta}}p(\boldsymbol{\beta},\mathbf{X},\boldsymbol{\xi}|\mathbf{Y})\\ &= \text{argmax}_{\mathbf{X},\boldsymbol{\beta}}p(\mathbf{Y},\boldsymbol{\beta},\mathbf{X},\boldsymbol{\xi})\\ &= \text{argmax}_{\mathbf{X},\boldsymbol{\beta}}p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X})p(\boldsymbol{\beta}|\boldsymbol{\sigma}_\beta)p(\mathbf{X}|\sigma_x,\delta_x),\end{aligned} \tag{4.28}$$

where the normalizing constant has been omitted and $\boldsymbol{\xi}$ now contains the set of known hyperparameters $\boldsymbol{\xi} = \{\boldsymbol{\sigma}_\beta,\sigma_x,\delta_x\}$. We then have the following joint distribution

$$\begin{aligned}p(\boldsymbol{\beta},\mathbf{X},\boldsymbol{\xi}|\mathbf{Y}) =&p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X})p(\boldsymbol{\beta}|\boldsymbol{\sigma}_\beta,N_{\text{cp}},\boldsymbol{\tau})p(\mathbf{X}|\sigma_x,\delta_x)\\ =&\prod_{i=1}^{N}\prod_{t=1}^{T}p(y_{i,t}|\boldsymbol{\beta}_i,\mathbf{x}_t)\prod_{i=1}^{N}\prod_{n=1}^{N_{\text{cp}}}p(\beta_{i,n}|\sigma_{\beta,[i,n]})\prod_{j=1}^{P}p(\mathbf{x}_j|\sigma_x,\delta_x)\\ =&\prod_{i=1}^{N}\prod_{t=1}^{T}\text{Poiss}(\exp(g_i(\mathbf{x}_t)))\prod_{i=1}^{N}\prod_{n=1}^{N_{\text{cp}}}\mathcal{N}(\beta_{i,n};0,\sigma_{\beta,[i,n]}^2)\prod_{j=1}^{P}\phi(\mathbf{x}_j;0,\mathbf{K}_t)\\ =&\prod_{i=1}^{N}\prod_{t=1}^{T}\frac{(\exp(g_i(\mathbf{x}_t)))^{y_{i,t}}}{y_{i,t}!}\exp(-\exp(g_i(\mathbf{x}_t)))\\ &\times\prod_{i=1}^{N}\prod_{n=1}^{N_{\text{cp}}}\frac{1}{(2\pi)^{\frac{1}{2}}\sigma_{\beta,[i,n]}}\exp\left(-\frac{1}{2}\frac{\beta_{i,n}^2}{\sigma_{\beta,[i,n]}^2}\right)\\ &\times\frac{1}{(2\pi)^{\frac{P}{2}}|\mathbf{K}_t|^{\frac{P}{2}}}\exp\left(-\frac{1}{2}\sum_{j=1}^{P}\mathbf{x}_j^T\mathbf{K}_t^{-1}\mathbf{x}_j\right),\end{aligned} \tag{4.29}$$

where conditional independence has been assumed, and ϕ and $\mathcal{N}$ have been specified in earlier sections. We will again stop referring to the hyperparamters in our notation, when dealing with densities, to simplify whenever possible.

Do note that N_{cp} and $\boldsymbol{\tau}$ may be considered hyperparameters of the B-Spline function, and as such could be included in the set $\boldsymbol{\xi}$. We will however not always consider these fixed, changing them depending on our problem of interest, and as such, special attention will be given to how N_{cp} and $\boldsymbol{\tau}$ are chosen in Chapter 5. They have therefore been left out from $\boldsymbol{\xi}$, our set of fixed hyperparameters.

The iterative MAP procedure, as we will see in Section 4.5, will be somewhat similar to the LMT-case. Thus, we first calculate the posterior distributions and gradients necessary for maximization.

4.4.1 MAP estimation of spline coefficients

From Equation 4.29 we have that

$$p(\boldsymbol{\beta}|\mathbf{Y},\mathbf{X}) \propto p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X})p(\boldsymbol{\beta}). \tag{4.30}$$

Taking the logarithm to simplify, and inserting expressions for the densities, we get the following

$$\begin{aligned}\log p(\boldsymbol{\beta}|\mathbf{Y},\mathbf{X}) &\propto \log p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X}) \log p(\boldsymbol{\beta}) \\ &= \sum_{i=1}^{N}\sum_{t=1}^{T}\Big[y_{i,t}g_i(\mathbf{x}_t) - \exp(g_i(\mathbf{x}_t))\Big] - \sum_{i=1}^{N}\sum_{n=1}^{N_{\text{cp}}}\frac{1}{2}\frac{\beta_{i,n}^2}{\sigma_{\beta,[i,n]}^2},\end{aligned} \tag{4.31}$$

which we will denote $\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta})$. As a reminder, $g_i(\mathbf{x}_t)$ refers to the B-Spline function of degree 3 corresponding to neuron i, as specified in Equation 4.10. We then find $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{SPL}}$ by maximizing this expression

$$\begin{aligned}\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{SPL}} &= \text{argmax}_{\boldsymbol{\beta}}\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta}) \\ &= \text{argmax}_{\boldsymbol{\beta}}\sum_{i=1}^{N}\sum_{t=1}^{T}\Big[y_{i,t}g_i(\mathbf{x}_t) - \exp(g_i(\mathbf{x}_t))\Big] - \sum_{i=1}^{N}\sum_{n=1}^{N_{\text{cp}}}\frac{1}{2}\frac{\beta_{i,n}^2}{\sigma_{\beta,[i,n]}^2}.\end{aligned} \tag{4.32}$$

As for an analytical expression for the gradient of our objective function, we differentiate row-wise with respect to the $\beta_{i,j}$'s, giving the following

$$\nabla\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta}) = \left[\frac{\partial}{\partial\beta_{1,1}}\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta})\dots\frac{\partial}{\partial\beta_{1,N_{\text{cp}}}}\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta})\dots\frac{\partial}{\partial\beta_{N,1}}\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta})\dots\frac{\partial}{\partial\beta_{N,N_{\text{cp}}}}\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta})\right]^T, \tag{4.33}$$

where $\nabla\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta}) \in \mathbb{R}^{NN_{\text{cp}}}$ is a vector of length N times N_{cp}. Subsequently, its elements take the form

$$\frac{\partial}{\partial\beta_{i,n}}\Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta}) = \sum_{t=1}^{T}\Big[y_{i,t}B_{n,3}(\mathbf{x}_t) - \exp(g_i(\mathbf{x}_t))B_{n,3}(\mathbf{x}_t)\Big] - \frac{\beta_{i,n}}{\sigma_{\beta,[i,n]}^2}, \tag{4.34}$$

where $B_{n,3}(\mathbf{x}_t)$ is the n'th B-Spline basis of degree 3, evaluated at $\mathbf{x}_t = \mathbf{x}(t)$.

4.4.2 MAP estimation of the latent variable

In the case of the Spline-based LVM, the spiking model still contributes to the likelihood, as the B-Spline function itself does not have a distribution and is dependent on the argument $\mathbf{X}$. The relationship, using Equation 4.29, takes the following form

$$p(\mathbf{X}|\mathbf{Y},\boldsymbol{\beta}) \propto p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X})p(\mathbf{X}), \tag{4.35}$$

and taking the logarithm

$$\begin{aligned}\log p(\mathbf{X}|\mathbf{Y},\boldsymbol{\beta}) &\propto \log p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X}) + \log p(\mathbf{X}) \\ &= \sum_{i=1}^{N}\sum_{t=1}^{T}\Big[y_{i,t}g_i(\mathbf{x}_t) - \exp(g_i(\mathbf{x}_t))\Big] - \frac{1}{2}\sum_{j=1}^{P}\left(\mathbf{x}_j^T\mathbf{K}_t^{-1}\mathbf{x}_j\right),\end{aligned} \tag{4.36}$$

gives the familiar MAP expression

$$\begin{aligned}\hat{\mathbf{X}}_{\text{MAP}}^{\text{SPL}} &= \text{argmax}_{\mathbf{X}} \Psi_{\mathbf{X}}^{\text{SPL}}(\mathbf{X}) \\ &= \text{argmax}_{\mathbf{X}} \sum_{i=1}^{N} \sum_{t=1}^{T} \Big[y_{i,t} g_i(\mathbf{x}_t) - \exp(g_i(\mathbf{x}_t)) \Big] - \frac{1}{2} \sum_{j=1}^{P} \left(\mathbf{x}_j^T \mathbf{K}_t^{-1} \mathbf{x}_j \right) .\end{aligned} \tag{4.37}$$

With the change from Gaussian process to a B-Spline function, we also circumvent to computational challenges that necessitated the use of inducing points, resulting in a much simpler expression for the gradient

$$\nabla \Psi_{\mathbf{X}}^{\text{SPL}}(\mathbf{X}) = \left[\frac{\partial}{\partial \mathbf{x}_1} \Psi_{\mathbf{X}}^{\text{SPL}}(\mathbf{X}) \dots \frac{\partial}{\partial \mathbf{x}_T} \Psi_{\mathbf{X}}^{\text{SPL}}(\mathbf{X}) \right]^T , \tag{4.38}$$

with elements

$$\frac{\partial}{\partial \mathbf{x}_t} \Psi_{\mathbf{X}}^{\text{SPL}}(\mathbf{X}) = \sum_{i=1}^{N} \Big[y_{i,t} \frac{\partial}{\partial \mathbf{x}_t} g_i(\mathbf{x}_t) - \exp(g_i(\mathbf{x}_t)) \frac{\partial}{\partial \mathbf{x}_t} g_i(\mathbf{x}_t) \Big] - \sum_{j=1}^{P} \mathbf{K}_t^{-1} \mathbf{x}_j . \tag{4.39}$$

The derivative of our B-Spline function $\frac{\partial}{\partial \mathbf{x}_t} g_i(\mathbf{x}_t) = \sum_{n=1}^{N_{\text{cp}}} \beta_{i,n} \frac{\partial}{\partial \mathbf{x}_t} B_{n,3}(\mathbf{x}(t))$ can be expressed in the following manner

$$\sum_{n=1}^{N_{\text{cp}}} \beta_{i,n} \frac{\partial}{\partial \mathbf{x}_t} B_{n,3}(\mathbf{x}(t)) = \sum_{n=1}^{N_{\text{cp}}} \beta_{i,n} \left[\frac{3}{\tau_{n+3} - \tau_n} B_{n,2}(\mathbf{x}(t)) - \frac{3}{\tau_{n+3+1} - \tau_{n+1}} B_{n+1,2}(\mathbf{x}(t)) \right] , \tag{4.40}$$

with τ-values originating from the augmented knot vector $\boldsymbol{\tau}$ (see Section 3.3.1 for definition). The differentiation is a known result and can be found in e.g. Piegl and Tiller (1996).

4.5 The iterative MAP procedure

The idea behind the iterative MAP procedure is the same for both the LMT and the Spline-based LVM. Starting from some initial value for $\mathbf{X}$ and either $\mathbf{F}$ or $\boldsymbol{\beta}$, we use optimization techniques to maximize the corresponding posterior expressions iteratively until convergence is satisfied. The choice of initialisation is an important one, as our target functions are not (necessarily) convex, and the result might not be the global maxima of interest, but rather some local maxima found due to poor initialisation. The discussion surrounding initialisation will be addressed further in the upcoming chapter. As for the optimization technique, we utilize the L-BFGS-B (see Byrd et al., 1995) algorithm, an efficient extension of the original gradient-based BFGS (Nocedal and Wright, 2006). The convergence criterion is similar for both MAP procedures, in that the algorithm terminates whenever the Euclidean distance between $\hat{\mathbf{X}}_{\text{MAP}}^{\dots(k)}$ and $\hat{\mathbf{X}}_{\text{MAP}}^{\dots(k-1)}$ becomes smaller than a specified tolerance, or if the iterations exceeds a predetermined limit for the number of iterations. The first of the MAP procedures is shown in Algorithm 1.

Algorithm 1: Iterative MAP procedure for LMT

```
   Input: observations $\mathbf{Y}$, initial guess $\mathbf{X}^0$, $\mathbf{F}^0$, hyperparameters
1  begin
2      while not converged do
3          for $i = 1, \dots, N$ do
4              Find $\hat{\mathbf{f}}_{i,\text{MAP}}^{\text{LMT}^{(k)}}$ by solving (4.18)
5          end
6          Find $\hat{\mathbf{X}}_{\text{MAP}}^{\text{LMT}^{(k)}}$ by solving (4.24)
7          Adjust $\sigma_\epsilon^2 = \sigma_\epsilon^2 \times \chi$
8      end
9      return $\hat{\mathbf{F}}_{\text{MAP}}^{\text{LMT}^{(k)}}$, $\hat{\mathbf{X}}_{\text{MAP}}^{\text{LMT}^{(k)}}$
10 end
```

We follow Myklebust (2020) by utilising a learning rate parameter $0 < \chi < 1$, which controls the noise term σ_ϵ^2 in our Gaussian process (see Equation 4.8), to allow for more efficient exploration during the early stages of the optimization. We will also use a similar initialisation for the $\mathbf{F}^0$ as Myklebust (2020), and so we will consequently also be skipping the first update of $\hat{\mathbf{F}}_{\text{MAP}}^{\text{LMT}^{(k)}}$, by their recommendation.

The quite similar MAP procedure for the Spline-based LVM is shown in Algorithm 2.

Algorithm 2: Iterative MAP procedure for Spline-based LVM

```
  Input: observations $\mathbf{Y}$, initial guess $\mathbf{X}^0$, $\boldsymbol{\beta}^0$, hyperparameters
1 begin
2     while not converged do
3         Find $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{SPL}^{(k)}}$ by solving (4.32)
4         Find $\hat{\mathbf{X}}_{\text{MAP}}^{\text{SPL}^{(k)}}$ by solving (4.37)
5     end
6     return $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{SPL}^{(k)}}$, $\hat{\mathbf{X}}_{\text{MAP}}^{\text{SPL}^{(k)}}$
7 end
```

Observe that as the B-Spline function has no distribution itself, we have no way of controlling how freely the targeted space can be explored, like we have with the noise parameter for the LMT. To combat this potential issue, a proposed solution is to not run the L-BFGS-B solver all the way to convergence when solving for $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{SPL}}$, but instead take a single gradient-step. That way, we avoid the possibility of having the solver converging prematurely to a local maxima too close to the initialisation. If the initialisation of $\mathbf{X}$ is good, however, this single-step might not be all that necessary. This, as well as choices regarding the number of control points and placement of the knot vector, will be discussed in the next chapter.

4.6 Feature Sharing, an extension to the Spline-based LVM

One of the strengths of the Spline-based LVM is its easily customisable log tuning curve function $g_i(\mathbf{x})$, in contrast to the non-parametric Gaussian process in the LMT-model. One such extension, inspired by the works of Klindt et al. (2017), is applicable when the neurons in question all share similarly shaped tuning curves of varying strength and position, like e.g. in the case neurons tuned to visual stimuli, head direction or spatial position (Albright, 1984; Taube et al., 1990; Hafting et al., 2005). In such cases, we may assume the B-Spline function modeling $g_i(\mathbf{x})$ to be shared by all N neurons, augmented with a scaling variable α_i controlling the strength of tuning, and two shift variables θ_i and $\boldsymbol{\gamma}_i$, which controls the position of the tuning curve along the first and second axis respectively

$$g_i^{\text{FS}}(\mathbf{x}(t)) = \alpha_i \sum_{n=1}^{N_{\text{cp}}} \beta_n B_{n,3}(\mathbf{x}(t) + \boldsymbol{\theta}_i) + \gamma_i. \tag{4.41}$$

The idea behind this feature sharing is hardly new, and has been utilised in the context of neural networks (as well as more traditional GLMs) by e.g. Klindt et al. (2017), Batty et al. (2017), but, to our knowledge, not in conjunction with a Spline-based LVM before. (That is, in a neuroscientific setting. The applications of a latent variable model with shared features are very broad, and might already be used in other fields.) As has been argued for by said authors, feature sharing lets us utilise the whole set of neuronal data to infer a single shared tuning curve, instead of only using the data from one neuron to infer its own tuning curve. Additionally, by assuming a known and suitable shape to the tuning curves, we can possibly infer the other variables more easily, compared to when one uses models where it is harder to make an educated guess with respect to the initialisation.

The notation for $\mathbf{Y}, \mathbf{X}$ will be kept the same as in the case without feature sharing. Note that $\boldsymbol{\beta}$ is now no longer a matrix of coefficents, as the coefficients are shared across all neurons, and thus instead represents a vector $\boldsymbol{\beta} \in \mathbb{R}^{N_{\text{cp}}}$. Similarly, the α_i's and γ_i's are gathered into the vectors $\boldsymbol{\alpha} \in \mathbb{R}^N$ and $\boldsymbol{\gamma} \in \mathbb{R}^N$, while the $\boldsymbol{\theta}_i$'s are collected into the matrix $\boldsymbol{\theta} \in R^{N \times P}$. Each α_i is given a Gamma prior distribution

$$\alpha_i \sim \text{Gamma}(s_i, r_i), \qquad \alpha_i > 0 \tag{4.42}$$

with some hyperparameters s_i, r_i. The $\boldsymbol{\theta}_i$'s are given a uniform prior,

$$\boldsymbol{\theta}_i \sim \text{Unif}(\boldsymbol{\theta}_{\text{lower}}, \boldsymbol{\theta}_{\text{upper}}), \tag{4.43}$$

where $\boldsymbol{\theta}_{\text{lower}}, \boldsymbol{\theta}_{\text{upper}}$ are determined based on the particular space of interest, similarly to how we define the augmented knot vector $\boldsymbol{\tau}$ for the B-Spline basis. Finally, each γ_i is given a Gaussian distribution

$$\gamma_i \sim \mathcal{N}(\mu_{\gamma_i}, \sigma^2_{\gamma_i}), \tag{4.44}$$

with some hyperparameters $\mu_{\gamma_i}, \sigma^2_{\gamma_i}$.

To motivate our choice of variables $\boldsymbol{\alpha}, \boldsymbol{\theta}, \boldsymbol{\gamma}$, we present a visualisation of how the three variables affect the shape of the tuning curve. Consider Figure 4.1, which shows a possible realisation of a tuning curve (note that $g_i^{\text{FS}}(\mathbf{x}(t))$ here has been exponentiated, as it models the log tuning curve, and not the tuning curve itself).

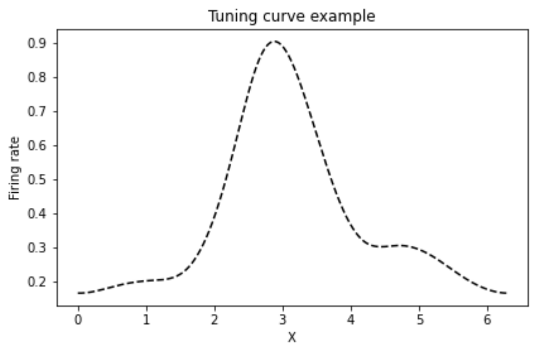

Figure 4.1: Figure showing a possible tuning curve shape for a neuron responding to a 2π-periodic variable.

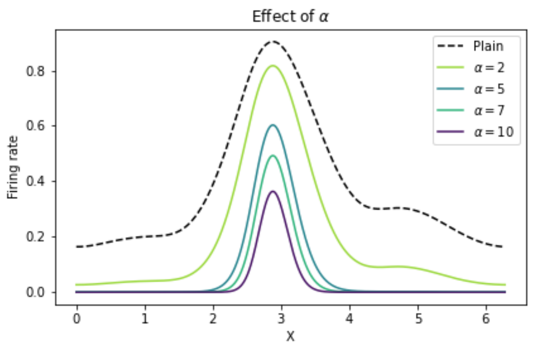

Figure 4.2: Figure showing the effect of the variable α_i on the shape of the tuning curve.

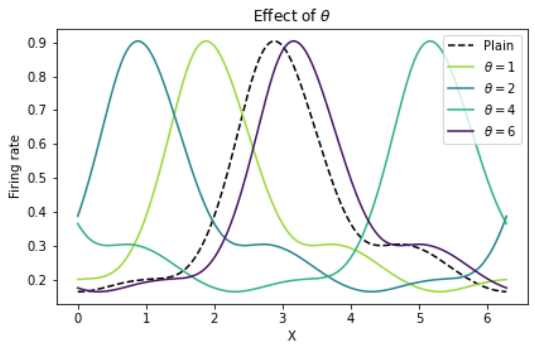

Figure 4.3: Figure showing the effect of the variable θ_i on the shape of the tuning curve.

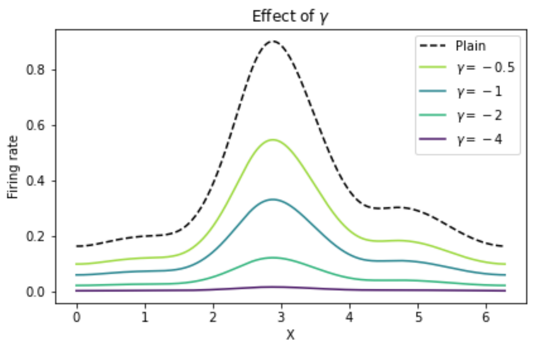

Figure 4.4: Figure showing the effect of the variable γ_i on the shape of the tuning curve.

First, we note that the sign of $g_i^{\text{FS}}(\mathbf{x}(t))$ in this case is negative, since the tuning curve is less than one at all points. The "Plain" tuning curve has variables set to $\alpha_i = 1, \theta_i = 0, \gamma_i = 0$. Increasing α_i scales the tuning curve down (due to the negative sign), for which the result can be see in Figure 4.2. Note that this not only shrinks the height of the tuning curve, but also the width. Changing θ_i, as seen in Figure 4.3, results in a pretty straightforward change; the tuning curve is shifted along the X-axis, where a shift of any multiple of $\theta_i = 2\pi$ returns the tuning curve to the original position. Finally, the change in γ_i (Figure 4.4) also shifts the log tuning curve, but along the second axis instead. Exponentiated, this has the same effects as multiplying the tuning curve with some constant. Instead of scaling both the width and height, this variable only affects the height, with the width of the tuning staying the same.

4.7 Inference, feature sharing

Inference is done in much the same way as has been shown in the earlier sections, so in an effort to not repeat ourselves too much, we briefly go over the resulting changes to the inference procedure when utilising feature sharing, and describe the resulting changes in the MAP procedure. Inference will be shown for a one-dimensional latent variable, since this will be the dimensionality of interest in our application chapter. Nonetheless, the inference is easily extended to a higher dimensional latent variable.

4.7.1 Expressions

Starting with the spline coefficients, we now simply state the relation between the log posterior, the log likelihood and the log prior. The new model, shown in Equation 4.41, slightly changes the expression, which we denote $\Psi^{\text{FS}}_{\boldsymbol{\beta}}(\boldsymbol{\beta})$, one of our target functions for the MAP estimation

$$\begin{aligned}\log p(\boldsymbol{\beta}|\mathbf{Y},\mathbf{X},\boldsymbol{\alpha},\boldsymbol{\theta},\boldsymbol{\gamma}) \propto& \log p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X},\boldsymbol{\alpha},\boldsymbol{\theta},\boldsymbol{\gamma}) \log p(\boldsymbol{\beta})\\ =&\sum_{i=1}^{N}\sum_{t=1}^{T}\Big[y_{i,t}g_i^{\text{FS}}(x_t)-\exp(g_i^{\text{FS}}(x_t))\Big]-\sum_{n=1}^{N_{\text{cp}}}\frac{1}{2}\frac{\beta_n^2}{\sigma^2_{\beta,[n]}}\\ :=&\Psi^{\text{FS}}_{\boldsymbol{\beta}}(\boldsymbol{\beta}).\end{aligned} \tag{4.45}$$

This target gives rise to the following formula for the gradient

$$\nabla\Psi^{\text{FS}}_{\boldsymbol{\beta}}(\boldsymbol{\beta})=\left[\frac{\partial}{\partial\beta_1}\Psi^{\text{SPL}}_{\boldsymbol{\beta}}(\boldsymbol{\beta})\dots\frac{\partial}{\partial\beta_{N_{\text{cp}}}}\Psi^{\text{SPL}}_{\boldsymbol{\beta}}(\boldsymbol{\beta})\right]^T, \tag{4.46}$$

with elements

$$\frac{\partial}{\partial\beta_n}\Psi^{\text{FS}}_{\boldsymbol{\beta}}(\boldsymbol{\beta})=\sum_{n=1}^{N}\sum_{t=1}^{T}\Big[y_{i,t}\alpha_iB_{n,3}(x_t+\theta_i)-\exp(g_i^{\text{FS}}(x_t))\alpha_iB_{n,3}(x_t+\theta_i)\Big]-\frac{\beta_n}{\sigma^2_{\beta,[n]}}. \tag{4.47}$$

The newly introduced variables follow roughly the same pattern, with target functions defined through the following relations

$$\begin{aligned}\log p(\boldsymbol{\theta}|\mathbf{Y},\mathbf{X},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{\gamma}) \propto& \log p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X},\boldsymbol{\alpha},\boldsymbol{\theta},\boldsymbol{\gamma})\\ =&\sum_{i=1}^{N}\sum_{t=1}^{T}\Big[y_{i,t}g_i^{\text{FS}}(x_t)-\exp(g_i^{\text{FS}}(x_t))\Big]\\ :=&\Psi^{\text{FS}}_{\boldsymbol{\theta}}(\boldsymbol{\theta}),\end{aligned} \tag{4.48}$$

$$\begin{aligned}\log p(\boldsymbol{\gamma}|\mathbf{Y},\mathbf{X},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{\theta}) \propto& \log p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X},\boldsymbol{\alpha},\boldsymbol{\theta},\boldsymbol{\gamma})\\ =&\sum_{i=1}^{N}\sum_{t=1}^{T}\Big[y_{i,t}g_i^{\text{FS}}(x_t)-\exp(g_i^{\text{FS}}(x_t))\Big]-\sum_{i=1}^{N}\frac{1}{2}\frac{(\gamma_i-\mu_{\gamma_i})^2}{\sigma^2_{\gamma_i}}\\ :=&\Psi^{\text{FS}}_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}),\end{aligned} \tag{4.49}$$

$$\begin{aligned}\log p(\boldsymbol{\alpha}|\mathbf{Y},\mathbf{X},\boldsymbol{\beta},\boldsymbol{\theta}) \propto& \log p(\mathbf{Y}|\boldsymbol{\beta},\mathbf{X},\boldsymbol{\alpha},\boldsymbol{\theta})+\log p(\boldsymbol{\alpha})\\ =&\sum_{i=1}^{N}\sum_{t=1}^{T}\Big[y_{i,t}g_i^{\text{FS}}(x_t)-\exp(g_i^{\text{FS}}(x_t))\Big]+\sum_{i=1}^{N}(s_i-1)\log\alpha_i-r_i\alpha_i\\ :=&\Psi^{\text{FS}}_{\boldsymbol{\alpha}}(\boldsymbol{\alpha}),\end{aligned} \tag{4.50}$$

and with gradient elements equal to

$$\frac{\partial}{\partial\theta_n}\Psi^{\text{FS}}_{\boldsymbol{\theta}}(\boldsymbol{\theta})=\sum_{i=1}^{N}\Big[y_{i,t}\frac{\partial}{\partial\theta_n}g_i^{\text{FS}}(x_t)-\exp(g_i^{\text{FS}}(x_t))\frac{\partial}{\partial\theta_n}g_i^{\text{FS}}(x_t)\Big], \tag{4.51}$$

$$\frac{\partial}{\partial\gamma_i}\Psi^{\text{FS}}_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})=\sum_{i=1}^{N}\Big[y_{i,t}-\exp(g_i^{\text{FS}}(x_t))\Big]-\frac{(\gamma_i-\mu_{\gamma_i})}{\sigma^2_{\gamma_i}}, \tag{4.52}$$

$$\frac{\partial}{\partial\alpha_i}\Psi^{\text{FS}}_{\boldsymbol{\alpha}}(\boldsymbol{\alpha})=\sum_{t=1}^{T}\left[y_{i,t}\sum_{n=1}^{N_{\text{cp}}}\Big[\beta_nB_{n,3}(x_t+\theta_i)\Big]-\exp(g_i^{\text{FS}}(x_t))\sum_{n=1}^{N_{\text{cp}}}\Big[\beta_nB_{n,3}(x_t+\theta_i)\Big]\right]+\frac{s_i-1}{\alpha_i}-r_i. \tag{4.53}$$

Note that $\frac{\partial}{\partial \theta_n} g_i^{\text{FS}}(x_t)$ actually takes the same form as the derivative $\frac{\partial}{\partial \mathbf{x}_t} g_i(\mathbf{x}_t)$ in Equation 4.40, only with the $\mathbf{x}(t)$ exchanged with $\mathbf{x}(t) + \theta_n$ (and $\mathbf{x}(t)$ being a one-dimensional variable), as well as being timed with α_i.

Finally, we have the latent variable itself, which more or less keeps its familiar expression

$$\begin{aligned}\log p(\mathbf{X}|\mathbf{Y}, \boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\theta}) &\propto \log p(\mathbf{Y}|\boldsymbol{\beta}, \mathbf{X}, \boldsymbol{\alpha}, \boldsymbol{\theta}) + \log p(\mathbf{X}) \\ &= \sum_{i=1}^{N} \sum_{t=1}^{T} \left[y_{i,t} g_i^{\text{FS}}(x_t) - \exp(g_i^{\text{FS}}(x_t)) \right] - \frac{1}{2} \left(\mathbf{x}^T \mathbf{K}_t^{-1} \mathbf{x} \right) \\ &:= \Psi_{\mathbf{X}}^{\text{FS}}(\mathbf{X}),\end{aligned} \tag{4.54}$$

including the expression for its derivative

$$\frac{\partial}{\partial x_t} \Psi_{\mathbf{X}}^{\text{FS}}(\mathbf{X}) = \sum_{i=1}^{N} \left[y_{i,t} \frac{\partial}{\partial x_t} g_i^{\text{FS}}(x_t) - \exp(g_i^{\text{FS}}(x_t)) \frac{\partial}{\partial x_t} g_i^{\text{FS}}(x_t) \right] - \mathbf{K}_t^{-1} \mathbf{x}. \tag{4.55}$$

With all five target functions, and their derivatives derived, we have acquired the necessary expressions to perform our estimation

$$\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{FS}} = \text{argmax}_{\boldsymbol{\beta}} \Psi_{\boldsymbol{\beta}}^{\text{FS}}(\boldsymbol{\beta}), \tag{4.56a}$$

$$\hat{\boldsymbol{\theta}}_{\text{MAP}}^{\text{FS}} = \text{argmax}_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}^{\text{FS}}(\boldsymbol{\theta}), \tag{4.56b}$$

$$\hat{\gamma}_{\text{MAP}}^{\text{FS}} = \text{argmax}_{\gamma} \Psi_{\gamma}^{\text{FS}}(\gamma), \tag{4.56c}$$

$$\hat{\boldsymbol{\alpha}}_{\text{MAP}}^{\text{FS}} = \text{argmax}_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}}^{\text{FS}}(\boldsymbol{\alpha}), \tag{4.56d}$$

$$\hat{\mathbf{X}}_{\text{MAP}}^{\text{FS}} = \text{argmax}_{\mathbf{X}} \Psi_{\mathbf{X}}^{\text{FS}}(\mathbf{X}). \tag{4.56e}$$

4.7.2 MAP procedure

As previously shown, we alternate between maximizing the various expressions to iteratively estimate our set of target variables. The MAP procedure when using feature sharing is showcased in Algorithm 3.

Algorithm 3: Iterative MAP procedure for Spline-based LVM, using feature sharing

```
   Input: observations $\mathbf{Y}$, initial guess $\mathbf{X}^0$, $\boldsymbol{\beta}^0$,$\boldsymbol{\theta}^0$,$\gamma^0$,$\boldsymbol{\alpha}^0$ hyperparameters
1  begin
2      while not converged do
3          Find $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{FS}^{(k)}}$ by solving (4.56a)
4          Find $\hat{\boldsymbol{\theta}}_{\text{MAP}}^{\text{FS}^{(k)}}$ by solving (4.56b)
5          Find $\hat{\gamma}_{\text{MAP}}^{\text{FS}^{(k)}}$ by solving (4.56c)
6          Find $\hat{\boldsymbol{\alpha}}_{\text{MAP}}^{\text{FS}^{(k)}}$ by solving (4.56d)
7          Find $\hat{\mathbf{X}}_{\text{FS}}^{\text{FS}^{(k)}}$ by solving (4.56e)
8      end
9      return $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{FS}^{(k)}}$,$\hat{\boldsymbol{\theta}}_{\text{MAP}}^{\text{FS}^{(k)}}$,$\hat{\gamma}_{\text{MAP}}^{\text{FS}^{(k)}}$, $\hat{\boldsymbol{\alpha}}_{\text{MAP}}^{\text{FS}^{(k)}}$, $\hat{\mathbf{X}}_{\text{MAP}}^{\text{FS}^{(k)}}$
10 end
```

Since we wish to utilise the assumption behind feature sharing in the most efficient way, we skip the first update of $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{FS}^{(k)}}$, to keep the initial tuning curve shape intact while estimating the other variables. This argument is similar to why the first $\hat{\mathbf{F}}_{\text{MAP}}^{\text{LMT}^{(k)}}$ update is proposed skipped in Algorithm 1, as the initialisation for the other variables $\mathbf{X}^0$, $\boldsymbol{\theta}^0$, γ^0, $\boldsymbol{\alpha}^0$ might be less accurate than the $\boldsymbol{\beta}^0$, thus worsening the estimate if we immediately solve for $\hat{\boldsymbol{\beta}}_{\text{MAP}}^{\text{FS}^{(k)}}$, conditioned on poorer initialised variables.

4.8 A note on inference regarding periodicity

When deriving the expressions for the models utilising a spline-based approach in the previous sections, an underlying assumption that the spline function is non-periodic has been made. However, in cases where the latent variable is indeed periodic, tuning curves are also assumed to be continuous across the boundaries of the defined interval of interest. This is simple to account for, as discussed in Subsection 3.3.1, but this slight change in how the control points behave also changes the inference slightly. Notably, since the first three (due to the use of a degree three B-Spline) control points must be exactly the same as the three last to obtain periodicity, we exchange $\beta_{N_{\text{cp}}-2}$ with β_1, $\beta_{N_{\text{cp}}-1}$ with β_2 and $\beta_{N_{\text{cp}}}$ with β_3. This results in three fewer variables in $\boldsymbol{\beta}$ ($3N$ fewer, in the non-feature sharing case), although the B-Spline function itself still has N_{cp} control points. Thus when calculating the gradients in Equations 4.33 and 4.46, the gradient elements associated with β_1, β_2 and β_3 will get the contributions from the likelihood that originally belonged to $\beta_{N_{\text{cp}}-2}$, $\beta_{N_{\text{cp}}-1}$ and $\beta_{N_{\text{cp}}}$. Note that the prior contribution is still the same, and is not ”doubled”.

Chapter 5

Data Analysis

In this chapter, we showcase the results from applying the models from Chapter 4 to simulated and real data. In Section 5.1, we discuss modeling choices and some challenges related to the different models presented in the previous chapter, before moving on to Section 5.2, where we compare how the models scale with respect to data size in various cases. Finally, we use the models to infer the head direction of a mouse based on real neural data in Section 5.3.

5.1 Modeling choices and challenges

5.1.1 Evaluating the results

Since we will be comparing performances in the upcoming sections, both between different models and depending on initialisations/choice of hyperparameters, we will need some form of measure to decide whether a model is ”better” than another. Our evaluation method of choice is the root mean squared error (RMSE), which measures how well the inferred latent path $\hat{\mathbf{X}}$ does compared to the true $\mathbf{X}$ by taking the square root of the mean squared error over all the time points

$$\text{RMSE} = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(\mathbf{x}_t - \hat{\mathbf{x}}_t)^2}. \tag{5.1}$$

Taking the square root ensures that the error is proportional to the scale of the inferred $\hat{\mathbf{X}}$. An argument can be made for using the mean absolute error instead (MAE), which differs from the RMSE by considering the absolute difference between the elements of $\hat{\mathbf{X}}$ and $\mathbf{X}$, instead of squaring them before averaging

$$\text{MAE} = \frac{1}{T}\sum_{i=1}^{T}|\mathbf{x}_t - \hat{\mathbf{x}}_t|. \tag{5.2}$$

Due to the squared difference used in the RMSE, elements of $\hat{\mathbf{X}}$ far away from the truth are penalized more compared to the MAE, which is a feature to be aware of. As we already discussed in Subsection 5.1.3, the Spline-based LVM has a tendency of producing results that might suddenly deviate from the true path by a large margin. While upon inspection, these deviations might be easy to spot and account for (we might for instance scale them back to the original area of interest), such a jump will be heavily penalized when using the RMSE measure. Where such large and obvious deviations occur, we will take the liberty to re-scale $\hat{\mathbf{X}}$ back to the area of interest. This is in accordance with how the LMT solution also will be scaled and shifted according to the true solution, before RMSE is calculated. The argument behind this is that since these models are intended to be used in an exploratory setting, the shape and dynamic of the inferred latent variable might still allow us to identify what the neurons are in fact responding to, thus also indicating how we might more properly scale or shift the inferred path.

Since the RMSE is a difference measure between inferred path and truth, it is also only available in settings with simulated data, when we know the true solution. In some real world cases, we can make an argument for when we have a recording of what we assume to be the true latent variable, but then again, is it really latent when we both know about it and have already recorded it? Nonetheless, in an exploratory setting, we do not have the luxury of evaluating the performance of our model in the same way as we do with simulated data. Myklebust (2020) explored whether the target function $\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X})$ from Equation 4.23, evaluated at the inferred $\hat{\mathbf{X}}$, could in fact be used as a measure of how accurate $\hat{\mathbf{X}}$ is. This is reminiscent of using the likelihood function to evaluate goodness of fit. However, their results indicate that the most optimal solution, with respect to RMSE evaluation, does not always have the highest posterior score, thus rendering this evaluation method ineffective.

We experience similar results when assessing whether $\Psi_{\mathbf{X}}^{\text{SPL}}(\mathbf{X})$ can be used more effectively than the LMT variant. Experiments show that $\Psi_{\mathbf{X}}^{\text{SPL}}(\mathbf{X})$ evaluated at the true solution can have a lower posterior score, compared to an inferred $\hat{\mathbf{X}}$ initialised from the true solution and with true tuning curves. This indicates that there are local maxima around the true solution that results in a better score, possibly due to variation in the spike data, which happens to give a better fit for a slightly noisier solution of the truth. Note that this is not only the case when initialised from the truth. Initialising the Spline-based LVM from ISOMAP, applied to a smoothed spiking matrix, results in an inferred $\hat{\mathbf{X}}$ that gives a higher posterior score than the true solution in about half of the cases. Although this is not a conclusive study, we speculate whether the posterior score could be an unsuitable metric for evaluating the performance of our model as well, similarly to what Myklebust (2020) discovered, when the RMSE is unavailable.

5.1.2 Interior knots

The number of knots in the knot vector, and thus also the number of control points, is an important choice when working with splines, along with the placement of the knot vector. One possibility is to consider them free parameters, estimating them using e.g. cross-validation, or through MCMC, like in for instance DiMatteo et al. (2001). In our case, however, we will rely on the somewhat simpler ”elbow” method, where one decides the value of a parameter based on the location of the ”elbow” of a curve (Thorndike, 1953), plotted against some measure. We consider the number of interior knots, and plot them against the RMSE of an interpolating spline which interpolates a Gaussian bump tuning curve. The results can be observed in Figure 5.1, where we can see that the accuracy quickly saturates when the number of knots increase.

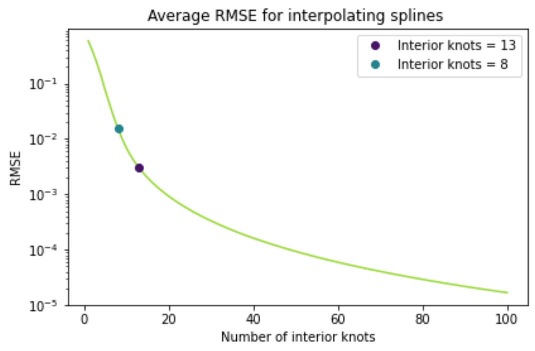

Figure 5.1: RMSE as a function of the number interior knots for an interpolating spline interpolating a Gaussian bump tuning curve. RMSE axis shown in log scale.

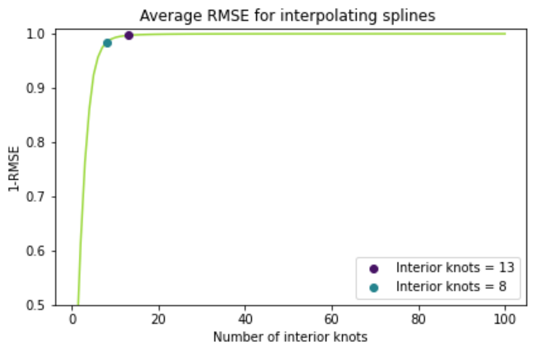

Figure 5.2: $(1 - \text{RMSE})$ as a function of the number interior knots for an interpolating spline interpolating a Gaussian bump tuning curve.

For easier inspection of the ”elbow”, we also plot $(1 - \text{RMSE})$ as a function of the number of knots, and forgo the log scale on the RMSE axis.

From Figure 5.2, there seems to be an observable elbow at eight knots, while at 13 there seems to considerable diminishing returns. Considering Figure 5.1 again, to reduce the average RMSE by another order of magnitude would require a doubling of the number of interior knots. We therefore decide to continue with eight interior knots, resulting in B-Splines with $N_{\text{cp}} = 12$ control points, although an argument could also be made for 13 interior knots. Note that when we work with a periodic latent variable, we increase the number of interior knots to 11, to accommodate for the fact that we now have three less unique control points.

5.1.3 Scaling, shifting and flipping

As mentioned by Myklebust (2020), there are various challenges associated with the usage of the LMT model. Most of them can be attributed to the target function $\Psi_{\mathbf{X}}^{\text{LMT}}(\mathbf{X})$ from Equation 4.23 which, upon closer inspection, is a non-convex and even function. This means that there is no guarantee that any optimization algorithm will converge to the global maxima, as it might get stuck in some local maxima, depending on how good the initialisation is. Do note that this is also a problem that one encounters in the spline-variant, so initialisation is equally important for this model.

The evenness of the target function means that any solution of the latent variable $\mathbf{X}$, which we may refer to as the "path", is just as likely to be found as $(-\mathbf{X})$, so the event of flipped solutions are a possibility. In addition, the covariance kernels are unable to detect the difference between $\mathbf{X}$ and some shifted solution $(\mathbf{X} + c)$, and although they only make up parts of the target function, experiments show that shifted solutions do occur, depending on the initialisation. Wrong scaling of the inferred $\mathbf{X}$ is another issue that presents itself during simulation, which in theory should prove to be a less optimal solution, compared to the true latent variable. We speculate however, that due to the non-convexity of the problem, this is but one of the issues that might end up presenting itself when doing iterative MAP estimation. In Figure 5.3 we see how initially the estimate is both shifted and scaled wrong, while once corrected, the estimate is quite accurate.

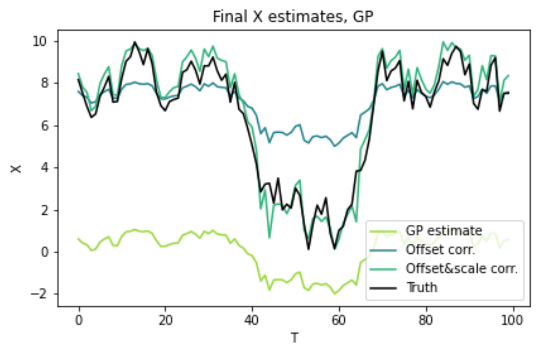

Figure 5.3: Showcase of how the LMT might produce solutions that are shifted and scaled wrong.

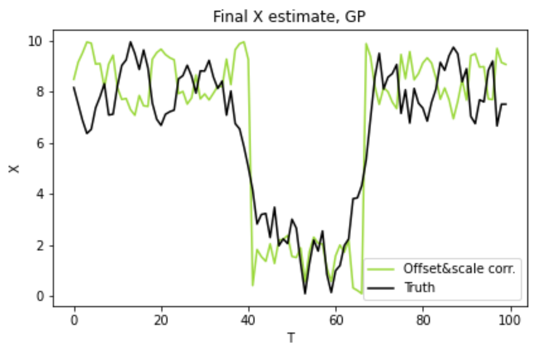

Figure 5.4: Showcase of how the LMT might produce flipped or even partially flipped solutions.

In Figure 5.4, we see the case where the estimate produced is a flipped variation of the true $\mathbf{X}$. More accurately, it is actually a partially flipped result, as in the interval from $T \approx 50$ to $T \approx 65$, the estimate is following the true trajectory correctly. While completely flipped results are still valuable, since they convey the behaviour of the latent variable in a way that is interpretable, partly flipped results are difficult to identify and correct for. Although the result from Figure 5.4 is almost completely flipped, if e.g. one-third of the inferred path was flipped, we would have no easy way of telling without consulting the underlying truth. While previous knowledge of the experimental setting or the inferred latent variable itself could be used in an exploratory setting to correct for offset and scaling issues, the challenge related to partial flipping is one without an immediate solution.

As for the Spline-based LVM, an immediate observation is that the change from Gaussian process to spline function means the target function (see Equation 4.37) is no longer even. Coupled with the removal of the isotropic covariance kernel in favour of a non-isotropic spline function (the splines are repeated along $\mathbb{R}$, but not mirrored), theoretically the chance of converging to a flipped estimate should now be a lot smaller, if even existent. However, even if early experiments show no sign of the Spline-based LVM producing flipped results, we cannot write off the possibility that flipped or partially flipped results might occur at some point, as we will see later, since the target function is still not convex. To give an indication of the different behaviour of the two methods, we initialise 6 estimates from random positions, then let the MAP procedures from Algorithm 1 and 2 run until convergence. The initialisations can be seen in Figure 5.5, while the resulting final estimates for the LMT model and the Spline LVM is shown in Figure 5.6 and 5.7, respectively. From Figure 5.6, we see that the final estimates when using the LMT model can end up flipped in a variety of ways, some completely, some partially and along different axes. This is not the case with the variant using splines, as more or less each simulation ends up converging towards the truth.

That does not mean the spline-variant is without troubles, however. Many of the final estimates in Figure 5.7 periodically make erratic jumps before returning to the trajectory of the true path. We speculate that this might partially be due to two reasons, both of whom are easier to envision when considering only one realisation, together with its initialisation. Such a case is shown in Figure 5.8.

The first thing that is immediately apparent is that the final estimate seems to have an unfortunate tendency of "sticking" to the initialisation, e.g. at around $T = 20$, $T = 60$ and $T = 85$. The other is that when initialised close to the borders of the interval which the latent path is defined on (in this case, $[0, 10]$), the inferred path might make a large jump that is reminiscent of a shift with length equal to the length of the interval that the path exists on. Given that we in Section 4.2.2 defined the tuning curves repeatedly along $\mathbb{R}$, it is perhaps not too surprising that the inferred path might be pulled towards shifted solutions. To combat this one could implement some form

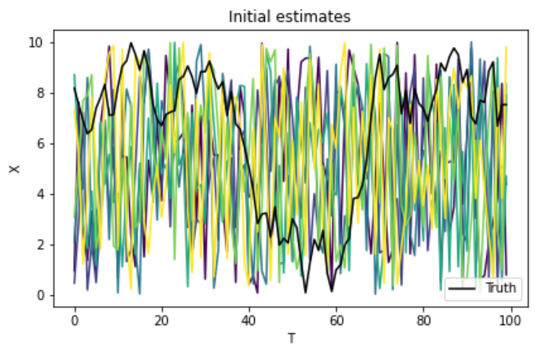

Figure 5.5: Six random initialisations, realised by drawing each $\mathbf{x}_t$ from a Unif$[0, 10]$ distribution, together with the true path.

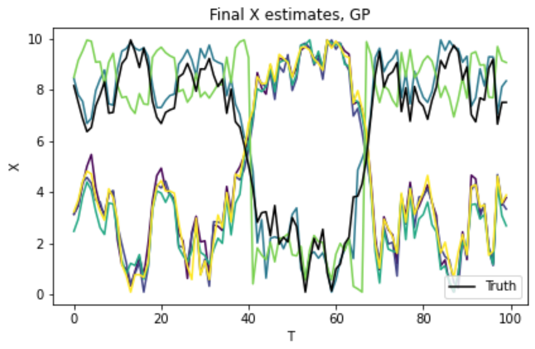

Figure 5.6: Final estimates of $\mathbf{X}$ using LMT, plotted against the true path.

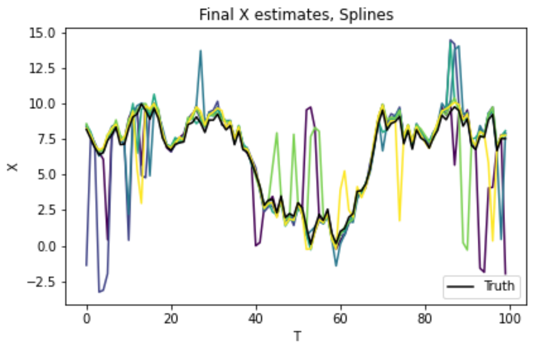

Figure 5.7: Final estimates of $\mathbf{X}$ using the Spline-based LVM, plotted against the true path.

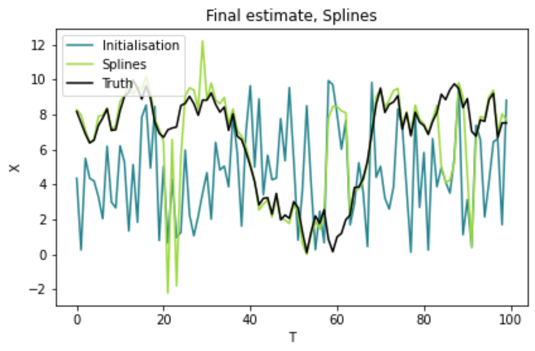

Figure 5.8: One random initialisation, and the corresponding final estimate of $\mathbf{X}$ using the Spline-based LVM, plotted against the true path.

of penalisation for making large jumps from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$, or utilise the constrained optimization implementation of the L-BFGS-B optimizer. However, early assessments indicate that results are on average worse when solving the problem as a constrained optimization problem, so it might simply be more useful to inspect the result from unconstrained optimization and then correct for obviously shifted jumps. Unfortunately, jumps may also occur at locations that are harder to explain, like at around $T = 30$, where the final estimate has converged towards the true path from an initialisation further away, but at one point makes a larger detour. We have no tangible explanation for why this is the case, other than that this particular point might belong to a local maxima for our non-convex problem. Couple this with the ”stickyness” with respect to some parts of the initialisation, and it becomes quite clear that a proper initialisation is a requisite for convergence to an optimal solution, as tends to be the case for non-convex optimization.

A note on the impact of the prior

Given that neural data sets can be quite large in size, either with respect to the recording length and/or the number of neurons recorded, there is always the possibility that when using a MAP procedure for estimation, the contribution from the likelihood might completely dominate that of the prior, resulting in a solution that approaches the MLE. On inspection, we observe that in the simulated cases we will be discussing in the upcoming sections, the contribution from the likelihood outweighs that of the prior by approximately one order of magnitude. Still, the prior may be useful to partially control the shape and behaviour of the final estimate by adjusting the hyperparameters. Consider now the σ_x and δ_x parameters in the covariance function in Equation 4.2. By reducing σ_x, we can control how much the inferred path varies, as well as strengthening the belief in the prior, since σ_x is inverse proportional to the contribution of the prior term in e.g. the MAP expression in Equation 4.37. This might help controlling the unwanted jumps in the final estimate shown in Figure 5.8. The effect of lower σ_x can be seen in Figures 5.9 and 5.10, which showcase two zoomed-in sections of a longer inferred latent path.

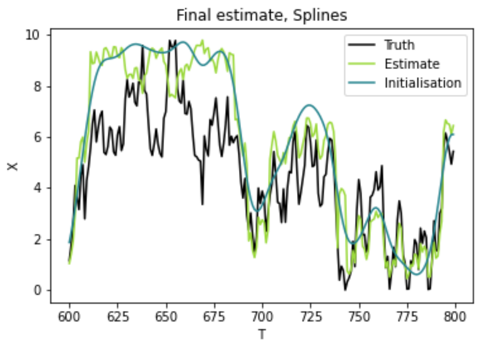

Figure 5.9: Section of final estimate of $\mathbf{X}$ using Spline-based LVM, plotted against true path. Initialisation based on first component of PCA applied to smoothed spike data.

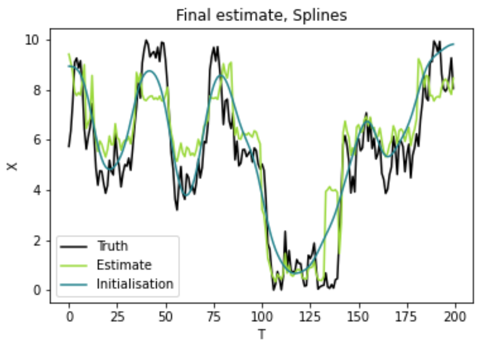

Figure 5.10: Another section of final estimate of $\mathbf{X}$ using Spline-based LVM, plotted against true path. Initialisation based on first component of PCA applied to smoothed spike data.

What's interesting here is that while there are no immediate signs of large shifted jumps, the stronger prior seems to have prompted a local maxima that now includes a partially flipped solution. Since the prior term is indeed even, it does make some sense that a stronger belief in it could provoke flipped solutions, even though the likelihood does not. This is akin to how the LMT model also struggles with local maxima including partial flipping, even though it is not entirely clear why by simply inspecting the target function.

We may also change the δ_x parameter, to control the smoothness of our inferred path. Although this does not influence the strength of our belief in the same way as changing σ_x does, it does allow us to pick up the general trend of the path, instead of trying to perfectly fit all the twists and turns, which might be a useful asset when doing exploratory analysis. A visualisation of how changing δ_x affects the result is shown in Figures 5.11 and 5.12.

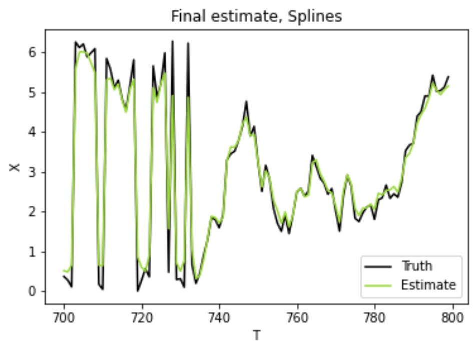

Figure 5.11: Section of final estimate of $\mathbf{X}$ using Spline-based LVM, with δ_x same as generative model. Initialised sufficiently close to the true path, for visualisation purposes. Periodic true path.

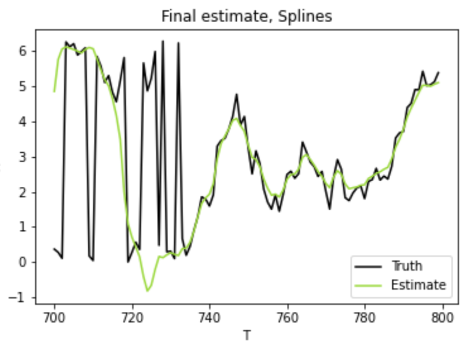

Figure 5.12: Section of final estimate of $\mathbf{X}$ using Spline-based LVM, with δ_x larger than generative model. Initialised sufficiently close to the true path, for visualisation purposes. Periodic true path.

As expected when making δ_x larger than that of the generative model, the inferred $\mathbf{X}$ now picks up on the general trend, but forgoes the large jumps that accompanies a periodic latent variable.

5.1.4 The choice of initialisation

As we have mentioned multiple times, initialisation is a crucial part when solving non-convex optimization problems. A poor initialisation means the solver might converge to a sub-optimal solution, or worse, something completely wrong. Both for the LMT model and the Spline-based LVM, there are two variables that need initialisation(assuming the hyperparameters are set): the latent variable $\mathbf{X}$ and the variable related to the tuning curves, be that $\mathbf{F}$ or $\boldsymbol{\beta}$.

Since the two models differ in how they model the tuning curves, they also require different initialisations for those variables. Myklebust (2020) explored different possibilities for the initialisation of $\mathbf{F}$, achieving success with what iss known as the square root transform (whose usefulness has also been discussed by Yu et al., 2009)

$$\mathbf{F}^0 = \sqrt{\mathbf{Y}} - \frac{\max\sqrt{\mathbf{Y}}}{2}, \tag{5.3}$$

which transforms Poisson count data into something approximately Gaussian, proven to be very valuable when using Gaussian processes in a setting with Poisson data. The existence of such a transform is an advantage to using the LMT model, as one is able to exploit the available data to make an informed initial guess for the tuning curves. As for the Spline-based LVM, to our knowledge, there exits no similarly useful transformation for initialising $\boldsymbol{\beta}$, which also incorporates the spike data. One possibility is initialing $\boldsymbol{\beta}$ from a flat prior, then performing a single MAP step by maximizing Equation 4.32, conditioned on the initial $\mathbf{X}$ of choice.

$$\boldsymbol{\beta}^0 = \text{argmax}_{\boldsymbol{\beta}} \Psi_{\boldsymbol{\beta}}^{\text{SPL}}(\boldsymbol{\beta}) | \mathbf{X}^0. \tag{5.4}$$

This initialisation is not ideal, as it is heavily dependent on a good initialisation for $\mathbf{X}$, and possibly couples the $\boldsymbol{\beta}$ and $\mathbf{X}$ too tightly together. However, experiments show that this initialisation is still better than simply assigning random uniform values from some interval to the elements of $\boldsymbol{\beta}$, hence we proceed with this choice of initialisation.

Consequently, the choice of initialisation for $\mathbf{X}$ becomes all the more important. Myklebust (2020) explored various contenders, but found no indication that one initialisation proved to be much better than the others. Although a flat initialisation was possible when using LMT, possibly due to the strong initialisation of $\mathbf{F}$, such an initialisation proves to be less fruitful for the Spline-based LVM, due to the initialisation of $\boldsymbol{\beta}$. We are therefore dependent on an $\mathbf{X}^0$ that captures the behaviour of the latent variable more efficiently.

One contender is to use a PCA initialisation, that is, first applying a Gaussian filter to the spike data $\mathbf{Y}$, then applying PCA to the smoothed spikes and extracting the first principal component, before re-scaling the estimate to match the domain of the latent variable. Two such initialisations, based on a smoothed filter with width 5, are shown in Figures 5.13 and 5.14, for 20 and 100 neurons respectively.

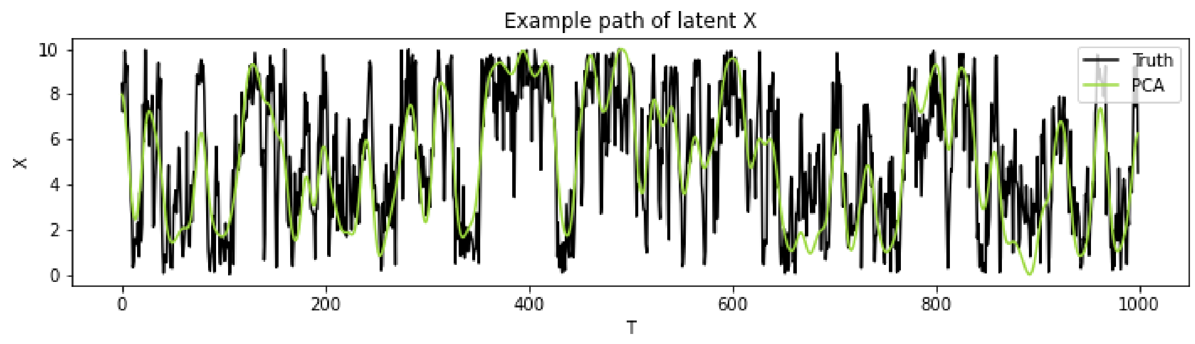

Figure 5.13: First principal component plotted against true path. PCA applied to smoothed simulated data with 20 neurons, $\sigma_x = 40, \delta_x = 16$.

We see here that the general trend is captured pretty well by PCA, irrespective of whether we have 20 or 100 recorded neurons, leading us to believe this initialisation is sufficient also when it comes to basing the initial $\boldsymbol{\beta}^0$ on it.

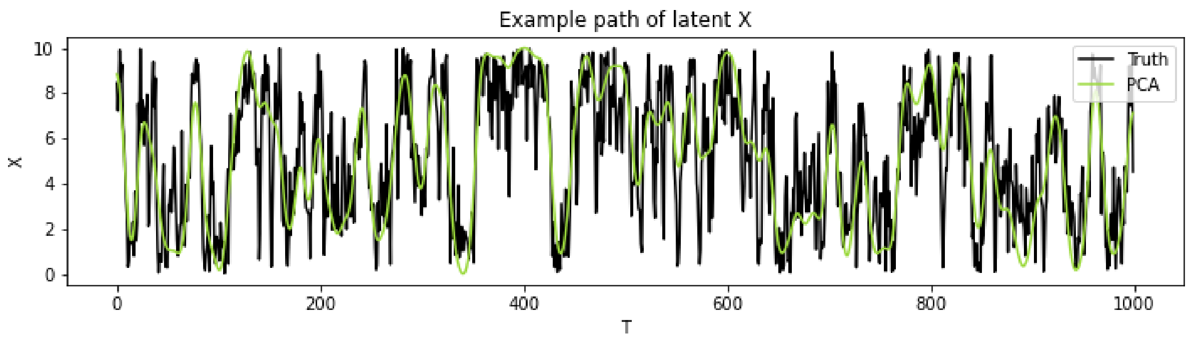

Figure 5.14: First principal component plotted against true path. PCA applied to smoothed simulated data with 100 neurons, $\sigma_x = 40, \delta_x = 16$.

Another possibility is to utilise a more advanced dimensionality reduction technique, like ISOMAP. In this simple simulated case, we can apply ISOMAP directly on the spike matrix $\mathbf{Y}$, re-scaling the results afterwards. The results from applying ISOMAP can be seen in Figures 5.15 and 5.16, with a similar number of neurons as in the PCA case.

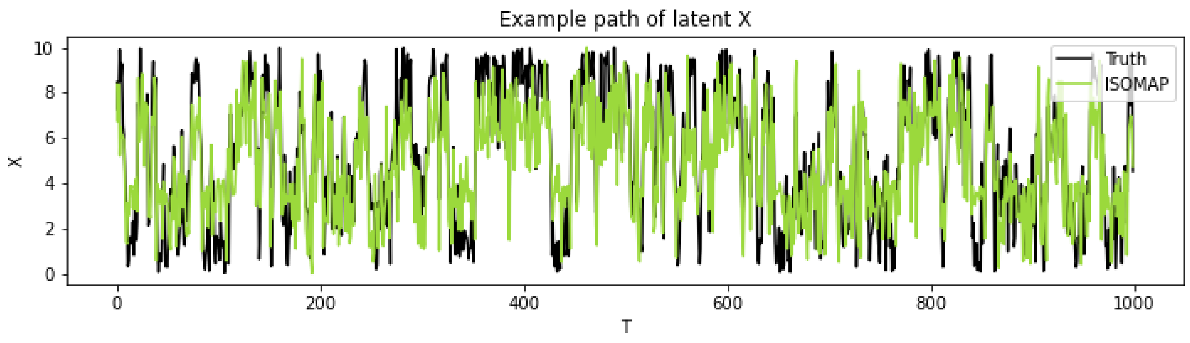

Figure 5.15: ISOMAP solution plotted against true path. ISOMAP applied to smoothed simulated data with 20 neurons, $\sigma_x = 40, \delta_x = 16$.

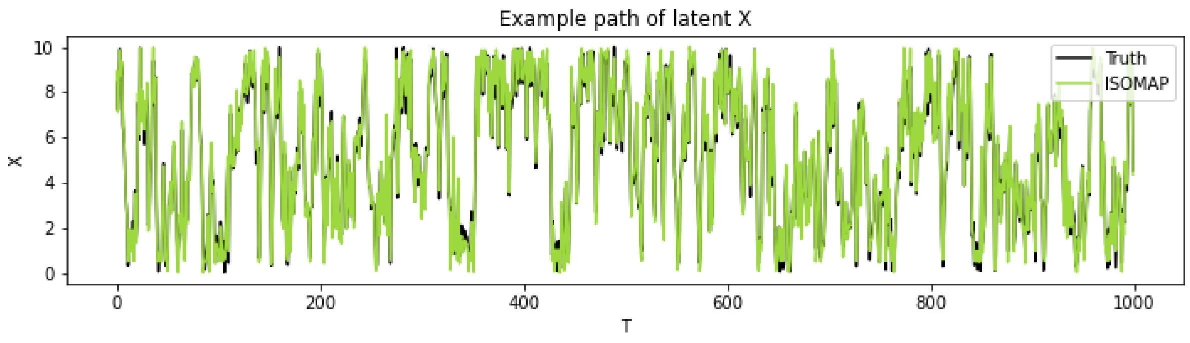

Figure 5.16: ISOMAP solution plotted against true path. ISOMAP applied to smoothed simulated data with 100 neurons, $\sigma_x = 40, \delta_x = 16$.

Here we observe that ISOMAP is noticeably better, in fact almost perfectly capturing the trajectory of the latent variable once given enough neural data. In such a simple case as this non-periodic simulated one, ISOMAP alone would probably suffice as prediction for the latent path. However, ISOMAP is not a probabilistic model and does not let us extract information about the tuning curves, so there is still merit to using it as an initialisation for our models in an effort to further improve our results. Also, in the case of real data, there may be other factors that influence the neurons, so a model with proper hierarchical setup might be preferable to incorporate as much knowledge as we can.

The case for feature sharing

As we have just discussed, LMT has an apparent advantage when it comes to the initialisation related to the tuning curve estimation. Luckily, splines are highly customisable, and using feature sharing when applicable allows us to overcome (among other things) the issues related to the initialisation of $\boldsymbol{\beta}$. In the upcoming sections, where feature sharing is applicable, we will consider the case where the latent variable is a 2π-periodic variable, with tuning curves exhibiting a Gaussian bump structure. Using this as a guide for initialisation, we fit an interpolating spline to a shifted cosine function (see Figure 5.17), then use the accompanying coefficients as the initialisation for $\boldsymbol{\beta}$.

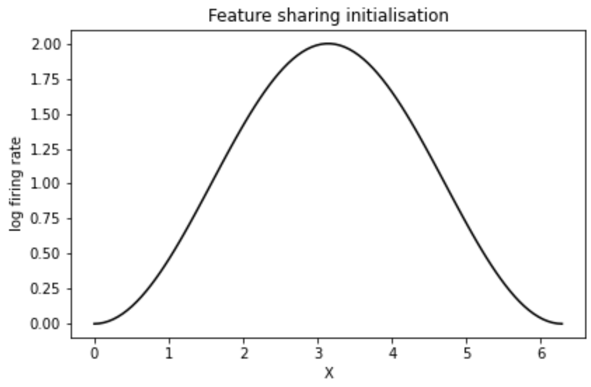

Figure 5.17: Cosine bump initialisation for Spline coefficients, when the latent variable is 2π-periodic.

This wide shape of the initialisation should allow the iterative estimation to relax towards the correct tuning width, and is the reasoning behind picking a cosine shape. Compared to a narrow Gaussian bump, which would be constant along much of the $[0, 2\pi]$ interval, we hypothesize that such an initialisation might be more easily trapped in local maxima due to the abundance of bins with equal firing rate.

As for the initialisation of $\boldsymbol{\alpha}$, $\boldsymbol{\theta}$ and $\boldsymbol{\gamma}$, one could use the observed firing rate of the neurons to make a more informed initial decision of the tuning strength. We will however be content with drawing initialisations based on the priors described in Section 4.6.

5.2 Simulated data

5.2.1 1D non-periodic case

We first consider the case where our simulated variable is a one-dimensional non-periodic latent variable. That is, $x_t \in [x_{\text{min}}, x_{\text{max}}], \forall t$, i.e. $\mathbf{x}$ is confined to some specified interval on $\mathbb{R}^1$. Tuning curves will be defined as Gaussian bumps, with peak firing rates randomly distributed along the interval that $\mathbf{x}$ exists on. The neuron's will also be given a background firing rate, meaning that as $\mathbf{x}$ moves further away from a neurons peak firing rate, its firing rate will approach some intensity $\neq 0$, to avoid the case where some neurons become completely inactive.

Such a situation with a one-dimensional latent variable can be related to a rat running in a corridor that is narrow enough to only permit movement along its longitudinal axis, except for when the rat turns (see e.g. Gothard et al., 1996, for a similar setup). The latent variable then represents the rats position, while the neurons have been assigned a peak firing rate at some random location in the corridor, for which we say the neuron is tuned to. Far enough away from its ”preferred location”, the neurons still fire with intensity equal to the background firing rate.

For this particular case, we pick $x_{\text{min}} = 0$, $x_{\text{max}} = 10$, with a background firing rate of 0.5 spikes per bin, while peak firing rate is set to 4 spikes per bin. The latent variable is then generated according to the prior model defined in Equation 4.3, with a small distinction: as the generative prior in this case is defined by a multivariate normal distribution, there is no guarantee that the path will not traverse outside of our specified boundaries. To ensure that the path is contained within our bounds, we ”flip” the path over whenever it would cross the boundaries x_{min}, x_{max}, which also slightly changes the distribution of $\mathbf{x}$, as it is no longer exclusively sampled from a multivariate normal distribution.

However, the basis for the generation of $\mathbf{x}$ is still the prior model, and thus requires us to specify the two hyperparameters σ_x and δ_x in Equation 4.2. As it is desirable for the path to cover the whole domain $[x_{\text{min}}, x_{\text{max}}]$, in order to make sure we capture distinct information from all the neurons, we set the variance parameter σ_x to a sufficiently high value so that the path is able to traverse the whole domain, e.g. $\sigma_x = 40$. A realisation of a possible path is shown in Figure 5.18. A corresponding set of tuning curves for 20 neurons, defined as Gaussian bumps, is presented in Figure 5.19.

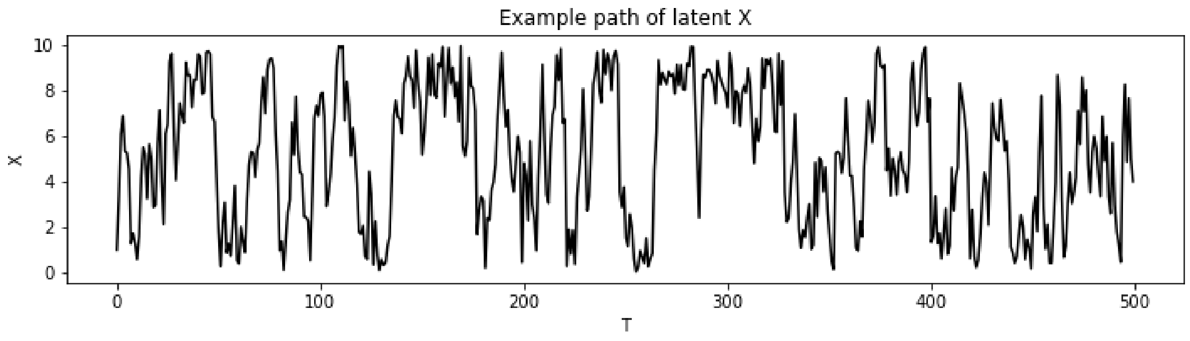

Figure 5.18: Example of path generated by the prior model, ensured to stay within boundaries. $\sigma_x = 40, \delta_x = 16$.

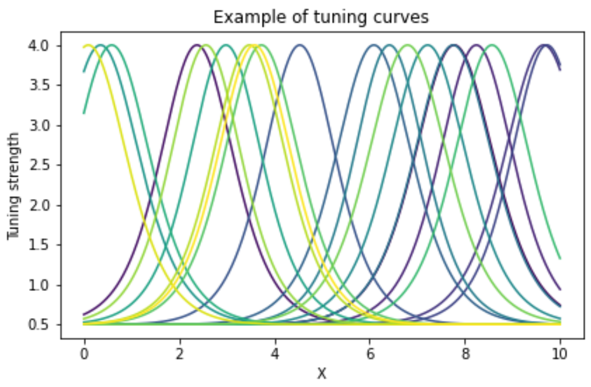

Figure 5.19: Selection of 20 tuning curves, defined as Gaussian bumps at random locations, with peak firing rate of 4 and background firing rate of 0.5 spikes per bin.

The hyperparameters for our two models have been set by trial and error, while keeping the parameters in the $\mathbf{X}$ prior the same as those assigned to the generative model. In the LMT case, we arrived at $\sigma_f = 2, \delta_f = 0.83$. For the version with splines, as discussed earlier, we chose 8 interior knots for our knot vector, padding the augmented knot vector $\boldsymbol{\tau}$ with repeats of the first and last knot, as is customary when working with non-periodic splines. We also set all $\sigma^2_{\beta,[i,n]} = 10^2$.

We begin by investigating how well the models perform with respect to scaling in the number of neurons N and recording length T. This is done by creating an array of different values for N and T

$$\begin{aligned} N &= [10, 15, 20, 25, 30, 35, 40, 45, 50, 75, 100, 150, 200, 250], \\ T &= [10, 15, 20, 25, 50, 75, 100, 200, 300, 400, 500, 750, 1000, 5000], \end{aligned} \tag{5.5}$$

then (considering scaling with respect to N first) for each element in the N-arrray, we generate 20 different instances of the latent variable $\mathbf{X}$, as well as a corresponding data set of neural activity $\mathbf{Y}$ based on the bump tuning curves, while fixing $T = 1000$. We run our models until convergence on these data sets, then compare $\hat{\mathbf{X}}$ with the true $\mathbf{X}$ by calculating the RMSE, reporting the average RMSE across those 20 cases. Correspondingly, we do the exact same for the T-array, this time fixing $N = 100$.

$\mathbf{X}$ will be initialised using the PCA initialisation, applied to a spike matrix that has been smoothed with a Gaussian filter using a width of five. We also include the results when initialising the Spline-based LVM from the final result of the LMT model, to investigate whether they share similar local maxima, or if it is possible to improve upon the LMT solution. The PCA initialisation will also be included, for reference. Results can be seen in Figure 5.20 and 5.21, where the LMT model has been dubbed "GP" and the Spline-based LVM "Splines".

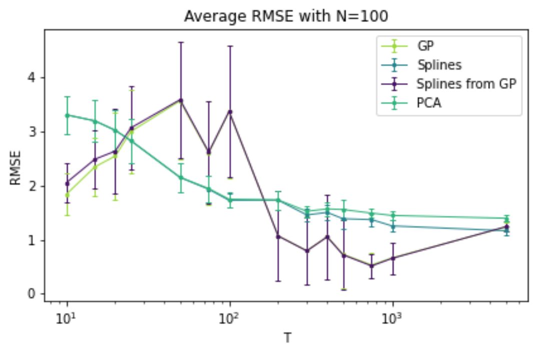

Figure 5.20: Average RMSE over 20 runs, as a function of T, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 40, \delta_x = 100$. Domain of $\mathbf{X}$: $[0, 10]$.

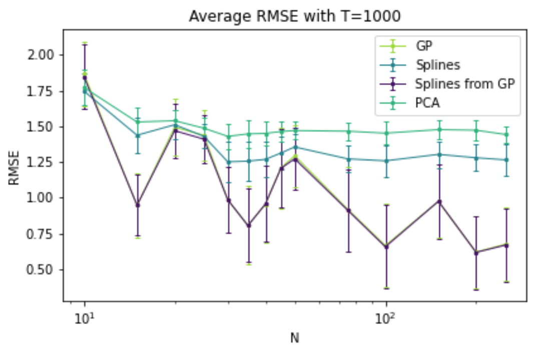

Figure 5.21: Average RMSE over 20 runs, as a function of N, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 40, \delta_x = 100$. Domain of $\mathbf{X}$: $[0, 10]$.

Immediately one notices, in Figure 5.20, the dip in the RMSE for the LMT results, before it slowly starts making its way up again. Although one could expect that more data (larger T) would lead to better results, this is not the case here. One explanation, also mentioned by Myklebust (2020) in the context of optimal tuning strength, is that due to the existence of multiple local maxima for the LMT solution, a larger T (i.e. more time bins) means more possibilities for such maxima to occur. This results in the model performing better in a "sweet-spot" where T is large enough for there to be sufficient data, but not so large that the occurence of local maxima solutions become prominent. This quirk is not present in the Splines-based LVM results, but the "Splines" also improves much less on the PCA initialisation, compared to the "GP". We note some odd behaviour for lower values of T, but with so few time bins, a situation that would hardly occur when applying the models to real data, the results are more a curiosity than an indication of how well the models perform. We also see that when initialised from the "GP" solution, the "Splines" stay in place, indicating the the solution is also a maximum for the "Splines" (due to this fact, we observe that the "GP"-line is mostly obscured by that of "Splines from GP").

As for model performance with respect to scaling in N, the "Splines" solution now seems to improve slightly more on the PCA initialisation, but that they both saturate pretty fast with respect to the number of neurons. The "GP", on the other hand, behaves in a more desirable way, improving further when increasing N. We do see that the RMSE from the "GP" model exhibits some sort of wavy tendency, still with the desirable downward trend. Although the confidence intervals are quite big, and cover almost the whole wavy pattern, we speculate that this behaviour maybe due to the random placement of the peak firing rate for our neurons. In the event that some of the tuning curves might be overlapping quite heavily, there will be less activity from the neurons in some particular areas with poor coverage. If the path spends a good amount of time in said areas, the resulting $\hat{\mathbf{X}}$ might be slightly distorted, and thus result in a worse RMSE on average. We can also not completely exclude that the wavy behaviour might simply be due to chance.

We note, however, that these results are all based on a fixed set of hyperparameters, and as such, might not be representative for all possible modeling situations. This is, of course, unavoidable, as some choices regarding the modeling must be made in order to produce results. In an effort to accommodate for this fact, we run the simulation again, this time considering $\delta_x = 16$, which results in a true path where previous values of $\mathbf{x}_t$ have much less influence on upcoming ones. Scaling of RMSE with respect to T and N is shown in Figures 5.22 and 5.23.

These results are in quite the stark contrast to those seen in Figures 5.20 and 5.21. Although Figure 5.22 showcases the same "sweet-spot" for values of T with respect to the "GP" performance, the "Splines" now improve much more on the PCA initialisation, performing better than the "GP" as T becomes larger.

Even more noticeable is the improvement with respect to scaling in N, as the "GP" is now the model that saturates quickly, while the "Splines" continue improving with increasing N. This only goes to show that changing the hyperparameters, and also the truth of the problem we are considering, can have a tremendous effect on the results, as the behaviour of the two models have entirely switched. Note that when the Spline-based LVM is initialised from the LMT solution, it actually performs worse, compared to if it is initialised from PCA (which in and of itself has a worse RMSE than the LMT). This indicates that the local maxima of the LMT model are strong

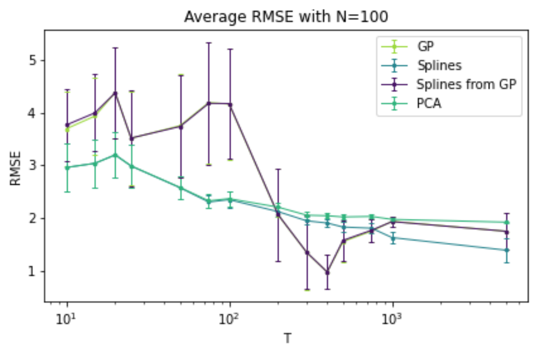

Figure 5.22: Average RMSE over 20 runs, as a function of T, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 40, \delta_x = 16$. Domain of $\mathbf{X}$: $[0, 10]$.

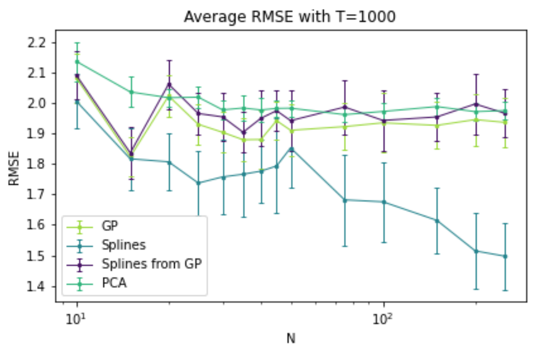

Figure 5.23: Average RMSE over 20 runs, as a function of N, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 40, \delta_x = 16$. Domain of $\mathbf{X}$: $[0, 10]$.

enough to keep the Spline-based LVM from escaping and finding a more optimal solution. Parallels can be drawn to Myklebust (2020), who showed that there is no fixed "optimal" initialisation for LMT, and sometimes even a flat initialisation can give better results than when initialised from PCA. Similarly, the "better" initialisation that is the LMT solution here proves to be more like a local maxima trap.

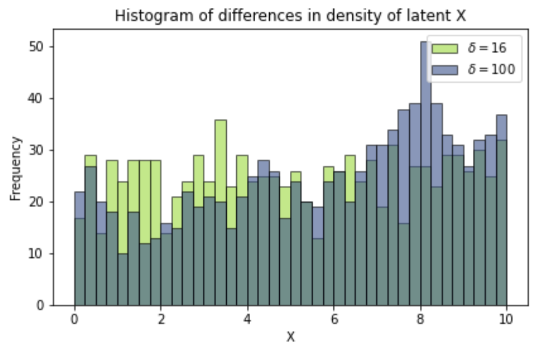

Figure 5.24: Histogram showing difference in amount of time spent in various bins of the latent variable domain $[0, 10]$. Simulation number: 1.

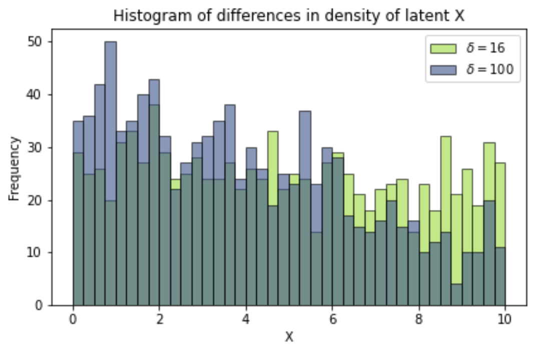

Figure 5.25: Histogram showing difference in amount of time spent in various bins of the latent variable domain $[0, 10]$. Simulation number: 2.

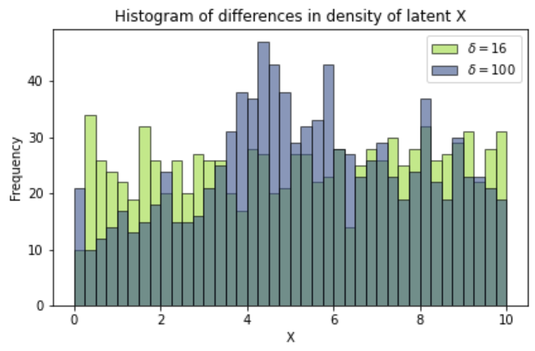

Figure 5.26: Histogram showing difference in amount of time spent in various bins of the latent variable domain $[0, 10]$. Simulation number: 3.

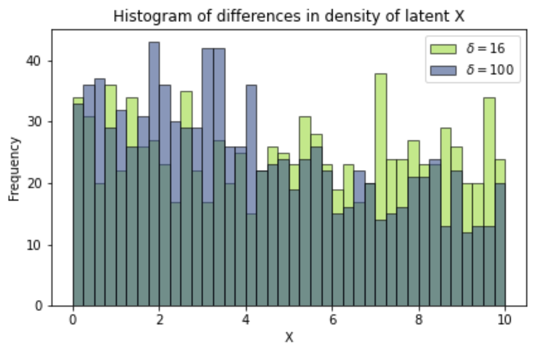

Figure 5.27: Histogram showing difference in amount of time spent in various bins of the latent variable domain $[0, 10]$. Simulation number: 4.

We also observe that the RMSE is in general lower when the truth is generated from a prior with smoothness parameter $\delta_x = 100$, compared to $\delta_x = 16$. To investigate this further, we simulate paths from the generative prior with both $\delta_x = 100$ and $\delta_x = 16$, discretise the domain $[0, 10]$ into 40 evenly sized bins, then count the number of times the path visit each bin. The results are presented as histograms in Figures 5.24 - 5.27, showing four different simulations.

From these four plots it is clear that the path simulated with $\delta_x = 16$ generally has a more even distribution of visits compared to the path with $\delta_x = 100$, which shows more pronounced peaks and valleys along various parts of the interval. Considering how the smoothness parameter influences the behaviour of the simulated latent variable, and inspecting one possible set of the two different instances of the latent trajectory visualised in Figures 5.28 and 5.29, it is perhaps not entirely surprising that the histograms take the shape that they do.

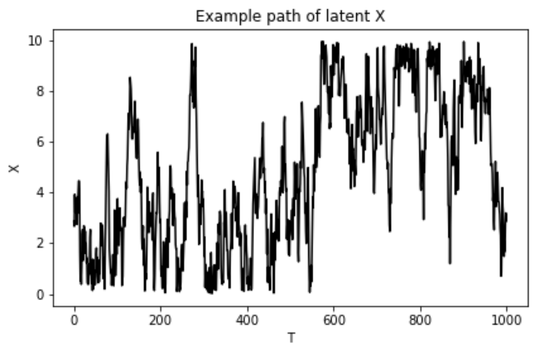

Figure 5.28: Example path generated by the prior model, hyperparamers: $\sigma_x = 40, \delta_x = 100$.

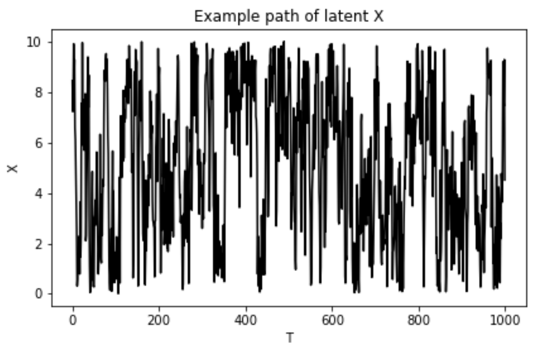

Figure 5.29: Example path generated by the prior model, hyperparamers: $\sigma_x = 40, \delta_x = 16$.

A smaller δ_x means the latent variable can vary more freely along the domain, being less governed by its previous positions. This, as we see in Figure 5.29, results in larger jumps and more efficient covering of the whole domain, which is reflected in the more uniform histograms. On the other hand, the path in Figure 5.28 explores the domain slower, staying longer in subareas of the whole domain, while occasionally making larger detours. Thus we get the more pronounced peaks in the corresponding histograms, where the path has more or less "settled in". This explains the differences in RMSE between the two choices of δ_x, as a path that varies wildly with sharp turns and more erratic behaviour is in general more difficult to accurately infer. There is also the fact that the tuning curve prior for the LMT model is based on the behaviour of $\mathbf{X}$ (it is a spatial prior based on the position of $\mathbf{X}$, not the domain of $\mathbf{X}$), thus rendering it less accurate when the path is harder to infer. Meanwhile, the Spline-based LVM does not assume anything about the particular amount of time the true path spends in each bin. In that sense, we can say there is an underlying uniform assumption, which matches better with the more uniform histogram produced by the path with hyperparameter $\delta_x = 16$. This leads to the Spline-based LVM performing better than the LMT model is able to do, although still not as good as when the path is generated with $\delta_x = 100$.

Another point of interest is how the two models scale with respect to algorithm run time. We time the algorithm, excluding the parts shared by both algorithms, like during the initialisation, and plot how the run time scale with respect to T and N. The results can be seen in Figures 5.30 and 5.31.

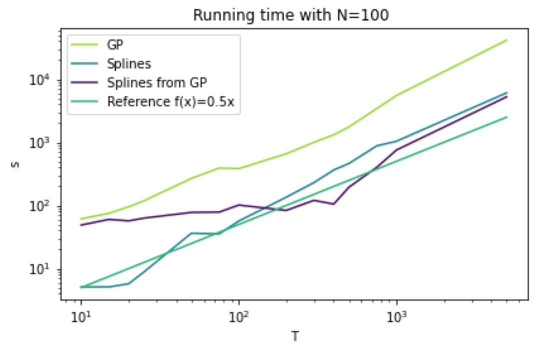

Figure 5.30: Scaling of algorithm run time, as a function of T. GP and Splines initialised from PCA.

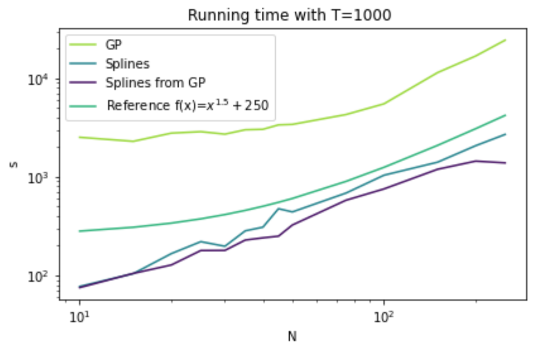

Figure 5.31: Scaling of algorithm run time, as a function of N. GP and Splines initialised from PCA.

Here we see one of the evident advantages to using the Spline-based LVM. Scaling in T seems to be linear for both models, while in N it seems to be somewhere in between linear and quadratic. That is, for the LMT model, as the Spline-based LVM seems to perhaps more closely resemble linear scaling instead of to the power of 1.5. There is however an order of magnitude offset between the two models (more than one order, in fact, when considering scaling with respect to N), which quite clearly puts limitations on how efficient LMT can be with respect to large data sets. We also note that for these simulations, we considered the hyperparameters fixed, which ideally should be found by estimation as well (either by optimization, or using cross-validation in a supervised setting). This would further complicate the problem, and consequently also increase the run time.

It is perhaps more interesting and fruitful to discuss the possibilities that accompany the lower run time of the Spline-based LVM. With a faster algorithm, we are able to further reduce the bin size of our problem (resulting in larger T) to the point where we are able to assume a Bernoulli spiking model instead of Poisson. This is advantageous because a Bernoulli random variable has finite support, while a Poisson does not, making the latter harder to extensively sample. And while the models may in fact be considered equal, given small enough time bins, computational limitations makes it difficult to achieve close to infinite temporal precision. There is also the fact that some problems are simply more difficult to solve when using a Poisson spiking model. As pointed out by Davidovich et al. (2020), when reconstructing the hidden node problem, the inference of neuronal couplings is much less accurate when assuming a Poisson spiking model. Thus if we can get away with using a Bernoulli spiking model with the Spline-based LVM, something that would be less feasible with the LMT model, other findings would indicate that the results might improve even further.

Having observed the effect that the different values of δ_x can have on the models' performance, we wish to investigate whether another initialisation can also have a similar effect. Earlier we discussed various possibilities for initialising $\mathbf{X}$, where we saw that ISOMAP seemed to be much more effective at capturing the trends of the latent variable, compared to PCA. We therefore perform similar experiments as those already reported, that is scaling with respect T and N based on the average RMSE for a set of different T and N's, only this time initialised from ISOMAP (which does not require the use of a Gaussian filter, as ISOMAP can be applied directly on the spike matrix $\mathbf{Y}$). We will consider both the case where the generative prior has hyperparameter $\delta_x = 100$, as well as $\delta_x = 16$. Before reporting the results, however, we make an adjustment to the augmented knot vector τ. Having investigated the inferred tuning curves based on $\hat{\boldsymbol{\beta}}^{\text{SPL}}_{\text{MAP}}$, we observed that the results were less accurate at the edges of the domain of the latent variable, possibly due to the path spending less time at the borders (it is "reflected" at the border, after all) or perhaps due to the non-continuity of the spline at the end points. To achieve a more accurate estimate of the tuning curve along the borders, we propose to extend the interval the knot vector $\mathbf{t}$ is defined over. Instead of picking knot end points equal to x_{min} and x_{max}, we set them to be $x_{\text{min}} - \delta_\tau$ and $x_{\text{max}} + \delta_\tau$, where δ_τ is the length between two interior knots. This way, the discontinuous end property is forced slightly outside of the interval of interest, possibly resulting in a better estimate. We also investigate whether the single gradient-step discussed in Section 4.5 improves the final result. We compare the average RMSE for a selection of T, N-values, for the different configurations of $\mathbf{t}$ and gradient steps.The results are shown in Table 5.1.

Average RMSE	1 step, $\mathbf{t}$	1 step, ext. $\mathbf{t}$	to conv, $\mathbf{t}$	to conv, ext. $\mathbf{t}$
$N = 100, T = 5000$	0.2422	0.2334	0.2453	0.2425
$N = 100, T = 1000$	0.3180	0.3026	0.3328	0.3187
$N = 200, T = 1000$	0.2689	0.2533	0.2683	0.2568

Table 5.1: Average RMSE, with four digit precision, for three different sets of T, N-values. The different configurations for the estimation are: using one gradient step with $0, 10$ as endpoints for $\mathbf{t}$, using one gradient step with $0 - \delta_\tau, 10 + \delta_\tau$ as endpoints for $\mathbf{t}$, running the optimizer for $\boldsymbol{\beta}$ until full convergence with $0, 10$ as endpoints for $\mathbf{t}$, and running the optimizer for $\boldsymbol{\beta}$ until full convergence with $0 - \delta_\tau, 10 + \delta_\tau$ as endpoints for $\mathbf{t}$.

We see that the difference in average RMSE for the distinct configurations are quite small (possibly due to the impressive accuracy of ISOMAP, as we will see a bit later). Nonetheless, using one gradient step together with a knot vector interval that extends over the domain of $\mathbf{X}$ results in a consistent lower RMSE, compared to the other configurations. We speculate that the difference could be even greater if either one uses a less accurate initialisation than ISOMAP, or if the problem is more complex (as is often the case with real data). We therefore stick with the one gradient step, extended knot vector configuration as default when utilising the Spline-based LVM further.

Having investigated these variations of the MAP estimation procedure, we turn our attention back to the scaling in T and N, using ISOMAP as our initialisation. Scaling plots can be seen in Figures 5.32 - 5.35.

From these figures, it is clear that the biggest change brought by using ISOMAP as an initialisation is the closing of the performance gap between the two other models, in no doubt due to ISOMAPs efficient decoding

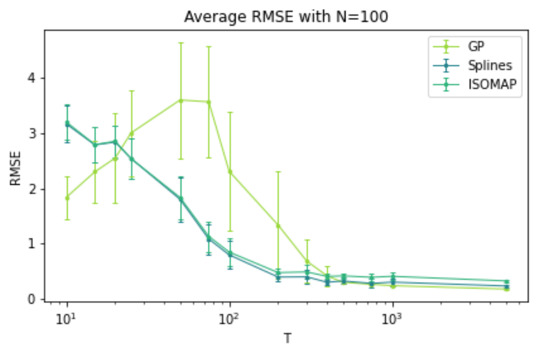

Figure 5.32: Average RMSE over 20 runs, as a function of T, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 40, \delta_x = 100$. Domain of $\mathbf{X}$: $[0, 10]$.

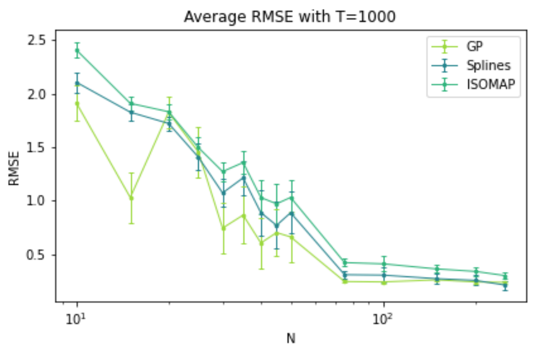

Figure 5.33: Average RMSE over 20 runs, as a function of N, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 40, \delta_x = 100$. Domain of $\mathbf{X}$: $[0, 10]$.

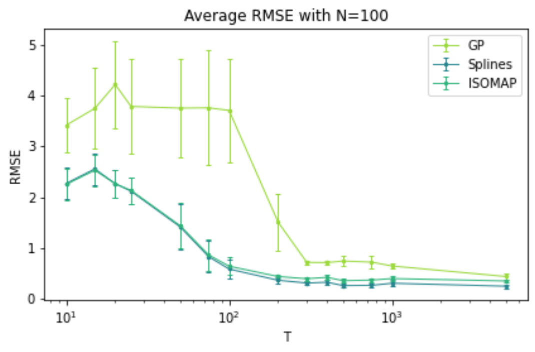

Figure 5.34: Average RMSE over 20 runs, as a function of T, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 40, \delta_x = 16$. Domain of $\mathbf{X}$: $[0, 10]$.

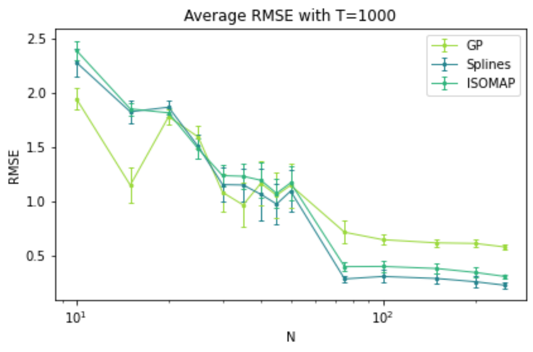

Figure 5.35: Average RMSE over 20 runs, as a function of N, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 40, \delta_x = 16$. Domain of $\mathbf{X}$: $[0, 10]$.

of the neural data. In Figures 5.32 and 5.33, as both T and N becomes sufficiently large, the two models seem to be doing equally well, slightly improving upon the ISOMAP initialisation. We also see the effects of changing δ_x in the prior, although not on such a large scale as when we used the PCA initialisation. With $\delta_x = 100$, we observe that "GP" improves slightly more upon the ISOMAP initialisation than the "Splines" do (Figure 5.33), while for $\delta_x = 16$, "GP" actually does worse than ISOMAP itself. Having touched upon this in Subsection 5.1.1, we speculate that this is due to some local maxima (a slightly noisier fit to the true solution) proving to be stronger than the actual solution when doing MAP estimation, thus pulling the algorithm towards the worse and more noisy solution. This results in the "GP" solution also saturating faster and at a higher RMSE, similarly to what can be seen in Figure 5.23. All in all, this provides an indication that the behaviour related to the change in δ_x is consistent across different initialisations.

For good measure, we also report the algorithm run time when initialised from ISOMAP, which more or less follow the same trends as the run time scaling when initialised from PCA, unsurprisingly. Run time plots are shown in Figures 5.36 and 5.37.

5.2.2 1D periodic case

Having considered the case where we assume non-periodicity for our latent variable, we move over to the more complicated issue that is inferring a periodic latent variable. The setup is somewhat similar to the non-periodic one, in that we consider a one-dimensional latent variable, $x_t \in [x_{\text{min}}, x_{\text{max}}], \forall t$, but where the boundaries simply wrap around, instead of acting as barriers like in the non-periodic case. Tuning curves are similarly defined as Gaussian bumps, with peak firing rates and a background firing rate.

By picking values $x_{\text{min}} = 0, x_{\text{max}} = 2\pi$, we construct a situation similar to the one we have with the data from

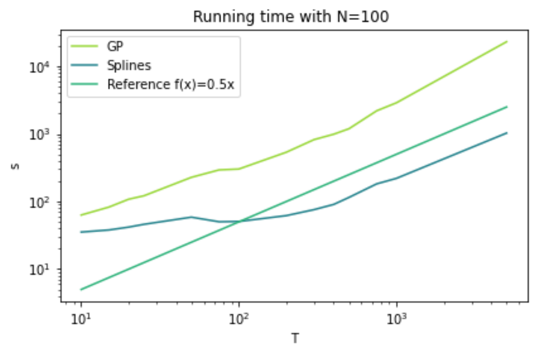

Figure 5.36: Scaling of algorithm run time, as a function of T. GP and Splines initialised from ISOMAP.

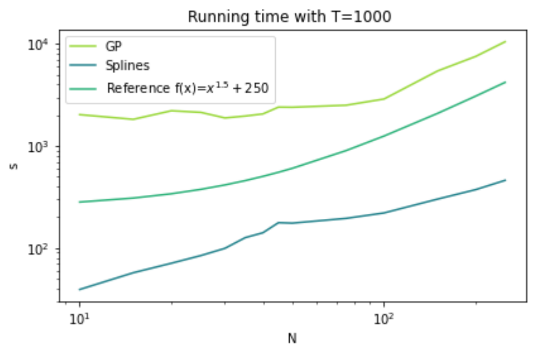

Figure 5.37: Scaling of algorithm run time, as a function of N. GP and Splines initialised from ISOMAP.

Peyrache et al. (2015); the latent variable represents the head direction of some animal, e.g. a mouse, doing some particular action while having the head direction tracked. Neurons have a peak firing rate assigned to some specific head direction, and far enough away resort to the background firing rate.

In our simulation, we keep the peak firing rate and background firing rate the same as in the non-periodic case, that is 4 spikes per bin and 0.5 spikes per bin. The path is also generated by the same prior model (the multivariate normal distribution), although in this case with a periodic covariance function. Due to the paths periodic nature, we have no need to impose a ”flip” at the boundaries to keep the latent variable between its boundaries. Instead, we shift its values to be contained inside the interval by taking the modulo in the end. With hyperparameters $\sigma_x = 5$, $\delta_x = 50$, we generate a possible realisation and present it in Figure 5.38.

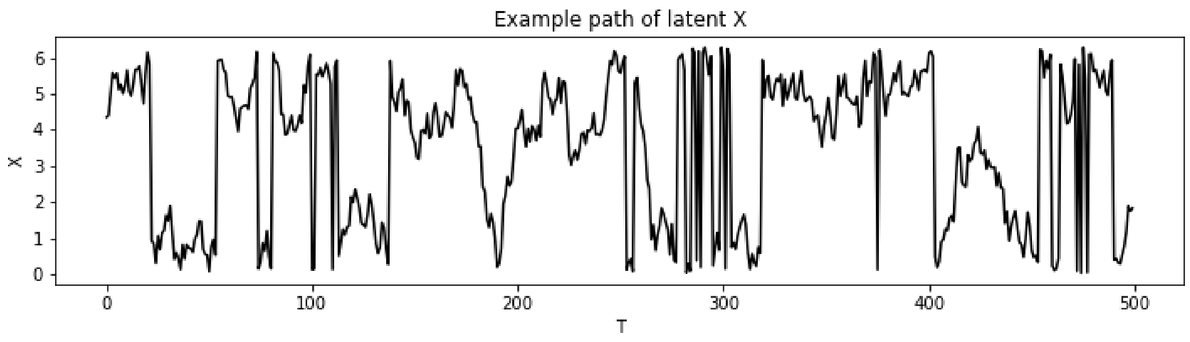

Figure 5.38: Example of path generated by the prior model, with periodic boundary conditions. $\sigma_x = 5, \delta_x = 50$

Notice how the latent variable suddenly makes a big jump from one end of the interval to the other, for example at around $T = 100$, representing when the mouse’s head direction passes over a predetermined reference direction (such as straight forward). These particular points where the path ”wraps around” the interval are known to be difficult to pick up, as e.g. PCA and ISOMAP have no way of imposing our knowledge of periodicity on the problem, and will simply try to fit their solution to the data. A set of 20 tuning curves, defined as Gaussian bumps, are also shown in Figure 5.39. Notice how the tuning curves may also now wrap around the boundary, in contrast to those shown for the non-periodic case in Figure 5.19.

This particular setup is exactly what inspired the feature sharing extension to the Spline-based LVM. A setting where the initialisation of $\mathbf{X}$ is less accurate, but where knowledge of the behaviour of the neurons allows us to make a more informed initialisation for the shape of the tuning curves.

As for the hyperparameters, we now use 11 interior knots for both models, and keep $\sigma^2_{\beta,[i,n]} = 10^2$ and $\sigma^2_{\beta,[n]} = 10^2$ as well. For the feature sharing variant, we set $s_i = 1$, $r_i = 3$, assign $\mu_{\gamma_i} = 0$, $\sigma_{\gamma_i} = 10$, and match $\boldsymbol{\theta}_{\text{lower}}, \boldsymbol{\theta}_{\text{upper}}$ to $x_{\text{min}}, x_{\text{max}}$, that is, 0 and 2π.

We then move on to investigate whether we can improve upon the Spline-based LVM by utilising the theory about feature sharing. We use the same arrays of T and N values as in the non-periodic case (see Equation 5.5), and compare how the average RMSE scaled with respect to those, with $\mathbf{X}$ initialised from ISOMAP.

The results can be seen in Figures 5.40 and 5.41, and are firmly in the favour of utilising feature sharing,

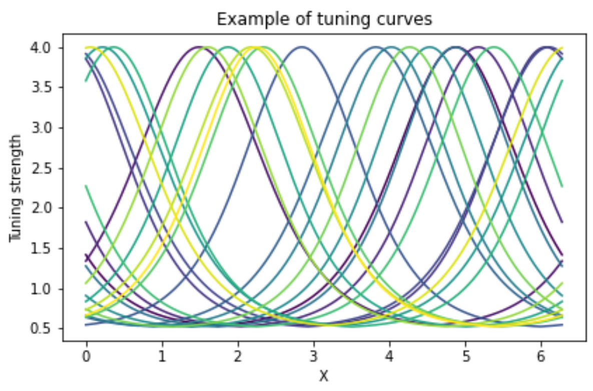

Figure 5.39: Selection of 20 tuning curves, defined as Gaussian bumps at random locations, with peak firing rate of 4 and background firing rate of 0.5 spikes per bin. Periodic boundary conditions.

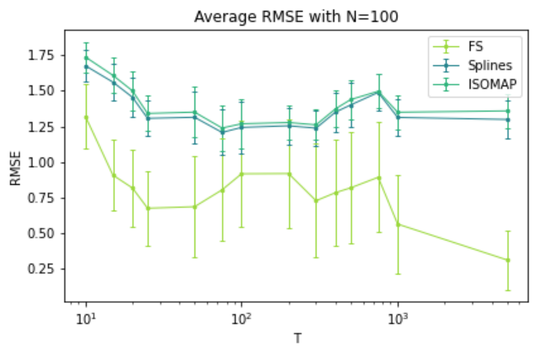

Figure 5.40: Average RMSE over 20 runs, as a function of T, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 5, \delta_x = 50$. Domain of $\mathbf{X}$: $[0, 2\pi]$ (periodic).

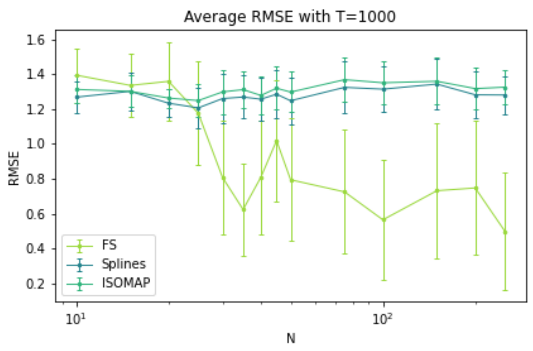

Figure 5.41: Average RMSE over 20 runs, as a function of N, with corresponding 95% confidence intervals. Path hyperparameters: $\sigma_x = 5, \delta_x = 50$. Domain of $\mathbf{X}$: $[0, 2\pi]$ (periodic).

here dubbed ”FS”. While both the initialisation ISOMAP and the standard Spline-based LVM saturate quickly, both with respect to T and N, we see that the model incorporating feature sharing has a much lower RMSE, all while trending downwards with increasing amounts of data. For smaller values of T, we also see that the RMSE decreases rapidly, which is in accordance with what Klindt et al. (2017) discovered; if you have enough neurons ($N = 100$ in this case), you can get away with a shorter recording length, under the assumption that you can pool all the neuronal data together, due to their similarly shaped tuning. In theory, this should also be the case for when you have a smaller amount of neurons, but longer recording periods, although we cannot observe this phenomenon in Figure 5.41. We do note that the confidence intervals are in general quite wide, but the improvement of using feature sharing is still significant.

As for the algorithm run time, we would expect that incorporating feature sharing would lead to the algorithm taking longer time. Even though we are reducing the dimension of $\boldsymbol{\beta}$, we are introducing three new variables, $\boldsymbol{\alpha}$, $\boldsymbol{\theta}$ and $\boldsymbol{\gamma}$, that each need to be found independently by MAP estimation. Figures 5.42 and 5.43 showcase how the scaling behaves, and while we do see a slight increase in run time, it is nowhere near as large of a differences as when we compared the LMT model with the Spline-based LVM. The scaling in N, as seen in Figure 5.43, is now also clearly linear, compared to the almost quadratic behaviour in Figure 5.37.

One interesting observation is that the run time is quite a bit longer for the feature sharing model where values of T are very small. We imagine that since the recording length is quite short, the latent variable ends up not covering the whole space, thus leaving multiple segments of the interval $[0, 2\pi]$ unvisited. Since we have a large number of neurons, with many of them possibly tuned to locations $\mathbf{X}$ does not visit, we might end up with quite a few neurons exhibiting no activity at all. This makes it hard to infer a general shape for the tuning curve that fits all the neurons' activity, as well as the placement of the peak intensity, resulting in the algorithm taking longer time to find an optimal solution.

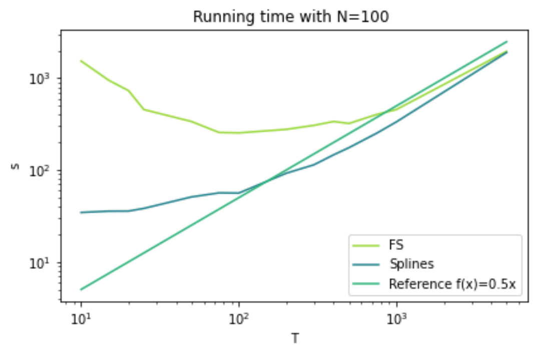

Figure 5.42: Scaling of algorithm run time, as a function of T. GP and Splines initialised from ISOMAP.

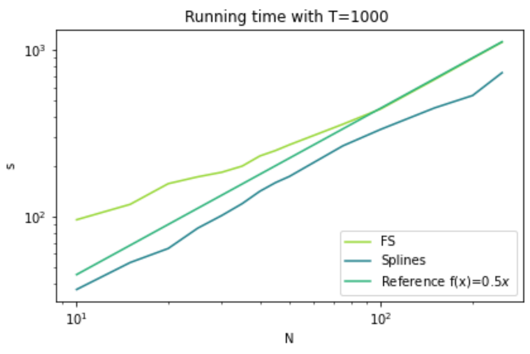

Figure 5.43: Scaling of algorithm run time, as a function of N. GP and Splines initialised from ISOMAP.

5.3 Head direction data

Having evaluated our models and compared them with respect to simulated data, we now move over to real head direction data, recorded by Peyrache et al. (2015). We treat the recorded head direction as the latent variable $\mathbf{X}$, a one-dimensional periodic variable existing on the domain $[0, 2\pi]$. Our goal will be to infer the head direction $\mathbf{X}$, comparing the results of LMT and the feature sharing model. We will be investigating both how well the tuning curves are reconstructed, as well as how close the inferred path is to the true head direction.

Although it is known that many of the neurons recorded does indeed respond to the head direction, there is however no guarantee that other factors does not influence the neurons as well. We will for simplicity assume that the head direction is the only driving force behind the neural spiking.

For the analysis, we pick an interval of 5000 time bins, for which the development of the head direction can be seen in Figure 5.44. We note that the behaviour of the path in general is pretty similar to that of the simulated latent variable in Figure 5.38, perhaps wrapping around the border a bit more.

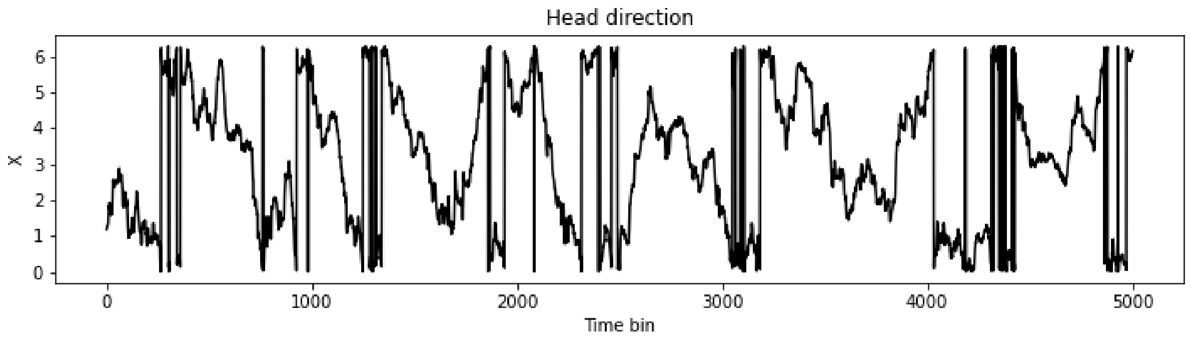

Figure 5.44: Head direction developement for Mouse12-120806 on a 5000 bin interval, with bin size of 25.6 ms.

For this interval, we inspect the activity and tuning of the 73 recorded neurons, identifying 15 which seem to be both tuned to head direction and produce a satisfactory amount of spikes to be considered informative. Observed firing rate for these 15 neurons can be seen in Figure 5.45.

We immediately note that these neurons are less active than the ones we simulated in the previous sections. The strongest tuned neurons average around $1.5 - 1.8$ spikes per bin, while about half average around one spike per bin, the rest even lower than that. In the simulated case, all neurons were set to spike with an intensity of four spikes per bin, to contextualise the difference. There is also the fact that we now only have 15 neurons for a data set of length 5000, which is in stark contrast to the 100 neurons used for the simulated data set of same length when we investigated the scaling of model performance. For completeness sake, we also showcase the binned spike data for the 15 neurons in Figure 5.46.

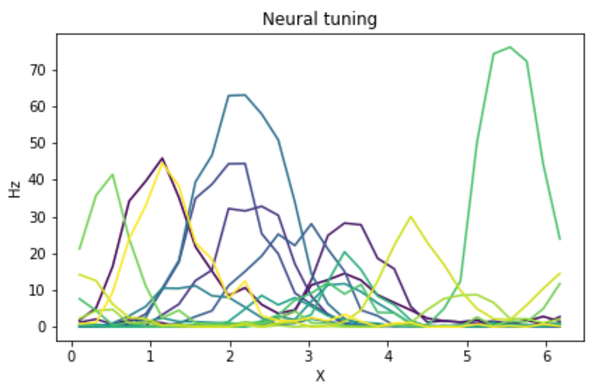

Figure 5.45: Observed firing rates (in Hz) for the 15 neurons, bin size 25.6 ms.

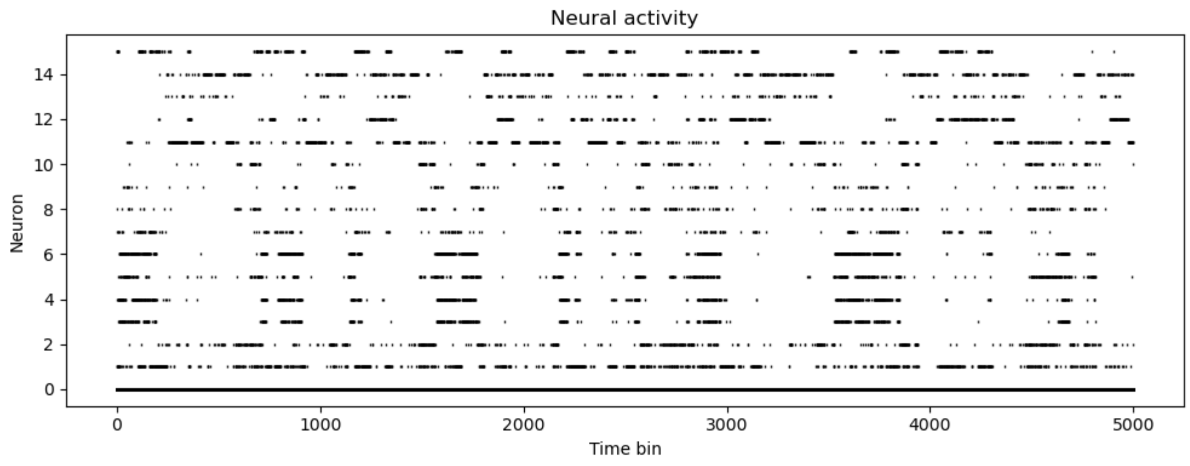

Figure 5.46: Binned and binarised spike data for the 15 chosen neurons for 5000 bins. The 15 neurons are placed on the y-axis, and a black dot means that at least one spike was observed in that particular time bin (of width 25.6 ms).

5.3.1 Inferring the tuning curves

As a starting point, we investigate how well the tuning curves can be reconstructed when initialising the feature sharing model from the true head direction, without updating the $\mathbf{X}$ estimate. Given the truth, the model should be able to recover the tuning curves reasonably well. For the hyperparameters, in the prior for $\mathbf{X}$ we set $\sigma_x = 5$, $\delta_x = 50$, while we keep the same hyperparameters as those used in Subsection 5.2.2 for the rest of the feature sharing model. These hyperparameters will be consistent throughout the rest of the chapter. We present four different tuning curves, with results shown in Figures 5.47 - 5.50.

The model does a fairly good job of reconstructing the tuning curves, attempting to also capture the extra bump that is present in Figure 5.47. This also affects the shape of some of the other tuning curves, like in Figure 5.49, where we can see the contours of the same bump. We suspect that this is due to the first neuron being more active compared to the others, having a higher firing rate, and thus contributing more to the likelihood than the less active neurons. Thus when doing MAP estimation, the resulting spline function might be biased towards providing a good fit to neuron number one. Other than this, the results seem to be satisfactory.

5.3.2 Initialisation from true path

Moving on, we now investigate how well the tuning curves can be reconstructed when initialising the model from the true head direction, as well as whether the inferred path deviates from the truth. Since Myklebust (2020) already established that the LMT model is able to mostly reconstruct the tuning curves, and infer the path of the head direction (although slightly shifted due to the initialisation of $\mathbf{F}$), we will focus our attention on only evaluating the feature sharing model in this part as well. The inferred $\mathbf{X}$ can be seen in Figure 5.51, while tuning

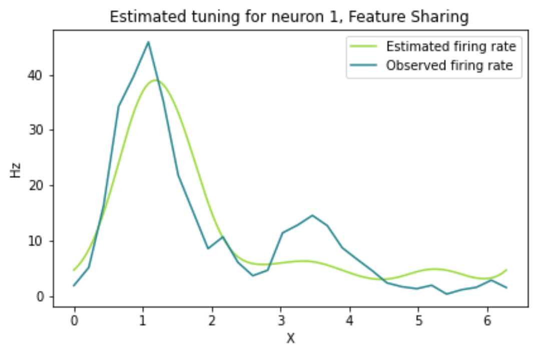

Figure 5.47: Inferred tuning curve for neuron 1, using the feature sharing model.

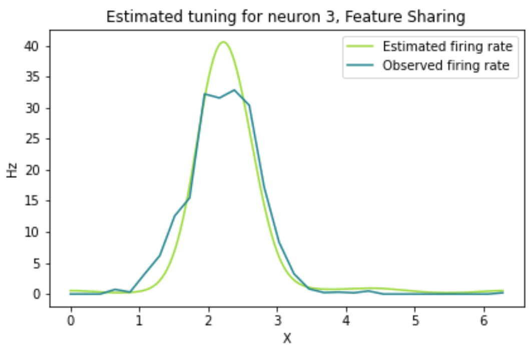

Figure 5.48: Inferred tuning curve for neuron 3, using the feature sharing model.

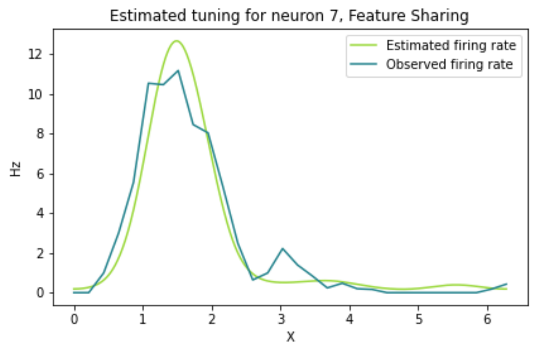

Figure 5.49: Inferred tuning curve for neuron 7, using the feature sharing model.

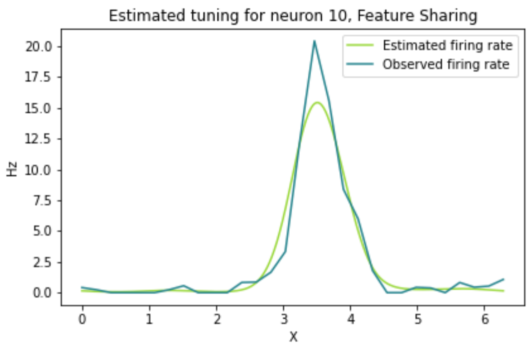

Figure 5.50: Inferred tuning curve for neuron 10, using the feature sharing model.

curves are shown in Figures 5.52 - 5.55.

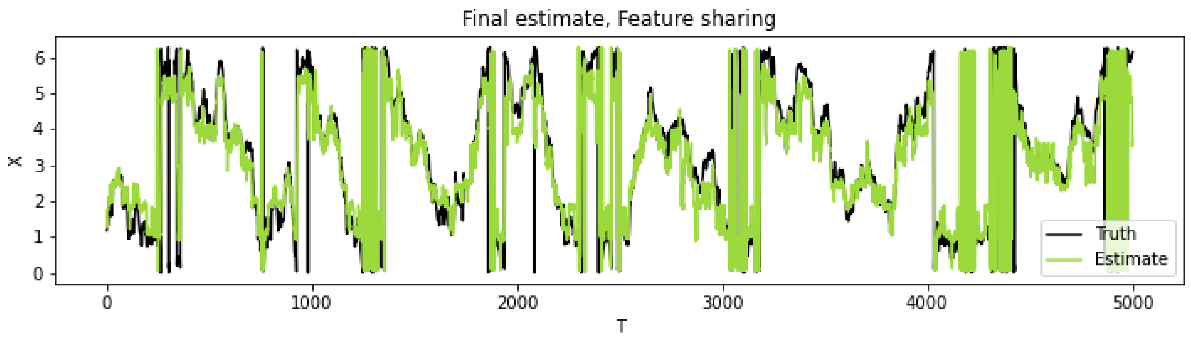

Figure 5.51: Inferred head direction development plotted against the true development, using feature sharing.

For reference, the resulting RMSE is 0.8449. For the most part, the inferred path seems to stay pretty close to the truth, while alternating between over-estimating ($T \in [0, 300]$, $T \in [700, 1000]$) and under-estimating ($T \in [300, 600]$, $T \in [3100, 3500]$) the path at various parts of the interval. The tuning curves, however, are in a worse shape. While the location of the tuning curve is mostly reconstructed, the shape and height seem to be quite far from the truth. Investigating this further, it seems to be the result of the spline-function immediately being fit to a too narrow shape, thus rendering the intentional effect of α and γ obsolete. With a too narrow fit to begin with, the model seems to overestimate the intensity, to compensate for the small width. This overestimated intensity appears to be some sort of local maxima, as trying to adjust the shape of the tuning curve by controlling α and γ only makes the too narrow shape even more narrow, or resulting in a curve with much smaller area under curve, compared to the observed tuning curve. One possible way to compensate for this is to lock the initial shape of the

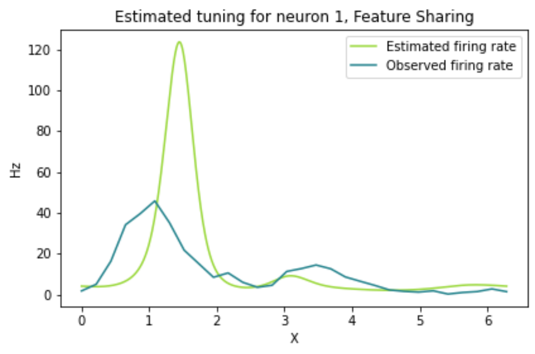

Figure 5.52: Inferred tuning curve for neuron 1, using the feature sharing model.

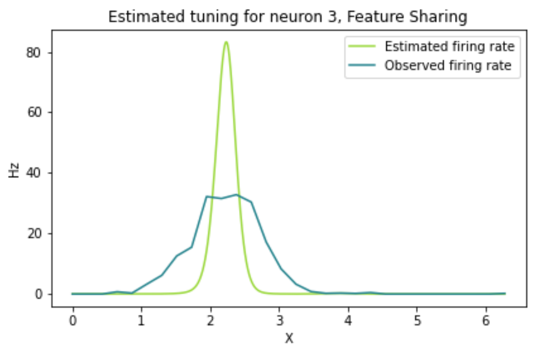

Figure 5.53: Inferred tuning curve for neuron 3, using the feature sharing model.

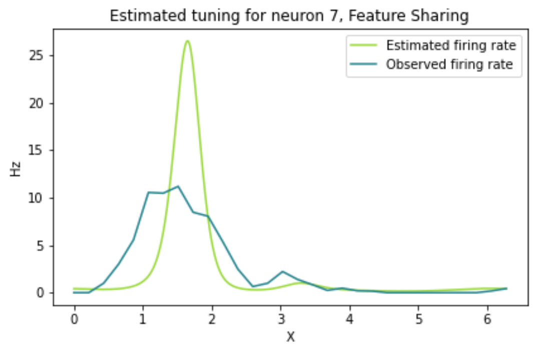

Figure 5.54: Inferred tuning curve for neuron 7, using the feature sharing model.

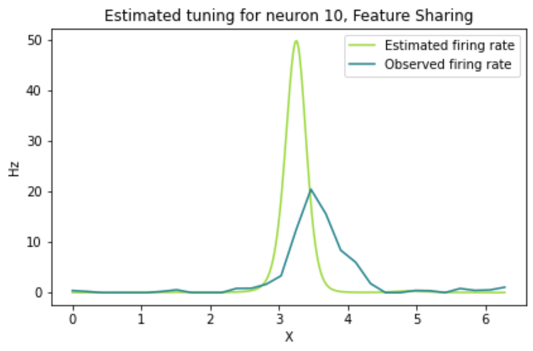

Figure 5.55: Inferred tuning curve for neuron 10, using the feature sharing model.

spline-function for more iterations during the MAP procedure. Experiments show that this gives a better estimate of the intensity, however, the variables $\boldsymbol{\alpha}, \boldsymbol{\theta}, \boldsymbol{\gamma}$ then seem to hone in on some local maxima, resulting in the initial shape of the spline-function not changing at all. As one of the intentions behind using the feature sharing technique is to be able to infer a suitable fit with the spline-function, it is not obvious that initially locking the shape is the preferred solution to this problem. Further investigation would be necessary to identify a proper solution to this issue.

5.3.3 Initialisation from PCA

Finally, to compare the results from the LMT model and the Spline-based LVM with feature sharing, we initialise $\mathbf{X}$ with PCA applied to the neural data (smoothed with a Guassian filter of length 4). Both PCA and ISOMAP are valid initialisations, but ISOMAP picks up more of the underlying noise, resulting in a more noisy initialisation. We therefore stick to PCA for this experiment, but note that neither PCA nor ISOMAP are very good at picking up the parts of the path where the head direction wraps around the border.

For the hyperparameters, in the prior for $\mathbf{X}$ we set $\sigma_x = 5$, $\delta_x = 50$, while the LMT specific hyperparameters are set to $\sigma_f = 8$, $\delta_f = 0.5$, a similar setup as the one used by Myklebust (2020). For the feature sharing model, we keep the same hyperparameters as those used in Subsection 5.2.2. The results can be seen in Figures 5.56 and 5.57.

Immediately, it looks like the results from the two models are somewhat similar. Some parts are captured fairly well, while others are completely off. Interestingly, it seems like both models have produced inferred paths that are partially flipped around the interval $T \in [2500, 3000]$, as well as $T \in [4000, 4300]$. Neither model seems to be particularly good at capturing the points where the path wraps around. As for the RMSE, the results from the model using feature sharing has an RMSE of 1.6759, while the LMT result has an RMSE of 1.7934. For reference, the PCA initialisation has an RMSE of 1.7249. Although the model using feature sharing in this case has the lowest RMSE, the results are far from optimal, and we find it difficult to say for certain whether one model truly is better than the other. However, what we do see indications of is that the model using feature sharing attempts to correct

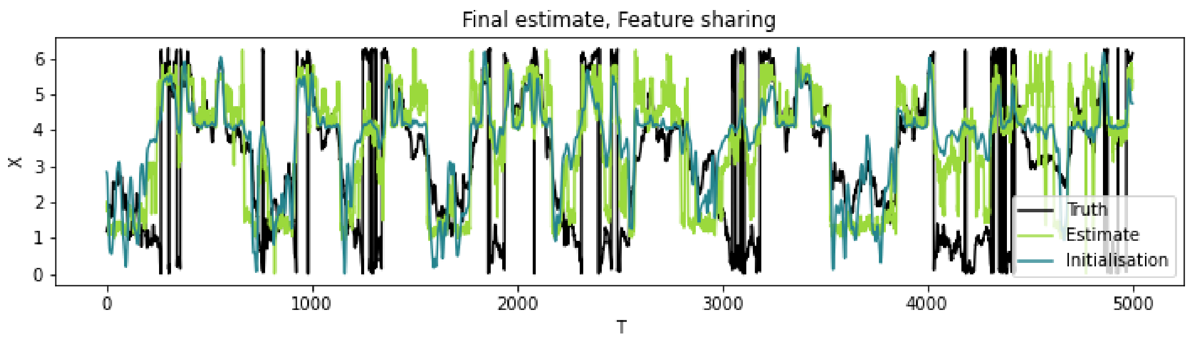

Figure 5.56: Inferred head direction development plotted against the true development, using feature sharing.

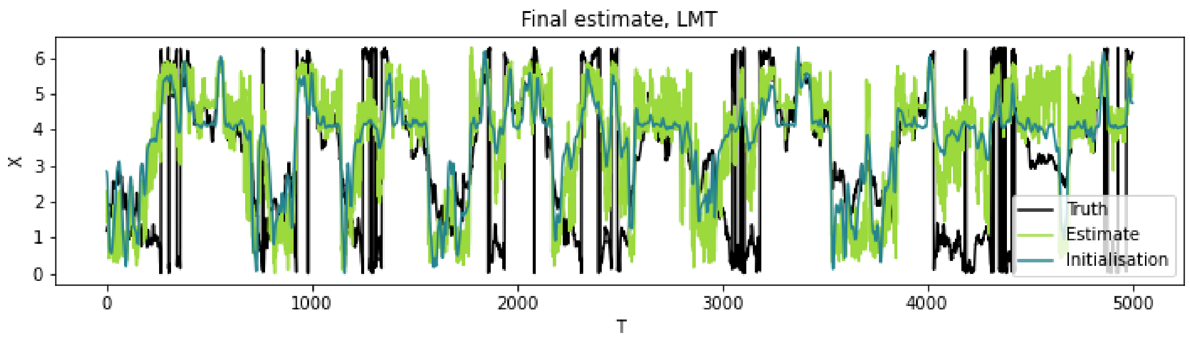

Figure 5.57: Inferred head direction development plotted against the true development, using LMT.

the PCA failure to pick up on the path wrapping around, like for instance at around $T = 1900$ and $T = 4100$. A selection of inferred tuning curves are also shown in Figures 5.58 - 5.65.

For neurons number one, three and seven, it seems like the fit would be better if we were to mirror the tuning curves around $x = 2$, which is in accordance with how the inferred paths seemed to also be partially flipped around this value in the intervals mentioned above. The intensity is generally better captured by the LMT model, compared to the results from the feature sharing model, which massively overestimates the peak firing rate. Especially the tuning curve for neuron 10 seems to be completely wrong, both with respect to placement and intensity. As for why, the cause has yet to be determined, but we suspect it might be linked to the same reasons discussed in Subsection 5.3.2.

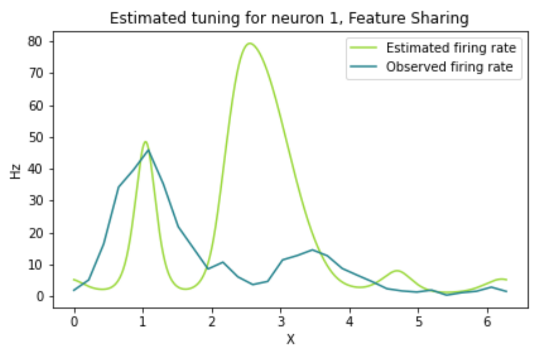

Figure 5.58: Inferred tuning curve for neuron 1, using the feature sharing model.

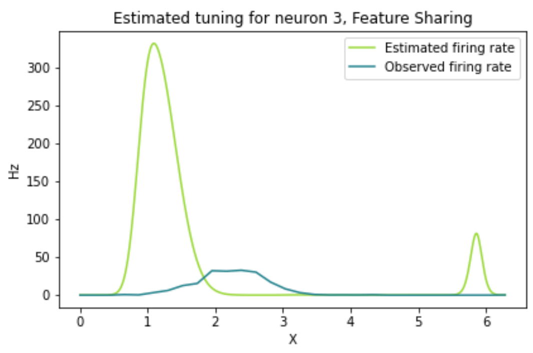

Figure 5.59: Inferred tuning curve for neuron 3, using the feature sharing model.

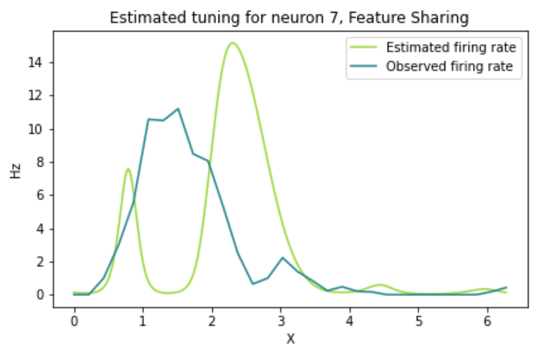

Figure 5.60: Inferred tuning curve for neuron 7, using the feature sharing model.

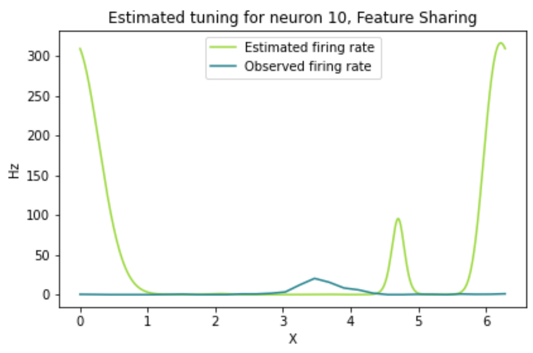

Figure 5.61: Inferred tuning curve for neuron 10, using the feature sharing model.

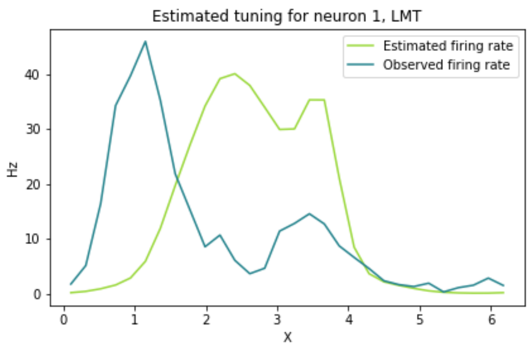

Figure 5.62: Inferred tuning curve for neuron 1, using the LMT model.

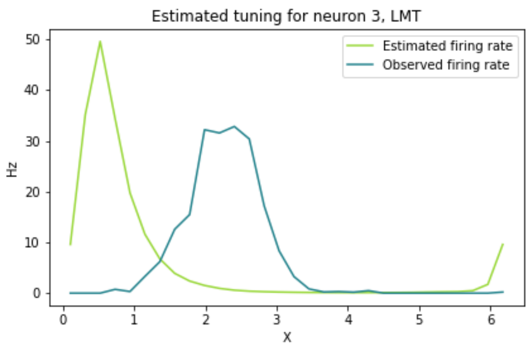

Figure 5.63: Inferred tuning curve for neuron 3, using the LMT model.

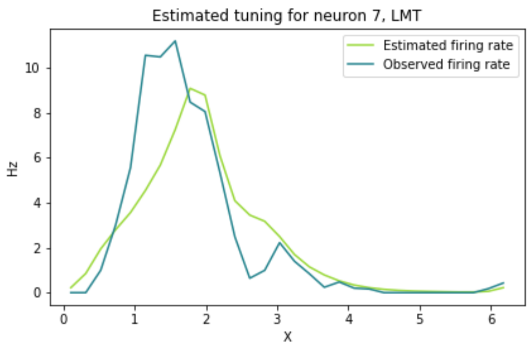

Figure 5.64: Inferred tuning curve for neuron 7, using the LMT model.

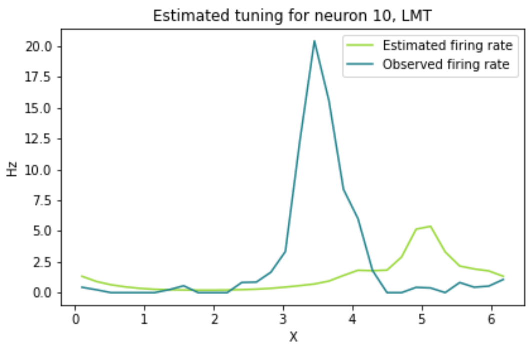

Figure 5.65: Inferred tuning curve for neuron 10, using the LMT model.

Chapter 6

Discussion and conclusion

In this chapter, we discuss the results and findings from Chapter 5. To begin with, we summarise and interpret our results in Section 6.1. In Section 6.2 we mention possible ways to extend the research further, and build upon the results, before we close off with a conclusion in Section 6.3.

6.1 Discussion

6.1.1 Modeling choices and challenges

In Section 5.1, we address the choice of spline hyperparameters, and comment on the need for adjustments and transformations of the inferred path before comparison with the true latent variable can be performed. Indications show that the Spline-based LVM might be less susceptible to the issue of partially flipped solutions, although it in turn suffers from partially shifted solutions. We recognise that changing prior hyperparameters might affect the frequency of these unwanted local maxima appearing, for better or worse.

We describe the importance of initialisation, and mention how the LMT model may have an advantage in this area, utilising the information in the spike data for a valuable initialisation of the log tuning curves. The Spline-based LVM, lacking a sophisticated log tuning curve initialisation, becomes increasingly more reliant on a good initialisation of the latent variable. Under the right circumstances, however, this can be offset by utilising feature sharing for an advantageous log tuning curve initialisation. We also comment on various initialisations for the latent variable, and how the available data affects their accuracy.

We also touch upon the subject of evaluation, and note how the posterior score is ineffective for evaluating the performance of the Spline-based LVM in the absence of ground truth. This result is similar to what Myklebust (2020) discovered, in that the log posterior score is not a valid measure for the performance of the LMT model.

6.1.2 Simulated data

In Section 5.2, we compare models with respect to their performance scales with increasing T and N, and discover that with respect to scaling in N, the LMT model does better when the true path exhibits less oscillatory behaviour, while the Spline-based LVM does better when the opposite is true. We equate this to the LMT model's spatial prior, which is dependent on the position and density of the latent variable, in contrast to the ”uniform” assumption in the Spline-based LVM. We also observe that, regardless of oscillatory behaviour, the LMT model performs poorer for larger values of T. Attributing this to the increase in opportunities for a local maxima to occur, we speculate whether such maximas might be less prominent for the Spline-based LVM, or if its continued improvement with respect to larger T is a product of the tight coupling between $\boldsymbol{\beta}$ and $\mathbf{X}$, thus only resulting in a slight improvement on the initialisation (which we also see does not suffer from poorer performance with large T). Doing only a single gradient step during optimization, instead of running the optimizer until full convergence, seems to results in an overall more accurate estimate, and might help combat this possible issue.

Investigating other initialisations, we see that on simulated data, the efficiency of ISOMAP as an initialisation closes much of the performance gap between the two models. However, the LVM model still performs worse when the true path oscillates more, to the point where it converges to a worse solution than the original initialisation.

This seems to strengthen the claim that the local maxima issue is more prominent in the LVM model, and not necessarily only a product of poor initialisation.

On another note, there seems to be a clear difference in algorithm run time, regardless of which initialisation is used. Being an order of magnitude faster (both with respect to scaling in T and N), the Spline-based LVM might be able to accommodate a Bernoulli spiking model, in contrast to a Poisson spiking model, which we speculate might improve results further. This would also implicate a smaller bin width, which should yield a more accurate representation of the neural spike data. Additionally, one must not forget that algorithm run time can be an important factor when considering model choices, given that neural data sets continue to grow exponentially in size.

We compare the results of the Spline-based LVM incorporating theory from feature sharing, to the ordinary Spline-based LVM. Under the right circumstances (i.e. when the tuning curves all share a similar shape), utilising feature sharing provides a significant upgrade, resulting in a more accurately inferred path in a setting where the latent variable is periodic. We see that the increase in computational complexity is not too large, compared to the gain in accuracy. However, while we in theory should be able to perform more accurate inference with fewer neurons, as we are now able to pool their data together, results are not indicative of this. We can only speculate that a recording length of $T = 1000$ might be to short to harness this power, in the absence of other explanations.

Do note that, as mentioned in Subsection 5.2.1, some modeling choices have been made with respect to e.g. hyperparameter choices, tuning strength and tuning curve shape. Our results are (partially) a product of these choices, and we recognise different choices might give rise to different results.

6.1.3 Head direction data

In Section 5.3, we apply our models to data gathered from real head direction neurons by Peyrache et al. (2015). Compared to simulated data, these neurons spike with vastly lower intensity, as well as being fewer in numbers. Given that they also might be tuned to other variables (in addition to head direction), be linked to each other through connectivity and have autoregressive properties, none of which are accounted for in our models, we would expect the inferred path to be less accurate compared to the simulated case.

We observe that the feature sharing model is able to reconstruct the tuning curves when given the true path, although it struggles to accurately estimate the peak intensity when also inferring the latent variable $\mathbf{X}$. We mention the possibility of keeping the initial shape of the spline-function constant during additional steps of the MAP procedure, but we are, without further investigation, unable to conclude whether this would be a suitable solution for inferring the tuning curves more accurately.

Comparing the feature sharing model with the LMT model, we see that the two models produce somewhat similar results, both showing indications of partially flipped solutions. This is also reflected in the tuning curves, where mirroring some would provide a more accurate result. Although the LMT model is more effective at estimating the firing rate, it has an overall higher RMSE compared to the feature sharing model, which does improve upon the RMSE of the PCA initalisation. Whether one of the results could truly be deemed "better" would require further investigation.

6.2 Further work

As the framework of these latent variable models are made to account for various dimensionalities and manifolds, it would be interesting to investigate whether the Spline-based LVM would be able to infer a latent variable of higher dimensionality than one (as was done in the case of head direction data). This is akin to how Jensen et al. (2020) used the LMT to explore different possible manifolds.

We also recognise that for both of our models, the hyperparameters, which were set according to a trial and error process, should optimally be found through optimization, in a similar manner to the latent variable and the tuning curve variables.

Investigating other possibilities for doing variable estimation, as alternatives to the iterative MAP procedure, is also of interest. Given the dimensionality of our problem, and the existence of local maxima, we recognise that the MAP procedure might not be an optimal choice. Although Wu et al. (2017) devised the decoupled Laplace approximation for doing inference, it is not given that this method is also applicable for the Spline-based LVM (both with and without feature sharing).

On the note of feature sharing; since the variables that control the general shape of the tuning curve (the $\boldsymbol{\beta}$'s) were found through MAP estimation, and very active neurons contribute more towards the likelihood, the general shape of the tuning curve might be biased in the direction of the shape of these neurons' tuning curves. We imagine incorporating something akin to a weight parameter, to account for the differences in neural activity, would ensure that the most active neurons do not dominate the fit of the general tuning curve shape.

Additionally, since the Spline-based LVM is much faster computationally, it would be interesting to explore whether a change from spike count data to spike presence data would improve the results, since this model might be able to accommodate the much smaller bin width.

6.3 Conclusion

We showed how one can exchange the Gaussian process modeling the tuning curve in the LMT model with a B-spline function, devising a model we named the Spline-based LVM. We compared its performance with the LMT model on simulated data, evaluating how the models scaled with respect to data size, changes in hyperparameters and initialisation. While no single model was overall superior in each and every one of the cases, the Spline-based LVM was consistently less computationally demanding. Besides being a highly useful trait when evaluating large data sets, it could allow for the consideration of a smaller bin width and a different spiking distribution, which in turn could improve results further.

Furthermore, we detailed how the Spline-based LVM can be expanded upon by incorporating feature sharing between neurons. This yielded more accurate results on simulated data, compared to the generic Spline-based LVM, without too large of a cost in computational complexity.

Finally, we applied the models to head direction data, where we inferred the head direction with higher precision using the Spline-based LVM incorporating feature sharing, compared to PCA and this particular configuration of the LMT model.

Bibliography

Agresti, A., 2015. Foundations of linear and generalized linear models. John Wiley & Sons.

Ahrens, M.B., Li, J.M., Orger, M.B., Robson, D.N., Schier, A.F., Engert, F., Portugues, R., 2012. Brain-wide neuronal dynamics during motor adaptation in zebrafish. Nature 485, 471–477.

Ahrens, M.B., Orger, M.B., Robson, D.N., Li, J.M., Keller, P.J., 2013. Whole-brain functional imaging at cellular resolution using light-sheet microscopy. Nature methods 10, 413–420.

Albright, T.D., 1984. Direction and orientation selectivity of neurons in visual area mt of the macaque. Journal of neurophysiology 52, 1106–1130.

Batty, E., Merel, J., Brackbill, N., Heitman, A., Sher, A., Litke, A., Chichilnisky, E., Paninski, L., 2017. Multilayer recurrent network models of primate retinal ganglion cell responses. 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings .

Briggman, K.L., Abarbanel, H.D., Kristan, W.B., 2005. Optical imaging of neuronal populations during decision-making. Science 307, 896–901.

Brody, C.D., 1999. Correlations without synchrony. Neural computation 11, 1537–1551.

Byrd, R.H., Lu, P., Nocedal, J., Zhu, C., 1995. A limited memory algorithm for bound constrained optimization. SIAM Journal on scientific computing 16, 1190–1208.

Cunningham, J.P., Byron, M.Y., 2014. Dimensionality reduction for large-scale neural recordings. Nature neuroscience 17, 1500.

Davidovich, I., Dunn, B., Hertz, J., Roudi, Y., 2020. Mean field theory inference and learning in networks with stochastic natural exponential family neurons. 29th Annual Computational Neuroscience Meeting: CNS2020. BMC Neuroscience .

De Boor, C., 1978. A practical guide to splines. volume 27. springer-verlag New York.

DiMatteo, I., Genovese, C.R., Kass, R.E., 2001. Bayesian curve-fitting with free-knot splines. Biometrika 88, 1055–1071.

Ecker, A.S., Berens, P., Cotton, R.J., Subramaniyan, M., Denfield, G.H., Cadwell, C.R., Smirnakis, S.M., Bethge, M., Tolias, A.S., 2014. State dependence of noise correlations in macaque primary visual cortex. Neuron 82, 235–248.

Fahrmeir, L., Kneib, T., Lang, S., Marx, B., 2013. Regression: models, methods and applications. Springer Science & Business Media.

Fetz, E.E., 1992. recognizably coded in the activity of single neurons? Behavioral and brain sciences , 154.

Gothard, K.M., Skaggs, W.E., McNaughton, B.L., 1996. Dynamics of mismatch correction in the hippocampal ensemble code for space: interaction between path integration and environmental cues. Journal of Neuroscience 16, 8027–8040.

Gundersen, G.W., Zhang, M.M., Engelhardt, B.E., 2020. Latent variable modeling with random features. arXiv preprint arXiv:2006.11145 .

Haberman, S.J., 1977. Maximum likelihood estimates in exponential response models. The annals of statistics , 815–841.

Hafting, T., Fyhn, M., Molden, S., Moser, M.B., Moser, E.I., 2005. Microstructure of a spatial map in the entorhinal cortex. Nature 436, 801–806.

Hatsopoulos, N.G., Ojakangas, C.L., Paninski, L., Donoghue, J.P., 1998. Information about movement direction obtained from synchronous activity of motor cortical neurons. Proceedings of the National Academy of Sciences 95, 15706–15711.

Hubel, D.H., Wiesel, T.N., 1979. Brain mechanisms of vision. Scientific American 241, 130.

Jensen, K.T., Kao, T.C., Tripodi, M., Hennequin, G., 2020. Manifold gplvms for discovering non-euclidean latent structure in neural data. arXiv preprint arXiv:2006.07429 .

Klindt, D., Ecker, A.S., Euler, T., Bethge, M., 2017. Neural system identification for large populations separating "what" and "where", in: Advances in Neural Information Processing Systems, pp. 3506–3516.

Kulkarni, J.E., Paninski, L., 2007. Common-input models for multiple neural spike-train data. Network: Computation in Neural Systems 18, 375–407.

Lawrence, N., 2005. Probabilistic non-linear principal component analysis with gaussian process latent variable models. Journal of machine learning research 6.

Lawrence, N.D., 2003. Gaussian process latent variable models for visualisation of high dimensional data., in: Nips, Citeseer. p. 5.

Mazor, O., Laurent, G., 2005. Transient dynamics versus fixed points in odor representations by locust antennal lobe projection neurons. Neuron 48, 661–673.

Myklebust, E.M., 2020. A robustness evaluation of the latent manifold tuning model. Master's thesis. Norwegian University of Science and Technology.

Nocedal, J., Wright, S., 2006. Numerical optimization. Springer Science & Business Media.

Paninski, L., Cunningham, J.P., 2018. Neural data science: accelerating the experiment-analysis-theory cycle in large-scale neuroscience. Current opinion in neurobiology 50, 232–241.

Peyrache, A., Lacroix, M.M., Petersen, P.C., Buzsáki, G., 2015. Internally organized mechanisms of the head direction sense. Nature neuroscience 18, 569.

Piegl, L., Tiller, W., 1996. The NURBS book. Springer Science & Business Media.

Quinonero-Candela, J., Rasmussen, C.E., 2005. A unifying view of sparse approximate gaussian process regression. The Journal of Machine Learning Research 6, 1939–1959.

Rasmussen, C., Williams, C., 2006. Gaussian Processes for Machine Learning. MIT press.

Smith, A.C., Brown, E.N., 2003. Estimating a state-space model from point process observations. Neural computation 15, 965–991.

Steinmetz, N.A., Koch, C., Harris, K.D., Carandini, M., 2018. Challenges and opportunities for large-scale electrophysiology with neuropixels probes. Current opinion in neurobiology 50, 92–100.

Stevenson, I.H., Kording, K.P., 2011. How advances in neural recording affect data analysis. Nature neuroscience 14, 139–142.

Taube, J.S., Muller, R.U., Ranck, J.B., 1990. Head-direction cells recorded from the postsubiculum in freely moving rats. i. description and quantitative analysis. Journal of Neuroscience 10, 420–435.

Tenenbaum, J.B., De Silva, V., Langford, J.C., 2000. A global geometric framework for nonlinear dimensionality reduction. science 290, 2319–2323.

Thorndike, R.L., 1953. Who belongs in the family? Psychometrika 18, 267–276.

Titsias, M., 2009. Variational learning of inducing variables in sparse gaussian processes, in: Artificial intelligence and statistics, PMLR. pp. 567–574.

Titsias, M., Lawrence, N.D., 2010. Bayesian gaussian process latent variable model, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings. pp. 844–851.

Wu, A., Pashkovski, S., Datta, S.R., Pillow, J.W., 2018. Learning a latent manifold of odor representations from neural responses in piriform cortex., in: NeurIPS, pp. 5383–5393.

Wu, A., Roy, N.A., Keeley, S., Pillow, J.W., 2017. Gaussian process based nonlinear latent structure discovery in multivariate spike train data, in: Advances in neural information processing systems, pp. 3496–3505.

Yu, B.M., Cunningham, J.P., Santhanam, G., Ryu, S.I., Shenoy, K.V., Sahani, M., 2009. Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. Journal of neurophysiology 102, 614–635.

Yuste, R., 2015. From the neuron doctrine to neural networks. Nature reviews neuroscience 16, 487–497.