# Gaussian Processes on Graphs

University of Oxford

St Anne's College


*Doctor of Philosophy*

Supervisor: Xiaowen Dong

Yin-Cong Zhi

March 28, 2024

# Abstract

In the ever-growing field of machine learning research, the use of graphs has recently gathered significant interest for modelling on data with relational structures. Graphs and network-based data now exist ubiquitously in the real world, with examples including social networks, transportation, financial exchanges, and brain networks. Therefore, developing models on graphs is essential to allow users to understand and predict the complex nature observed in everyday phenomena. Currently, there is an abundance of literature on graph neural networks, but limited options are available that are probabilistic and Bayesian. In addressing this issue, we develop a series of Gaussian processes (GPs) for graph data in this thesis. Building GPs on graphs is now more feasible thanks to the emergence of graph signal processing, providing us with the tools to handle graph-structured information and smoothness modelling. The first problem we tackle is predicting the evolution of signals with a multi-output Gaussian process. We use kernels defined from the graph Laplacian with learnable spectral filters to predict with the smoothness level that matches the data. We then turn our focus to semi-supervised classification, designing three models for this task each emphasizing on a particular approach: multi-scale modelling, transductive learning, and sheaf modelling. The first approach provides a novel utilization of wavelets on graphs to fully exploit their ability to capture multi-scale properties in the data. Next, we present a unified definition of kernels on graphs with transductive properties, aiming to utilize the distribution of the full dataset to better inform the prediction. This naturally suits semi-supervised problems on graphs where training and testing nodes are generally connected and available at the same time. Finally, we introduce sheaves as a higher order representation of graphs, to design GPs with stronger separation power by learning additional topological structures. Collectively, this thesis represents not only a valuable contribution to the study of GPs for discrete and non-Euclidean data, but also useful alternatives to the more broadly used graph neural networks.

# Declaration

I declare that this thesis is my own work, except when stated otherwise.

# Acknowledgements

Everyone I met and interacted with during this DPhil project has become more than just people I worked with, they are now also great friends that will last beyond this degree. My sincere gratitude firstly goes to Xiaowen Dong, my supervisor, for taking me on as a DPhil student, and the substantial amount of time he has put into helping me shape my research. He was there to offer fresh ideas and guidance when I needed them, while also giving me the freedom to explore what I found interesting. Under his supervision I grew from a shy novice to now a confident and knowledgeable scientist in my field of study. My thanks also go to Stephen Roberts, for always being available to offer advice when we sought them.

Amongst the people I worked with, I would like to first give a mention to my collaborators at Cambridge, Felix Opolka and Pietro Lio, who I was introduced to thanks again to Xiaowen. Their expertise complemented particularly well with my background, and proved invaluable in the projects we worked together on to this day. Thanks to my acquaintance with them, I was able to visit their lab at Cambridge during my final year of study, something that hugely broadened my knowledge and ideas for my research going forward. I would also like to thank Yin Cheng Ng, who kindly offered to work with me as a close collaborator from the start of my DPhil. As someone who has been through the process of a Ph.D., he offered the guidance for me to raise my level of work to that of today.

I would also like to give my thanks to Henry Kenlay and Stacy Xingyue Pu who began the journey with me at OMI and became dependable support every week, where we shared ideas, papers, and stimulating discussions that helped me learn much more than I could by myself. Henry in particular, also offered up his model reports at each milestone of the DPhil, and they became integral when it came to my time to produce the same documents. Our group meetings became even more engrossing when more members joined the group, so credit to Alan Chau, Deborah Sulem, Pierre Osselin, Valentina Semenova, Baskaran Sripathmanathan, Bohan Tang, Dragos Gorduza, and Li Zhang for being part of my DPhil.

I also could not have completed this DPhil without the support of my parents, Zheng Liang Zhi and Hua Zhang, who set me off on the road to this DPhil, and were with me every step of the way. The most important of all was being able to stay with them during the unfortunate lockdown, shielding me from the risk of Covid-19, which has been the toughest period of this degree.

My mental and physical health are accredited to my partner Anna Liu, who supported me in and outside of work, encouraging me to be more outgoing and take up opportunities when they came, and to take part in more sports and extra-curricular activities and was always there as someone I can talk to outside of work.

My college MCR at St Anne's is also a big reason why my time here at Oxford holds so deeply to my heart. To name a few: Vedang Narain, Riddhi Jain, Ramani Chandramohan, Isabella Martus, Brittany Hause, Lisa Zhang, Theresa Rule, Zhen Shao, May Sin Ke, Amanda Ang, Will van Noordt, Harry Tappenden, Dragos Gorduza, Ryan Gidda, Malaika Ivey, Julian Kaptanian, Mark Hughes, Lintong Zhang, Rosanna Jackson, Merve Bilici, Daniel Hortelano, and Siyun Chen.

Last but not least, I'd like to express my gratitude to EPSRC for providing me with the studentship (Project Reference: EP/S515541/1; 2265637) that allowed me to complete this thesis at Oxford.

# Contents

# Chapter 1

# Introduction

Graphs have become ubiquitous tools to represent more complex and irregular data [1, 2]. The combination of nodes and edges provides a coherent representation of pairwise relationships and interactions between entities, with the connectivities generally dictated by the physics of the problem at hand or inferred from the data. Real world structures that can be efficiently represented as graphs have become increasingly common, such as social networks, citations, transportation, sensor images, and brain networks. Thus, many problems can be solved with the help of graphs, including classification of nodes, prediction of links and relationships, and simulation of the evolution of systems. Therefore, modelling data as graphs can help us better understand the complex nature observed in many modern-day phenomena.

Graphs are not only useful for representing structured data, they also play a key role in modern day machine learning. Recently, graph-based data has provided many challenges for popular models including Gaussian processes (GPs) [3] and neural networks, and as a result, studies in adapting to graphs have led to significant advancements in learning on geometric and non-Euclidean data. Learning on graphs has been studied from both machine learning and signal processing perspective [2, 4, 5, 6], and with the rise of graph neural networks (GNNs) [7, 8], this has led to substantial interest in designing models for graph-structured data, and tackling the various challenges that come with using graphs. Notably, modelling on graphs has provided fresh perspective on existing neural networks and GPs, and the use of simple procedures such as aggregating information over neighbouring nodes has already led to many performance improvements compared to the non-graph counterparts that ignore the graph connectivity. As we will see in this thesis, such improvements are only apparent under certain assumptions, but now with more advanced tools available, it is possible to design models that are more flexible and

adaptable to the different smoothness profiles of each problem setting, leading to more generalizable machine learning models.

Amongst the huge library of models in machine learning, we focus on GPs and their adaptation to graphs. GPs are powerful models routinely employed for regression and classification tasks. The model is known for its flexibility and having a wide range of applications, often making them the preferred and optimal choice to solve real-life problems. There are many advantages of using a GP, the first of which is in leveraging the power of non-parametric modelling and data efficiency of regression, thus parameters are generally fewer and more easily inferred, allowing the model to learn from very few data points. Despite the few parameters, GPs still have the flexibility due to the use of powerful kernel functions that maps data into higher dimensions where separating into the label classes is possible. Another element that comes with such models is clear interpretability, to allow the user to understand the predictions, as opposed to the black-box that deep learning models exhibit. Furthermore, GPs also belong to the Bayesian family of models, characterized by the use of priors to allow the user to incorporate additional knowledge. Subsequently, this leads to the ability to predict with confidence which can contribute to the user's decision to trust the model. Commonly compared to neural network and deep learning models, with each having its distinctive advantages, GPs have reliably produced rivalling performances against models known for having great learning power [9, 10, 11].

While GPs are potent tools, they encounter challenges when employed with graph-structured data. The use of kernels focuses on covariance structures, and therefore lacks an explicit structural modelling of the graph connectivity. The graph provides additional information on top of the feature data (often on the nodes), and is governed by discrete operators that do not combine easily with kernels defined on the continuous space. Kernel selection therefore becomes non-trivial and one has to carefully decide on which graph operators to use, and how to integrate this with the feature data kernel. Currently, there is yet a unifying framework of kernels for graph data, and therefore one does not have the options to specify any desired properties such as smoothness modelling, transductive learning, and how to combine kernels to produce higher order interactions between the feature data and the graph. The work in this thesis therefore provide a range of options to the GP literature, allowing for modelling of more complex graph-based problems. [12]

As the central element of GPs, adapting to graph data is predominantly achieved by embedding graph information in the kernel, and one can think of this as adding

information to the prior. In the recent developments of graph GPs, the relational structure of the graph is generally utilized to average the input data over direct neighbourhoods determined by the graph. This has already shown promising results, performing competitively against state-of-the-art GNNs on numerous datasets. Using a GP also offered other advantages of being a Bayesian model, compared to GNNs which generally represent a non-probabilistic approach. Building on this foundation will have certain challenges, as naturally there exists important information for prediction beyond the immediate neighbours. How to best manage such information is non-trivial, as empirical studies have shown that there is the need to maintain a balance between local and global information. On the other hand, the graph spectral domain offers an alternative modelling perspective that is less explored, and can be key to capturing behaviours that are not apparent in the spatial domain, of which many current approaches operate in. Beyond this, we also have the task of translating the relevant information into a valid kernel function, for which there is currently limited literature.

There are different prediction tasks when it comes to graph data, here we broadly categorize them as: *Graph signal prediction* - where we wish to model the evolution of the values on the nodes over the full graph; *Semi-supervised node classification* - working within a single graph that is partially labelled and we aim to predict the unlabelled nodes; and *Graph classification* - where each data point is an individual graph for which we infer the label for. Within these problems, we also tackle more technical challenges throughout this thesis, we now introduce them in the following paragraphs.

From a signal processing perspective, many operators used in the design of graph-based models may be interpreted as a result of low-pass functions [13, 14], the GP model of [15] is an example that makes use of low-pass filters to induce spatial structures within the predicted graph signals. Although information is generally in the lower frequency elements, incorporating higher frequencies of the likes of band- and high-pass are required to fully capture the behaviours of modern-day data, where labels can exhibit less smooth or high frequency information. In addressing this limitation we allow the kernel's spectral function to be free to take any shape instead of being restricted to low-pass, opening the possibility to band- or high-pass profiles (in addition to low-pass). Learning the kernel via its spectral function has been an effective technique on the continuous space such as [16, 17], but translating to discrete kernels defined on graphs is non-trivial and yet to be attempted. As graph data can exhibit various spectral profiles, the shape of the spectral function can be made learnable as part of the GP training, allowing the optimal kernel to

be found in a data driven manner, and thus having the capacity to predict graph signals with different levels of smoothness based on the data.

In addition to the graph Fourier transform, wavelet transforms have been developed to analyze graph signals at different scales [18, 19, 20, 21]. For semi-supervised classification, we generally work with larger graphs, and this means multi-scale patterns can appear more regularly. In the node feature data, this is characterized by the potential existence of one particular smoothness profile at node level, but a different profile at cluster level, clusters of clusters and so on; one can imagine this as zooming out on the graph. The uniqueness of the wavelet transform is in providing a localization trade-off between the graph spatial and frequency domains. Whereas one would make use of fixed-hop neighbourhood information in typical spatially defined models, using wavelets leads to continuously weighted neighbours, and with the scale of the localization controllable by the user. Concurrently, multi-scale properties are reflected in the graph structure by the existence of eigen-gaps in the eigenvalues of the Laplacian, or the spectrum. Wavelets meanwhile appear as band-pass filters in this domain, with the location of the peaks also controllable by the user, making them ideal for capturing the spectral patterns of each "cluster of eigenvalues" that are close together. In the current graph learning literature, there have been very little attempts in using wavelets as an alternative to the graph Fourier basis in spectral-based GNNs [22, 23]. Furthermore, the basis was chosen to be a set of low-pass profiles only, which meant they did not take full advantage of wavelet's ability to handle higher frequency elements, and more importantly they do not capture any multi-scale patterns. Hence, for effective modelling on graphs, improvements can come from being able to incorporate higher frequency elements, as well as capturing multi-scale properties, and both features can be achieved by leveraging the qualities of wavelets that have previously been ignored.

The graphs in semi-supervised classification are often attributed (there exist feature data on the nodes), however the kernels used for graph GPs are currently limited to linear transformations of a non-graph base kernel, where the transformation is defined from graph operators (such as the adjacency matrix or wavelet transform). This corresponds to multiplying two kernels together, which along with adding kernels are generally simple but limited ways of combining prior information. Therefore, in designing graph GPs one may seek a direct kernel that takes into account both the graph and node feature data. Currently, kernels have been defined for feature data [24] and for graphs [25] separately from regularization theory, however no attempts have been made for attributed graphs that contain both sets of information. There is a need for such kernels as graph data generally come with node attributes.

In addition, the learning aspect of the model can be further improved by better utilization of the transductive properties that come with using kernels on graphs. This is different to the common inductive assumption, instead allowing the model access to not just the training set, but also the test and unlabelled inputs and the overall graph, with only the test labels not used during training. As it is often the case that training and testing data are both available from the start when working with graph data, one should look to leverage transductive models to make use of the distribution of the full dataset to better inform the complex and non-Euclidean patterns in the labels.

Finally, we look into modifying the graph with the goal to improve the model performance, this differs from the previous approaches where we assumed the input graph is fixed and unchanging. The given graph is generally useful but only up to a certain point, and will suffer from problems relating to over-smoothing and poor handling of heterophily. We therefore look into sheaves [26] as a higher order representation of graph data. Sheaves represent a new trend of lifting the input graph into a higher-order structure before operating on such object. Representing graph data as sheaves has been shown theoretically to lead to higher separation power [27], and thus any node classification problem can be thought of as finding the right sheaf with respect to the distribution of node labels. Meanwhile, graph GPs have yet to incorporate any modification of the graph during training, and therefore integrating sheaf learning offers an important step in this direction. The higher order information of the sheaf is represented by a set of restriction maps describing the relationship between each node-edge pair. Learning of such mappings has been shown previously to be possible through a neural network [27], we therefore adapt a similar approach into the GP training. In doing so, we are also integrating a neural network as part of the kernel learning for the GP, leveraging neural networks' ability to model non-linear relationships between data points while still maintaining a Bayesian model and its useful properties.

## 1.1 Technical Overview

The objective of this thesis is to develop novel models that better integrate graphs with Gaussian processes, and to demonstrate new approaches to building a kernel-based model for such data. To this end, the problems we tackle are presented as 4 technical chapters each demonstrating how to build GPs on graphs. Here we give a brief outline of each chapter:

1. **Gaussian Processes on Graphs via Spectral Kernel Learning**: We first aim to learn the optimal kernel on the graph [25] via learning the (discrete) Fourier dual, or the spectral function. We then use this kernel to predict vectorial responses that fall on the graphs, called graph signals, using kernels defined from the connectivity of the graph. This is used as part of a multi-output GP with another kernel taking in regression covariates to form the separable kernel for vectorial functions of [12]. We focus on finding the optimal spectral function of the kernel on the graph, using a flexible polynomial function defined in the spectral domain so that it can adapt to the data's characteristics. The main novelty of this work is not only to demonstrate the ability to learn a discrete kernel on graphs via its spectral function, but also by using a bespoke optimization algorithm, the polynomial is learnt in a constrained manner such that we can establish a connection with filtering on graphs. This provides a new form of interpretability, allowing the user to understand the model from a signal-processing perspective.

2. **Adaptive Gaussian Processes on Graphs via Spectral Graph Wavelets**: More common problems on graphs are semi-supervised and require dealing with node attributes. In this chapter we also work in this setting where the graph is partially labelled and we want to predict the remaining unlabelled nodes, with node attributes acting as feature data for the model. The limitations with many existing models on graphs are firstly the reliance on convolutions that only emphasize low frequency elements (such as the averaging in a GCN layer), as well as the inability to capture multi-scale patterns in the data. We find an efficient solution to both of these issues by utilizing wavelets on graphs. Spectrally, wavelets are band-pass filters with a set of scales to tune the location of the filter peaks. This translates to continuous neighbourhoods in the spatial domain where the weighting is determined by the scale. We found that many common graphs such as citation networks exhibit multiple scales, and thus using a set of wavelets complements the modelling of such data. Furthermore, the wavelets are made adaptive by learning a set of scales that best fit the data. Lastly for scalability, to avoid the expensive eigen-decomposition from the wavelet transform, a weighted polynomial of the graph Laplacian is used to approximate the transform matrix, allowing the model to handle larger datasets. As models on graphs generally ignore any multi-scale properties, this work provides a more custom handling of such information, and demonstrates a more extensive usage of wavelets on graphs.

3. **Transductive Kernels for Gaussian Processes on Graphs**: To treat attributed graphs correctly, one should realize that the node feature data fall on a different space to that of the graph. From theories on regularization and reproducing kernel Hilbert spaces (RKHSs), we find kernel functions are derived by first specifying a regularizer, followed by computing the inverse of this function. Typical kernels such as RBF, Matern, etc, can all be derived this way using a regularizer as a function of the Laplace operator. Furthermore it has been established in [25] that there is an equivalence between the Laplace operator and the graph Laplacian for continuous and discrete spaces respectively, thereby choosing regularizers with the graph Laplacian leads to kernels on (unattributed) graphs. In the often case when the graph has feature data on its nodes, there are now two pieces of information at play in different domain spaces. We therefore start with the sum of two regularizers - one dependent on the Laplace operator that corresponds to the feature data kernel on the nodes, and the other on the graph Laplacian and takes into account the graph connectivity. The resulting kernel is the inverse of the two regularizers. Evaluating this kernel between any two points will depend on the whole graph and all the node features, this includes both the training and testing data and any unlabelled nodes, hence the "transductive" property. We present this definition in the most generic form, and show that all current graph GP kernels correspond to an instance of our design. However what we proposed is unique in that it makes use of two regularizers while previous models only included one. Overall, this work provides a more unified definition of kernels for attributed graphs.

4. **Sheaf Laplacian Gaussian Processes**: The last piece of work goes in the direction of modifying the input graph as part of the GP training for semi-supervised classification. Instead of working on the given graph, we lift the data into a sheaf, a structure that encodes higher order relational information. We specify the sheaf by equipping the graph with topological information, assigning vector spaces on the nodes and edges, called stalks, and restriction maps between each node-edge pair. The restriction maps act as describing the relationship between each node-edge pair, and determine the edge values in the sheaf. In the case when the edge stalks are the space of scalars, this leads to the one-dimensional sheaves which are the same dimension as the original graph Laplacian, and has a similar effect to attention on graphs [28]. We have so far established various GPs on graphs constructed from the graph Laplacian, making the choice of one-dimensional sheaves appropriate as the Laplacian can

be fed directly into any readily made graph GP kernel. The sheaf's restriction maps are learnt using a linear neural network layer, with weights inferred as part of the GP training. Through this procedure, our analysis found the homophily ratios of the labels are increased due to the sheaf assigning generally larger weights to edges connecting nodes with the same label, indicating that sheaf learning is steering the model towards the correct labels. Overall, this work contributes towards the literature of modifying the graph during training, and is a first attempt into building GPs on sheaves.

## 1.2 Publications and Statement of Originality

My work during this DPhil contributed to the following publications:

1. *Yin-Cong Zhi, Yin Cheng Ng, and Xiaowen Dong. Gaussian processes on graphs via spectral kernel learning.* Transactions on Signal and Information Processing over Networks*, 2022*
2. **Felix Opolka, *Yin-Cong Zhi, Pietro Liò, and Xiaowen Dong. Adaptive gaussian processes on graphs via spectral graph wavelets. In* International Conference on Artificial Intelligence and Statistics*, pages 4818–4834. PMLR, 2022*
3. *Yin-Cong Zhi, Felix L Opolka, Yin Cheng Ng, Pietro Liò, and Xiaowen Dong. Transductive kernels for gaussian processes on graphs.* arXiv preprint arXiv:2211.15322*, 2022*
4. The final technical chapter "Sheaf Laplacian Gaussian Processes" is in preparation.
5. (Omitted in this thesis) *Felix Opolka, Yin-Cong Zhi, Pietro Liò, and Xiaowen Dong. Graph Classification Gaussian Processes via Spectral Features. 2023*

The work in 1. was accepted at the peer-reviewed workshop GCLR 2021 `https://sites.google.com/view/gclr2021/accepted-papers?authuser=0`, and the full paper was accepted for publication in the journal *Transactions on Signal and Information Processing over Networks.*

The work in 2. was accepted at the *International Conference on Artificial Intelligence and Statistics* 2022.

The work in 3. was accepted for oral presentation at the *Workshop on statistical learning for LARge scale GRaphs (LARGR)* 2023 `https://statlearngraph23.sciencesconf.org/` and *Graph Signal Processing Workshop* 2023 `http://gspworkshop.`

`org/`. This will be submitted to *Signal Processing Letters* for publication.

The final technical chapter will also be sent to a workshop environment shortly.

Chapters 4 and 6 of this thesis will be based on the first and third publications respectively, and chapter 7 will be based on the fourth piece of work currently in preparation. These three works were predominantly produced by me, this includes the model design, writing and technical derivation, experiments, and any figures presented in the papers. The other authors contributed towards the discussion and improving the quality of the writing. The second publication is joint work with Felix Opolka, and we take equal contribution for the publication. This paper forms Chapter 5 of the thesis, and will contain both our contributions: Mine is in developing the model design and producing the synthetic results, while Felix Opolka developed the method for the scalability of the model, and produced the results on real world experiments. We contributed equally toward the writing of the paper, and producing the figures for our respective sections and the Appendix.

Finally, I also contributed to the work in 5. as second author, and this work has been submitted to *The Conference on Uncertainty in Artificial Intelligence (UAI)* 2023 and accepted for oral presentation at *Graph Signal Processing Workshop* 2023. The content of the paper was predominantly produced by Felix Opolkaa, I was part of the discussions of the model designs and helped with the writing of the Related Work section.

# Chapter 2

# Preliminaries

## 2.1 Graph Signal Processing

Signal processing on graphs is defined by its parallelism with signal processing over the continuous domain. The study focuses on the representation of signals through basis functions, transforms, and modifications such as filtering and de-noising. In classical signal processing, analysis is often carried out in both the time and frequency domains (also called spectral domain), and one can navigate between the two through the Fourier transform. The definition on a graph is analogous in that the interplay is between the spatial and spectral domains, with spatial replacing the typical time domain.

Signal processing on graphs can be derived through various perspectives, for instance, by starting from the graph Laplacian $\mathbf{L}$ and its spectral decomposition [32], or from a general graph shift $z$-transform [33, 34] typically based on the adjacency matrix $\mathbf{A}$. In this review we focus on the former as the graph Laplacian, a discrete operator, has many relations to Fourier analysis in the continuous domain. Through this, we can then explore the connections between models in the continuous and discrete domains.

The Fourier transform is a convolution of a signal or function with a basis, picking out the element in the function at each frequency. The commonly used Fourier basis $e^{-i\omega x}$ is chosen based on the Laplace operator $\Delta = \sum_i \frac{\partial^2}{\partial x_i^2}$, which provides a measure of smoothness of a function. The basis of $e^{-i\omega x}$ are the eigen-functions of this operator, and we can see this by $\Delta e^{i\omega x} = -\omega^2 e^{-i\omega x}$, where $\omega$ is Fourier variable.

When we move to the discrete domain, we consider first a regular lattice mesh, which can form the discrete approximation of the Euclidean space. This would correspond

to a regular graph $\mathcal{G}$ where the nodes have a uniform degree. The operator $\Delta$ can then be found by applying finite difference approximation, and this has been shown to lead to the graph Laplacian

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{2.1}$$

as demonstrated in [25], for diagonal degree matrix $\mathbf{D}$, and adjacency matrix $\mathbf{A}$ with entries from $\{0, 1\}$, where $\mathbf{A}_{ij}$ is 1 if the nodes $v_j$ and $v_j$ are connected and 0 otherwise. One can then obtain the same measure of smoothness on non-regular graphs using the same definition, as well as weighted graphs with $\mathbf{A}_{ij} = \alpha_{ij} \in \mathbb{R}^+$ for edge weight $\alpha_{ij}$ connecting node $i$ and $j$. Graphs can also be directed which means generally $\alpha_{ij} \neq \alpha_{ji}$, and this would lead to a non-symmetric Laplacian.

When deriving a Fourier basis on graphs, it is important to note the Laplace operator and the graph Laplacian are analogous

$$\Delta \equiv \mathbf{L}, \tag{2.2}$$

with the former governing the continuous space and the latter governing the discrete graph space. When we look to analyze functions and signals that fall on a graph, we need to exploit this equivalence to define the Fourier basis. Previously we take the eigen-functions of $\Delta$, but in the discrete case the basis becomes the eigenvectors of the graph Laplacian

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top. \tag{2.3}$$

Taking the graph Fourier transform then becomes applying the eigenvector matrix $\mathbf{U}$ to a function or signal on the graph $\mathbf{f}$ as

$$\mathbf{U}^\top\mathbf{f}, \tag{2.4}$$

and we can see this picks out the amplitude of each basis element within the function or signal. This operation also takes $\mathbf{f}$ into the graph spectral domain.

Much like the classic Fourier basis, an eigenvector $\mathbf{u}$ that corresponds to a larger eigenvalue $\lambda$ in $\mathbf{\Lambda}$ will appear less smooth on the graph, and we can evaluate this through the graph smoothness measure defined as $\mathbf{u}^\top\mathbf{L}\mathbf{u}$. Signals can therefore be divided into exhibiting low-, band-, and high-pass profiles, characterized by the signal consisting mainly of eigenvectors corresponding to small, medium, and large eigenvalues respectively. Moreover this presents a clear path to define filtering on a

graph, which are important operations in graph signal processing.

A filter is a function defined in the spectral domain, chosen by the user to reduce or amplify certain frequencies when applied to a signal. This corresponds to a function on $\omega$ in classical filtering, and in the graph case this becomes the eigenvalues $\lambda$ of the graph Laplacian. Because the basis eigenvectors consistently become less smooth as eigenvalues increase, one can specify the desired spectral profile by using a filter that is low-, band- or high-pass shaped. A filtered graph signal is then computed as

$$\hat{\mathbf{f}} = \mathbf{U}g(\Lambda)\mathbf{U}^\top\mathbf{f} \tag{2.5}$$

where $g$ determines the profile of the filter, and $\mathbf{U}$ is the inverse graph Fourier transform returning $g(\Lambda)\mathbf{U}^\top\mathbf{f}$ back to the spatial domain. The graph filter matrix therefore refers to the following terms

$$\mathbf{U}g(\Lambda)\mathbf{U}^\top. \tag{2.6}$$

Occasionally, this can be expressed as a function of the graph Laplacian directly, and one would then skip the representation in the graph spectral domain, for example $g(\Lambda) = \exp(-\alpha\Lambda)$ would lead to $\mathbf{U}g(\Lambda)\mathbf{U}^\top = \exp\{-\alpha\mathbf{L}\}$.

Lastly, we can define the normalized Laplacian as

$$\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}. \tag{2.7}$$

This has similar properties with the un-normalized Laplacian (2.1), as the Fourier transform and filtering using the eigen-basis of $\tilde{\mathbf{L}}$ can be interpreted in the same way. The difference is in the boundedness of the values in $\tilde{\mathbf{L}}$ as the eigenvalues $\lambda \in \mathbf{\Lambda}$ will now be in the range [0,2], because of this property the normalized version is often preferred in practical situations.

In the common case when we expect signals to be smooth, $\mathbf{f}$ should consist more of lower frequency elements, while higher frequencies are more likely attributed to noise and therefore one would wish to reduce. In this case the choice of $g$ would be a decreasing function - a low pass - such that the eigenvectors with larger eigenvalues are reduced more in magnitude.

Machine learning relating to networks and graphs have become common problems within the community, with the more irregular structure of the data posing many new challenges. Graph signal processing provides many tools to better handle graphs, the concept introduced in this section will be frequently used to build more

effective models.

The recent review of [35] has highlighted the many applications of graph signal processing for modelling and prediction purposes. In particular, filtering and frequency analysis on the graph has been applied to multi-response regression, graph classification, and unsupervised learning of clustering and dimensionality reduction. Furthermore, defining models in the graph spectral domain provides interpretability from a filtering perspective, allowing the user to understand the smoothing operations acting on the data and in the predictions. We will present some examples when used in conjunction with Gaussian processes in Chapter 3.

Lastly, another topic of graph signal processing related to the work in this thesis is the study of graph stationarity. Although not explicitly stated, the graph data we use are assumed to be stationary, and this ensures certain models are suitable for the prediction tasks. Succinctly, it can be shown that a random graph signal is stationary on $\mathbf{L}$ if its covariance can be written as a polynomial of $\mathbf{L}$ [36, 37]. As we will see in our reviews, the class of graph GPs that are of particular interest are those whose kernels, and as a result the covariances, are polynomials of $\mathbf{L}$, making them suitable for modelling graph signals.

## 2.2 Gaussian Processes

Gaussian processes (GPs) are powerful machine models known for their data efficiency and Bayesian properties when making predictions. One can interpret GPs as a distribution over functions with inference taking place directly in the functional space. As a Bayesian model, the prediction from a GP model takes the form of a full distribution, providing the added benefit of allowing the user to obtain uncertainties with the prediction.

We start with data of the form $(\mathbf{x}_i, y_i)$. The Bayesian approach dictates that a prior is assumed on the data beforehand, and this is taken as a Gaussian process due to its generality and closed form solutions to conditioning. More importantly, the Gaussian process is fully specified by the second order moment, and thus one only needs to choose the covariance function to determine the prior. In general, a mean of 0 can be assumed in the prior as the inference step in computing the posterior first de-means the prior $(\mathbf{y} - \mu(\mathbf{X}))$ (as we will see later), and a 0 mean would lead to a simplified formula. The key is therefore to find the covariance between the data a-priori, and this can be specified by a powerful family of functions called kernels.

For a given kernel $k$, the prior is expressed as the infinite dimension GP as

$$f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \quad \forall \mathbf{x}, \mathbf{x}', \tag{2.8}$$

and we model the input-label pairs as $y_i = f(\mathbf{x}_i) + \epsilon_i$ where $\epsilon_i$ is the noise term in the model. The prior on the full dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ is then a subset of the Gaussian process with the following distribution

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left( \begin{pmatrix} \mu(\mathbf{X}) \\ \mu(\mathbf{X}_*) \end{pmatrix}, \begin{pmatrix} k(\mathbf{X}, \mathbf{X}) & k(\mathbf{X}, \mathbf{X}_*) \\ k(\mathbf{X}_*, \mathbf{X}) & k(\mathbf{X}_*, \mathbf{X}_*) \end{pmatrix} + \sigma_\epsilon^2 \mathbf{I} \right), \tag{2.9}$$

for training set $(\mathbf{X}, \mathbf{y})$, while $\mathbf{f}_*$ is the GP prior at test points $(\mathbf{X}_*, \mathbf{y}_*)$, and the noise term $\boldsymbol{\epsilon}$ has independent covariance $\sigma_\epsilon^2 \mathbf{I}$. The choice of $k$ is determined by the user, while this kernel may also have optimal parameters $\boldsymbol{\theta}$ to be found which we refer to as hyperparameters. These are inferred from the training data, and this procedure is referred to as the training stage of the GP. The hyperparameters are optimized with respect to the marginal log-likelihood

$$\log \mathbb{P}(\mathbf{y}|\boldsymbol{\theta}) = -\frac{1}{2} \log |\mathbf{K}_{\boldsymbol{\theta}}| - \frac{1}{2} (\mathbf{y} - \mu(\mathbf{X}))^\top \mathbf{K}_{\boldsymbol{\theta}}^{-1} (\mathbf{y} - \mu(\mathbf{X})) - \frac{1}{2} \log(2\pi). \tag{2.10}$$

for $\mathbf{K}_{\boldsymbol{\theta}} = k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I}$. This can be interpreted as how likely the prior can predict the training data, therefore we want to maximize this function with respect to $\boldsymbol{\theta}$. We will omit $\boldsymbol{\theta}$ in the notation from here onward for simplicity.

The Gaussian process then learns the pattern of the data by computing the posterior distribution for the test data by conditioning on the training. This distribution provides the user with a point prediction from the mean, and confidence intervals that can be constructed from the covariance. Conditioning on the training data $(\mathbf{X}, \mathbf{y})$, the posterior distribution of $\mathbf{f}_*$ for predicting at test points $\mathbf{y}_*$ is

$$\mathbb{P}(\mathbf{f}_*|\mathbf{y}) \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \tag{2.11}$$

where the posterior parameters have closed form solutions of

$$\boldsymbol{\mu}_* = \mu(\mathbf{X}_*) + \mathbf{K}_*^\top \mathbf{K}^{-1} (\mathbf{y} - \mu(\mathbf{X})) \tag{2.12}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^\top \mathbf{K}^{-1} \mathbf{K}_* \tag{2.13}$$

for $\mathbf{K} = k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I}$, $\mathbf{K}_* = k(\mathbf{X}, \mathbf{X}_*)$, and $\mathbf{K}_{**} = k(\mathbf{X}_*, \mathbf{X}_*)$

As one can see, the predictive distribution is heavily reliant on the covariances obtained through the kernel function. Thus, the choice of the kernel determines the

performance of the model, and designing a GP boils down to designing an effective kernel as part of the prior.

### 2.2.1 Variation Inference and Classification with Gaussian Processes

In most regression settings, the inference step of the GP can be computed by the closed form solution of (2.12) and (2.13), however this only applies when a Gaussian likelihood is assumed on the data and we first show where that enters the model. Before computing the posterior $\mathbb{P}(\mathbf{f}|\mathbf{y})$ we start with the joint distribution of the GP $\mathbb{P}(\mathbf{f}, \mathbf{y})$, here, $\mathbf{f}$ represents the GP at any general point in the input domain

$$\mathbb{P}(\mathbf{f}, \mathbf{y}) = \mathbb{P}(\mathbf{y}|\mathbf{f})\mathbb{P}(\mathbf{f}). \tag{2.14}$$

Here, $\mathbb{P}(\mathbf{f})$ is the prior chosen by the user, while $\mathbb{P}(\mathbf{y}|\mathbf{f})$ is the likelihood and this is assumed to be

$$\mathbb{P}(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma_\epsilon^2 \mathbf{I}). \tag{2.15}$$

As a product of two Gaussians, this means $\mathbb{P}(\mathbf{f}, \mathbf{y})$ will remain Gaussian, and a direct result of this is that $\mathbb{P}(\mathbf{f}|\mathbf{y})$ can be computed using the Gaussian conditioning formulae of (2.12) and (2.13).

More generally, two problems can arise in the GP inference step: 1. If a non-Gaussian likelihood is assumed on the data, for classification tasks for example, then the posterior will be analytically intractable, which means the formulae of (2.12) and (2.13) do not apply. 2. The inference step will be computationally costly if the number of training data $N$ becomes large as it requires an expensive $\mathcal{O}(N^3)$ matrix inversion. Both problems can be addressed by approximating the posterior through a variational approach: first, a set of inducing points $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_M]^\top$ are introduced and form the inducing random variables $\mathbf{u} = [f(\mathbf{z}_1), \ldots, f(\mathbf{z}_M)]^\top$ that is a subset of the GP $f(\mathbf{x})$. Assuming the GP prior of $\mathbf{u} \sim \mathcal{N}(0, \mathbf{K_{zz}})$ where $[\mathbf{K_{zz}}]_{ij} = k(\mathbf{z}_i, \mathbf{z}_j)$, the conditional GP has the following distribution

$$f(\mathbf{x})|\mathbf{u} \sim \mathcal{GP}(\mathbf{k_{zx}}^\top \mathbf{K_{zz}^{-1}} \mathbf{u} \ , \ \mathcal{K}(\mathbf{x}, \mathbf{x}) - \mathbf{k_{zx}}^\top \mathbf{K_{zz}^{-1}} \mathbf{k_{zx}}) \tag{2.16}$$

where $\mathbf{k_{zx}}$ are the cross covariances $[\mathcal{K}(\mathbf{z}_1, \mathbf{x}), \ldots, \mathcal{K}(\mathbf{z}_M, \mathbf{x})]^\top$. The variational posterior distribution $q(\mathbf{u})$ is assumed to be a multivariate Gaussian with mean $\mathbf{m}$ and covariance matrix $\mathbf{S}$ to be found through maximizing the Evidence Lower Bound

(ELBO)

$$\mathcal{L}(\theta, \mathbf{Z}, \mathbf{m}, \mathbf{S}) = \sum_{n=1}^{N} \mathbb{E}_{q(f(\mathbf{x}_n))}[\log \mathbb{P}(y_n|f(\mathbf{x}_n))] - \mathrm{KL}[q(\mathbf{u})||\mathbb{P}(\mathbf{u})]. \tag{2.17}$$

Typically we find a lower triangular matrix $\mathbf{H}$ such that $\mathbf{H}\mathbf{H}^\top = \mathbf{S}$, to ensure the variational covariance is symmetric and reduce the number of parameters that need to be found. The terms in $\mathbf{H}$ and $\mathbf{m}$ are found freely, maximized with respect to the ELBO (2.17) via gradient based optimizers such as Adam and L-BFGS-B.

With the use of inducing variables, the $\mathcal{O}(N^3)$ computational runtime can be reduced to $\mathcal{O}(NM^2)$ where $M$ is the number of inducing variables. This means the number of data points $N$ still affects the overall complexity, and at high volumes can still make inference infeasible. The choice of $M$ is down to the user, but generally $M$ needs to grow with $N$ to ensure high quality approximation [38]. Variational GPs will also contain additional variational parameters $\mathbf{m}$ and $\mathbf{S}$, leading to a further $\mathcal{O}(M^2 + M)$ of memory usage. We refer readers to Chapter 3 in [3] and [39] for a more comprehensive overview of GPs for classification.

In this thesis, we will be making use of GPs for classification at various points, and the likelihood function that will be used when tackling such problems is the robust maximum likelihood: assuming there are $n$ GPs $f_{i1} \ldots, f_{in}$ for the $n$ classes associated with the label $y_i$, these are evaluated as

$$\mathbb{P}(y_i|f_{i1} \ldots, f_{in}) = \begin{cases} 1 - \eta & \text{if } i = \operatorname{argmax}(f_{i1} \ldots, f_{in}) \\ \frac{\eta}{n-1} & \text{otherwise} \end{cases} \tag{2.18}$$

where $\eta$ is chosen to be a small value (0.001 in GPflow [39]). $\mathbb{P}(\mathbf{y}|\mathbf{f})$ is then the product of $\mathbb{P}(y_i|f_{i1} \ldots, f_{in})$ over $i$. The robust maximum likelihood has become the preferred option (than for example the *softmax*) when working with GPs, and this is also observed in our experiments when we test on real world data.

## 2.3 Kernels and Regularization

Many machine learning problems can be formulated as a minimization problem of a loss function and a potential regularization function, or regularizer. The regularizer is desirable as it enforces certain structures on the model to prevent problems such as over-fitting. Nowadays, it is more common to find the solution to the minimization problem numerically, the approach in this section however goes towards finding the

solution analytically, and describes the procedure for deriving kernel functions.

We start in the continuous domain, as described by [24], to find a model $f$ to predict the responses $\mathbf{y}$. As a minimization problem this can be written generally as

$$\min_f \quad \text{loss}(f, \mathbf{y}) + \Omega(||f||^2) \tag{2.19}$$

for a loss function with respect to the problem, between model $f$ as a function of input $\mathbf{x} \in \mathbf{X}$, and the labels $\mathbf{y}$. We consider the regularizer to be functions that can be represented by an inner product

$$\Omega(||f||^2) = \langle Pf, Pf \rangle = \langle f, r(\Delta)f \rangle = \langle f, f \rangle_{\mathcal{H}}, \tag{2.20}$$

where the Hilbert space $\mathcal{H}$ represents the space where the distance function is determined by the inner product specified by either $P$ or a function of the Laplace operator $r(\Delta)$. The operator $P$ is a collection of smoothness measures on $f$, with a few examples including:

1. $P = 1$, this would correspond to $||f||^2$ which is similar to the ridge regression.
2. $P = \nabla = [\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_m}]^\top$ which enforces the gradients of $f$ to be minimized, providing a stronger smoothing operator than 1.
3. $P = (1, \nabla)^\top$, called splines that combine various smoothness measures.

Due to $\nabla \cdot \nabla = \Delta = \sum_i \frac{\partial^2}{\partial x_i^2}$ and $\langle Pf, Pf \rangle = \langle f, P^\top P f \rangle$, if $P$ contains only gradients operators $\nabla$ we can combine $P^\top P$ into a function of $\Delta$, and the Hilbert space $\mathcal{H}$ can be characterized by the function $r(\Delta)$ instead. We will specify to the Hilbert space $\mathcal{H}$ by $r(\Delta)$ from this point onward.

Next, we first present a definition followed by a theorem that will be important for deriving kernels functions [40]:

**Definition.** *(Reproducing Kernel Hilbert Space): Let $\mathcal{H}$ be a Hilbert space, $\mathcal{H}$ is also a Reproducing Kernel Hilbert Space (RKHS) if there exists a kernel $k$ such that:*

- *$k(\mathbf{x}, \cdot)$ is a function in $\mathcal{H}$ for all $\mathbf{x}$.*
- *$\langle f, k(\mathbf{x}, \cdot) \rangle_{\mathcal{H}} = f(\mathbf{x})$ for all $\mathbf{x}$ and $f$ (reproducing property).*

**Theorem.** *(Representer Theorem): If $\mathcal{H}$ is an RKHS, then there exists a solution to (2.19) that is a linear combination of the kernel*

$$f^*(\mathbf{x}') = \sum_i w_i k(\mathbf{x}_i, \mathbf{x}'). \tag{2.21}$$

This theorem tells us that if we can find a kernel that makes $\mathcal{H}$ an RKHS, then the solution to (2.19) boils down to finding a set of linear weights $w_i$, significantly reducing the problem complexity. Such kernel is required to meet the two properties of the definition of RKHS, the first is trivially satisfied by the choice of the space, so we focus on the more important reproducing property, which can be written more explicitly as follows

$$\langle f(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\rangle_{\mathcal{H}} = \langle f(\mathbf{x}), r(\Delta)k(\mathbf{x}, \mathbf{x}')\rangle \tag{2.22}$$

$$= \int f(\mathbf{x})r(\Delta)k(\mathbf{x}, \mathbf{x}')\ d\mathbf{x} = f(\mathbf{x}'). \tag{2.23}$$

If we look at the terms inside the integral, the above problem is equivalent to finding $k(\mathbf{x}, \cdot)$ such that $k(\mathbf{x}, \mathbf{x}')r(\Delta) = \delta(\mathbf{x}-\mathbf{x}')$ where $\delta$ is the Dirac delta function. Finding $k$ therefore boils down to finding the Green's function, and the solution is generally written as $k(\mathbf{x}, \mathbf{x}') = r^{-1}(\Delta)_{(\mathbf{x}-\mathbf{x}')}$, or in matrix format

$$\mathbf{K} = r^{-1}(\Delta), \tag{2.24}$$

where the function $r$ on the operator is inverted. We note that $r(\Delta)$ is a function of an operator, and therefore does not depend on any variable, however the function $r^{-1}(\Delta)$ does depend on the variable $\mathbf{x}$, and is evaluated at $\mathbf{x} - \mathbf{x}'$.

Lastly, we present details on how Green's functions can be solved generally as well as an example to further illustrate the steps:

First we note that $\Delta e^{-i\omega\mathbf{x}} = -\omega^2 e^{-i\omega\mathbf{x}}$, and more generally we can write $r(\Delta)e^{-i\omega\mathbf{x}} = g(\omega)e^{-i\omega\mathbf{x}}$ where $g$ represents the function on $\omega$ induced by $r(\Delta)$. The Green's function is found by taking the inverse Fourier transform of the reciprocal of $g$ as described in [24]:

$$r^{-1}(\Delta)_{\mathbf{x},\mathbf{x}'} = \int_{-\infty}^{\infty} \frac{1}{g(\omega)} e^{i\omega(\mathbf{x}-\mathbf{x}')}\ d\omega = k(\mathbf{x}, \mathbf{x}'). \tag{2.25}$$

To illustrate further, we show an example derivation below for the Gaussian regularizer

$$r(\Delta) = \sum_{i=0}^{\infty} \frac{1}{i!}\left(\frac{\sigma^2\Delta}{2}\right)^i \equiv \exp\left\{\frac{\sigma^2\Delta}{2}\right\}. \tag{2.26}$$

This leads to $g(\omega) = \exp\{\sigma^2\omega^2/2\}$, and following suit we then compute the inverse

Fourier transform of the reciprocal

$$\int_{-\infty}^{\infty} \exp\left\{ -\frac{\sigma^2\omega^2}{2} \right\} e^{i\omega(\mathbf{x}-\mathbf{x}')}\, d\omega \propto \exp\left\{ -\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{x}')^2 \right\} \tag{2.27}$$

where we note the last term corresponds to the popular Gaussian RBF kernel. More examples of choices of regularizers and the kernels they lead to can be found in Appendix C.

### 2.3.1 Kernels and Regularization on Graphs

The regularization approach works with Hilbert spaces both continuous and discrete, and therefore one can also derive RKHSs that satisfy the same requirement in the discrete graph domain. To this end, deriving kernels has been translated to the graph domain in [25] by exploring the equivalence between the Laplace operator and the graph Laplacian.

Let $\mathbf{f}$ be a function or signal on the graph, we again start from the regularization problem of (2.19), but instead of $\Delta$ the regularizer is now replaced by a function of the graph Laplacian

$$\Omega(||\mathbf{f}||^2) = \langle \mathbf{f}, r(\mathbf{L})\mathbf{f} \rangle = \mathbf{f}^\top r(\mathbf{L})\mathbf{f} \tag{2.28}$$

where the discrete nature of graphs means the inner product becomes matrix multiplication instead of integration. Next, we again require the reproducing property, and we do this by looking at the $i$th element

$$[\mathbf{f}^\top r(\mathbf{L})\mathbf{K}]_i = \mathbf{f}_i \quad \forall i. \tag{2.29}$$

The $i$th element corresponds to the $i$th node in the graph, therefore this is similar to examining the reproducing property at point $\mathbf{x}$ in the previous section. If we look at the matrices as a whole, then we find that

$$\mathbf{f}^\top r(\mathbf{L})\mathbf{K} = \mathbf{f} \implies \mathbf{K} = r^{-1}(\mathbf{L}). \tag{2.30}$$

This corresponds to the continuous setting where we take the reciprocal of a function in the spectral domain before taking the inverse Fourier transform, thus, we can write the solution in a form aligned with the previous section as

$$\mathbf{K} = \mathbf{U} r^{-1}(\Lambda)\mathbf{U}^\top. \tag{2.31}$$

This kernel does have behavioural differences compared to the continuous kernels from the previous section, the most notable of which is its transductive nature. This can be seen more visibly when examining the kernel elementwise between two nodes $v_i$ and $v_j$, for which we find that $k(v_i, v_j)$ can only be obtained by computing the kernel over the full graph followed by taking the $(i, j)$th element

$$k(v_i, v_j) = [\mathbf{U} r^{-1}(\Lambda)\mathbf{U}]^{\top}_{ij}. \tag{2.32}$$

This means computing the kernel between any two nodes requires the whole graph, and the term $k(v_i, v_j)$ depends on all the nodes rather than the two input nodes only. This is an important difference to the inductive approach of typical GPs. Although inductive models are better at generalizing to new data, transductive models are more powerful predictors when both the training and testing data are available from the start. As we will see, this will play a big role in our technical chapters.

## 2.4 Connections with Other Models

Generally, kernels and regularization are related to several models from the previous sections, we briefly discuss these relationships in this section.

**Kernels and Filtering:** Firstly, observing equations (2.25) and (2.31), we notice that kernels consist of functions defined on the frequency variable (the eigenvalues), with the inverse Fourier transform applied to take the function to the time/spatial domain. Functions operating in the frequency domain have clear smoothness interpretations, with the larger Fourier variables corresponding to higher frequency elements. The example Gaussian regularizer of (2.26) is an increasing function, and therefore can be interpreted as a function that increases exponentially in the penalization of higher frequencies. The resulting kernel is the inverse of this penalization, which reduces high-frequency elements and thus mimics the nature of a low-pass filter. We further note that a kernel defined on the graph of (2.31) has the same form as graph filtering in equation (2.6). Hence, kernel-based modelling can be interpreted as a filtering system, the difference being filters enforce a certain smoothness on the data while the kernel measures it.

**Regularization and Gaussian Processes:** Secondly, as a kernel method, Gaussian processes also fall in the class of machine learning via regularization. Indeed, the training stage of the Gaussian process involves maximizing the marginal log-likelihood, but this can be easily turned into a minimization problem to match the regularization approach. We follow the derivation of Chapter 6 of [3] to get the loss

function and regularizer of the GP to be

$$\text{loss}(f) = \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))^2 + \frac{1}{2}||f||^2_{\mathcal{H}}. \tag{2.33}$$

This is based on taking the log of the Gaussian likelihood such that the exponential disappears. At a new test point $\mathbf{x}_*$ we know from Representer Theorem that

$$f^*(\mathbf{x}_*) = \sum_{i=1}^{N} w_i k(\mathbf{x}_i, \mathbf{x}_*). \tag{2.34}$$

Therefore, we can plug this into the loss function of $f$, noting that $\langle k(\mathbf{x}_i, \cdot), k(\mathbf{x}_j, \cdot)\rangle = k(\mathbf{x}_i, \mathbf{x}_j)$ from reproducing property, equation (2.33) becomes

$$\frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{K}\mathbf{w})^2 + \frac{1}{2}\mathbf{w}^\top\mathbf{K}\mathbf{w} = \frac{1}{2\sigma^2}\mathbf{y}^\top\mathbf{y} - \frac{1}{\sigma^2}\mathbf{y}^\top\mathbf{K}\mathbf{w} + \frac{1}{2\sigma^2}\mathbf{w}^\top\mathbf{K}^2\mathbf{w} + \frac{1}{2}\mathbf{w}^\top\mathbf{K}\mathbf{w}. \tag{2.35}$$

Finding $\mathbf{w} = (w_1, \ldots, w_N)^\top$ is now a quadratic problem which makes the solution much easier to find. In this case, we can solve for them by matrix differentiation with respect to $\mathbf{w}$ and equating to 0:

$$\frac{1}{\sigma^2}\mathbf{w}^\top\mathbf{K}^2 - \frac{1}{\sigma^2}\mathbf{y}^\top\mathbf{K} + \mathbf{w}^\top\mathbf{K} = 0 \tag{2.36}$$

$$\implies \mathbf{w}^\top(\frac{1}{\sigma_2}\mathbf{K}^2 + \mathbf{K}) = \frac{1}{\sigma^2}\mathbf{y}^\top\mathbf{K} \tag{2.37}$$

$$\implies \mathbf{w} = (\mathbf{K}^2 + \sigma^2\mathbf{K})^{-1}\mathbf{K}\mathbf{y} = (\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}. \tag{2.38}$$

Plugging $\mathbf{w}$ into (2.34), we get the solution

$$f(\mathbf{x}_*) = \mathbf{K}_*^\top(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \tag{2.39}$$

where we denote $\mathbf{K}_*^\top = (k(\mathbf{x}_1, \mathbf{x}_*), \ldots, k(\mathbf{x}_N, \mathbf{x}_*))$, and this we recognize to be the posterior mean of the Gaussian process.

One way to interpret this result is that adding the regularizer contributed to adding the noise covariance term $\sigma^2\mathbf{I}$ to the kernel matrix. This means the GP $f$ will not be forced to interpolate every data point, instead fitting the GP with the noise allows the posterior mean to be a small distance away from each point determined by the standard deviation of $\sigma^2$, this makes it possible to form a smoother shape compared to the un-regularized version.

# Chapter 3

# Literature Review

The work in this thesis will be predominantly compared with GP and kernel-based models on graphs, therefore, the focus of this chapter will be on the recent development of graphs GPs and how they are applied to various problems. In addition, we will also cover some common GNN models at the end of the chapter to provide a wider perspective of the most commonly used models on graphs.

## 3.1 Graph Signal Prediction

From a signal processing perspective, graphs generally act as the domain, while the values on the nodes form a signal or function that falls on this graph. Thus, one may be interested in predicting the evolution of signals, which models the change in the patterns over the whole graph. For example, a diffusion process with $\exp\{-t\mathbf{L}\}\mathbf{x}_0$ models the heat diffusion flow on a graph depending on the time $t$ starting from some initial condition $\mathbf{x}_0$, evaluating at each time step simulates how the concentration spreads as time increases. This is a simple example, but generally we do not know the underlying process and were only given a number of realizations of signals.

The data we have are of the form $(\mathbf{x}_i, \mathbf{y}_i)$, where each $\mathbf{y}_i \in \mathbb{R}^M$ is a signal on the graph $\mathcal{G}$ of size $M$, and in this section $\mathbf{x}_i$ will represent the regression covariates or input features. A training set of signals are given at $i = t_1, \ldots, t_N$, and the task then becomes predicting the signals at $t_{N+1}, \ldots, t_{N+M}$ given the previous signals.

This setting is a multi-response problem where the model outputs vectorial predictions, we will refer to the signals that we are predicting as the response variable. As the model needs to output a vector of values instead of a single label, handling of such data will require a multi-output GP, and henceforth to define the prior would

require kernels for vector valued functions presented in [12]. Of the various choices, the separable kernel has become more prominent due to the clean formulation and clear interpretation of each element

$$\tilde{\mathbf{y}} \sim \mathcal{N}(0, \mathbf{P} \otimes \mathbf{K}) \tag{3.1}$$

where $\tilde{\mathbf{y}}$ is the concatenation of a training set of responses $\mathbf{y}_1, \ldots, \mathbf{y}_N$ on the graph $\mathcal{G}$ into a single vector. To specify such prior, there is now the need to choose two kernels, $\mathbf{K}$ providing the covariance between responses $\mathrm{Cov}(\mathbf{y}_i, \mathbf{y}_j)$ computed by covariates $k(\mathbf{x}_i, \mathbf{x}_j)$, and $\mathbf{P}$ measuring $\mathrm{Cov}(\mathbf{y}_{ij}, \mathbf{y}_{ik})$ the $j$th and $k$th elements of each $\mathbf{y}_i$. In the general multi-response problem, the user will need to find a kernel for $\mathbf{P}$ based on prior knowledge. In the case of the graph signals, we know each signal $\mathbf{y}_i$ is governed by an underlying graph, and subsequently we can make use of this graph to provide the information needed to define the kernel. One example of this is [15], where $\mathbf{P}$ is chosen based on the graph filtering matrix

$$\mathbf{P} = (\mathbf{I} + \alpha\mathbf{L})^{-2} \tag{3.2}$$

while $\mathbf{K}$ continues to be the feature data kernel computed on the input covariates. The matrix $\mathbf{P}$ above follows the general form of kernels on graphs of [25] that act at the node level (it corresponds to Laplacian kernel raised to the power of 2, and noting that multiplying kernels will result in another valid kernel). A similar approach to solving this problem can also be found in [41] in the context of kernel regression, which is a frequentist counterpart to Gaussian processes where we do not compute any uncertainty with the prediction.

In both cases, the choice of $\mathbf{P}$ enforced smoothing properties in the model that may not necessarily suit the data. The predictions visualized in [41] appeared overly smooth, and this is mainly due to the low-pass nature of $\mathbf{P}$. Part of the work in this thesis will therefore go into addressing this issue to design a model that predicts with a smoothness level that matches the training signals.

## 3.2 Semi-Supervised Classification

Semi-supervised node classification has been the most popular task for machine learning on graphs, in this setting, one works with graphs that are partially labelled and the task is to predict the remaining unlabelled nodes. This form of learning on graphs becomes semi-supervised, where training and testing nodes are connected and during the training stage the model also has access to the test information. Im-

portantly, the model does not get given the test labels, so the prediction step is still comparable to the typical supervised learning framework. In tackling these problems, many creative solutions have been proposed in the GNN literature, however building GPs on graphs is less explored and has yet to demonstrate the capacity to handle graphs that are less smooth or heterophilic. Having well designed graph GPs is still important to provide a different outlook to the use of neural networks, as they offer many advantages over GNNs such as superior learning on fewer data, confidence intervals with the predictions, and interpretability.

In semi-supervised problems, a graph $\mathcal{G}$ is given by the node and edge sets $(\mathcal{V}, \mathcal{E})$, and each node $v_i \in \mathcal{V}$ has associated nodes features and label $(\mathbf{x}_i, y_i)$. During training, the user will have access to the full graph, or $\mathcal{E}$, the set of training nodes $v_1, \ldots, v_N$ and their features and labels $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, as well as the testing node features $\mathbf{x}_{N+1}, \ldots, \mathbf{x}_{N+M}$. The aim is to use all the information to infer the labels of the set of testing nodes $v_{N+1}, \ldots, v_{N+M}$.

**Spatial Transformation of the Kernel Matrix:** Combining graph information with the GP is first achieved by applying a linear matrix transformation of an initial GP prior [42], where the prior alone is independent of the graph. The advantage of using a linear transformation on a Gaussian is that the resulting prior remains Gaussian, and so the subsequent inference steps will not change. The linear approximation is applied as follows

$$\mathbf{B}f \sim \mathcal{N}(0, \mathbf{B}\mathbf{K}\mathbf{B}^\top) \tag{3.3}$$

for a Gaussian process $f \sim \mathcal{N}(0, \mathbf{K})$ with kernel matrix $\mathbf{K}$ computed on the node feature data only. The matrix $\mathbf{B}$ can be interpreted as part of the prior chosen by the user; in this context it will come from graph information, and is chosen in [42] and later in [43] to be

$$\mathbf{B} = (\mathbf{I} + \mathbf{D})^{-1}(\mathbf{I} + \mathbf{A}) \tag{3.4}$$

for adjacency matrix $\mathbf{A}$ and diagonal degree matrix $\mathbf{D}$. This operator collects each node's 1-hope neighbourhood from the graph, averaging the data over this group of nodes to allow the neighbours to influence the label given to the central node, identity is added to include the central node itself in the averaging.

This resembles the graph convolutional network (GCN) [44] where the averaging operation is applied over the 1-hop neighbours in each layer of the neural network. The 1-hop neighbourhood choice is effective despite the simplicity due to the ho-

mophily assumption on the graph, this is determined by the homophily ratio [45] defined as

$$h = \frac{|\{\{u, v\} : \{u, v\} \in \mathcal{E} \text{ and } y_u = y_v\}|}{|\mathcal{E}|}. \tag{3.5}$$

This ratio calculates the proportion of edges that connect nodes of the same label, typically $h > 0.7$ is considered homophilic and $h < 0.3$ is heterophilic. The datasets tested on in [42, 44] have relatively high $h$, therefore node-level information is the most relevant within the one-hop neighbourhood, making (3.4) an effective choice. On heterophilic data, intuitively one can see the use of the graph in this manner becomes less appropriate, and our empirical analysis in this thesis also indicated that using (3.4) only led to mixed improvements. Meanwhile, going beyond the 1-hop neighbourhood does not always improve the performance, instead it can very quickly lead to over-smoothing problems: [42] found its assumption failed when the neighbourhood was increased to just 2-hop, while [44] showed degrading performances with more than 3 layers (here the number of layers corresponds to the number of hops of the neighbourhood in the convolution) and the neural network cannot be made deep. In both cases, the graph operators in-use are low-pass functions, and oversmoothing is due to these operators being applied repeatedly. Thus, the ability to deal with heterophilic data and avoid over-smoothing in the model remain as ongoing problems [45, 46].

Going beyond the 1-hop neighbourhood is generally required when taking on more irregular data, this can be heterophilic graphs, or having higher frequency elements in the spectral domain. More generally, the ability to aggregate larger neighbourhood information also indicates a more powerful and complete model. When a convolution operates on a bigger neighbourhood, there are more information for determining the class of each node, but at the same time more focused information will be lost by the averaging effect over a larger set of data points, converging all nodes to a global value that is harder to differentiate. Thereby, to utilize larger hop neighbourhoods in a way that will improve performance, there needs to be a balance between the information gained and lost.

A Spatial operator that takes into account multi-hop neighbourhood can be defined by applying a single hop matrix operator to the $n$th power, this can therefore be written generally as

$$\mathbf{S}_n = \left[(\mathbf{I} + \mathbf{D})^{-\frac{1}{2}}(\mathbf{I} + \mathbf{A})(\mathbf{I} + \mathbf{D})^{-\frac{1}{2}}\right]^n, \tag{3.6}$$

when $n = 1$ the effect is a similar operator to (3.4), but this definition has sometimes been the preferred choice in the GNN literature due to it being symmetric, and is more commonly referred to as graph convolution. These can be found in GNN layers, and more recently in the context of a GP, the work in [47] included multiple $\mathbf{S}_n$ to define

$$\mathbf{B} = (\tilde{\mathbf{S}}_1 + \cdots + \tilde{\mathbf{S}}_K) \tag{3.7}$$

such that

$$\tilde{\mathbf{S}}_n = \lambda_n \mathbf{S}_n + (1 - \lambda_n)\mathbf{I} \tag{3.8}$$

with parameters $\lambda_n \in [0, 1]$ to be optimized. Each $\lambda_n$ parameter controls the influence of the convolution up to $n$-hop neighbours, if the neighbourhood convolution over-smooths this can be overcome by setting $\lambda_n$ close to 0 to reduce the effect of the averaging at the $n$th power, leaving the identity function which will only highlight the central node's information. The over-smoothing problem that comes with a larger $n$ is reduced by a suitably small $\lambda_n$, which is optimized as part of the GP training. This method does not fully solve the problems associated with using larger hop neighbours, as it can only reduce the influence of a neighbourhood, rather than extracting the most relevant information for the model.

As a side note, the above graph GPs operate on the node-level, but once the prior kernel is defined, one can easily convert this to a model for predicting the existence of edges, which we call link prediction. The task then becomes one where given the location of a training set of edges, we aim to predict the existence of edges between given pairs of nodes. The technique to turn node-based kernels to edge-based was shown in [48] where after the prior kernel matrix is chosen, the kernel between two edges is defined as

$$k_e((\mathbf{x}_i, \mathbf{x}_j), (\mathbf{x}_k, \mathbf{x}_l)) = k(\mathbf{x}_i, \mathbf{x}_j)k(\mathbf{x}_k, \mathbf{x}_l) + k(\mathbf{x}_i, \mathbf{x}_l)k(\mathbf{x}_j, \mathbf{x}_k) \tag{3.9}$$

for a kernel between edges connecting $(v_i, v_j)$ with features $(\mathbf{x}_i, \mathbf{x}_j)$ and $(v_k, v_l)$ with features $(\mathbf{x}_k, \mathbf{x}_l)$.

Although link prediction is a different problem setting, they have important applications in the literature of learning graphs from data, which is an important topic of study that focuses on inferring graphs from data, in particular when there exist relational structures but the underlying graph is not available. We will refer readers to [49] for further reading on such studies.

**Kernels with GNN Architecture:** Learning the optimal kernel has also found success through incorporating neural network architectures in the likes of [50, 51]. In dealing with graph data, the graph convolution defined in the GCN [44] has also been adapted into a GP kernel learning. This has been demonstrated in [52], with the following kernel for node features $\mathbf{x}_i$

$$k(\mathbf{x}_i, \mathbf{x}_j) = k_c(V(\mathbf{x}_i), V(\mathbf{x}_j)) \tag{3.10}$$

$$V(\mathbf{x}_i) = \sigma\left(\mathbf{W} \sum_n \tilde{\mathbf{A}}_{in} \mathbf{x}_n\right) \tag{3.11}$$

where $V(\mathbf{x}_i)$ represents a graph convolutional layer where $\tilde{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \hat{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the operator often found in a GCN layer, such that $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ is the degree matrix from $\tilde{\mathbf{A}}$. $\mathbf{W}$ is the neural network weights shared for every $V(\mathbf{x}_i)$, and they become hyperparameters to be found. In doing so, the kernel between any two node features is also dependent on the neighbouring features. Even though the addition of neural network architectures will generally increase the flexibility of the model, the convolutions utilized here are still limited to local neighbourhood averaging that relies on a homophilous assumption. On heterophilic graphs, it is still unclear how this design will behave when trained in conjunction with a GP.

**Matern Kernels on Graphs:** Finally, another approach for node classification is from the direction of kernels on graphs derived from regularization theory that we detailed in Chapter 2.3. As a reminder, a node level kernel on graphs is derived through first choosing a regularization function based on the graph Laplacian $r(\mathbf{L})$. This is due to the equivalence of $\mathbf{L}$ and the Laplace operator $\Delta$ and how kernels can be derived from a regularizer $r(\Delta)$ in the continuous domain. For instance, the diffusion kernel on graphs [25] obtained from $r(\mathbf{L}) = \exp\left\{\frac{\sigma^2}{2}\mathbf{L}\right\}$ is the equivalent of $r(\Delta) = \exp\left\{\frac{\sigma^2}{2}\Delta\right\}$, which leads to the RBF kernel we showed in the example of (2.26).

In the same procedure of replacing $\Delta$ by $\mathbf{L}$, the Matern class of kernels has also been translated to the graph case [53]. Matern kernels have first been defined on manifolds [54] through a differential operator in terms of $\Delta$ that has similar interpretations as regularization. In translating to the graph space we simply use the following regularizer

$$r(\mathbf{L}) = \left(\frac{2\nu}{\kappa^2} + \mathbf{L}\right)^{\frac{\nu}{2}} \tag{3.12}$$

where in usual Matern kernel notations, $\kappa$ is the lengthscale, and $\nu$ is the smoothness

such that when $\nu \to \infty$ this converges to the diffusion kernel in the same way the continuous Matern kernel would converge to the RBF. Much like the continuous counterpart, this regularizer provides a less-smooth kernel compared to the diffusion that is commonly used, providing another option to choose from for kernel-based problems on graphs. However, an important limitation is the lack of use of node features, as this kernel only depends on the graph connectivity induced by the graph Laplacian. The node classification accuracy from using this kernel is therefore not as good as previous graph GPs that contain a node feature kernel, although they generally improved on the example kernels on graphs presented in [25].

An interesting study from [53] is on the variance of the Matern kernel, which also applies to Laplacian-based kernels on graphs in general. Due to the irregularity of the graph, the variances are altered and become non-constant for each node. This is noticeably different to continuous kernels where you would expect a constant value on the diagonal of the kernel matrix. The authors cited [55] in explaining the reason behind the irregular variances in graph kernels, stating that the node variances depend not only on the number of neighbours each node has, but more specifically on how quickly a random walk starting from each node will return to itself. So far, there are yet to be studies that analyze the quality of the variances from these kernels, and this can lead to a user not trusting the uncertainties produced. Therefore, finding a definition of kernels on the graph with regular variances across all nodes is still an ongoing problem, and consequently, kernels on graphs in the current state are less suitable for variance based applications such as Bayesian optimization.

With the exception of the Matern kernel on graphs, the methods presented in this section mainly revolved around spatial operations; the Matern kernel has spectral interpretations but lacks the use of node features. As a result, spectral-based models are relatively under-explored, even though they can provide the solutions to many problems current models suffer from. This thesis will focus more on spectral-based models, as well as other mechanisms such as multi-scale modelling, transductive learning, and graph modification, to provide different perspectives for tackling semi-supervised problems.

## 3.3 Graph Classification

Another use of GPs in a graph context is to classify graphs. In this setting each data point is a graph with potential node features, and one aims to find the label of a given graph. The data on the nodes are generally the same dimension for each dataset, but the individual graphs come in different sizes and this is the main challenge of this

problem as typical kernels cannot take in differently sized inputs directly. There is therefore a need to map the graphs into embeddings, or using additive kernels taking in parts of the graphs that are the same size. The novelty of such models is therefore in the integration of graph features within a GP kernel, while how the graphs are utilized and manipulated can sometimes provide transferable techniques to other learning tasks on graphs.

The data in this problem comes in the form of $(\mathcal{G}_1, y_1), (\mathcal{G}_2, y_2), \ldots, (\mathcal{G}_N, y_N)$ such that each $\mathcal{G}_i$ has label $y_i$, and one aims to predict the labels of a testing set of graphs $\mathcal{G}_{N+1}, \ldots, \mathcal{G}_{N+M}$. Each graph $\mathcal{G}_i$ will also have node features of the form $\mathbf{X}_i = (\mathbf{x}_{1i}, \ldots, \mathbf{x}_{P_i i})$ of size $K \times P_i$ for feature dimension $K$, and $P_i$ is the size of $\mathcal{G}_i$. Each graph may also have node labels $\mathbf{b}_i \in \mathbb{R}^{P_i}$, and depending on the design some classifiers may also utilize this information.

**Graph Kernels:** The most straightforward graph level kernels can be found in the surveys of [56, 57, 58]. One can find various definitions and designs from such surveys, the first ones we would like to highlight are additive kernels that rely on a base kernel $k_b$ taking in as inputs single node data from each graph, and the full kernel is defined by the sum over all possible node pairs from the two graphs:

$$k(G_1, G_2) = \sum_{i \in G_1} \sum_{j \in G_2} k_b(\mathbf{x}_i, \mathbf{x}_j). \tag{3.13}$$

A similar kernel can be defined for any potential edge data instead to measure the similarity between two graphs.

Another design we would like to mention is through forming a product graphs $\mathcal{G}_1 \times \mathcal{G}_2$ with the node set defined as

$$\mathcal{V}_\times = \{(v_1, v_2) \mid v_1 \in \mathcal{G}_1, v_2 \in \mathcal{G}_2 \text{ and } l(v_1) = l(v_2)\} \tag{3.14}$$

$$\mathcal{E}_\times = \{((v_1, v_2), (u_1, u_2)) \mid (v_1, u_1) \in \mathcal{G}_2, (v_2, u_2) \in \mathcal{G}_2\}, \tag{3.15}$$

where $l(v_i)$ is the node label of $v_i$. Each node in $\mathcal{G}_1 \times \mathcal{G}_2$ represents a pair of nodes from the two input graphs that have the same label, while two nodes in the direct product graph are connected if and only if the associated pairs of nodes are connected in $\mathcal{G}_1$ and $\mathcal{G}_2$. The intuition to such designs is based on random walks - when this is performed on the product graph it becomes equivalent to walks on the two input graphs over the same label, with the return times providing a measure of similarity. This kernel is computationally more expensive due to the size of the product graph, and hyperparameters are generally more difficult to tune depending

on the formulation of the kernel. We will refer readers to the Random Walk Kernels section in [57] for more details.

The additive kernels of (3.13) between nodes have a shortcoming in that the connectivity of each input graph is ignored. Thus, one potential extension is instead of defining kernels between nodes, a kernel can take "patches" as inputs instead. For a simple example, a regular grid graph can be defined for image data where each node is a pixel and the nodes are connected to their nearest neighbours. A patch is then selected as $\mathbf{x}^{[p]}$ to be the concatenation of data at node $p$ and its 1-hop neighbours similar to CNN architecture, while the additive kernel takes as inputs $\mathbf{x}^{[p]}$ and $\mathbf{x}^{[p']}$ as shown in [59]. When moving onto irregular graphs where each 1-hop neighbourhood is generally of different sizes, a convolution method with a coordinate system was proposed in [60] to transform the patches to that of the same size before passing into the kernel. An example is the polar coordinates of [61], and the number of bins chosen beforehand forms the dimension each patch is mapped to. However, generally choosing the most suitable coordinates system is difficult to determine, and this method also requires all input graphs to be the same size (but they can have different edges), and thus cannot be applied to common datasets of irregularly sized graphs.

**Weisfeiler-Lehman:** A more prominent kernel that has been used more recently is the Weisfeiler-Lehman (WL) graph kernels [62], derived from the WL-test for graph isomorphism [63]. The algorithm follows a node gathering and relabelling procedure: initialize a value at each node (using potential node attributes or choosing all 1s), collect neighbourhood labels into a superset, relabel, and compare the new labels between all nodes from two graphs. If the new node labels differ at any of the nodes then this indicates the two graphs are not isomorphic (the labelling and relabelling refer to a value assigned to each unique superset, they are used as input embeddings and not the target variable we aim to predict), and moreover the labels themselves provide an embedding that can then be fed into a kernel function to measure the level of similarity between the two graphs. In the kernel, the labelling from WL is mapped to a histogram through a hashing function before passing as the inputs to overcome the dimension differences. Fig. 2 in [62] provides detailed visualizations of the steps described. An extension is by using the Wasserstein distance between two sets of labels as shown in [64], as an alternative to comparing the histogram. Lastly, this algorithm is known to exhibit similarities to the message passing in GNNs such as [65, 66, 67], where these models are known to be at best as good as the WL test. The analysis on the connections between the WL test and GNNs has led to the graph isomorphism network (GIN) proposed in [68] where the design ensures it

is as powerful as the WL test, whereas the likes of GCN and GraphSAGE [69] are not. Such models are still reliant on the expressive power of WL, and therefore it is important to note the limitation of the WL test in that there exist graphs that WL cannot tell apart despite being clearly different. Generally, the WL test can conclude for certain that two graphs are not isomorphic, but if the test indicates isomorphism there is still the possibility that the two graphs are different.

**Multi-Scale Laplacian Kernel:** Previously mentioned approaches have a degree of locality when collecting neighbourhood information for each node, this is effective as local information generally are more discriminative, but larger neighbourhoods will behave more like the global mean and can also have important information. The final approach is therefore in operating at a multi-scale level, comparing the graphs based on subgraphs of increasingly larger sizes. The base kernel defined here takes graph Laplacians as inputs, and they are passed in as matrix norms in the graph Laplacian kernel [70]

$$k(\mathcal{G}_1, \mathcal{G}_2) = k_b(p_1, p_2) = \frac{|(\frac{1}{2}\mathbf{L}_1 + \frac{1}{2}\mathbf{L}_2)^{-1}|^{1/2}}{|\mathbf{L}_1^{-1}|^{1/4}|\mathbf{L}_2^{-1}|^{1/4}} \tag{3.16}$$

defined by distributions $p_1 \sim \mathcal{N}(0, \mathbf{L}_1^{-1})$ and $p_2 \sim \mathcal{N}(0, \mathbf{L}_2^{-1})$ (using pseudo-inverse of the Laplacian), and assuming the graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are sampled from the distributions. A Laplacian can be obtained for any subgraph of each $\mathcal{G}$, and passing this sub-Laplacian into the kernel defined above provides a measure at a different scale. The subgraphs are obtained firstly through local neighbourhoods around each node, followed by the union of neighbourhoods for each level up. This approach can potentially improve on the previous kernels as it acts both locally and more globally. However, the limitation is in the computational cost as there exist a large number of possible pairs of subgraphs, numerous approximations were proposed in the reference which we will refer readers to for more details.

## 3.4 Specialized Graph Gaussian Processes

Gaussian processes are known to be flexible machine learning models, and naturally they have also been applied to different graph applications and settings to the previously mentioned work. We present a few more examples here to demonstrate the variety of graph problems that can be solved using a GP.

**Spatio-Temporal GPs on Graphs:** Firstly, our previously mentioned signal prediction problem on graphs is closely connected to the spatio-temporal GPs [71]. An

example of such applications on graphs is [72], making use of kernels on graphs as part of the spatial modelling. This kernel corresponds to the definition of Section 2.3.1 and precisely equation (2.31), where we express the function in the graph spectral domain

$$\mathbf{K}_G = \mathbf{U}\text{diag}(r(\mathbf{\Lambda}))\mathbf{U}^\top. \tag{3.17}$$

A vectorial variable $\mathbf{x}$ will have this covariance if

$$\mathbf{x} = \mathbf{T}\mathbf{z} \quad , \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}) \quad , \quad \mathbf{T} = \mathbf{U}\text{diag}(r(\Lambda)^{\frac{1}{2}}). \tag{3.18}$$

Thus, we can define a multi-output GP on graphs using the $\mathbf{T}$ matrix as

$$\mathbf{z}_t = \mathbf{f}(\mathbf{x}_{t-1}) + \boldsymbol{\epsilon}_t \tag{3.19}$$

$$\mathbf{x}_t = \mathbf{T}\mathbf{z}_t \tag{3.20}$$

such that $\mathbf{f}$ is $M$ dimensional GP corresponding to the size of the graph with a chosen node feature kernel, making $\mathbf{z}_t$ also the same dimension. The temporal element is in the time dependent $\mathbf{x}_t$, and so the aim is in predicting the features $\mathbf{x}_{t+1}$ given the features at time $t$. The entries in $\mathbf{f}$ are assumed to be independent and identically distributed, with $\mathbf{T}$ enforcing the spatial structure to produce $\mathbf{x}_{t+1}$. This draws similarity to Section 3.1 where the same vectorial prior is use in $\mathbf{f}(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}')\mathbf{I})$ for feature kernel $k$, but the full covariance cannot be written in the compact separable form due to the recursive dependencies defined in (3.19) and (3.20).

**Evolving Graph GPs:** In the previous settings, there is the assumption that the graph is fixed and unchanging, and this is in place for all problems such as semi-supervised tasks, graph signal prediction, and spatio-temporal graph GPs. In real world situations it is possible that relationships and connections may change over time, the modelling should therefore be able to reflect on the change in information. To this end we take a look at the work of [73] on evolving graphs.

In a simple case, nodes are assumed to be moving particles reflected by a change in coordinates, and the edges are dependent on the Cartesian distance between nodes thresholded at some cut-off distance. The graph at the next time point will depend on the current time graph, plus some velocity function defining how the nodes change. This is described in the motion as

$$\mathbf{v}_{t+1} = \mathbf{v}_t + f(\mathcal{N}(\mathbf{v}_t)) \tag{3.21}$$

where $\mathbf{v}_t$ represents the node as coordinates at time $t$, and the authors chose the velocity change to be dependent on the set of 1-hop neighbours and the nodes itself $\mathcal{N}(\mathbf{v}_t)$. Writing $\Delta\mathbf{v}_t = f(\mathcal{N}(\mathbf{v}_t))$, we can then easily determine the joint distribution on the set $(\Delta\mathbf{v}_1, \ldots, \Delta\mathbf{v}_M)$ for graphs with $M$ nodes. From past graphs we can find the data pairs $\{(\mathbf{v}_i, \Delta\mathbf{v}_i)\}_{i=1}^N$ as model input and output, using a multi-output GP to predict the $\Delta\mathbf{v}_i$ we then find $\mathbf{v}_{i+1}$ using the above equation. Lastly, the kernel used in this problem is defined on sub-trees, which we will refer readers to the referenced paper [73] for more details.

**Multi-Task GPs:** Lastly, we would like to highlight an approach of [74], that focuses on a specific problem of vectorial regression when the response can have multiple readings. We start with a multi-output $N$ dimensional Gaussian prior that is currently independent of any graph or spatial structure

$$\mathbf{f}(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}')\mathbf{I}_N) \tag{3.22}$$

where $\mathbf{I}_N$ is the identity matrix of dimension $N$. When multiple of the above GPs are concatenated together as $\tilde{\mathbf{f}} = (\mathbf{f}_1, \ldots, \mathbf{f}_M) \in \mathbf{R}^{NM}$ the kernel becomes

$$\tilde{\mathbf{f}} \sim \mathcal{GP}(0, \mathbf{I}_M \otimes k(\mathbf{x}, \mathbf{x}')\mathbf{I}_N). \tag{3.23}$$

The distribution over a set of inputs $\mathbf{X} = (\mathbf{x}_i \ldots, \mathbf{x}_N)$ then becomes

$$\tilde{\mathbf{f}}(\mathbf{X}) \sim \mathcal{N}(0, (\mathbf{I}_M \otimes \mathbf{I}_N) \otimes \mathbf{K}). \tag{3.24}$$

One can induce spatial structures to this model through a matrix $\mathbf{B}$, some examples of such from a graph have been explained in Section 3.2. In this context, this is applied to $\mathbf{f}$ as

$$\mathbf{Bf} \sim \mathcal{N}(0, \mathbf{B}k(\mathbf{x}, \mathbf{x}')\mathbf{I}_N\mathbf{B}^\top) = \mathcal{N}(0, \mathbf{B}\mathbf{B}^\top k(\mathbf{x}, \mathbf{x}')) \tag{3.25}$$

leading to the full prior over inputs $\mathbf{X}$ as

$$\mathcal{N}(0, (\mathbf{I}_M \otimes \mathbf{B}\mathbf{B}^\top) \otimes \mathbf{K}). \tag{3.26}$$

This GP is a further extension of the multi-output GP to multi-tasks, to model signals that have multiple readings at each given time step $t$. For instance an example given in [74] is a distribution grid having readings for active power, reactive power injections, and voltage, and thus there are three sensors, each producing a graph signal at time $t$. In this case $\mathbf{I}_M$ will be chosen as the $3 \times 3$ matrix since $M = 3$. $\mathbf{I}_M$

can also be replaced by some correlation matrix that is known beforehand or can be computed between the multiple sources of signals.

## 3.5 Graph Neural Networks

Deep learning and neural network-based models have found success in many application domains such as language models, computer vision, recommender systems, and more. At the same time, graph neural networks (GNNs) is also a fast growing field that extends the use of deep learning to graph data, and can be applied to any of the problems presented in this chapter. We now briefly discuss some of the most commonly used GNNs in the literature.

The mechanics of GNNs involves information propagation through layers where node embeddings are convoluted with selected graph operators. Each layer also contains appropriate quantities of weights that are trained by minimizing a loss function, and these form the parameters that allow the model to fit towards the data.

**Spatial GNNs:** We write $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times D_l}$ to represent the node embeddings in the $l$th layer of the GNN for the number of nodes $N$ and embedding dimension $D_l$ in layer $l$, with $\mathbf{H}^{(0)} = \mathbf{X}^\top$ as the node features matrix in the input layer. The node embeddings of subsequent layers are computed recursively as

$$\mathbf{H}^{(l+1)} = f(\mathbf{H}^{(l)}, \mathcal{G}) \tag{3.27}$$

for some generally non-linear $f$ that contains neural network and graph operators. One of the most commonly used GNNs is the graph convolutional network (GCN) [44], which has the following setup

$$\mathbf{H}^{(l+1)} = f(\mathbf{H}^{(l)}, \mathcal{G}) = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}), \tag{3.28}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, and $\tilde{\mathbf{D}}$ is diagonal with $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$, identity matrix is added so that the convolution of each node embedding is over the neighbours and itself. The matrix $\mathbf{W}^{(l)}$ refers to the weights of layer $l$, which are trained through back-propagation, $\sigma$ is any non-linear activation function such as ReLU or tanh.

At node level, we can express the embedding $\mathbf{h}_i^{(l)}$ for each node $v_i$ as a row vector from $\mathbf{H}^{(l)}$ in layer $l$, and forward propagation of the GCN can more formally be

written as

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}\mathbf{h}_j^l\mathbf{W}^{(l)}\right) \tag{3.29}$$

where $\mathcal{N}_i$ is the set of neighbours of $v_i$ and the node itself.

A more flexible version of the GCN is graph attention networks (GAT) [28], where the learnable attention mechanism is added to the layer as

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l)}\mathbf{h}_j^{(l)}\mathbf{W}^{(l)}\right), \tag{3.30}$$

and the term $\alpha_{ij}^{(l)}$ can be interpreted as an importance weighting on the node embedding $\mathbf{h}_j^{(l)}$ to node $v_i$ in layer $l$. The attention weights are normalized through a softmax function over the possible neighbours of each node $v_i$ to get

$$\alpha_{ij}^{(l)} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^{(l)\top}[\mathbf{h}_i^{(l)}\mathbf{W}^{(l)}||\mathbf{h}_j^{(l)}\mathbf{W}^{(l)}]^\top))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\mathbf{a}^{(l)\top}[\mathbf{h}_i^{(l)}\mathbf{W}^{(l)}||\mathbf{h}_k^{(l)}\mathbf{W}^{(l)}]^\top))} \tag{3.31}$$

where $\mathbf{a}^{(l)} \in \mathbb{R}^{2D_l}$ is a vector of trainable weights, and $||$ represents concatenation of two vectors. This ensures the attention weights are well bounded and within [0,1].

**Spectral GNNs:** Convolution can also be carried out in the graph spectral domain as an alternative to the previously spatially defined GNNs. Spectral GNNs operate over the eigen-basis of the normalized graph Laplacian $\tilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, and learn a filter in the spectral domain that is then convoluted with the node embeddings. To demonstrate this, let $\mathbf{s}$ be a signal on $\mathcal{G}$, the signal can be convoluted with a filtering function $\mathbf{g}$ as

$$\mathbf{s} * \mathbf{g} = \mathbf{U}(\mathbf{U}^\top\mathbf{s} \odot \mathbf{U}^\top\mathbf{g}) \tag{3.32}$$

where $\odot$ represents elementwise product. Let $\hat{\mathbf{g}} = \mathbf{U}^\top\mathbf{g}$, the above then takes the form of filtering on graphs as

$$\mathbf{s} * \mathbf{g} = \mathbf{U}\text{diag}(\hat{\mathbf{g}})\mathbf{U}^\top\mathbf{s}. \tag{3.33}$$

This forms the convolution of a spectral GNN layer as demonstrated in [75] as

$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{U}\text{diag}(\hat{\mathbf{g}})\mathbf{U}^\top\mathbf{H}^{(l)}), \tag{3.34}$$

with the entries in $\hat{\mathbf{g}}$ becoming trainable parameters/weights instead. The choice of $\hat{\mathbf{g}}$ can be such that all its terms are freely optimized, or $\hat{\mathbf{g}}$ can be some chosen filtering function over the eigenvalues with a set of trainable parameters.

Spectral GNNs are limited in scalability due to the eigen-decomposition required to specify each layer, thus, ChebNet [76] has been proposed to overcome this issue by approximating the spectral convolution matrix with a Chebyshev polynomial of the graph Laplacian. Denoting $\bar{\mathbf{L}} = \tilde{\mathbf{L}} - \mathbf{I}$, a ChebNet layer is defined as

$$\mathbf{H}^{(l+1)} = \sigma\Bigg(\sum_{k=1}^{K-1} \theta_k T_k(\bar{\mathbf{L}})\mathbf{H}^{(l)}\Bigg), \tag{3.35}$$

for Chebyshev polynomial $T_k$, and $\tilde{\mathbf{L}} - \mathbf{I}$ is used due to the Chebyshev polynomial being restricted to the range [-1,1]. The trainable parameters then become the polynomial coefficients $\theta_k$, and each polynomial can be computed recursively as $T_k(\bar{\mathbf{L}}) = 2\bar{\mathbf{L}}T_{k-1}(\bar{\mathbf{L}}) - T_{k-2}(\bar{\mathbf{L}})$, with $T_0(\bar{\mathbf{L}}) = \mathbf{I}, T_1(\bar{\mathbf{L}}) = \bar{\mathbf{L}}$. As the Chebyshev polynomials act linearly on the Laplacian, this can still be interpreted as a filtering function on the eigenvalues of $\bar{\mathbf{L}}$ as $\mathbf{U}\sum_{k=1}^{K-1}\theta_k T_k(\bar{\mathbf{\Lambda}})\mathbf{U}^\top$ for shifted eigenvalues $\bar{\mathbf{\Lambda}} = \mathbf{\Lambda} - \mathbf{I}$.

Overall, the architecture of the GNN such as the number of hidden layers and layer dimensions (also called the hyperparameters of the neural network) are determined based on a separate validation set, with the combination that led to the best validation performance selected as the final model. Meanwhile, the final layer of the GNN is called the read-out layer, and it outputs embeddings of relevant dimensions depending on the problem at hand (scalar, vectorial, etc.). This forms the prediction of the model for which we can compute the loss against the training labels, we then proceed to minimize the loss function with respect to all trainable weights in the GNN. Examples of loss functions include MSE (continuous) and cross entropy (categorical).

# Chapter 4

# Gaussian Processes on Graphs via Spectral Kernel Learning

Studies in graph signal processing assume the graph is a domain for which signals fall on, and the graph Laplacian offers a basis for which one can apply smoothness analysis in the spectral/frequency domain. In a predictive modelling context, the GP can be applied to model the evolution of signals, while GP regression also has filtering interpretations making them the ideal model of choice. In this chapter we further establish the connections between regression, filtering, and defining kernel functions in the spectral domain, followed by demonstrating finding the optimal kernel on graphs by learning the spectral function.

## 4.1 Introduction

Graph signal prediction aims to make vectorial predictions that fall on a graph given some previously observed signals. Multi-output Gaussian processes (MOGP) are particularly suitable for graph data of this form as each signal can be viewed as a vector response where the dependency between elements is encoded in the graph structure. At the same time, the dependency between different graph signals can be modelled using a typical kernel on the input covariates (e.g., the squared exponential kernel). This leads to the formulation of separable kernels for MOGP, as is the case in the co-regionalization model in [12], and the two kernels can be designed separately and combined by means of a Kronecker product. We refer to the kernel operating on the inputs covariates as kernel on the input space, and the kernel operating on the output signals as node-level kernel, where the latter provides a measure of smoothness between data observed on the nodes.

Smola et al. [25] have introduced the notion of kernel on graphs, where kernel functions between nodes were derived from a regularization perspective by solving for a reproducing kernel Hilbert space (RKHS). The resulting kernel is based on the graph Laplacian, and this is closely related to graph signal processing, which makes use of tools such as graph Fourier transform and filtering [32, 34, 77]. One particular low-pass filter defined in [32], commonly used to denoise graph signals, also assumes the form of kernels on graphs. This was subsequently used in [15, 41] to construct a GP model on graphs for predicting low frequency signals. However, the filter as defined in [32, 15, 41] has only been demonstrated to perform well on low-pass signals, and modifications are required to adapt to band- or high-pass signals. The same limitation also applies to other existing GP models developed for graph-structured data such as [42], where the relationship between the node observations is defined a priori based on the assumption of low-frequency characteristics. Models will need to make use of high frequency information to better handle less smooth data such as heterophilic graphs [45, 78]. Addressing this limitation requires a different choice of kernels with a spectrum that better adapts to the characteristics of the data.

Learning kernels in the spectral domain have been studied in the continuous case such as [16, 79, 80], but the extension of the approach to a discrete graph space has yet to be explored. In this chapter, we propose a novel MOGP model for graph-structured data, which uses a kernel on graphs to measure node-level relationships in the data. We explicitly relate this kernel to a graph filter, which is used to obtain the target graph signals according to our generative model. Importantly, the frequency response of the filter (or the spectrum of the kernel) is learned by adapting to the data, thus making the resulting MOGP flexible in capturing different signal characteristics.

Our model constitutes several unique contributions to the literature: First, the model is designed to capture various graph signal structures by incorporating a flexible polynomial function in the graph spectral domain, producing a highly adaptable model. Second, the polynomial function is learned by maximizing the log-marginal likelihood while respecting a constraint to enforce the positivity of the spectrum. The positivity constraint allows for a meaningful interpretation of the learned models as graph filters, giving the users insights on the characteristics of the data. Finally, we demonstrate that our algorithm can recover ground truth filters applied to synthetic data, and show the adaptability of the model on real-world data with different spectral characteristics.

## 4.2 Background

### 4.2.1 Kernels for Vector Valued Functions

The tasks of predicting vectorial values will require the model $\mathbf{f}$ to be a multi-output function. Kernels for this type of functions are formulated by the Kronecker product of two kernels, one for the inputs and the other on the elements of $\mathbf{f}$. This is described as separable kernels in [12], where between any two inputs the function $f$ will have the following form

$$\text{Cov}(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}')\mathbf{H} \tag{4.1}$$

where $\mathbf{H}$ is of size $M \times M$ such that $M$ is the dimension of the output of $\mathbf{f}$. This matrix operates on the output elements of $\mathbf{f}$ and thus it is referred to as the kernel on the output space. When applied to $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top$, the matrix can be written in a compact manner through a Kronecker product

$$\mathbf{K}(\mathbf{X}, \mathbf{X}) = \mathbf{K} \otimes \mathbf{H} \tag{4.2}$$

with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. $\mathbf{K}$ is therefore referred to as the kernel on the input space.

### 4.2.2 Kernels and Regularization on Graphs

A property of kernel functions is provided by Bochner's theorem [81], which states that positive definite functions have non-negative measures as the spectrum in the spectral domain. On the discrete graph space, kernels are derived by the graph Fourier transform and a non-negative transfer function. In this section we briefly summarize the formulation of kernels on graphs described in [25].

The graph Laplacian is the discrete counterpart of the Laplace operator, therefore it has the property of quantifying the smoothness of a function on the graph [82, 83]. We briefly recap kernels and regularization on graphs first: when finding a smooth model $\mathbf{f}$ for graph signal $\mathbf{y}$, it is common to solve for the following regularized problem

$$\min_{\mathbf{f}} ||\mathbf{f} - \mathbf{y}||_2^2 + R(||\mathbf{f}||^2), \tag{4.3}$$

where we have the regularization function $R$ on $\mathbf{f}$. In the graph case, $R(||\mathbf{f}||^2) = \mathbf{f}^\top \mathbf{P} \mathbf{f}$ where $\mathbf{P}$ often takes the form of a penalty function of the graph Laplacian, i.e. $\mathbf{P} = r(\mathbf{L})$, that penalizes specific graph spectral components of $\mathbf{f}$. The kernel

function is then computed by $\mathbf{K} = \mathbf{P}^{-1}$ in order for Eq. (4.3) to have a representation in an RKHS, with Moore-Penrose pseudoinverse used if $\mathbf{P}$ is singular [84]. The solution of Eq. (4.3) then exists by the Representer Theorem [40]. Furthermore, this definition is flexible in that different variations of the Laplacian such as the normalized Laplacian

$$\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} \tag{4.4}$$

and scaled Laplacian

$$\mathbf{L}_S = \frac{1}{\lambda_{max}(\mathbf{L})}\mathbf{L} \tag{4.5}$$

will both lead to valid kernels.

## 4.3 Proposed Model

### 4.3.1 Gaussian Processes for Graph Signals

Consider data pairs of the form $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ where each output $\mathbf{y}_n \in \mathbb{R}^M$ is a signal on a graph $\mathcal{G}$ of $M$ nodes indexed by some input covariates $\mathbf{x}_n \in \mathbb{R}^C$. One way to generate data of this form is to consider $\mathcal{G}$ as a sub-graph of a bigger graph $\mathcal{G}_{\text{full}}$, and the values on the remaining nodes $\mathcal{G}_{\text{full}} \backslash \mathcal{G}$ are used as $\mathbf{x}_n$. For example, in the context of predicting traffic flow in a city, the network between the junctions will be $\mathcal{G}_{\text{full}}$ and we use the values at a fixed number of junctions as input $\mathbf{x}_n$ to predict the flow at the rest of the nodes used as the outputs $\mathbf{y}_n$. When predicting a new signal, this makes the assumption that if the traffic flows on two different days are similar on the input junctions then they will be similar at the output junctions. How each junction in $y_n$ behaves is then modelled by the sub-graph containing only the output junctions (which form $\mathcal{G}$). Other setups are also possible depending on the problem. We will refer readers to our experiments in Section 4.5.3 for details.

From a generative model perspective, we assume each $\mathbf{y}_n$ is a realization of a filtering system $\mathbf{B}\mathbf{f}(\mathbf{x}_n)$ where $\mathbf{B} \in \mathbb{R}^{M \times M}$ is the graph filter, and $\mathbf{f}(\cdot) \in \mathbb{R}^M$ is a simple MOGP function with independent components evaluated at $\mathbf{x}_n$ - the elements in $\mathbf{f}$ are assumed to be independent GPs with identical kernel function $k$ on any two inputs $\mathbf{x}_n$ and $\mathbf{x}'_n$. This leads to $\text{Cov}(\mathbf{f}(\mathbf{x}_n), \mathbf{f}(\mathbf{x}'_n)) = k(\mathbf{x}_n, \mathbf{x}'_n)\mathbf{I}_M$, where $\mathbf{I}_M \in \mathbb{R}^{M \times M}$ is the identity matrix. Graph information in $\mathbf{y}_n$ is therefore induced by the filtering

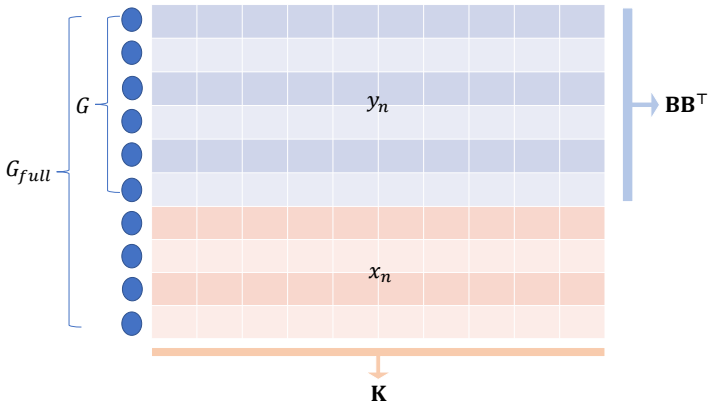



Figure 4.1: Illustration of graph data construction into input-output pairs for a GP. Each column in blue is a graph signal that indicates the value on the nodes, with a corresponding column in red of input covariate below.

matrix $\mathbf{B}$, giving rise to the following model

$$\mathbf{y}_n = \mathbf{B}\mathbf{f}(\mathbf{x}_n) + \boldsymbol{\epsilon}_n, \tag{4.6}$$

where $\boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \sigma_\epsilon^2 \mathbf{I}_M)$. The model in Eq. (4.6) is generic in the sense that, depending on the design of $\mathbf{B}$, we can incorporate any characteristics of the signal $\mathbf{y}_n$ in the graph spectral domain.

The prior covariance between two signals $\mathbf{y}_n$ and $\mathbf{y}_m$ can be computed as $\text{Cov}(\mathbf{y}_n, \mathbf{y}_m) = \mathbb{E}(\mathbf{y}_n \mathbf{y}_m^\top) = \mathbf{B}\mathbb{E}(\mathbf{f}(\mathbf{x}_n)\mathbf{f}(\mathbf{x}_m)^\top)\mathbf{B}^\top = k(\mathbf{x}_n, \mathbf{x}_m)\mathbf{B}\mathbf{B}^\top$, and if we let $\tilde{\mathbf{y}} = \text{vec}([\mathbf{y}_1, \ldots, \mathbf{y}_N])$, the covariance of the full data becomes

$$\text{Cov}(\tilde{\mathbf{y}}) = \mathbf{K} \otimes \mathbf{B}\mathbf{B}^\top + \sigma_\epsilon^2 \mathbf{I}_{MN}, \tag{4.7}$$

where $\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m)$, and $\otimes$ denotes the Kronecker product. The $\mathbf{B}\mathbf{B}^\top$ term can be thought of as a kernel between elements of each output $\mathbf{y}_n$, while $\mathbf{K}$ operates on the signals' corresponding inputs $\mathbf{x}_n$ and $\mathbf{x}_m$. Generally, $\mathbf{K}$ will be referred to as the input kernel, while we will call $\mathbf{B}\mathbf{B}^\top$ the node-level kernel.

We now state our main model for prediction of graph signals. Given the GP prior on $\mathbf{f}(\mathbf{x})$, the vectorized training signals $\tilde{\mathbf{y}}$ and test signal $\mathbf{y}_* \in \mathbb{R}^M$ with given input $\mathbf{x}_*$ follow the joint distribution

$$\mathbb{P}\left(\begin{bmatrix}\tilde{\mathbf{y}} \\ \mathbf{y}_*\end{bmatrix}\right) \sim \mathcal{N}\left(0\,,\, \begin{bmatrix}\mathbf{K} \otimes \mathbf{B}\mathbf{B}^\top & \mathbf{K}_* \otimes \mathbf{B}\mathbf{B}^\top \\ \mathbf{K}_*^\top \otimes \mathbf{B}\mathbf{B}^\top & \mathbf{K}_{**} \otimes \mathbf{B}\mathbf{B}^\top\end{bmatrix} + \sigma_\epsilon^2 \mathbf{I}_{M(N+1)}\right), \tag{4.8}$$

where $\mathbf{K}_* = (k(\mathbf{x}_1, \mathbf{x}_*), \ldots, k(\mathbf{x}_N, \mathbf{x}_*)^\top \in \mathbb{R}^N$ and $\mathbf{K}_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$. For the inputs,

the kernel $k$ can be any existing kernel such as the squared exponential or Matérn kernel. For node-level, we consider $\mathbf{B}$ as a kernel on graphs that is based on the scaled graph Laplacian of Eq. (4.5), and follows the general form in Eq. (2.31) as $\mathbf{B} = \sum_{i=1}^{M} g(\lambda_i)\mathbf{v}_i\mathbf{v}_i^\top = g(\mathbf{L}_S)$. From this point onwards, $\lambda_i$ and $\mathbf{v}_i$ correspond to the eigenvalues and eigenvectors of $\mathbf{L}_S$, and $g(\lambda)$ is the function in the scaled graph spectral domain.

It is worth noting that in choosing a non-negative $g$, the resulting $\mathbf{B}$ gives us two different interpretations of the model. From a kernel perspective $\mathbf{B}\mathbf{B}^\top$ forms the node-level kernel to measure similarities on the elements of each signal $\mathbf{y}_i$, and this corresponds to the kernel on the output space from separable kernels defined in [12]. From the filtering perspective, we identify that all kernels on graphs defined in [25] are of low-pass nature, and although all definitions of $g$ are non-negative, we find that this is restrictive and less suitable to data that does not exhibit smoothness or a low-frequency characteristic. For the model to become more adaptive, we propose to use a more flexible spectral function so that it can pick up on the likes of band- or high-pass data profiles.

### 4.3.2 Graph Spectral Kernel Learning

To achieve an increase in model flexibility, we use a polynomial for the function $g$, while we give the model the ability to adapt to the data by learning the polynomial shape as part of the training step. We parameterize $g$ as follows

$$g(\lambda) = \beta_0 + \beta_1\lambda + \cdots + \beta_P\lambda^P \implies \mathbf{B} = \sum_{i=0}^{P} \beta_i \mathbf{L}_S^i, \tag{4.9}$$

with coefficients $\beta_0, \ldots, \beta_P$ learned via log-marginal likelihood maximization. Learning of the coefficients is done by optimizing them as hyperparameters which we will go into more details in section 4.4.

There are a number of advantages of our model setup, in particular:

- The kernel on graphs is learned rather than chosen a priori, and the function that characterizes the kernel is a flexible polynomial making the model highly adaptable to data with different spectral properties. Moreover, existing choices provided in [25] all consist of functions that have polynomial expansions. Hence our model provides suitable approximations if data came from a more complex generative model.
- The application of the $P^{th}$ power of the Laplacian corresponds to filtering

restricted to the $P$-hop neighbourhood of nodes. Our polynomial is finite, thus the user can control the localization in the kernel, a property that is often desirable in graph-based models such as the graph convolutional network (GCN) [44].

- The scaled Laplacian ensures the eigenvalues lie in the full range $[0, 1]$ regardless of the graph. Other alternatives such as the normalized Laplacian $\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ often found in the literature of graph signal processing [32] bounds the eigenvalues to be within $[0, 2]$ and, by subtracting the identity matrix, shifts the eigenvalues to the range $[-1, 1]$. However, the eigenvalues of $\tilde{\mathbf{L}}$ are often not spread over the full domain $[-1, 1]$, thus the polynomial is only defined partially over this range.

As a remark, a suitable choice for the degree $P$ is based on a balance between the number of hyperparameters and flexibility. A higher degree means more hyperparameters to optimize but the polynomial can fit towards a more complex shape. While we want $g$ to have enough curvature, the degree should be kept small to ensure $g$ is smooth and easy to learn. Practically we found a choice of $P = 3$ often leads to good performances, and this is consistent with the observations that have been made in the context of oversmoothing in graph neural networks. In particular, [44] suggested that information propagation on graphs should be kept within the 3-hop neighbourhood, and beyond this the repeated application of low-pass functions can lead to embeddings becoming overly similar, leading to a lost of distinguishability.

In addition, our setting is a more generalized version of that in Section II.B of [15], where the terms in the regularizer $\mathbf{J}_p$ can in theory be learnt, but it does not have the advantage of the proposed polynomial methods: first, it does not correspond to a localized filtering; second, it has a high learning complexity, which can reduce the regularizing effect of the norm and as a result makes it more prone to over-fitting.

### 4.3.3 Equivalence to the Co-regionalization Model

The prior model in (4.8) follows the form of separable kernels similar to the co-regionalization model in the literature of kernels for vector-valued functions [12]. Our derivation specifies the kernel on the output space more directly, but in this section we show how we can arrive at our model from the co-regionalization setup. Starting with the model $\mathbf{y}_n = \mathbf{f}(\mathbf{x}_n) + \epsilon_n$ for a GP function $\mathbf{f}(\mathbf{x}_n) \in \mathbb{R}^M$, under the

setup of intrinsic co-regionalization model (ICM) [12], we have

$$\mathbf{f}(\mathbf{x}_n) = \sum_{i=1}^{Q} \mathbf{b}_i u^i(\mathbf{x}_n) \tag{4.10}$$

where $u^1(\mathbf{x}), \ldots, u^Q(\mathbf{x})$ are i.i.d. variables following $\mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$ and $\mathbf{b}_i \in \mathbb{R}^M$ for all $i$. This leads to a model whose covariance is

$$\begin{aligned}
\mathrm{Cov}(\mathbf{f}(\mathbf{x}_n), \mathbf{f}(\mathbf{x}_m)) &= \sum_{i=1}^{Q}\sum_{j=1}^{Q} \mathbf{b}_i \mathbf{b}_j^\top \mathbb{E}(u^i(\mathbf{x}_n) u^j(\mathbf{x}_m)) \\
&= \sum_{i=1}^{Q} \mathbf{b}_i \mathbf{b}_i^\top \mathbb{E}(u^i(\mathbf{x}_n) u^i(\mathbf{x}_m)) \\
&= k(\mathbf{x}_n, \mathbf{x}_m) \sum_{i=1}^{Q} \mathbf{b}_i \mathbf{b}_i^\top .
\end{aligned} \tag{4.11}$$

Denoting $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_Q)$, we can see that $\mathbf{B}\mathbf{B}^\top = \sum_{i=1}^{Q} \mathbf{b}_i \mathbf{b}_i^\top$, thus the covariance can be written as $\mathrm{Cov}(\mathbf{f}(\mathbf{x}_n), \mathbf{f}(\mathbf{x}_m)) = k(\mathbf{x}_n, \mathbf{x}_m)\mathbf{B}\mathbf{B}^\top$. When we have $N$ input-output data pairs, the full covariance of $\tilde{\mathbf{f}} = \mathrm{vec}(\mathbf{f}(\mathbf{x}_1), \ldots, \mathbf{f}(\mathbf{x}_N))$ will follow the separable form $\mathrm{Cov}(\tilde{\mathbf{f}}) = \mathbf{K} \otimes \mathbf{B}\mathbf{B}^\top$. Since a kernel on graphs is usually a square matrix, our graph GP model is equivalent to ICM if $Q = M$ and the vectors $\mathbf{b}_i$ combine into a matrix that takes the general form of Eq. (2.31).

As an additional note, the covariance we derive is dependent on the manner in which $\mathbf{f}(\mathbf{x}_1), \ldots, \mathbf{f}(\mathbf{x}_N)$ are stacked into a single vector (the covariance of $\tilde{\mathbf{y}}$ is then formed from the covariance of $\mathbf{f}$ plus a noise term). If we take $\tilde{\mathbf{f}} = \mathrm{vec}((\mathbf{f}(\mathbf{x}_1), \ldots, \mathbf{f}(\mathbf{x}_N))^\top)$ instead, we will get the covariance $\mathbf{B}\mathbf{B}^\top \otimes \mathbf{K}$. These are simply different ways to represent the prior covariance, and $\mathbf{B}\mathbf{B}^\top$ and $\mathbf{K}$ still correspond to the input and node-level kernels, respectively.

## 4.4 Optimizing GP Log-Marginal Likelihood

The polynomial coefficients $\beta_i$ in the kernel on graphs are found by maximizing the log-marginal likelihood on a training set using gradient optimization. Let $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_P)^\top$, and let $\Omega$ contain $\boldsymbol{\beta}$ and all other hyperparameters, the GP log-marginal likelihood is

$$\begin{aligned}
L(\Omega) &= \log \mathbb{P}(\tilde{\mathbf{y}}|\Omega) \\
&= -\frac{1}{2}\log|\boldsymbol{\Sigma}_\Omega| - \frac{1}{2}\tilde{\mathbf{y}}^\top \boldsymbol{\Sigma}_\Omega^{-1}\tilde{\mathbf{y}} - \frac{NM}{2}\log(2\pi),
\end{aligned} \tag{4.12}$$

where $\mathbf{\Sigma}_\Omega$ is the covariance of (4.7). As described in Eq. (4.6), the term $\mathbf{B} = g(\mathbf{L}_S)$ also acts as a filter on the GP prior to incorporate information from the graph structure. In order for (4.6) to be a valid filtering system, we need to constrain $\mathbf{B}$ to be positive semi-definite (PSD); in other words, we need to have $g(\lambda) \geq 0$ for all eigenvalues [32, 12]. Just optimizing $\boldsymbol{\beta}$ alone in an unconstrained fashion will not guarantee this, thus we utilize Lagrange multipliers to combine constraints with our main objective function.

Assuming all hyperparameters are fixed (details of optimizing for hyperparameters are presented in the experiments section), our constrained optimization problem for finding the optimal kernel on graphs is the following

$$\begin{aligned} &\min_{\beta} - L(\boldsymbol{\beta}) \\ &\text{subject to } -\mathbf{B}_v\boldsymbol{\beta} \leq 0, \end{aligned} \tag{4.13}$$

where we express the log-marginal likelihood $l$ as a function of $\boldsymbol{\beta}$ and $\mathbf{B}_v \in \mathbb{R}^{M\times(P+1)}$ is the Vandermonde matrix of eigenvalues of the Laplacian with the following form

$$\mathbf{B}_v = \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^P \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^P \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_M & \lambda_M^2 & \dots & \lambda_M^P \end{pmatrix}. \tag{4.14}$$

It is easy to see that to have $g(\lambda) \geq 0$ for all eigenvalues is equivalent to setting $-\mathbf{B}_v\boldsymbol{\beta} \leq 0$. Hence, our objective function now becomes

$$\mathbb{L}(\boldsymbol{\beta}, \boldsymbol{\mathcal{L}}) = -L(\boldsymbol{\beta}) + \boldsymbol{\mathcal{L}}^\top(-\mathbf{B}_v\boldsymbol{\beta}) \tag{4.15}$$

$$= -L(\boldsymbol{\beta}) - \boldsymbol{\mathcal{L}}^\top\mathbf{B}_v\boldsymbol{\beta} \tag{4.16}$$

where $\boldsymbol{\mathcal{L}} \in \mathbb{R}^M$ is a vector of Lagrange multipliers. The solution to this problem is guided by the Karush–Kuhn–Tucker (KKT) conditions [85], which specifies that $\boldsymbol{\beta}^*$ is the optimal solution to Eq. (4.13) if $(\boldsymbol{\beta}^*, \boldsymbol{\mathcal{L}})$ is the solution to

$$\min_{\boldsymbol{\beta}} \max_{\boldsymbol{\mathcal{L}}\geq 0} \mathbb{L}(\boldsymbol{\beta}, \boldsymbol{\mathcal{L}}). \tag{4.17}$$

Due to the non-convexity of the log-likelihood, the Lagrangian is non-convex with respect to both variables and we instead solve for the dual problem

$$\max_{\boldsymbol{\mathcal{L}}\geq 0} \min_{\boldsymbol{\beta}} \mathbb{L}(\boldsymbol{\beta}, \boldsymbol{\mathcal{L}}) \tag{4.18}$$

**Algorithm 1** Constrained optimization of polynomial coefficients for GP log-marginal likelihood

1: **Input:** Initialization of $\boldsymbol{\beta}$ and $\mathcal{L}'$
2: Solve for $\min_{\boldsymbol{\beta}} \mathbb{L}(\boldsymbol{\beta}, S_m(\mathcal{L}'))$ using gradient descent:
$\beta_i \rightarrow \beta_i - \gamma_\beta \frac{\partial \mathbb{L}}{\partial \beta_i}(\boldsymbol{\beta}, S_m(\mathcal{L}'))$ for $i = 0, \ldots, P$
3: Update $\mathcal{L}'$:
$\mathcal{L}' \rightarrow \mathcal{L}' + \gamma_\mathcal{L} \frac{\partial \mathbb{L}}{\partial \mathcal{L}'}(\boldsymbol{\beta}, S_m(\mathcal{L}'))$
4: Repeat 2 and 3 until $\mathbb{L}$ converges
5: **Output:** $\boldsymbol{\beta}$

as this makes the function concave with respect to $\mathcal{L}$ [86, 87] leading to an easier problem overall.

We find the solution by alternatively updating $\boldsymbol{\beta}$ and $\mathcal{L}$ described in Algorithm 1. Here, we place a *softmax* function on $\mathcal{L}$ defined as

$$S_m(\mathcal{L}') = \log(1 + e^{\mathcal{L}'}) \tag{4.19}$$

to keep the Lagrange multipliers positive during the optimization. Due to the non-convexity of Eq. (4.18), Algorithm 1 may only find a local optimum depending on the initialization. A simple strategy to obtain a sensible initialization is to maximize the log-marginal likelihood (problem (4.13) without the constraint on $\boldsymbol{\beta}$), with initialization chosen from a small set of values that lead to the highest log-marginal likelihood. The solution to this unconstrained optimization is then used as the initialization for $\boldsymbol{\beta}$ in Algorithm 1. Compared to $\boldsymbol{\beta}$, the algorithm is much more stable with respect to the initialization of the log-Lagrange multiplier $S_m(\mathcal{L})$, and we found using either a fixed or random initialization worked well in practice.

### 4.4.1 Scalability

By exploiting the Kronecker product structure of the covariance matrix, inversion of Eq. (4.7) needed for Algorithm 1 and GP inference can be reduced to a runtime of $\mathcal{O}(N^3 + M^3)$ and thus avoiding the expensive $\mathcal{O}(N^3 M^3)$. We manipulate the covariance matrix in a similar fashion to [88], first re-writing it as follows

$$\begin{aligned}
\boldsymbol{\Sigma} &= \mathbf{B}\mathbf{B}^\top \otimes \mathbf{K} + \sigma_\epsilon^2 \mathbf{I} \\
&= (\mathbf{I} \otimes \mathbf{K})(\mathbf{B}\mathbf{B}^\top \otimes \mathbf{I}) + \sigma_\epsilon^2 (\mathbf{B}\mathbf{B}^\top \otimes \mathbf{I})^{-1} \mathbf{B}\mathbf{B}^\top \otimes \mathbf{I} \\
&= \left[\mathbf{I} \otimes \mathbf{K} + \sigma_\epsilon^2 ((\mathbf{B}\mathbf{B}^\top)^{-1} \otimes \mathbf{I})\right] \mathbf{B}\mathbf{B}^\top \otimes \mathbf{I} \\
&= \left[\sigma_\epsilon^2 (\mathbf{B}\mathbf{B}^\top)^{-1} \oplus \mathbf{K}\right] \mathbf{B}\mathbf{B}^\top \otimes \mathbf{I}
\end{aligned} \tag{4.20}$$

for Kronecker sum $\oplus$. Next, take the eigen-decomposition $\mathbf{K} = \mathbf{U}_K \mathbf{\Lambda}_K \mathbf{U}_K^\top$ and $\mathbf{B}\mathbf{B}^\top = \mathbf{U}_B \mathbf{\Lambda}_B \mathbf{U}_B^\top$, the above equation becomes

$$\begin{aligned}\mathbf{\Sigma} &= \left[\sigma_\epsilon^2 \mathbf{U}_B \mathbf{\Lambda}_B^{-1} \mathbf{U}_B^\top \oplus \mathbf{U}_K \mathbf{\Lambda}_K \mathbf{U}_K^\top\right] \mathbf{U}_B \mathbf{\Lambda}_B \mathbf{U}_B^\top \otimes \mathbf{I} \\ &= \sigma_\epsilon^2 (\mathbf{U}_B \otimes \mathbf{U}_K)(\mathbf{\Lambda}_B^{-1} \oplus \mathbf{\Lambda}_K) \\ &\times (\mathbf{U}_B^\top \otimes \mathbf{U}_K^\top)(\mathbf{U}_B \mathbf{\Lambda}_B \mathbf{U}_B^\top \otimes \mathbf{I}).\end{aligned} \tag{4.21}$$

Each bracket can then be individually inverted by utilizing the orthogonality of the eigenvector matrices and the full matrix inverse becomes

$$\begin{aligned}\mathbf{\Sigma}^{-1} =& \frac{1}{\sigma_\epsilon^2} (\mathbf{U}_B \mathbf{\Lambda}_B^{-1} \mathbf{U}_B^\top \otimes \mathbf{I})(\mathbf{U}_B \otimes \mathbf{U}_K) \\ &\times (\mathbf{\Lambda}_B \oplus \mathbf{\Lambda}_K^{-1})(\mathbf{U}_B^\top \otimes \mathbf{U}_K^\top).\end{aligned} \tag{4.22}$$

Computational complexity is therefore dominated by the two eigen-decomposition of matrices of size $N \times N$ and $M \times M$ giving an overall cost of $\mathcal{O}(N^3 + M^3)$.

Potential further reduction can come in the form of graph coarsening to reduce signals' dimension $M$ while preserving the spectral characteristics [89], as well as sparse GP variational inference [90]. We will leave these as future work.

## 4.5 Experiments

In this section, we first present results on synthetic experiments to demonstrate our algorithm's ability to recover ground truth filter shapes. We then apply our method to several real-world datasets that exhibit different spectral characteristics to show the adaptability of our model.

In all experiments, the GP prior will be in the form of Eq. (4.8) and we consider baseline GP models from [15, 42], and kernels on graphs defined in [25] in Eq. (17-20):

- Standard GP: $\mathbf{B} = \mathbf{I}$
- Global filtering: $\mathbf{B} = (\mathbf{I} + \alpha \mathbf{L})^{-1}$ [15]
- Local averaging: $\mathbf{B} = (\mathbf{I} + \alpha \mathbf{D})^{-1}(\mathbf{I} + \alpha \mathbf{A})$ [42] where we also added a weighting parameter $\alpha$.
- Graph Laplacian regularization: $\mathbf{B}\mathbf{B}^\top = \mathbf{L}^\dagger$ (pseudo-inverse of the Laplacian) [12]
- Regularized Laplacian: $\mathbf{B}\mathbf{B}^\top = (\mathbf{I} + \alpha \tilde{\mathbf{L}})^{-1}$ [25]

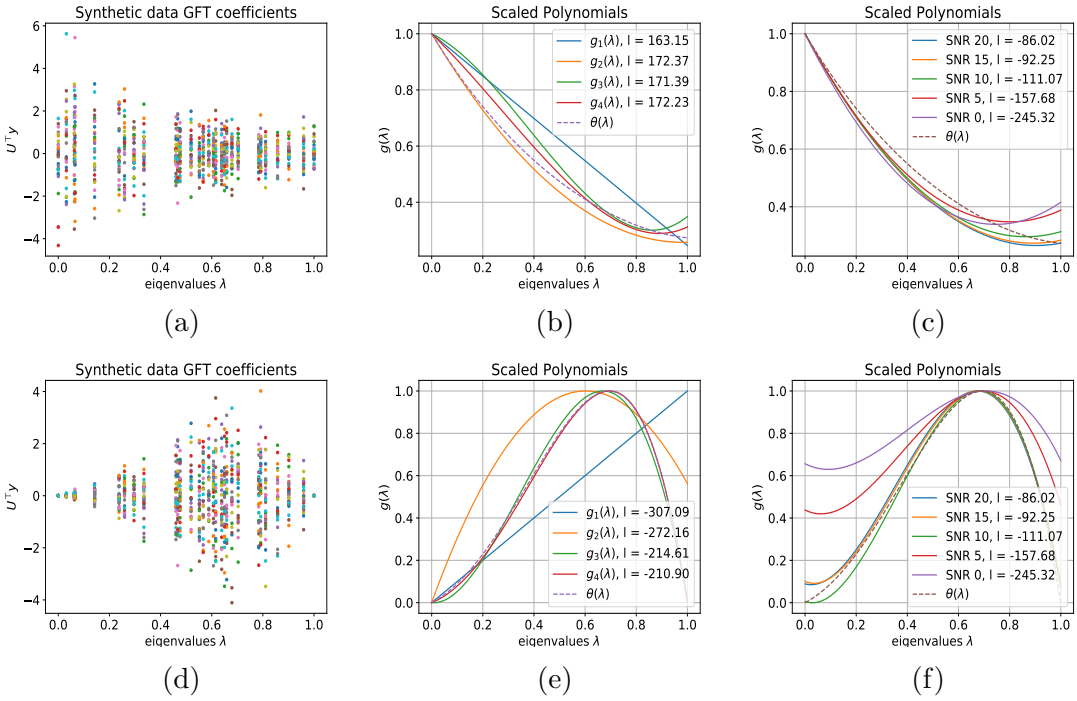



Figure 4.2: Spectral kernel learning on synthetic data, where we show the synthetic graph Fourier coefficients, and the scaled polynomials learned with their log-marginal likelihoods, for data with low- and band-pass spectrums. Ground truth polynomials are $\theta = (1, -1.5, 1.5^2/2, -1.5^3/6, 1.5^4/24)$ for the low-pass (first 5 terms of $e^{-1.5}$), and $\theta = (0, 1, 4, 1, -6)$ for the band-pass. figures/chapter1 (b) and (c) compare the degrees of the polynomial, (c) and (f) compare the SNR noise levels.

- Diffusion: $\mathbf{B}\mathbf{B}^\top = \exp\{(-\alpha/2)\tilde{\mathbf{L}}\}$ [25]
- $p$-step random walk: $\mathbf{B}\mathbf{B}^\top = (\alpha\mathbf{I} - \tilde{\mathbf{L}})^p$ [25]
- Cosine: $\mathbf{B}\mathbf{B}^\top = \cos(\tilde{\mathbf{L}}\pi/4)$ [25]

The kernel on the input space will be the squared exponential $\mathbf{K}_{ij} = \sigma_w^2 \exp\{-\frac{1}{2l}||\mathbf{x}_i - \mathbf{x}_j||_2^2\}$ applied to inputs $\mathbf{x}_n$ and $\mathbf{x}'_n$. The set of hyperparameters is $\Omega = \{\alpha, l, \sigma_w^2, \sigma_\epsilon^2\}$, and these will be found in conjunction with the model parameters $\boldsymbol{\beta}$ in our polynomial. The hyperparameters in the baselines are found by maximizing the log-marginal likelihoods by direct gradient ascent in the same manner as our models. The predictive performance will be evaluated by the posterior log-likelihoods $\log \mathbb{P}(\mathbf{y}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ for test signals $\mathbf{y}_*$, with GP posterior mean $\boldsymbol{\mu}_*$ and covariance $\boldsymbol{\Sigma}_*$ computed by Gaussian conditioning (see [3]).

We would also like to investigate the effect of the constraint on learning performance. To this end, we include our model where we only solve the problem of Eq. (4.13) without the constraint. In the real-world experiments, we include this for polynomials of degrees 3 and 4, where the resulting spectral functions are generally not

always valid graph filtering functions.

### 4.5.1 Initialization Strategy

Due to the highly non-convex structure of the GP log-marginal likelihood, optimizing hyperparameters is heavily reliant on the initializations. Here, we propose a procedure of steps to get the best and most stable solution before passing it to Algorithm 1.

The model parameters $\boldsymbol{\beta}$ are optimized with the hyperparameters giving the set of parameters to learn as $\Omega = \{\boldsymbol{\beta}, l, \sigma_w^2, \sigma_\epsilon^2\}$. Based on a training set of $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, we initialize

$$l = \text{Mean}(\{||\mathbf{y}_1||_2^2, \ldots, ||\mathbf{y}_N||_2^2\}) \tag{4.23}$$

$$\sigma_w^2 = \text{Var}(\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}). \tag{4.24}$$

We set the other parameters by empirically testing a small range of values, using the combination that leads to the highest log-marginal likelihood as the initialization. Our procedure is as follows:

1. Find the optimal $\boldsymbol{\beta}$ and $\sigma_\epsilon^2$ that maximize the log-marginal likelihood by a grid search. We use $\sigma_\epsilon^2 \in \{\frac{1}{10}\sigma_w^2, \frac{1}{5}\sigma_w^2\}$, while for each element $\beta_i$ we use $\beta_i \in \{-10, -9, \ldots, 10\}$.

2. Use the best combinations from grid search as initializations (along with initial $l$ and $\sigma_w^2$) for the unconstrained problem, i.e., maximizing the log-marginal likelihood with respect to $\{\boldsymbol{\beta}, l, \sigma_\epsilon^2\}$ by gradient ascent ($\sigma_w^2$ is indirectly optimized through $\boldsymbol{\beta}$.

3. Use the solution found in step 2 as the initializations to Algorithm 1 and solve for $\boldsymbol{\beta}$, while keeping all other hyperparameters fixed.

As a final note, we follow a general rule for selecting the learning rate for each parameter in the gradient optimization as: choosing the largest $r \in \mathbb{Z}$ such that $\gamma_p = 10^r$ for hyperparameter $p$, that leads to a consistent increase/decrease in the objective function. This will require some tuning from the user beforehand in order to ensure the algorithm converges in a reasonable time.

Table 4.1: Synthetic test log-likelihoods (higher the better) and standard errors.

| Model | Low | Band | High |
|---|---|---|---|
| Degree 1 | -13.79 (0.13) | -81.02 (0.24) | -132.05 (0.29) |
| Degree 2 | -13.48 (0.14) | -76.39 (0.22) | -127.70 (0.30) |
| Degree 3 | **-13.23 (0.14)** | **-65.54 (0.21)** | **-127.24 (0.30)** |
| Degree 4 | -13.88 (0.13) | -67.43 (0.22) | **-127.24 (0.31)** |
| Standard GP | -33.41 (0.13) | -82.41 (0.19) | -171.50 (0.37) |
| Laplacian [1] | -156.44 (0.12) | -96.94 (0.25) | -205.54 (0.92) |
| Global Filtering [6] | -15.90 (0.15) | -83.03 (0.19) | -204.43 (0.86) |
| Local Averaging [8] | -30.23 (0.18) | -99.26 (0.29) | -424.96 (17.58) |
| Regularized Lap [2] | -16.38 (0.15) | -83.08 (0.19) | -206.02 (0.95) |
| Diffusion [2] | -15.81 (0.15) | -82.42 (0.19) | -205.91 (0.94) |
| 1-Step RW [2] | -17.74 (0.15) | -83.13 (0.19) | -356.29 (0.32) |
| 3-Step RW [2] | -19.00 (0.15) | -88.03 (0.24) | -173.31 (0.36) |
| Cosine [2] | -19.56 (0.15) | -86.35 (0.19) | -185.03 (0.29) |

### 4.5.2 Synthetic Experiments

#### Synthetic Filter Reconstruction

For the first experiment we use synthetic signals which are generated following Eq. (4.6) using a $\mathbf{B}$ with a known polynomial chosen beforehand. The aim is to demonstrate that our model is able to recover the polynomial shapes of the ground truth filters through optimizing the GP log-marginal likelihood.

We set the underlying graph to be a 30-node Sensor graph from the `PyGSP` library [91]. The Sensor graph has an even spread of eigenvalues which helps the visualization of the polynomial. Signals are first sampled independently as $\mathbf{y}'_1, \mathbf{y}'_2, \cdots \sim \mathcal{N}(0, \mathbf{I})$. Using the scaled graph Laplacian $\mathbf{L}_S$, we denote the ground truth filter as $\theta(\mathbf{L}_S)$ with coefficients $(\theta_0, \ldots, \theta_Q)$. Each synthetic signal is then set as $\mathbf{y}_n = \theta(\mathbf{L}_S)\mathbf{y}'_n$ and we corrupt it with noise at a signal-to-noise ratio (SNR) of 10 dB. As the signals are sampled independently, the kernel function is $\mathbf{B}\mathbf{B}^\top \otimes \sigma_w^2\mathbf{I} + \sigma_\epsilon^2\mathbf{I}$ where $\sigma_w^2$ is set to signal variance. Input covariates $\mathbf{x}_i$ are not required in this context as the input kernel matrix is already defined as $\mathbf{K} = \mathbf{I}$. We denote the polynomials learned from our algorithm as $g_d$ for degree $d$ which has $d+1$ coefficients. If the $g_d(\lambda)$ goes above 1 for any $\lambda \in [0, 1]$, we can scale it down as $g'_d(\lambda) = \frac{1}{c} g_d(\lambda)$ for $c = \max_{x\in[0,1]} g_d(x)$. The resulting $g'_d$ will be in the range $[0, 1]$ making it easier to compare different filters, and the $c$ term can be absorbed into the variance of the full kernel function, alleviating the need to optimize for $\sigma_w^2$.

In Fig. 4.2, we show the results from learning on synthetic data with low- and band-pass spectrum (a high-pass spectrum will simply have the reversed shape of the low-pass so we will not present here due to the similarity). In Fig. 4.2a and Fig. 4.2d we plot the graph Fourier coefficients $\mathbf{U}^\top\mathbf{y}$ of the generated signals, with each colour corresponding to one signal. The learned polynomials with different degrees

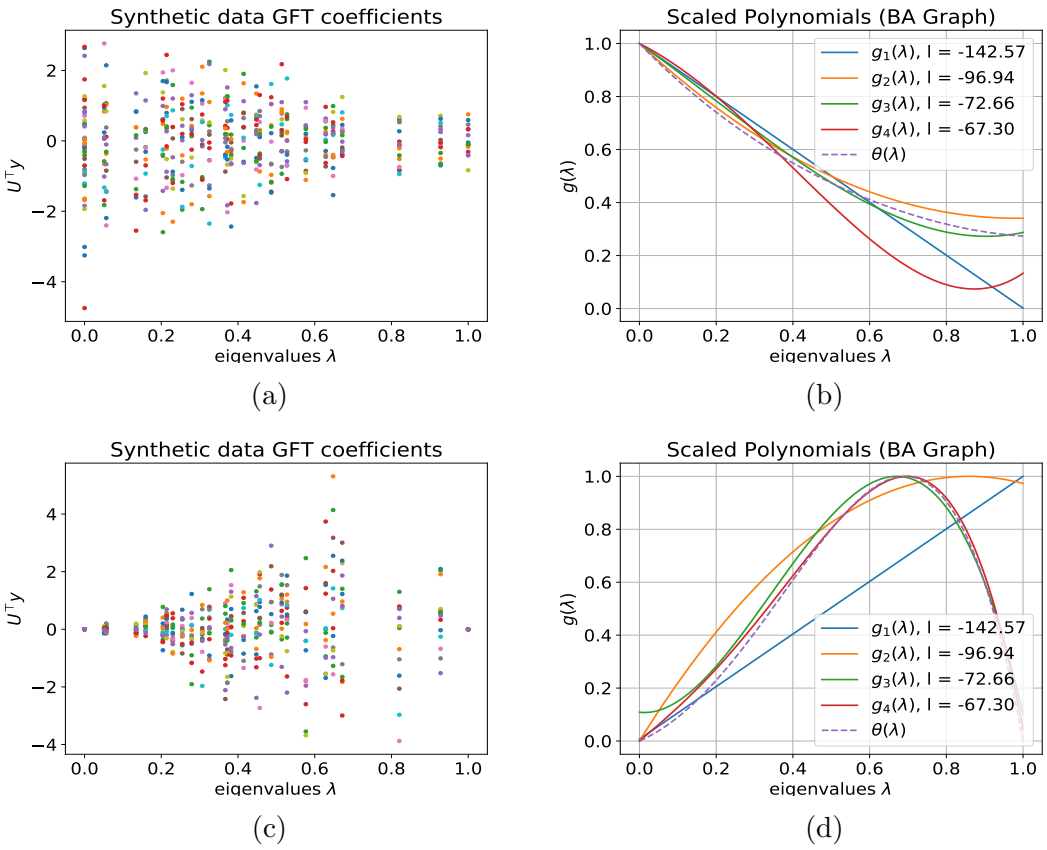



Figure 4.3: Synthetic filter reconstruction on a BA graph. (a) and (c) are signal graph Fourier coefficients, (b) and (d) are the recovered polynomial filters of degree 1 - 4.

can be found in Fig. 4.2b and 4.2e along with the ground truth polynomial $\theta(\cdot)$. Visually we can see that using a polynomial with $d = 2$ and 3 respectively capture the ground truths of low- and high-pass filters well enough that higher degree no longer offers clear improvement. This is also evident in the log-marginal likelihood, where we see only little improvement for $d > 2$ for low-pass and $d > 3$ for band-pass spectra.

We next study the effect of noise on learning the spectrum, using a degree 2 polynomial for low-pass and degree 3 for band-pass. Fig. 4.2c and 4.2f show the spectrum learned for various SNRs, where we can see visually that our model recovers the true spectrum well for SNR 10 dB or higher. As expected, the corresponding log-marginal likelihood steadily decreases as SNR decreases when data becomes noisier.

### Synthetic Filter Reconstruction Using Barabási–Albert Random Graph

To show that our algorithm can generalize to different graphs, we also try recovering graph filters on a Barabási–Albert (BA) random graph in place of the Sensor graph.

Table 4.2: Test log-likelihoods (higher the better) and standard error in bracket.

| **MODEL** (Training) | **fMRI** (21 signals) | **fMRI** (42 signals) | **WEATHER** (15 signals) | **WEATHER** (30 signals) | **UBER** (10 signals) | **UBER** (20 signals) |
|---|---|---|---|---|---|---|
| Degree 2 Polynomial | 35.36 (0.36) | 36.26 (0.49) | -11.58 (3.37) | -7.10 (2.27) | -13.51 (4.40) | -8.68 (2.84) |
| Degree 3 (unconstrained) | 35.33 (0.36) | 36.00 (0.70) | -12.77 (2.48) | -6.61 (2.23) | -12.50 (4.17) | -8.27 (2.68) |
| Degree 3 Polynomial | **36.15 (0.37)** | **36.45 (0.35)** | -9.09 (2.49) | -5.03 (2.39) | -12.48 (4.16) | -8.27 (2.69) |
| Degree 4 (unconstrained) | **36.15 (0.37)** | 36.00 (0.54) | -9.48 (2.22) | **-4.85 (2.43)** | -12.36 (4.13) | -8.47 (2.76) |
| Degree 4 Polynomial | 35.34 (0.36) | 36.00 (0.54) | **-7.47 (2.28)** | **-4.85 (2.43)** | **-12.34 (4.13)** | **-8.26 (2.67)** |
| Standard GP | 11.92 (0.09) | 11.57 (0.12) | -55.59 (4.40) | -53.91 (3.97) | -27.72 (1.21) | -26.70 (1.44) |
| Laplacian [12] | -17.09 (0.10) | -16.41 (0.11) | -67.04 (1.60) | -66.58 (1.60) | -29.16 (0.93) | -28.42 (0.95) |
| Local Averaging [42] | 11.50 (0.10) | 11.44 (0.13) | -51.88 (5.05) | -51.54 (5.09) | -28.93 (1.22) | -27.81 (1.44) |
| Global Filtering [15] | 9.38 (0.11) | 10.49 (0.13) | -50.97 (4.98) | -50.37 (5.22) | -29.15 (1.33) | -28.06 (1.57) |
| Regularized Laplacian [25] | 11.66 (0.10) | 11.44 (0.13) | -52.29 (5.05) | -52.01 (5.01) | -27.52 (1.22) | -26.59 (1.44) |
| Diffusion [25] | 11.55 (0.10) | 11.45 (0.13) | -51.27 (5.27) | -50.88 (5.40) | -27.84 (1.26) | -26.90 (1.49) |
| 1-Step Random Walk [25] | 10.86 (0.12) | 11.13 (0.14) | -60.28 (4.95) | -55.93 (4.22) | -28.65 (1.24) | -26.99 (1.46) |
| 3-Step Random Walk [25] | 11.36 (0.09) | 11.39 (0.09) | -73.09 (8.25) | -76.99 (8.84) | -32.59 (1.64) | -28.29 (1.59) |
| Cosine [25] | 10.09 (0.11) | 10.55 (0.14) | -54.99 (4.60) | -53.83 (4.01) | -27.64 (1.18) | -26.64 (1.44) |

Generally, BA graphs exhibit characteristics closer to real world behaviours. We sample a graph of 30 nodes generated from an initial 10 nodes, and each node added is randomly connected to 5 existing nodes. Filter recovery follows the same procedure as the previous section. Fig. 4.3 show the low- and band-pass filter shapes are still recovered well using Algorithm 1.

### Synthetic Predictive Signals

The main advantage of our spectral kernel learning approach is that we no longer need to worry if the kernel suits the profile of the data. As the models we considered for baselines have low-pass designs, they will not suit the likes of band- and high-pass data. In the previous section, we used $\mathbf{K} = \mathbf{I}$ due to the signals being sampled independently. Although this allowed us to see the full effect of the node-level kernel on the log-marginal likelihood, it cannot be used for inference as predictions from the posterior will be the same as the prior. We now carry out a similar synthetic experiment as that in [41] which defines a non-identity $\mathbf{K}$. This means we can compute the GP posteriors which are the predictive performances of GP models, therefore demonstrating that spectral kernel learning also translates into better predictive performances, especially on band- and high- pass data.

We start by sampling a positive definite matrix $\mathbf{C}$ from the inverse Wishart distribution with identity hyperparameter. The size of $\mathbf{C}$ is of $N \times N$ where $N$ corresponds to the number of signals, here $N = 30$. We then draw $M$ samples from $\mathcal{N}(0, \mathbf{C})$ to create our data matrix $\Delta$ of size $M \times N$. Each column in $\Delta$ is of dimension $M$, forming a signal on the graph. We use a Sensor graph again with $M = 25$ nodes and compute the Laplacian $\mathbf{L}_S$. Let $\mathbf{r}_i$ denote the $i$th row of $\Delta$, we filter this signal

by

$$\mathbf{y}_i = \theta(\mathbf{L}_S)\mathbf{r}_i. \tag{4.25}$$

We use the same low- and band-pass ground truth polynomials $\theta$ as previous section while also including the high-pass profile as $\theta = (0, 1.5, 1.5^2/2, 1.5^2 3/6, 1.5^4/24)$ (first 5 terms of $e^{1.5} - 1$). The prior covariance between signals $\mathbf{y}_i$ and $\mathbf{y}_j$ will be $\mathbf{C}_{ij}\mathbf{B}\mathbf{B}^\top$. Hence $\mathbf{C}$ can be used as the covariance matrix on the input space, and input covariates $\mathbf{x}_i$ are again not required. The full kernel of the GP becomes

$$\mathbf{C} \otimes \mathbf{B}\mathbf{B}^\top + \sigma_\epsilon^2 \mathbf{I}. \tag{4.26}$$

After running Algorithm 1 of log-marginal likelihood maximization, we compute the posterior by conditioning on the first 20 signals $\mathbb{P}(y_*|y_1, \ldots, y_{20})$, to get the posterior log-likelihood on the remaining test signals. We present these performances against the baselines in Table 4.1 where we see the significant improvements on band- and high-pass data from the adaptability of our model.

From the two synthetic experiments, we can conclude that a degree 1 polynomial can be too restrictive as we are fitting the spectrum into a straight line. Thus, when applied to real world data, we will only consider a lowest degree of 2 as this will ensure a level of curvature in the filter we learn.

### 4.5.3 Real World Data

In real world experiments, the graph may not always be available and sometimes needs to be constructed. We will detail how the graph is constructed in each experiment, with the requirement for the graph to be connected and not having multiple components. This is due to the fact that each component may have different spectral profiles, while we only learn a single filter. In the case graphs have more than one component, it would be more suitable to use multiple GPs.

#### fMRI Dataset

In the first real-world experiment we consider data from functional magnetic resonance imaging (fMRI) where an existing graph of 4465 nodes corresponds to different voxels of the cerebellum region in the brain (we refer to [15, 92] for more details on graph construction and signal extraction). A graph signal is the blood-oxygen-level-dependent (BOLD) signal observed on the voxels. Due to the size of the full graph, we use a small subset of nodes. To achieve a connected sub-graph, we first

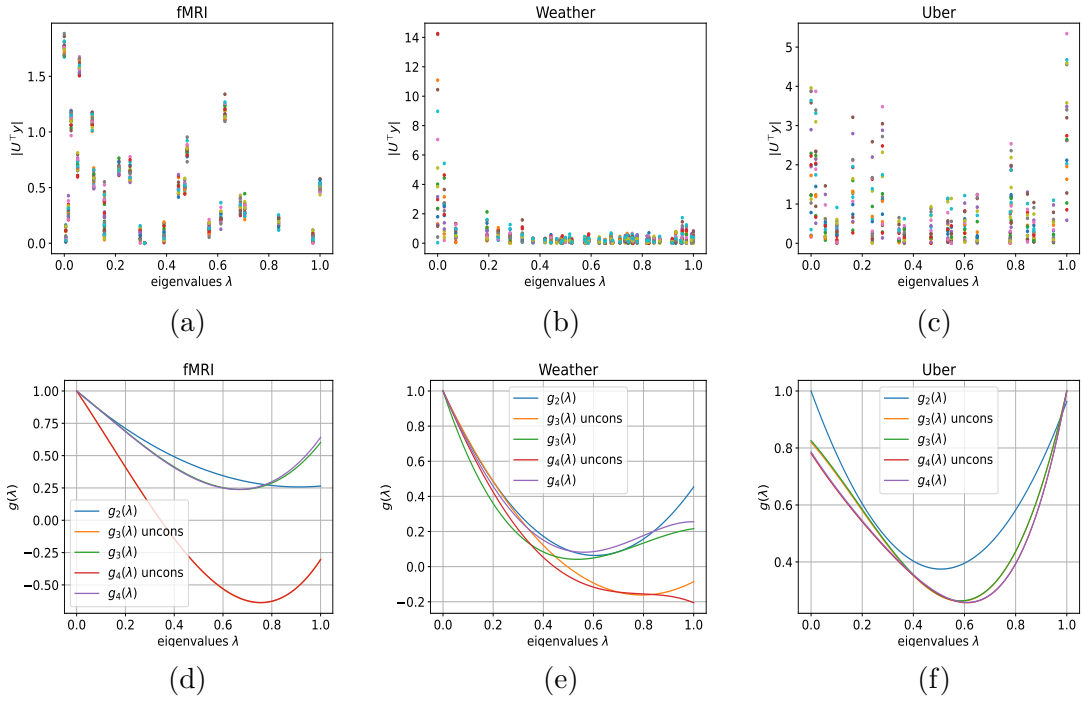



Figure 4.4: (a) - (c) Real world data magnitude of graph Fourier transform coefficients of the training signals $|\mathbf{U}^\top y|$. (d) - (f) Polynomial filters learnt on the corresponding datasets (on the larger training set). The polynomials generally picked up non-low-pass elements, leading to the predictions in Fig. 5 varying in a more similar manner to the test signals.

sample 500 nodes randomly and pick the largest component, which gives us a $\mathcal{G}_{\text{full}}$ of size 37. We then take the readings on the first 10 nodes as $\mathbf{x}_n$ along with the corresponding outcome signals $\mathbf{y}_n$ on the remaining 27 nodes ($\mathbf{y}_n \in \mathbb{R}^{27}$) which form the underlying graph on the outcome signals. The dataset contains 292 signals for which we train on a sample of up to 42 signals to learn the hyperparameters, we then compute the posterior to predict the remaining 250 test signals. To provide a measure of robustness, the test set is split into 10 subsets for which we compute the posterior log-likelihoods on each subset, and report the mean test log-likelihood and standard error - $\mu(\{l_1, \ldots, l_{10}\})$ and $\text{std}(\{l_1, \ldots, l_{10}\})/\sqrt{10}$ respectively in Table 4.2, while the polynomial filters are presented in Fig. 4.4d learnt from the GFT coefficients in Fig. 4.4a, where we see the posterior log-likelihoods are significantly better than the baseline models. The spectrum of the data has a relatively smooth shape as shown in Fig. 4.4a so a low degree polynomial was able to capture the spectrum well and using a higher degree resulted in only marginal improvements in performance. Our adaptive approach to training the GP resulted in much higher posterior log-likelihoods, indicating that we get a prediction with much higher certainty. In our next experiment, we also visualize the improved certainty from our

model.

### Weather Dataset

We now consider the temperature measurement in 45 cities in Sweden available from SMHI [93]. Using the cities' longitude and latitude, we construct a $k$-nearest neighbour graph for $k = 10$ using the function from `PyGSP` [91]. For this dataset, we perform the task of next-day prediction, where each $\mathbf{x}_n \in \mathbb{R}^{45}$ is the temperature signal on day $n$, and the corresponding $\mathbf{y}_n \in \mathbb{R}^{45}$ is the temperature signal on day $n + 1$. The weather dataset is the smoother of the examples we consider, but some minor high frequency elements can still be observed in Fig. 4.4b. We have a total of 90 input-output pairs, and we randomly sample 30 signal pairs $(\mathbf{x}_n, \mathbf{y}_n)$ for training and hyperparameters learning, and predict the signals on the other 60 pairs divided into 10 subsets to give us a mean and standard error in the same way in the previous dataset. The results are presented in Table 4.2 middle two columns and the polynomial filters are shown in Fig. 4.4e, where again, we can see the significant difference between the polynomial models and the baselines. There is also a visible improvement as the degree increases, indicating that a more flexible polynomial suited the profile of the data. Furthermore, by doing next-day prediction, our signals are on the whole graph allowing us to visualize them in Fig. 4.5. Here, we compared the predictions of a degree 3 polynomial, and that of [15] which represents a typical low-pass filter, something all baseline models have in common. In Fig. 4.5, we also reported the graph smoothness of each signal, defined as $\mathbf{z}^\top \mathbf{L}\mathbf{z}$, and generally we would want the graph smoothness of the prediction to be similar to that of the test signal. We can conclude from Fig. 4.5 that our model is superior in terms of both the prediction and uncertainty: visually the polynomial predictions better resembled the ground truth due to the small amount of high-pass picked up by the polynomial. This is also reflected in the predictive mean having a level of graph smoothness more similar to the test signals, while the graph smoothness from the low-pass is much smaller implying its predictions are over-smoothed. The standard deviations are also much lower The standard deviations from the polynomials are significantly lower, indicating that our model predicts with greater certainty.

### Uber Dataset

The final dataset is Uber rides in New York City for the month of September 2014. The dataset contains locations for pickups at $M = 29$ taxi zones which form the nodes of a graph, and edges are constructed based on a $k$-nearest neighbour graph using $k = 4$. The hourly number of Uber pickups in each zone is a signal on the

graph (more information on the dataset can be found in [94]). We randomly select 9 zones as inputs such that each $\mathbf{x}_n \in \mathbb{R}^9$; on the remaining 20 zones, we ensure they form a connected graph, and the values form the output signals $\mathbf{y}_n \in \mathbb{R}^{20}$. The test performances can be found in the final two columns in Table 4.2, the mean and standard errors are over 10 splits like in previous experiments. The data is of low- and high-pass nature as shown by the GFT coefficients in Fig. 4.4c and reflected by the filters learnt in Fig. 4.4f, thus all our models offered improvements compared to the baselines as expected.

## 4.6 Conclusion

We have developed a novel GP-based method for graph-structured data to capture the inter-dependencies between observations on a graph. The kernel on graphs adapts to the characteristics of the data by using a bespoke learning algorithm that also provides a better interpretability of the model from a graph filtering perspective. Our model was superior in capturing the smoothness of the data, and predicting with a higher level of certainty. Promising future directions include the extension of the model for application in classification and improvement in the scalability of the model.

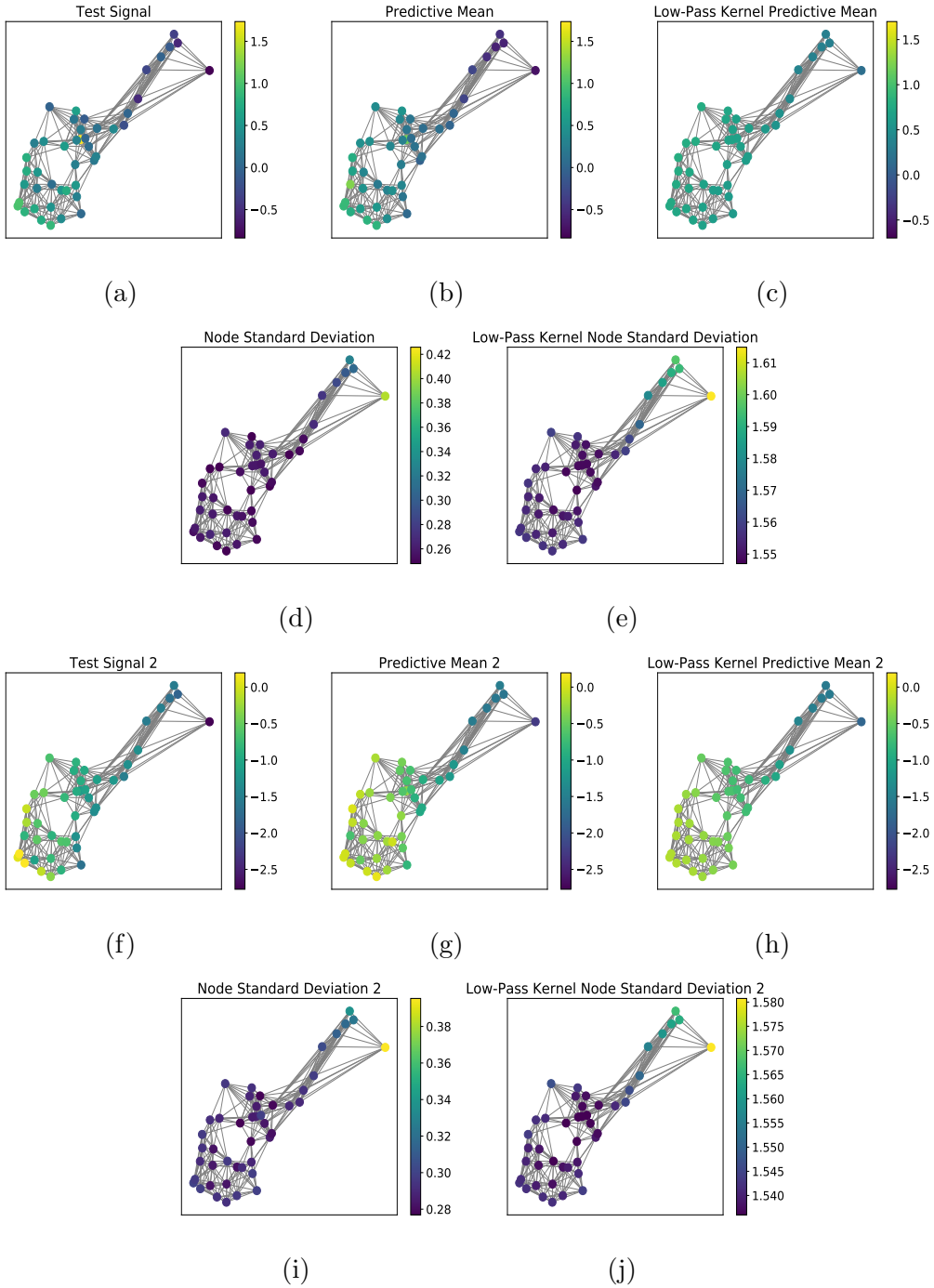



Figure 4.5: Test signals $\mathbf{z}_t$ from Weather dataset (a) $\mathbf{z}_t^\top \mathbf{L}\mathbf{z}_t = 19.84$ & (f) $\mathbf{z}_t^\top \mathbf{L}\mathbf{z}_t = 36.35$; GP predictions from degree 3 polynomial $\mathbf{z}_p$ (b) $\mathbf{z}_p^\top \mathbf{L}\mathbf{z}_p = 16.28$ (g) $\mathbf{z}_p^\top \mathbf{L}\mathbf{z}_p = 29.56$ (d) & (i); from a low-pass of [15] $\mathbf{z}_l$ (c) $\mathbf{z}_l^\top \mathbf{L}\mathbf{z}_l = 6.69$, (h) $\mathbf{z}_l^\top \mathbf{L}\mathbf{z}_l = 8.06$, (e) & (j) representing the low-pass nature of all baselines. The difference in graph smoothness between the test signals and the predictions are bigger in the low-pass, showing that the model over-smooths compared to the degree 3 polynomial, while the standard deviations show that our model is also far more certain in the predictions.

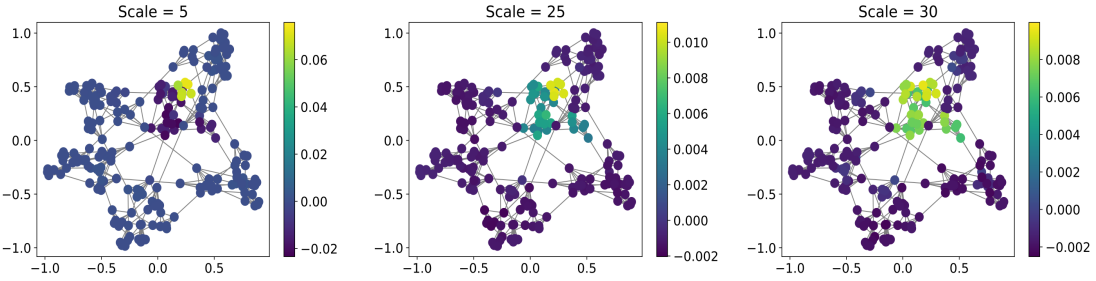
# Chapter 5

# Adaptive Gaussian Processes on Graphs via Spectral Graph Wavelets

Many problems on graphs involve dealing with partially labelled graphs, learning on the labelled nodes to predict the classes of the unlabelled. Furthermore, the graphs in such problems also exhibit multi-scale properties, described by the existence of clusters when we "zoom in and out" of the graph. We thus also shift our focus to designing a GP for semi-supervised node classification, with a focus on multi-scale modelling. The novelty is in the use of wavelets in defining the graph convolution, which offers a balance between the information available in the spatial and spectral domains, with an adjustable trade-off between the two. The flexibility of wavelets also allows for convolution to happen on multiple scales to accurately capture the patterns in the data.

## 5.1 Introduction

Semi-supervised classification on graphs involves inferring class labels of given nodes based on some training set. The predictions are often determined by the information on neighbouring nodes, which can provide indications of the likely class of the node of focus. The core consideration when incorporating graph structure into a GP model design is how much neighbours at varying distances should influence the prediction at a certain node. Early spectral approaches rely on the Fourier basis when designing graph-based operators [95, 76], which are fully localized in the frequency domain but not in the spatial domain, hence requiring a polynomial approximation



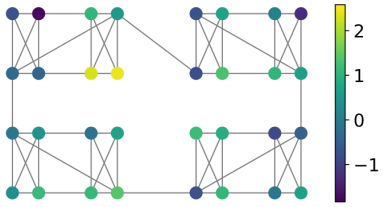

(a) (b) (c)

Figure 5.1: The Mexican Hat wavelet transform of a $\delta$ signal on the focal node. With different scales, the wavelet is able to capture different neighbourhood information weighted in a continuous manner.

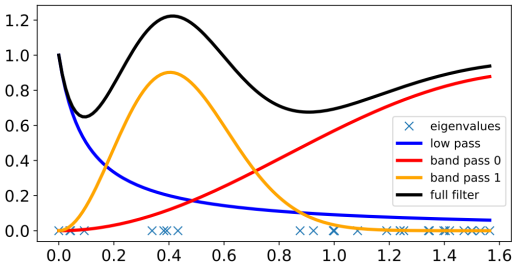
of the graph Laplacian to enforce spatial localization. We propose an approach using wavelets, which offer a natural way of trading off between spectral and spatial resolution, and thus localization, in both domains. The degree of spatial localization is implicitly controlled by a single wavelet scale parameter defined in the spectral domain (visualized in Figure 5.1), which makes graph wavelets a natural tool to enable a more flexible notion of neighbourhood of varying sizes. Moreover, the single scale parameter enables the model to adjust the effective neighbourhood sizes to the properties of the data when incorporated into a model that allows learning hyperparameters, such as a GP.

Beyond flexible control of neighbourhood size, using wavelets allows for combining filters of different scales. Real-world networks such as connectivity patterns in the brain or metabolic or social interactions networks often exhibit such multi-scale community structure [96, 97, 98], where sets of densely connected nodes in turn form densely connected communities (see Figure 5.2a for a synthetic example). These graphs often naturally form the domain of multi-scale signals, which can be modelled through wavelets by combining filters of multiple scales. Figure 5.2b shows an example of how low-pass and band-pass filters are combined into a more complex wavelet filter, which then captures signal components varying at different scales (Figures 5.2c-e).

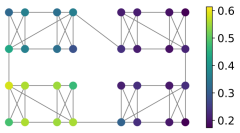
In this work, we introduce a novel graph GP model that uses spectral graph wavelets to incorporate graph structure into the GP kernel. Building on the convenient properties of the wavelet transform, the wavelet graph GP can naturally model continuous neighbourhoods of varying sizes and by extension multi-scale graph signals. The kernel filters are learnable such that their responses can adapt to the observed graph and data. To bypass the expensive eigen-decomposition of the graph Laplacian, we



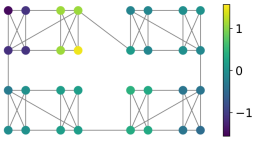

(a) full multi-scale signal



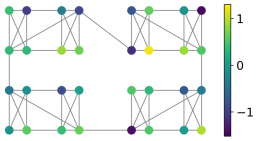

(b) signal spectrum




(c) low-pass filtered signal




(d) band-pass filtered signal




(e) high-pass filtered signal

Figure 5.2: Visualisation of how wavelet filters can be used to capture multi-scale properties in both the graph structure and the graph signal. Figure (a) shows a graph with two levels of clusters (4-node clusters and 8-node clusters). These clusters are reflected in the gaps (around 0.2 and 0.6) in the spectrum of the graph in Figure (b). The signal is obtained by filtering a random signal with the filter in (b), purposefully highlighting the three eigenvalue clusters. Figures (c) - (e) show how the full signal from (a) decomposes into the three filter components. As expected, the low-pass signal varies mostly on the highest cluster level (between 8-node clusters), the band-pass signal mostly on the second cluster level (between 4-node clusters), and the high-pass signal from node to node.

develop a fast approximation to the wavelet-based filtering, which still allows us to directly optimize the wavelet scales and reduces the approximation error on parts of the spectrum with most eigenvalues. We show that our approximation is more suitable for wavelet filters than the Chebyshev polynomial approximation commonly used for existing low-pass filtering approaches. Through experiments, we demonstrate accurate recovery of scales on a synthetic graph and evaluate our model on benchmark data sets, showing our model performance is competitive against state-of-the-art graph-based models.

## 5.2 Preliminaries

### 5.2.1 Spectral Filtering and Wavelets on Graphs.

We refer to the filtering of a signal as the process of highlighting specific frequency components in the signal while de-emphasizing others with the aim of obtaining a function more suitable for the prediction task. Let $\mathcal{G} = (\mathcal{V}, \mathbf{A})$ be a graph with

vertex set $\mathcal{V} = \{v_1, \ldots, v_N\}$ and adjacency matrix $\mathbf{A}$, we define the notion of spectral filtering on graphs [32] based on the graph Laplacian defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal degree matrix. Additionally, the commonly used normalized graph Laplacian is computed as $\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$. This is often preferred due to the boundedness of its eigenvalues to the interval $[0, 2]$ and the scaling of the graph edge weights [32], hence our model will make use of this normalized version throughout.

Assuming that $\mathcal{G}$ is undirected, the Laplacian is symmetric and admits the eigendecomposition $\tilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, and a filtering matrix is achieved by applying a function on the eigenvalues

$$\mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^\top. \tag{5.1}$$

The graph Fourier transform $\mathbf{U}$ is localized in the graph spectral domain as each eigenvector only contributes a single frequency to the construction of $\mathbf{f}$. However, they are not localized in space as each eigenvector of $\mathbf{f}$ is on the entire spatial domain. Wavelet transform addresses this issue by decomposing a function $\mathbf{f}$ into a linear combination of basis function that are both localized in space and frequency. The definition of graph wavelets is derived from spectral graph theory by [18] and will form the basis of the wavelets we utilize. The transform is an operator function of the graph Laplacian determined by a function $g$ as follows:

$$b_\beta(\tilde{\mathbf{L}}) = \mathbf{U}g(\beta\mathbf{\Lambda})\mathbf{U}^\top. \tag{5.2}$$

The function $g$ is applied in the graph spectral domain, but spatially it will also be localized if chosen from the library of mother wavelets. The scale parameter $\beta$ then plays the role of controlling the localization of the transform. We make use of the Mexican Hat wavelet, which we will present later on along with our model formulation. The spatial localization can be demonstrated by applying the wavelet transform to an impulse signal on the graph $b_\beta(\tilde{\mathbf{L}})\delta_n$, where $\delta_n = 1$ at node $n$ and 0 elsewhere. This is presented in Figure 5.1 where the various scales $\beta$ lead to different proximity of neighbourhoods. For each scale, the different hop neighbourhoods are also weighted in a continuous manner that decays to 0 once far enough away from the centre node. This allows the aggregation to happen in a non-linear manner to extract additional information for each node.

## 5.3 Methodology

### 5.3.1 Graph Wavelet GP.

We describe a Gaussian process model for the task of semi-supervised node-level prediction on a graph $\mathcal{G} = (\mathcal{V}, \mathbf{A})$ with $N$ nodes. The nodes of the graph are commonly associated with a set of features $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, which form the feature matrix $\mathbf{X} \in \mathbb{R}^{N \times K}$. As we have seen in Section 5.2, given a graph signal $\mathbf{f} \in \mathbb{R}^N$ on the graph domain, we can apply a wavelet filter $g_\theta(\lambda)$ with scale parameters $\theta$ as follows:

$$\hat{\mathbf{f}} = \mathbf{U} g_\theta(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{f}, \tag{5.3}$$

where $\mathbf{U}$ and $\mathbf{\Lambda}$ are the eigenvectors and eigenvalues of the graph Laplacian of $\mathcal{G}$ such that $\tilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \in \mathbb{R}^{N \times N}$ and $\mathbf{\Lambda}$ is a diagonal matrix. The wavelet filter $g_\theta$ is applied element-wise to $\Lambda$. For brevity, we define $\mathbf{W}_\theta \coloneqq \mathbf{U} g_\theta(\mathbf{\Lambda}) \mathbf{U}^\top$ and refer to it as the *wavelet filter matrix*.

For the sake of conducting Bayesian inference, we assign a Gaussian process prior to the function $\mathbf{f}$

$$\mathbf{f} \sim \mathcal{GP}\left(m(\mathbf{x}), k_\psi(\mathbf{x}, \mathbf{x}')\right), \tag{5.4}$$

with the mean function $m$ and kernel function $k_\psi$ with parameters $\psi$ operating on the node features. On domains described by graphs with a finite number of nodes this prior is equivalent to a multivariate normal distribution with mean $\mathbf{m} = m(\mathbf{X}) \in \mathbb{R}^N$ and covariance $\mathbf{K} = k(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N}$ respectively. As the wavelet filtering described in Equation 5.3 is a linear operation the filtered signal $\hat{\mathbf{f}}$ follows a Gaussian process prior

$$\hat{\mathbf{f}} \sim \mathcal{N}\left(\mathbf{W}_\theta \mathbf{m}, \mathbf{W}_\theta \mathbf{K} \mathbf{W}_\theta^\top\right). \tag{5.5}$$

When combined with a likelihood $p(\mathbf{y} \mid \hat{\mathbf{f}})$, the model is capable of Bayesian inference of an output signal $\mathbf{y} \in \mathbb{R}^N$ by computing the posterior distribution $p(\hat{\mathbf{f}} \mid \mathbf{y})$. In case of regression, the likelihood is commonly assumed to be a normal distribution $p(\mathbf{y} \mid \hat{\mathbf{f}}) = \mathcal{N}(\mathbf{y} \mid \hat{\mathbf{f}}, \sigma^2 \mathbf{I})$ with observation noise $\sigma^2$ and the posterior distribution can be computed in closed form [3]. In the case of classification, a categorical likelihood is assumed, leading to an intractable posterior. We then opt to approximate it with a variational posterior $q(\hat{\mathbf{f}})$ following [90].

### 5.3.2 Adaptive GP via Learning Wavelet Scales.

A key part of the model design is the choice of wavelet filter $g_\theta$ (cf. Equation 5.3). A wide variety of mother wavelet functions are available, here, we choose the Mexican

Hat wavelet function for the band-pass filters, defined as

$$b_\beta(\lambda) = \frac{2\sqrt{2}}{\sqrt{3}\pi^{\frac{1}{4}}} \left(\frac{\lambda}{\beta}\right)^2 \exp\left(-\frac{1}{2}\left(\frac{\lambda}{\beta}\right)^2\right) \tag{5.6}$$

with scale $\beta$. A band-pass filter emphasizes the frequencies in an interval (or band) of the spectral domain. The location of that interval is controlled by the scale $\beta$, which thereby controls the localization of the transform in the spatial and frequency domain. To model lower frequencies of the signal we choose a scaling function with a relatively fast decay as the low-pass filter, defined as

$$h_\alpha(\lambda) = \frac{1}{1+\alpha\lambda} \tag{5.7}$$

with scale $\alpha$. A low-pass filter emphasizes the lower frequencies of a signal, corresponding to its smoother components, where smoothness is measured by the Dirichlet energy $\|\mathbf{f}\|_{\mathcal{G}} = \mathbf{f}^\top \mathbf{L}\mathbf{f}$. The scale $\alpha$ controls how much the filter smooths the signal. To obtain the combined effect of the low-pass and all band-pass filters, we can compute a full filter function as the sum of the individual filters. For $L$ scales, this leads to the spectral filter function

$$g_\theta(\lambda) = h_\alpha(\lambda) + \sum_{l=1}^{L} b_{\beta_l}(\lambda) \tag{5.8}$$

with $\theta = \{\alpha, \beta_1, \ldots, \beta_L\}$, which is used to compute the wavelet filter matrix $\mathbf{W}_\theta = \mathbf{U} g_\theta(\mathbf{\Lambda})\mathbf{U}^\top$, where the subscript highlights the dependence of the filter matrix on the scale parameters.

When the wavelet filter is applied to the GP prior as in Equation 5.5, the scale parameters $\theta$ can be treated as kernel hyperparameters and can be optimized as part of the model fitting process. This is achieved by maximising the marginal log-likelihood $p(\mathbf{y} \mid \theta, \psi)$ with respect to both the scale parameters $\theta$ and the parameters $\psi$ of the node feature kernel $k_\psi$ (cf. Equation 5.4):

$$\begin{aligned} \theta, \psi &= \arg\max_{\theta,\psi} p(\mathbf{y} \mid \theta, \psi) \\ &= \arg\max_{\theta,\psi} \int p(\mathbf{y} \mid \hat{\mathbf{f}}) p(\hat{\mathbf{f}} \mid \theta, \psi)\, \mathrm{d}\hat{\mathbf{f}}, \end{aligned} \tag{5.9}$$

where we highlight the dependence of the GP prior $p(\hat{\mathbf{f}} \mid \theta, \psi)$ on the hyperparameters by explicitly conditioning on them. In the case of classification, which prescribes a non-Gaussian likelihood, the marginal likelihood is intractable and we therefore

resort to maximizing a variational lower bound (Equation 2.17) on the marginal likelihood, again following [90]. This setup enables the model to learn to emphasize frequencies in the data that best describe the output signal $\mathbf{y}$ at hand. In Section 5.4, we examine the model's ability to recover the correct scale in a synthetic data experiment.

### 5.3.3 Spectrum-adaptive Polynomial Approximation.

The model formulation described in previous sections requires computing the eigendecomposition of the Laplacian of the input graph $\mathcal{G}$, which has computational complexity in $\mathcal{O}(N^3)$ and is therefore intractable for larger graphs. To alleviate this limitation, we opt for choosing to approximate the wavelet filter $g_\theta(\lambda)$ with a polynomial $p_\theta(\lambda) = \gamma_0 + \gamma_1\lambda + \ldots + \gamma_K\lambda^K \approx g_\theta(\lambda)$ of degree $K$, as previously suggested by [18]. This allows rewriting the filtering operation in Equation 5.3 as

$$\hat{\mathbf{f}} = \mathbf{U}g_\theta(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{f} \approx \mathbf{U}p_\theta(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{f} = p_\theta(\tilde{\mathbf{L}})\mathbf{f}. \tag{5.10}$$

This formulation circumvents the expensive eigendecomposition of the graph Laplacian and furthermore allows exploiting the sparsity of the Laplacian by using sparse matrix-vector multiplication to compute $p_\theta(\tilde{\mathbf{L}})\mathbf{f}$, which reduces the complexity of the filtering operation to $\mathcal{O}(KE)$, where $E$ is the number of edges in the graph. Existing approaches have relied on a truncated Chebyshev polynomial approximation of the filtering operation and freely optimizing the polynomial coefficients $\boldsymbol{\gamma} \in \mathbb{R}^{K+1}$ [18, 76]. In contrast, our approach is based on optimizing the scale parameters (see previous sections) and we therefore require a polynomial approximation that is parameterized by the wavelet scales $\theta$. A natural choice is the least squares approximation to the filter function $g_\theta(\lambda)$

$$\boldsymbol{\gamma}_\theta = (\mathbf{V}_\xi^\top\mathbf{V}_\xi)^{-1}\mathbf{V}_\xi^\top g_\theta(\boldsymbol{\xi}), \tag{5.11}$$

where $\boldsymbol{\xi} \in \mathbb{R}^S$ is a set $\{\xi_i\}_{i=1}^S$ of linearly spaced points on the spectral domain in the interval $[0, 2]$ and $\mathbf{V}_\xi \in \mathbb{R}^{S\times(K+1)}$ is the Vandermonde matrix for $\boldsymbol{\xi}$ up to degree $K$.

The above least-squares approximation minimizes the approximation error uniformly on the spectral domain. However, in graphs with multi-scale characteristics, the eigenvalues are not uniformly distributed on the spectral domain but rather display *spectral gaps* corresponding to the different scales in the data (cf. Figure 5.2). As the filter function $g_\theta(\lambda)$ is only ever evaluated at the eigenvalues of the graph, a high approximation error of the polynomial approximation at those spectral gaps can be

accepted in turn for a lower approximation error on parts of the spectrum with a higher density of eigenvalues. Following the ideas of [99, 100], we achieve this by computing a weighted least square approximation of the filter function $g_\theta(\lambda)$, where the weights are chosen to be proportional to the spectral density of the graph [101, Chapter 6], which is defined as

$$p_\lambda(z) \coloneqq \frac{1}{N}\sum_{l=1}^{N} \mathbb{1}_{\{\lambda_l = z\}}. \tag{5.12}$$

*Spectral density estimation* aims to approximate this function without performing the expensive eigendecomposition of the graph Laplacian. We opt to employ the Kernel Polynomial Method [102, 103, 104, 105, 106] to find an estimate of the spectral density function by first finding an estimate for the cumulative spectral density function $P_\lambda(z) \coloneqq \frac{1}{N}\sum_{l=1}^{N} \mathbb{1}_{\{\lambda_l \leq z\}}$. For each $\xi_i$ from the set $\{\xi_i\}_{i=1}^{S}$ of $S$ linearly spaced points on the spectral domain, we aim to find the number of eigenvalues less than or equal to $\xi_i$. This can be achieved via stochastic trace estimation [107], which provides us with a randomized algorithm for computing the trace of a matrix $\mathbf{B}$ and we use the Gaussian estimator

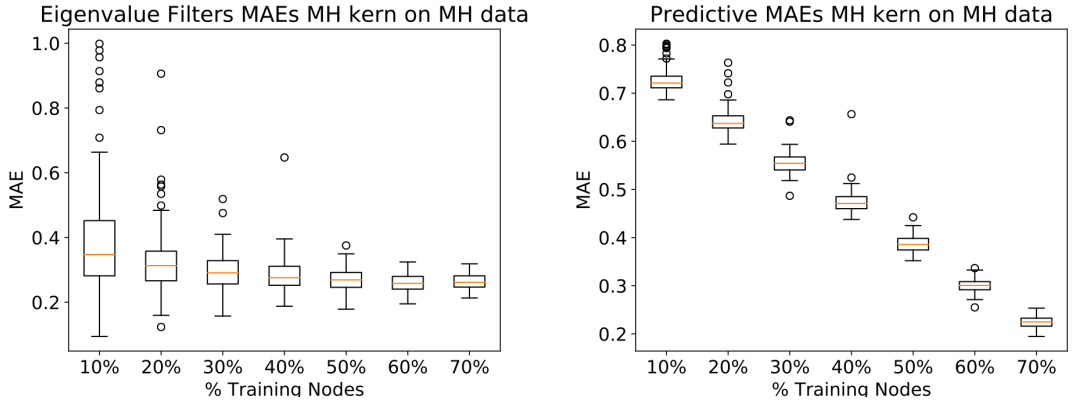
$$\operatorname{tr}(\mathbf{B}) = \mathbb{E}\left[\mathbf{z}^\top \mathbf{B}\mathbf{z}\right] \approx \frac{1}{R}\sum_{r=1}^{R} \mathbf{z}^\top \mathbf{B}\mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{5.13}$$

where $R$ is the number of Monte Carlo samples drawn for approximating the expectation. We thus require a matrix function $\Theta_{\xi_i}$ that maps the Laplacian $\tilde{\mathbf{L}}$ to a matrix whose trace equals the number of eigenvalues less or equal to $\xi_i$. This mapping is trivially given by $\Theta_{\xi_i}(\lambda) = \mathbb{1}_{\{\lambda \leq \xi_i\}}$. While we are not able to cheaply compute $\Theta_{\xi_i}$ exactly, we can approximate it using a Jackson-Chebyshev polynomial $\tilde{\Theta}_{\xi_i}$ (details of this approximation can be found in [108, 109]). We obtain an approximation $\tilde{P}_\lambda(z)$ to the cumulative spectral density function by interpolating between the estimates at points $\xi_i$ using monotonic piece-wise cubic interpolation $\mathcal{I}$

$$\begin{aligned} \tilde{P}_\lambda(z) = \mathcal{I}\left(\left\{\left(\xi_i, \frac{1}{N}\left[\frac{1}{R}\sum_{r=1}^{R} \mathbf{z}_r^\top \tilde{\Theta}_{\xi_i}(\tilde{\mathbf{L}})\mathbf{z}_r\right]\right)\right\}_{i=1}^{S}\right), \\ \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \end{aligned} \tag{5.14}$$

Finally, differentiating $\tilde{P}_\lambda(z)$ with respect to $z$ gives an approximation $\tilde{p}_\lambda(z)$ to the spectral density.

Using this estimate of the spectral density, we can compute weights $\boldsymbol{\omega} \in \mathbb{R}^S$ for




(a) (b)

Figure 5.3: (a) MAE between the recovered wavelet filter against ground truth at the eigenvalues. (b) MAE between predicted values at testing nodes and ground truth labels

each of the $S$ sample points $\xi_i$ on the spectral domain. We can then compute the weighted least squares coefficients

$$\boldsymbol{\gamma}_\theta = \underbrace{\left(\mathbf{V}^\top \mathrm{diag}(\boldsymbol{\omega})\mathbf{V}\right)^{-1}\mathbf{V}^\top \mathrm{diag}(\boldsymbol{\omega})}_{\text{projection matrix } \mathbf{P}} g_\theta(\boldsymbol{\xi}) \tag{5.15}$$

to be used in the polynomial approximation $p_\theta(\tilde{\mathbf{L}})$. The spectral density weights $\boldsymbol{\omega}$ may be pre-computed before training and combined into the projection matrix $\mathbf{P} \in \mathbb{R}^{(K+1)\times S}$, which projects from the exact filter values $g_\theta(\boldsymbol{\xi})$ to the polynomial coefficients $\boldsymbol{\gamma}_\theta$. Finally, these coefficients are used to approximate the wavelet filter matrix

$$\mathbf{W}_\theta \approx \gamma_0\mathbf{I} + \gamma_1\tilde{\mathbf{L}} + \gamma_2\tilde{\mathbf{L}}^2 + \ldots, \tag{5.16}$$

where we have dropped the coefficient's explicit dependence on the scale parameters $\theta$ for notational clarity.

## 5.4 Experiments

### 5.4.1 Synthetic Multi-Scale Graphs for Scales Recovery and Predictions.

The concept of multi-scale corresponds to different things depending on if we are in the graph spatial or spectral domain. In the spectral domain, this is characterized

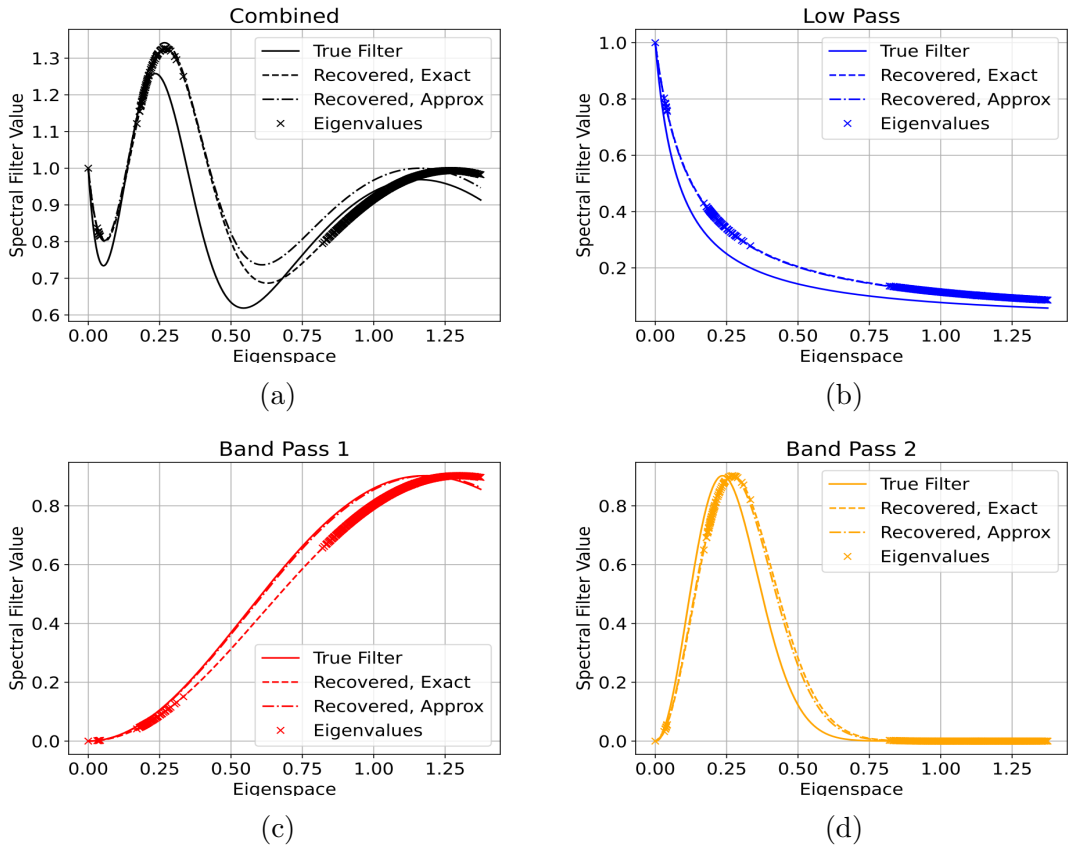



Figure 5.4: Scale recoveries using exact wavelets and polynomial approximations on 50% of data. The ground truth (a) is made of a low-pass $a = 12$ and two band-passes $s = 1.2$ & 6 shown in (b)-(d).

by different dilation of the band-pass filter, whereas spatially we often associate higher level scales as clustering of clusters. If the graph is spatially multi-scale, the different levels of clusters translate to gaps in the eigenvalues, which means we can apply certain characteristics to each level by adjusting a number of ground truth wavelet filters. In this synthetic setting we apply our model to graph data simulated to have both spectral and spatial multi-scale properties. We show that in optimizing the GP prior for the model fitting process, we can accurately recover different scales in the wavelets of the ground truth.

We start by sampling a multi-scale graph through a specially designed algorithm. We use the Erdős-Rényi (ER) random graph as the base generator, and the algorithm involves repeatedly sampling ER graphs to replace the nodes in that level. Continuous labels are then generated for the nodes by sampling from a Gaussian prior with wavelets in the kernels. Let $\mathbf{W}_\phi$ represent a set of wavelets with pre-chosen set of scales $\phi = \{a = 12, s_1 = 1.2, s_2 = 6\}$ such that $\mathbf{W}_\phi = h_a(\tilde{\mathbf{L}}) + g_{s_1}(\tilde{\mathbf{L}}) + g_{s_2}(\tilde{\mathbf{L}})$. We do not specify any node attributes, hence an identity kernel is assumed for $\mathbf{K}$. To

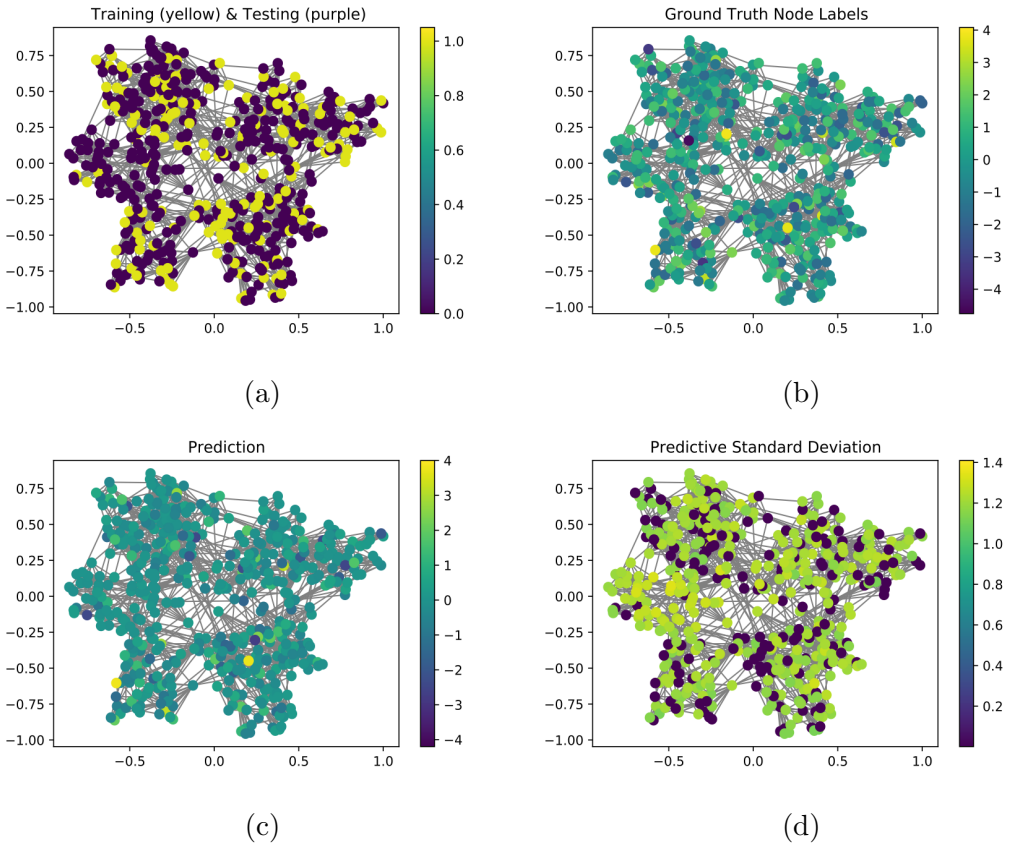



Figure 5.5: (a) graph split into training (yellow) and testing (purple) nodes, only training node labels are made available to the model. (b) node labels of full graph. (c) prediction of full signal using only the training nodes. (d) Node standard deviation of posterior (these are 0 at training nodes).

obtain the node labels, we sample from the Gaussian process

$$\mathbf{y} \sim \mathcal{GP}(0, \mathbf{W}_\phi \mathbf{W}_\phi^\top). \tag{5.17}$$

We split the labels $\mathbf{y}$ randomly into $\mathbf{y}_{\text{train}}$ and $\mathbf{y}_{\text{test}}$, with only $\mathbf{y}_{\text{train}}$ made available to the model for training. The model we use will take the form $\mathbf{f} \sim \mathcal{GP}(0, \mathbf{W}_\theta \mathbf{W}_\theta^\top)$ where $\mathbf{f}$ is the prior between the training and testing nodes and $\theta = \{\alpha, \beta_1, \beta_2\}$ are parameters to be found based on the training labels provided. As in the semi-supervised setting, the full graph will be made available to the model through computing the full $\mathbf{W}_\theta$ matrix, and $\theta$ is then found by maximizing the marginal log-likelihood $\mathbb{P}(\mathbf{y}_{\text{train}}|\theta, \mathcal{G})$. Once the hyperparameters are found we can condition on the training data to obtain the predictive distribution $\mathbb{P}(\mathbf{y}_{\text{test}}|\mathbf{y}_{\text{train}}, \theta, \mathcal{G})$. This distribution provides us with the mean prediction and confidence intervals as shown in Figure 5.5.

We look at two particular performance measures: the mean absolute error (MAE)

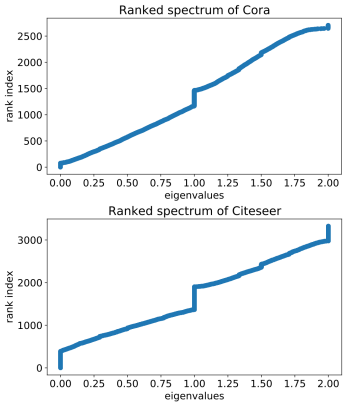


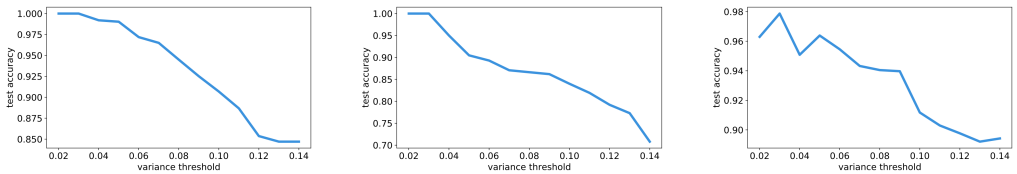

Figure 5.6: Ranked spectra of Cora and Citeseer. Both present distinct ranges of eigenvalues, suggesting multi-scale graph structure.

| Method | Cora | Citeseer | PubMed |
|---|---|---|---|
| **GCN** [44] | 80.5 ±0.8 | 68.1 ±1.3 | 77.8 ±0.7 |
| **GAT** [28] | 82.6 ±0.7 | 72.2 ±0.9 | 76.7 ±0.5 |
| **ChebNet** [76] | 78.0 ±1.2 | 70.1 ±0.8 | 69.8 ±1.1 |
| **LanczosNet** [110] | 79.5 ±1.8 | 66.2 ±1.9 | 78.3 ±0.3 |
| **AdaLanczosNet** [110] | 80.4 ±1.1 | 68.7 ±1.0 | 78.1 ±0.4 |
| **GP** [42] | 60.8 | 54.7 | 71.5 |
| **GGP** [42] | 80.9 | 69.7 | 77.1 |
| **GGP-X** [42] | 84.7 | 75.6 | 82.4 |
| **ChebGP** (ours) | 79.7 | 66.5 | 77.2 |
| **WGGP** (ours) | 84.7 | 70.8 | 78.4 |
| **WGGP-X** (ours) | 87.5 | 76.8 | 90.0 |

Table 5.1: Predictive accuracies of our proposed Wavelet Graph Gaussian Process model compared to a number of baselines. Results are reported with the mean and standard deviation over 10 runs except for Gaussian process models, which do not require random weight initializations.

between the ground truth wavelet filter and the recovered filter at the eigenvalues, and the MAE between $\mathbf{y}_{\text{test}}$ and the posterior mean of $\mathbb{P}(\mathbf{y}_{\text{test}}|\mathbf{y}_{\text{train}}, \theta, \mathcal{G})$. For each selection and percentage of nodes used during training, we sample multiple labels as in (5.17) to recover the filters from. The MAEs can be found in Figure 5.3. The ground truth and recovered filters (via both exact formulation and approximation) for one specific example are presented in Figure 5.4a, while the individual filters $h_a, g_{s_1}, g_{s_2}$ are also shown in Figure 5.4b - 5.4d. More results on scale recovery and predictions on synthetic data (including comparison against baselines) are presented in Appendix B.

### 5.4.2 Semi-Supervised Classification on Graphs.




(a) Cora (b) Citeseer (c) Pubmed

Figure 5.7: We evaluate the performance of our WGGP model when rejecting samples with high predictive variance, i.e. samples with high uncertainty. If the predictive variance estimates are well calibrated, as the variance threshold increases, fewer samples with high uncertainty are rejected and the accuracy should decrease.

We apply Wavelet Graph GP (WGGP) to three citation networks [111], which are

commonly used as benchmark data sets for graph-based models. Here, the underlying graph consists of citations and the node features are bag-of-words (BOW) re-weighed using the popular term frequency-inverse document frequency (TFIDF) transformation. The prediction targets are the topics of the scientific papers in the networks. For the base kernel of the GP, we use a degree 3 polynomial kernel on the TFIDF features, which has been empirically shown to work well with similar models. The wavelet kernel uses two band pass-filters and a low-pass filter. The wavelet kernel is approximated with a degree 5 polynomial for Cora and Citeseer and with a degree 3 polynomial for PubMed. Moreover, for Cora and Citeseer, the kernel is used as part of a non-sparse variational GP, whereas for PubMed we use a sparse variational GP to enable stochastic optimisation of the ELBO using mini-batches.

The hyperparameters of the model are the initial band-pass scales and whether a low-pass filter should be included in the kernel. We train all GP models for up to 300 epochs with a learning rate of 0.01. Similar to [42], we thus also report the result of WGGP trained on both the training and the validation set and refer to it as WGGP-X. The results are presented in Table 5.1 where our model is very competitive against a set of state-of-the-art baselines including graph neural network and GP models. In particular, LanczosNet and AdaLanczosNet [110] were included as, like the method proposed here, they are designed to extract multi-scale information from graphs. We also included a version of our model called ChebGP, which uses Chebyshev polynomials for the spectrum approximation method, to show the superiority of the polynomial approximation method we adopted. Our model outperforms both a vanilla GP model operating solely on the node features and the Graph Gaussian process (GGP) [42] aggregating information from the first-hop neighbourhood, thus highlighting the benefit of our multi-scale approach. Additional results and ablation studies are presented in the Appendix.

### 5.4.3 Uncertainty Estimates.

Unlike the neural network baselines, our proposed GP model performs approximate Bayesian inference and therefore outputs confidence estimates for its predictions at each node $v_i$ via the variance of the variational predictive distribution $q(y_i)$. We expect reliable variance estimates to be useful in deciding which samples to reject (and potentially send to a human labeller) because the model is unable to make a prediction with high enough confidence. We evaluate our model in this regard by computing its predictive accuracies for different variance thresholds. For a lower threshold, more low-confidence samples are rejected, which should lead to a higher predictive performance. We confirm that this property holds for the confidence

estimates of our model via Figure 5.7.

## 5.5 Discussion

In integrating wavelets with a GP, we have developed a model that is capable of capturing multi-scale information in the data. By including different wavelet scales, the model combines various levels of localization on graphs to capture beyond low-frequency elements. Even though the function is defined in the graph spectral domain, by adopting a polynomial approximation we avoid an expensive eigen-decomposition, allowing the model to scale to larger graphs. We show on synthetically generated data that different scales can be recovered accurately, and the multi-scale approach leads to competitive performance on real graph data sets against state-of-the-art graph models.

Applying the proposed wavelet model to a task at hand requires taking a number of practical considerations into account. Firstly, the number of scales in the wavelet kernel should ideally be chosen in a way such that the multi-scale graph data is captured by the different scales of the wavelets (although the model is robust to varying number of scales, cf. Appendix B). For example, we may aim to match the number of scales in the kernel with the number of gaps in the spectrum by estimating the eigenvalue distribution of the graph Laplacian, which is already part of the wavelet transform approximation. Secondly, given the nature of wavelets as dilated and shifted band-pass filters, an interesting question is which mother wavelet to choose for the GP model. While our model is robust to different choices of the mother wavelets (cf. Appendix B) certain options might be preferred for a given task based on their localization properties in the spatial and spectral domain. Finally, which nodes are selected for training can impact the learning process and final performance. If domain knowledge is available, one may look to find strategic ways to sample training nodes that will lead to the best possible characterisation of input data given a limited sampling budget.

# Chapter 6

# Transductive Kernels for Gaussian Processes on Graphs

Continuing with the work on semi-supervised problems on graphs, we now look to find a unified framework of kernel functions for attributed graphs. Kernels have been defined for general feature data, and for graphs (ignoring any node attributes) separately, but there is yet an option that takes into account both information. In this chapter we derive a kernel that depends on the graph connectivity as well as the node attributes (which will be used as feature data). This kernel will therefore be the most generalized definition for attributed graphs, and we provide the interpretability for each component so any user can easily design the overall kernel.

## 6.1 Introduction

Recent studies into building GPs on graphs have proved to be competitive against state-of-the-art graph neural network (GNN) models, giving rise to various kernel-based approaches for semi-supervised classification in [42, 53, 43] as well as Chapter 5 of this thesis. In this work, we focus on deriving more generalized kernel functions for semi-supervised problems on graphs. Graph data of this form often comes with node attributes, which we will use as feature data, and in designing a kernel for such graphs, the challenge is in embedding the graph connectivity as part of the kernel along with the feature data.

The most representative approaches to building GP on graphs for semi-supervised classification are [42] and Chapter 5, where the authors made use of a matrix transformation on a non-graph base kernel on the node feature data. This could be a limiting factor for the model as the graph information comes into the model through

a linear transformation. Furthermore, in this work we found if the base kernel is not particularly suitable, adding the graph elements will not be effective, and these models do not have a way to adjust the influence of the graph against the feature data kernel.

To address these issues, we use the principles of kernel design through regularization theory to derive a kernel with the ability to naturally handle the graph and feature data. In this approach, kernels are obtained by choosing regularization functions and finding a reproducing kernel Hilbert space (RKHS) [24], typical continuous kernels such as RBF and Matérn kernels can all be derived in this manner.

The regularization approach has been translated to the graph domain in [25], but currently this kernel depends on the graph only, and cannot incorporate node data. However, this method can be extended to graphs with node feature data, and in this chapter we show how this can be achieved. We start by introducing the regularization approach to derive kernels for feature data, and for graphs separately. We then present our approach by combining the two regularizers to obtain a kernel for graphs with node data, leading to our proposed model. The resulting model has transductive properties, meaning it trains on the full graph and all the nodes' data (but only the training node labels), and can better capture the distribution of the dataset. In addition, our kernel provides a clear way to control the influence of the graph compared to the feature data kernel. We then show that our setup is general, and that many graph learning models are actually instances of our design. We demonstrate the advantages of our model on synthetic data that are highly non-Euclidean while using relatively small training set. Lastly, we test on various real-world graph-data in semi-supervised classification, comparing against various graph GP and popular GNN models.

## 6.2 Background

### 6.2.1 Kernels via Regularization

We derive kernels based on the regularization theory introduced in Chapter 2.3, we reiterate the important details here. Many machine learning models can be formulated as a minimization problem of a loss function, and often regularizers can be added to the objective to enforce certain structures on the model (for instance to avoid over-fitting). By examining the regularizer, kernel functions can then be derived by assuming that solution comes from an RKHS. In the continuous domain,

as described by [24], we start with the following regularization problem

$$\arg\min_{f} \quad \text{loss}(f, \mathbf{y}) + \Omega(||f||^2) \tag{6.1}$$

between model $f$ as a function of $\mathbf{x}_i$, and labels $\mathbf{y}$. We consider the regularizing function that can be represented by an inner product

$$\Omega(||f||^2) = \langle f, r(\Delta)f \rangle = \langle f, f \rangle_{\mathcal{H}}. \tag{6.2}$$

The norm is specified as a Hilbert space $\mathcal{H}$ characterized by the smoothness measure $r(\Delta)$ on $f$. In Chapter 2.3, we established that

$$\mathbf{K} = r^{-1}(\Delta). \tag{6.3}$$

or elementwise

$$k(\mathbf{x}, \mathbf{x}') = r^{-1}(\Delta)_{(\mathbf{x}-\mathbf{x}')}. \tag{6.4}$$

This is derived by solving $\mathbf{K}r(\Delta) = \delta(\mathbf{x} - \mathbf{x}')$, ensuring $\mathcal{H}$ is a RKHS.

### 6.2.2 Kernels and Regularization on Graphs

A graph dataset $\mathcal{G}$ can be defined by $(\mathcal{V}, \mathbf{A}, (\mathbf{X}, \mathbf{y}))$, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ is the set of vertices each with associated node attributes $\mathbf{x}_i \in \mathbf{X}$ and label $y_i \in \mathbf{y}$, $\mathbf{A}$ is the symmetric adjacency matrix, and we will refer $\mathbf{x}_i$ as the feature data of the node from the full data matrix $\mathbf{X}$.

Let $\mathbf{f}$ be a function or signal on the graph, we again start from the regularization problem of (6.1), but instead of $\Delta$ the dependence is now replaced by the graph Laplacian

$$\Omega(||\mathbf{f}||^2) = \langle \mathbf{f}, r(\mathbf{L})\mathbf{f} \rangle = \mathbf{f}^\top r(\mathbf{L})\mathbf{f}. \tag{6.5}$$

In Chapter 2.3.1, we showed that the kernel on a graph is computed as follows

$$\mathbf{f}^\top r(\mathbf{L})\mathbf{K} = \mathbf{f} \implies \mathbf{K} = r^{-1}(\mathbf{L}). \tag{6.6}$$

The kernel derived here is a measure of similarity between nodes using solely the connectivities of the graph, and not on any node feature data. In the next section, we introduce how to derive a kernel on graphs that is also dependent on any additional node feature data.

The kernel in (6.6) can also be expressed in terms of the eigen-decomposition of $\mathbf{L}$ as $\mathbf{U}r^{-1}(\Lambda)\mathbf{U}^\top$. This we note is the same form as the filtering equation (2.6) introduced in Chapter 2.1 and therefore has interpretations as filtering on graphs. Since the function $r^{-1}$ acts on the graph Fourier spectrum, and the shape it takes will inform the user of the smoothness profile of the model.

## 6.3 Transductive Kernels for Graphs with Feature Data

Most graphs have feature data $\mathbf{x}_i$ associated with each node $v_i$, therefore there is the need to find a kernel of the form $k(\mathbf{x}_i, \mathbf{x}_j|\mathcal{G})$ for node-level Gaussian processes. We do this by noting that $\mathbf{x}_i$ belongs to a continuous space, while the nodes themselves fall on the graph space. Thus, if we are to take into account both spaces of information, we should use two regularizers

$$\Omega(||\mathbf{f}||^2) = \langle \mathbf{f}, [r_1(\Delta) + r_2(\mathbf{L})]\mathbf{f} \rangle. \tag{6.7}$$

The choice of adding $r_1(\Delta)$ and $r_2(\mathbf{L})$ is important as this leads to novel kernels later on. However, one also has the option to multiply the two regularizers, to be used solely or included as a third additional regularizer above. Which terms to include is down to what the user deems appropriate for the problem, and dependent on the complexity of optimizing additional kernel hyperparameters.

Like the previous section, by the reproducing property, the kernel is

$$\mathbf{K} = [r_1(\Delta) + r_2(\mathbf{L})]^{-1}. \tag{6.8}$$

Although we can simply take the $(i, j)$th element of the inverse to obtain $k(\mathbf{x}_i, \mathbf{x}_j|\mathcal{G})$, we can write this in a more explicit form by using the Woodbury formula [112, 113], first setting $\mathbf{K}_1 = r_1^{-1}(\Delta)$, we can express (6.8) as

$$[\mathbf{K}_1^{-1} + r_2(\mathbf{L})]^{-1} = \mathbf{K}_1 - \mathbf{K}_1[\mathbf{K}_1 + r_2^{-1}(\mathbf{L})]^{-1}\mathbf{K}_1. \tag{6.9}$$

A slight variation on the use of the Woodbury formula is the following, which practically we found to be more stable and is therefore the version we use in our implementation

$$[\mathbf{K}_1^{-1} + r_2(\mathbf{L})]^{-1} = \mathbf{K}_1 - \mathbf{K}_1[\mathbf{I} + r_2(\mathbf{L})\mathbf{K}_1]^{-1}r_2(\mathbf{L})\mathbf{K}_1 \tag{6.10}$$

from which we can see that elementwise

$$k(\mathbf{x}_1, \mathbf{x}_2) = k_1(\mathbf{x}_1, \mathbf{x}_2) - k_1(\mathbf{x}_1, \mathbf{X})^\top [\mathbf{I} + r_2(\mathbf{L})\mathbf{K}_1]^{-1} r_2(\mathbf{L}) k_1(\mathbf{X}, \mathbf{x}_2) \tag{6.11}$$

where $\mathbf{X}$ is the matrix of feature data on all the nodes.

The notable feature of (6.11) is that the kernel between any two points depends on the full graph Laplacian and the feature data on every node $\mathbf{X}$. This gives the kernel the transductive property, as during training the model has access to both the training and test node data (the node features), and any potential unlabelled nodes, but importantly only the training labels. This is similar to the resulting kernel of [114], but this approach solely modifies the Hilbert space by weight construction between points, instead of using separate graph information. Furthermore, with a learnable $r_2$, which we will introduce in the next section, this kernel not only captures the distribution of the input data, but also the smoothness of the labels.

The transductive nature of the kernel is due to the inversion of a set of matrices in (6.11), this means evaluating the prior requires a complexity of $\mathcal{O}(N^3)$ for size of the graph $N$. This is currently a limitation and therefore this kernel can only handle graphs of reasonable sizes.

### 6.3.1 Flexible Modelling of Regularization on Graphs

The regularizer $r_1(\Delta)$ does not enter the kernel when using the version of Woodbury formula to specify (6.10). Generally, we skip defining $r_1(\Delta)$ as it sometimes does not exist, for example for dot product kernels [115]. Instead, we can directly choose the kernel $\mathbf{K}_1$, and examples of such can be any typical kernel such as the RBF

$$\sigma_1^2 \exp\left\{ -\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2l^2} \right\}, \tag{6.12}$$

with $\sigma_1^2$ acting as the variance term. Meanwhile, examples of $r_2(\mathbf{L})$ are $r_2(\mathbf{L}) = \frac{1}{\sigma_2^2}(\mathbf{I} + \alpha\mathbf{L})$, or $\frac{1}{\sigma_2^2}\exp\{\alpha\mathbf{L}\}$ for $\alpha > 0$, which along with the choice in [114] are of low-pass nature when inverted to obtain the kernel [25]. A more flexible option is polynomials, and in order to specify this we first have $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ leading to the graph regularization term written as

$$r_2(\mathbf{L}) = \frac{1}{\sigma_2^2}\mathbf{U} r_2(\mathbf{\Lambda})\mathbf{U}^\top \tag{6.13}$$

where $r_2$ can be defined as an element-wise function on the graph Laplacian eigenvalues $\lambda$, and $\sigma_2^2$ acts as the variance term of $r_2(\mathbf{L})$. The polynomial coefficients can

| Model | Name | $r_1(\Delta)$ | $r_2(\mathbf{L})$ |
|---|---|---|---|
| Label Propagation | Label Propagation [116] | 0 | $\frac{1}{1-\alpha}(\mathbf{I}+\alpha\mathbf{L})$ |
| Kernels on graphs | Regularized Laplacian [25] | 0 | $(\mathbf{I}+\sigma^2\mathbf{L})$ |
| | Diffusion [25] | 0 | $\exp\{\frac{\sigma^2}{2}\mathbf{L}\}$ |
| | $p$-step random walk [25] | 0 | $(\alpha\mathbf{I}-\mathbf{L})^{-p}$ |
| | Cosine [25] | 0 | $(\cos(\mathbf{L}\pi/4))^{-1}$ |
| GP kernels | Matérn kernel on graphs [54] | 0 | $(\frac{2\nu}{\kappa^2}-\mathbf{L})^{\nu/2+d/4}$ |
| | Laplacian kernel [3] | $1+\|\|\Delta\|\|^2$ | 0 |
| | Gaussian kernel [3] | $e^{\frac{\sigma^2}{2}\|\|\Delta\|\|^2}$ | 0 |
| | Matérn kernel on manifolds [54] | $(\frac{2\nu}{\kappa^2}-\Delta)^{\nu/2+d/4}$ | 0 |
| Graph GP | GGP [42] | $(\mathbf{P}^\top)^{-1}r(\Delta)\mathbf{P}^{-1}$ | 0 |
| Wavelet Graph GP | WGGP [30] | $(\mathbf{W}^\top)^{-1}r(\Delta)\mathbf{W}^{-1}$ | 0 |
| Transductive kernel (ours) | TGGP (ours) | $(\frac{2\nu}{\kappa^2}-\Delta)^{\nu/2+d/4}$ | $\mathbf{U}[\text{softplus}(1+\sum_i(\beta_i\mathbf{\Lambda}^i))]\mathbf{U}^\top$ |

Table 6.1: Various types of graph learning models that can be considered instances of the transductive GP. For GGP and WGGP, the matrices are $\mathbf{P}=(\mathbf{I}+\mathbf{D})^{-1}(\mathbf{I}+\mathbf{A})$ and $\mathbf{W}=h(\mathbf{L})+g_1(\mathbf{L})+g_2(\mathbf{L})$ for low-pass $h$ and band-pass $g_1$ and $g_2$ defined in [30]. $r(\Delta)$ is not defined for these two models due to the kernel used being a dot product kernel, which does not have a corresponding regularization function.

be freely learnt during the optimization, but we require $r_2(\lambda)>0$ for all $\lambda$ in order to avoid any non-positive definite matrices which may lead to a singular kernel. We achieve this by passing the polynomial through a positive softplus function

$$r_2(\lambda)=1+\text{softplus}\left(\sum_{i=0}^{d}\beta_i\lambda^i\right) \tag{6.14}$$

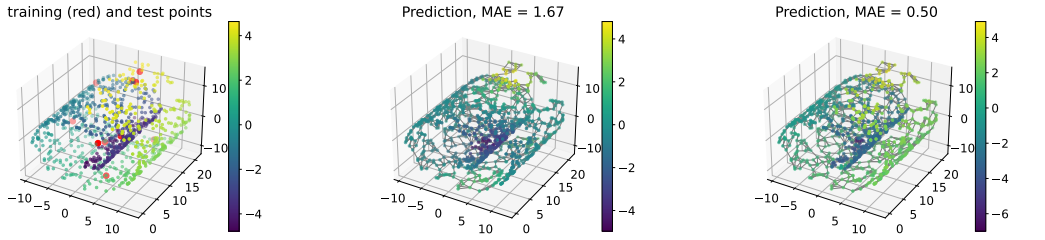
where each $\beta_i$ is learnt freely, $\text{softplus}(x)=\log(1+e^x)$, and the addition of 1 helps with numerical stability and leads to interpretable filters that are between [0,1].

The variance terms $\sigma_1^2$ and $\sigma_2^2$ not only capture the variance in the data, but also act as weighting functions on the two terms in the kernel in (6.8). Thus, by adjusting the size of these two hyperparameters, we can also tell which is more important between the node feature data and the graph structure.

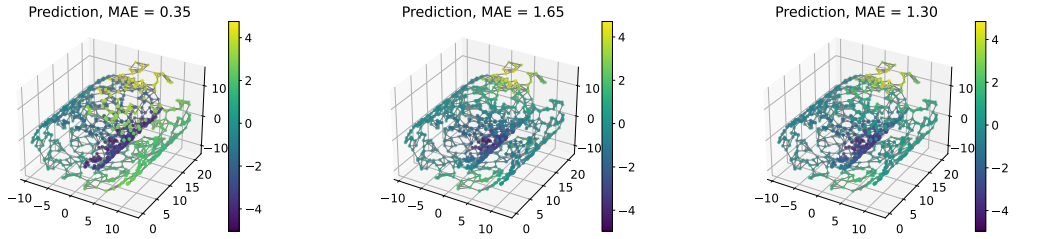
### 6.3.2 Relationships with Other Models

Our proposed kernel in (6.8) is a general framework for learning on graphs. Notably, when the two terms $r_1(\Delta)$ and $r_2(\mathbf{L})$ take certain forms, we recover well known models on graphs. We present a number of these in Table 6.1. The most noticeable element from this table is that all existing kernel-based models correspond to having only one of $r_1(\Delta)$ or $r_2(\mathbf{L})$, and importantly they lack a second regularizer. On the other hand, our model is unique in that it combines the two regularizers, therefore having the capacity to capture information in both the node features and the graph.

In addition to summing the two regularizers, it is also possible to include the product of two regularizers, but this is the less interesting case as it simply leads to product




(a) Ground truth (b) GP with RBF (c) Graph only $(\mathbf{I} + \alpha\mathbf{L})^{-1}$




(d) TGGP (ours) (e) GGP [42] (f) WGGP [30]

Figure 6.1: Swiss roll regression training on 10 points. Each plot shows the prediction from a GP model and the MAE is computed against ground truth in (a). Model (d) best resembled the ground truth and produced the lowest MAE. Results from repeated runs are presented in Fig. 6.2

kernels. This is comparable to **GGP** and **WGGP**, both of which combine the graph information with the feature kernel through multiplication (but with a transform matrix instead as $\mathbf{PKP}^\top$ and $\mathbf{WKW}^\top$). The properties of product kernels, as well as additive kernels, have been discussed in Chapter 2.4 of [117], describing multiplication and addition as "and" and "or" operations respectively. More precisely, product kernels will only have large values when both of the individual kernels are large, thereby assuming the labels have strong dependence on both similarity measures; additive kernels on the other hand can be large if either of the individual kernels is large, and this is equivalent to modelling multiple sources of information that can contribute to the prediction. In our case we have additive regularizers, but due to the transductive nature and the inversion in (6.10) make it capable of modelling more complex patterns compared to additive kernels. However, the role of the two regularizers play can be interpreted in a similar way - if two nodes are similar in their features or based on the graph, then they should be penalized less in the minimization of (6.1), and vice versa. On attributed graphs of various homophily levels, each pair of nodes can be similar due to their feature data, the graph, or both, therefore additive regularizers are the more flexible choice to control the influence of each source of information. However, the definition of the transductive kernel is flexible and the user can choose to include product of regularizers to replicate

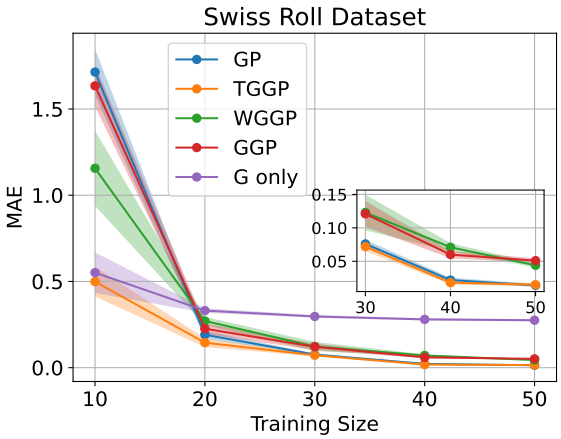



Figure 6.2: Mean MAE and standard error over 10 samples of Swiss roll synthetic data on different number of training nodes.

kernels used in existing graph GP such as **GGP** and **WGGP**.

## 6.4 Experiments

### 6.4.1 Synthetic Results

We first experiment on the Swiss-Roll dataset, where the non-Euclidean nature gives the need for using a graph. We sample 1000 points from the roll-like surface on which we construct a $k$-nearest-neighbour graph on the coordinates with $k = 4$, a relatively small choice while ensuring the graph is fully connected. Each point has a continuous label that is increasing as the roll moves from the inner-most end to the outer-most of the surface. The nodes have coordinates as attributes which we will use as feature data.

The transductive nature of the kernel heavily exploits the graph's ability to capture the roll-like structure, allowing the distribution of the test points to have a big effect on the prediction. As a result, the model picks up more information than simpler models, and this advantage is most apparent on fewer training data when other models struggle to learn. To demonstrate this, we randomly use only 10 labels as training, and predict the labels on the remaining 990 points.

We compare against kernels using just the feature data (standard GP), and graph only [25] as sanity checks, while GGP [42] and WGGP [30] are both kernel-based models, but with a different setup to our model. The predictions are visualized on the graph along with the MAE on the test node labels in Fig. 6.1.

Using the graph only kernel in Fig. 6.1c already improved on the MAE of the **GP**

| Method | Texas | Cornell | Wisconsin | Chameleon | Cora | Citeseer | Squirrel | Actor |
|---|---|---|---|---|---|---|---|---|
| # Nodes | 183 | 183 | 251 | 2,277 | 2,708 | 3,327 | 5,201 | 7,600 |
| Homophily Ratio | 0.11 | 0.30 | 0.21 | 0.23 | 0.81 | 0.74 | 0.22 | 0.22 |
| **GCN** [44] | 59.5 ±5.3 | 57.0 ±4.7 | 59.8 ±7.0 | 59.8 ±2.6 | 80.5 ±0.8 | 68.1 ±1.3 | 36.9 ±1.3 | 30.3 ±0.8 |
| **GAT** [28] | 58.4 ±4.5 | 58.9 ±3.3 | 55.3 ±8.7 | 54.7 ±2.0 | **82.6 ±0.7** | **72.2 ±0.9** | 30.6 ±2.1 | 26.3 ±1.7 |
| **ChebNet** [76] | 77.3 ±4.1 | **74.3 ±7.5** | 79.4 ±4.5 | 55.2 ±2.8 | 78.0 ±1.2 | 70.1 ±0.8 | 43.9 ±1.6 | 34.1 ±1.1 |
| **LP** [116] | 37.8 | 21.6 | 23.5 | 44.5 | 71.3 | 49.9 | 32.7 | 22.4 |
| **GP** [42] | 78.4 | 73.0 | 78.4 | 46.1 | 60.8 | 54.7 | 34.4 | **34.9** |
| **GGP** [42] | 78.4 | 62.1 | 60.8 | **73.5** | 80.9 | 69.7 | **64.8** | 26.3 |
| **ChebGP** [30] | **81.1** | 64.9 | **82.4** | **69.1** | 79.7 | 66.5 | 28.8 | 31.8 |
| **WGGP** [30] | 78.4 | 67.6 | **84.3** | 64.5 | **84.7** | **70.8** | **58.3** | 32.6 |
| **TGGP** (ours) | **81.1** | **75.7** | **82.4** | 63.2 | 80.3 | 70.5 | 53.8 | **34.9** |
| **GGP-X** [42] | 78.4 | 56.8 | 60.8 | **77.6** | 84.7 | 75.6 | **71.9** | 23.0 |
| **WGGP-X** [30] | 81.1 | 75.7 | 84.3 | 65.6 | **87.5** | **76.8** | 61.3 | 32.7 |
| **TGGP-X** (ours) | **86.5** | **81.1** | **86.3** | 63.4 | 83.8 | 76.7 | 54.2 | **36.9** |

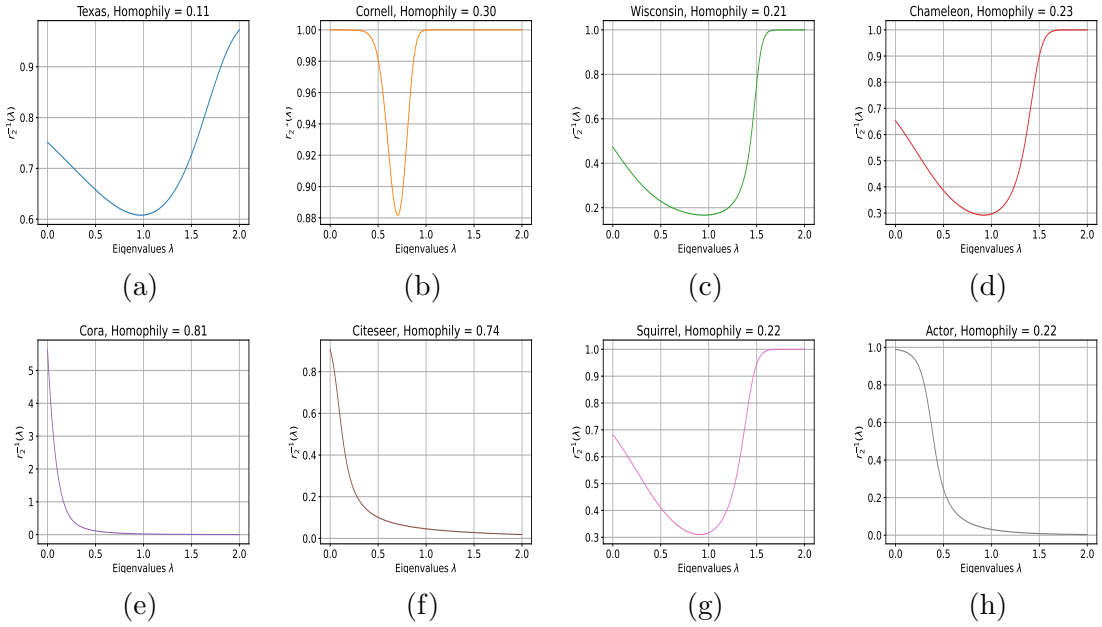
Table 6.2: Classification percentage accuracy on real-world datasets. The **-X** versions trains on both the training and validation sets. The two best performing models and the highest **-X** version are highlighted.

alone in Fig. 6.1b, indicating the graph is more informative than the feature data. This is due to the graph's ability to measure the inner and outer-most parts of the roll shape to be the furthest apart, and therefore detect the linearly increasing pattern of the node labels. **TGGP** on the other hand combines the two and was the best in terms of MAE and reproducing the pattern of the ground truth. Current graph GP models **GGP** and **WGGP** rely on the base kernel corresponding to the **GP**, this base kernel is the commonly used RBF, which does not capture the non-Euclidean nature of the data. Because the graph transforms $\mathbf{P}$ and $\mathbf{W}$ are multiplied with the base kernel, this induces the "and" relationship of product kernels to only take large values when both the base kernel and the graph transforms are large, and as the base kernel does not pick up any particular patterns in the data it limits the amount of influence of the graph information.

We further provide the MAEs for different training sizes and over multiple realizations of the Swiss-roll in Fig. 6.2. As the training points increase, the **GP** starts to show its predictive power, which indicates the feature data is now more informative. In this case, only **TGGP** continues to perform as well as **GP** as it has the ability to weigh the graph and feature data within the kernel, while **GGP** and **WGGP** only contains a single variance over both sets of information and therefore forces the graph convolution onto the feature kernel.

### 6.4.2 Real World Experiments

We lastly perform semi-supervised classification on a number of real-world graph datasets. We take $\mathbf{A}' = \frac{1}{2}(\mathbf{A} + \mathbf{A}^\top)$ as the adjacency matrix if the graph is directed to ensure eigenvalues are real. The $r_2$ function is chosen as a degree 4 polynomial,




(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 6.3: Polynomials $r_2^{-1}(\lambda)$ learnt on the real-world dataset, interpreted as graph filters. High homophily datasets (e) and (f) have clear low-pass natures, while the other 6 are low homophily and generally contain high-pass elements, except (h) which behaves more like a low-pass but with a slower decay.

and a Matérn12 is used for $\mathbf{K}_1$. We use the public split of training, validation, and testing sets, as sometimes the training set is smaller so this will showcase our model's ability to learn on less data. Hyperparameters in **TGGP** are inferred on the training labels, leaving the validation set usually required for GNN hyperparameter tuning unused. Therefore, much like the GP models of [42, 30], we also include the **-X** version where we include the validation labels in the training. As we are in a classification setting where uncertainty measure is less useful, we simply make use of a multi-output regression GP to predict continuous values for each class, followed up taking the argmax to obtain the prediction class. The classification accuracies are shown in Table 6.2, where we compared mainly against graph GP models as well as popular GNNs. **TGGP** was the best performing model in a number of datasets while still being amongst the top in others. **TGGP-X** also performed similarly compared to other **-X** models.

A way of measuring the smoothness of the data is through the homophily ratio, and current graph-based models are known to perform well when the ratio is high such as Cora and Citeseer, on which **TGGP** performed competitively. The other datasets exhibited relatively low ratios, and it was clear that none of the baseline models performed consistently well across all these datasets. **TGGP**, on the other hand, made use of the flexible polynomial to pick up non-smooth elements in the

data, as a result it was the only graph GP model that consistently improved on the **GP** which does not make use of the graph in any way.

The $r_2$ element in the transductive kernel can provide information on the smoothness nature over the graph. In particular, $r_2^{-1}(\lambda)$ takes the form of a graph filter (2.6), and as we chose to freely optimize the shape of the polynomial, thus adapting to the smoothness profile of the data. In Fig. 6.3 we display these filters for each of the real-world datasets. Firstly, we observe that in Fig. 6.3e and 6.3f, the high homophily of Cora and Citeseer lead to a low-pass filter, confirming that high homophily generally indicates existance of low-frequency elements. The more interesting behaviours are therefore the other 6 plots in Fig. 6.3, which are generally of low homophily. In such data the labels are less smooth and this leads to less predictable filter shapes, but the most notable pattern is the existence of high-pass information, indicating the existence of high frequency elements in the data. The exception is Actor (Fig. 6.3h) which resembles a low-pass but with a noticeably slower decay, thus still making it somewhat different to the other high homophily datasets.

## 6.5 Conclusion

We have proposed an extensive definition of kernels on graphs that can take into account both graph information and node feature data for semi-supervised learning. Using the graph inherently makes the kernel transductive, allowing the model to learn on the test information and better capture the distribution of the full dataset. Our proposed kernel is generic, and we have shown that numerous kernels and Laplacian-based models are instances of our design. Our model showed a superior ability to learn on synthetic data, particularly when operating on datasets with a small number of training points and exhibit a highly non-Euclidean nature. On real-world graphs, we also showed competitive performances in classification accuracy against various graph GP and GNN models.

# Chapter 7

# Sheaf Laplacian Gaussian Processes

The performance of node level classification can also be improved by modifying the input graph, this however is an under-explored approach particularly with GPs. Recently, learning with sheaves has become a trend for semi-supervised problems on graphs, lifting the graph into a higher order and more generalized structure that can account for less smooth data. On a sheaf, nodes and edges are assigned vector spaces that can encode more descriptive connections, while the sheaf Laplacian has also shown higher separation power compared to the original graph Laplacian. Thereby, in learning a sheaf over the graph, we are also modifying the graph to aid the predictions made by the model. For this piece of work, we look to incorporate a neural network to learn the optimal sheaf as part of the GP training, to design a sheaf Laplacian GP that makes use of the beneficial properties of sheaves.

## 7.1 Introduction

In the studies of GNNs, there has also been rigorous testing of the models to find when they perform well and when they become limited. Recently, two particular problems observed have been the focus of multiple studies: poor handling of heterophilic graphs [45], and over-smoothing [118]. In order to overcome such problems, researchers have looked into different approaches such as better utilization of larger neighbourhoods, as well as spectrally defined functions (e.g. [119, 120]) to build more powerful predictive models. Amongst the various approaches proposed, one new trend has revolved around the use of sheaves [26].

Sheaves come from a sub-field of algebraic topology and geometry and can be con-

sidered as a generalization of graphs [26]. Much like the advantages that graphs offer, sheaves also model relational data, but with the addition of vector spaces assigned to the nodes and edges and restriction maps between them to encode higher dimensional embeddings. The previously mentioned problems with GNNs can be ascribed to the given graph being too simple, as homophilic and heterophilic connections are all represented by the same type of edges. Therefore, there is the need to expand the information embedded in the edge connectivity in order to capture more complex relationships between nodes. Recently, GNNs operating on sheaves have consistently shown superior results in node classification particularly on graphs with relatively low homophily compared to ones operating on graphs [121, 27, 122]. The work of [27] also provided extensive theoretical analysis proving there is a higher separation power from the sheaf Laplacian compared to the graph Laplacian, and sheaf Laplacian operators form the building blocks for distinguishing nodes in each sheaf neural network layer. Furthermore, even one-dimensional sheaves can offer benefits, with the edges modified in a similar manner to attention on graphs. Though the sheaf structures may not be available in data in general, we can take note from the work in [27] that demonstrated the sheaf can be learnt as part of the model training, thus providing a data driven approach of modifying the input graph to improve the model predictions.

In our previous studies of graph GPs, we focused on the handling of less smooth data from a spectral perspective, where the model is often designed in the graph spectral domain to incorporate higher frequency elements (see Chapters 4 & 5). While heterophilic data are considered non-smooth, they do not always have a direct spectral interpretation. Generally, node labels that have high homophily tend to exhibit a low-pass nature, but the relationship between low homophily and high frequency elements is less clear-cut, and it may be possible that the information present in the spectral profiles alone cannot fully describe the heterophilic labels. Therefore, there is the need to tackle graphs of various homophily levels in a different way, and the sheaves perspective provides a novel direction in the form of modifying edges. In this chapter we look to incorporate sheaf Laplacian learning through a neural network as part of the kernel, this incorporates an additional learning aspect in the form of modifying the input graph as part of the training of a GP. This could be particularly useful in heterophilic graphs where nodes are more connected with differently labelled nodes, and the ability to modify the edge weights can help steer the model to focus on nodes with the same label. This approach is comparable to the work that integrates neural networks within GP kernels such as [52, 50, 51], however, our use of the neural network leads to a clear interpretation as learning of

the sheaf edge connections.

A notable design with current graph GPs such as [25, 53] as well as Chapter 5 & 6 is that the graph information is induced from the Laplacian, an element that we look to continue with. When working with sheaves, the sheaf Laplacian instead becomes the operator that governs the model setup. The dimension of the sheaf Laplacian is determined by the dimension of the vector spaces equipped on the sheaf edges (and the number of nodes), and although there are benefits to using higher dimensional sheaves, keeping the embeddings as scalars ensures the sheaf Laplacian has the same dimension as the graph Laplacian, making for an easy transition from graph to sheaf GPs. In this work, we thus look to incorporate sheaf learning through a linear layer as demonstrated in [27] to infer a one-dimensional sheaf Laplacian that is fed directly into a graph GP, replacing the previously graph Laplacian dependent kernel functions. In addition to the performance improvements this offers, we also analyze the resulting sheaf learnt from training, to show the difference that one-dimensional sheaf learning makes on the graph homophily, and further explain how the edge modifications are steering the model towards predicting the correct label.

## 7.2 Background

### 7.2.1 The Graph Laplacian

The graph Laplacian is often specified directly by the difference between the diagonal degree matrix and the adjacency matrix. However, it can be more formally defined through the notion of gradients, which measures the difference in the function values between two nodes. Gradient operators on graphs first require an ordering of nodes to determine the direction of the difference operator over each edge, once this is chosen, this operator $\Psi$ can be applied to any data over the nodes $\mathbf{x}$ as

$$(\Psi\mathbf{x})_{e=(v,u)} = x_v - x_u, \tag{7.1}$$

for $x_v, x_u \in \mathbf{x}$ corresponding to the data on node $v$ and $u$ respectively, and the node that comes first in the ordering is given the positive sign. This operator $\Psi$ is also known as the *incidence matrix*, with each row representing an edge in the graph and each column representing a node. The entries in each row consist of 1 and -1 for the two nodes connected by the edge, and zero everywhere else, with the sign determined by the ordering of the nodes. An example of $\Psi$ can be found in Fig. 7.1 for a simple small graph.

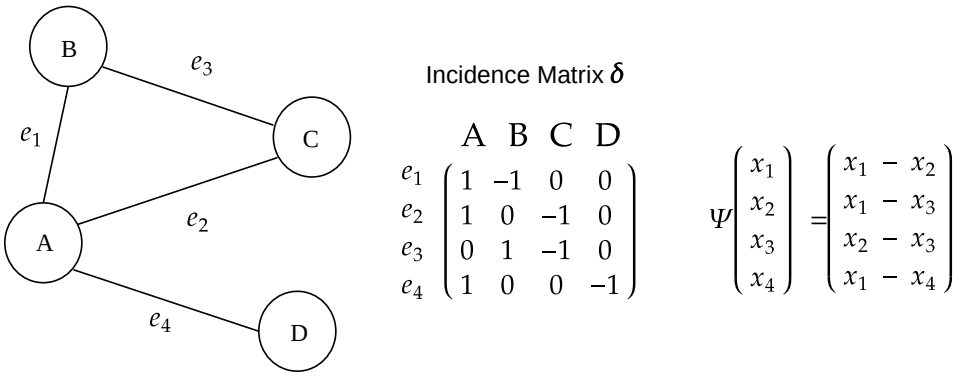



Figure 7.1: Incidence matrix of a graph representing the gradients $\Psi$, and how this is applied to the node data $\mathbf{x} = (x_1, x_2, x_3, x_4)^\top$

The graph Laplacian is then defined as the divergence of the gradient, this is computed by applying the transpose of $\Psi$, leading to

$$\mathbf{L} = \Psi^\top \Psi = \mathbf{D} - \mathbf{A}. \tag{7.2}$$

When applied to the node data, this leads to

$$(\mathbf{L}\mathbf{x})_v = \sum_u \mathbf{A}_{vu}(x_v - x_u). \tag{7.3}$$

Lastly, the normalized graph Laplacian can be computed as

$$\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} \tag{7.4}$$

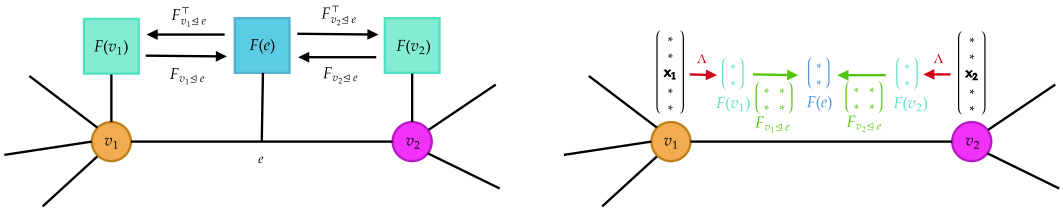
where we note $\mathbf{D}$ contains the diagonal terms of $\mathbf{L}$.

### 7.2.2 The Sheaf Laplacian

A sheaf can be defined on top of a graph by assigning vector spaces to the nodes and edges, and linear maps between the spaces. We present here the formal definition as found in [27, 122, 121]: a cellular sheaf $(\mathcal{G}, F)$ is defined on a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of

- A vector space $F(v)$ for each $v \in \mathcal{V}$, called *node stalks*.
- A vector space $F(e)$ for each $e \in \mathcal{E}$, called *edge stalks*.
- A mapping $F_{v \trianglelefteq e} : F(v) \to F(e)$, called *restriction maps*, for each node-edge pair $v \trianglelefteq e$ ($e = (v, u)$ connecting node $u$ and $v$).

We visualize this structure in Fig. 7.2a, where one can see the vector space $F(v)$




(a) (b)

Figure 7.2: (a) Illustration of a single sheaf edge $e$ between node $v_1$ and $v_2$. (b) $d = 2$ example: features $\mathbf{x}_i$ are mapped to the node space $F(v)$ through a linear layer $\Lambda$, the feature maps $F_{v \trianglelefteq e}$ then transform this to the edge space $F(e)$.

generally refers to the space relating to the node feature data. Meanwhile, $F(e)$ is a common space where two connected nodes "interact", allowing the encoding of different connections depending on the values taken in the edge space. When working with graphs alone, edges are generally pre-defined by edge weights, but the addition of restriction maps that come with defining a sheaf allows for edges to be varying in value and operate in potentially higher dimensions chosen by the user, (see Fig. 7.2b for an example). Dimensions of $F(e)$ and $F(v)$ are often assumed as equal and set to pre-chosen $d$, node feature data would be fed through a linear layer and mapped into the space $F(v)$. The restriction maps are defined as matrix transformations between $F(v)$ and $F(e)$, together with the node and edge stalks they have interpretations of opinion dynamics as explained in [123], where private opinions fall on the node stalks while the edge stalks act as the space in which these opinions are expressed publicly. The restriction maps go towards finding an agreement between the two nodes, and can be interpreted as explaining the difference between the node's public and private opinions.

The restriction maps are used to construct the sheaf Laplacian, and this governs the sheaf much like Laplacian operators on graphs. This definition will be integral to building a sheaf GP by adapting graph GPs that are currently built on the graph Laplacian. To define a $d$-dimensional sheaf Laplacian, we first define the gradient operator $\Psi$ as

$$(\Psi \hat{\mathbf{X}})_e = F_{v \trianglelefteq e} \hat{\mathbf{x}}_v - F_{u \trianglelefteq e} \hat{\mathbf{x}}_u \tag{7.5}$$

for transformed feature matrix $\Lambda(\mathbf{X}) = \hat{\mathbf{X}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_N)$ where the linear layer $\Lambda$ maps each $\mathbf{x}_i$ to dimension $d$, the operator $\Psi$ is therefore the sheaf version of the incidence matrix. We can interpret $\hat{\mathbf{x}}_v$ as the private embedded opinion of node $v$, while $F_{v \trianglelefteq e}$ with $e = (v, u)$ represents the change from private to public when

interacting with node $u$, the combination $F_{v \trianglelefteq e}\mathbf{x}_v$ then represents node $v$'s public opinion. The sheaf Laplacian is then defined by the matrix product $\mathbf{L}_F = \Psi^\top \Psi$ and when this is applied to the data matrix we get

$$(\mathbf{L}_F \hat{\mathbf{X}})_v = \sum_{v,u \trianglelefteq e} F_{v \trianglelefteq e}^\top (F_{v \trianglelefteq e}\hat{\mathbf{x}}_v - F_{u \trianglelefteq e}\hat{\mathbf{x}}_u). \tag{7.6}$$

As a Laplacian, the matrix will be positive semi-definite due to the inner matrix product of the gradient operator and has similar properties to the graph Laplacian.

When $d$ is chosen to be higher dimensional, the sheaf Laplacian will have block matrices in the positions of non-zero values of the graph Laplacian. The block matrices are also computed from the restriction maps, with firstly the diagonal computed as

$$\mathbf{L}_{F_{vv}} = \sum_{v \trianglelefteq e} F_{v \trianglelefteq e}^\top F_{v \trianglelefteq e} \tag{7.7}$$

while the non-diagonal block matrices are defined as

$$\mathbf{L}_{F_{vu}} = -F_{v \trianglelefteq e}^\top F_{u \trianglelefteq e}. \tag{7.8}$$

Finally, we can obtain the normalized sheaf Laplacian as

$$\mathbf{L}_S = \mathbf{D}_F^{-\frac{1}{2}} \mathbf{L}_F \mathbf{D}_F^{-\frac{1}{2}} \tag{7.9}$$

for $\mathbf{D}_F$ defined to be the block diagonals of $\mathbf{L}_F$.

If the vector spaces $F(v)$ and $F(e)$ are set to $\mathbb{R}$ ($d = 1$), this produces a sheaf Laplacian that is the same size as the graph Laplacian, but the difference being in the edge weights are modified by the restriction maps. Gaussian processes (GPs) and kernels defined on graphs typically make use of the graph Laplacian, and so a choice of $d = 1$ would make this compatible to be fed into a GP kernel for graph data. The effect of lifting to a sheaf is therefore similar to performing attention on the graph [124, 28], where each edge is modified based on the features of the two nodes it connects, and this can weaken or strengthen the connection depending on how much the two nodes are alike. Thus, a sheaf even at one dimension can still aid the prediction by acting as a graph modifier during training.

Another solution to the restriction maps can be found through vector diffusion maps, as explained in [122] these mappings can be computed by learning orthogonal transformations similar to on point clouds as described in [125]. Their construction

relies on the "manifold assumption", specifying that even though data lives in a high-dimensional space $\mathbb{R}^p$, the correlation between dimensions suggests that in reality, the data points lie on a $d$-dimensional Riemannian manifold $\mathcal{M}^d$ embedded in $\mathbb{R}^p$ with $d \ll p$. For a set of points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \in \mathbb{R}^p$, $\mathcal{M}^d$ has tangent spaces $T_{\mathbf{x}_i}\mathcal{M}$ at each point (that can be computed for instance via PCA), and any mechanism that can transform vectors between $T_{\mathbf{x}_i}\mathcal{M}$ and $T_{\mathbf{x}_j}\mathcal{M}$ would correspond to $F_{v_i \trianglelefteq e}^\top F_{v_j \trianglelefteq e}$, also called parallel transport. The algorithm is described in more details in [122], while Fig. 2.1 in [125] provides clear visualizations of how tangent spaces can form transport maps.

## 7.3 Sheaf Laplacian Learning and Gaussian Processes

The work of [27] demonstrated the ability to learn the sheaf through an MLP to model the mappings $F_{v \trianglelefteq e}$. We therefore propose a similar framework, but feeding the MLP's output into the GP kernel and thus the neural network weights are inferred as part of the training through optimizing the marginal log-likelihood of the GP. The MLP from node $v$ to edge $e = (u, v)$ will take as inputs the features on the two nodes connected by the edge $e = (v, u)$

$$\Phi : \mathbb{R}^{2k} \to \mathbb{R}^{2d} \to \mathbb{R}^{d \times d}, \tag{7.10}$$

$\Phi$ therefore encompasses both the mapping of both features $(\mathbf{x}_v, \mathbf{x}_u)$ into $F(v)$ in a hidden layer, and outputs the restriction maps as $d \times d$ matrices. Each layer is defined as

$$\text{hidden layer: } \begin{pmatrix} \hat{\mathbf{x}}_v \\ \hat{\mathbf{x}}_u \end{pmatrix} = \sigma\left( \mathbf{V} \begin{pmatrix} \mathbf{x}_v \\ \mathbf{x}_u \end{pmatrix} \right) \tag{7.11}$$

$$\text{output layer: } F_{v \trianglelefteq (v,u)} = \sigma\left( \mathbf{W} \begin{pmatrix} \hat{\mathbf{x}}_v \\ \hat{\mathbf{x}}_u \end{pmatrix} \right) = \Phi \begin{pmatrix} \mathbf{x}_v \\ \mathbf{x}_u \end{pmatrix}. \tag{7.12}$$

The output of the linear layer is $d \times d$ matrix valued, and with the choice of $d = 1$ the product $F_{v \trianglelefteq (v,u)}^\top F_{u \trianglelefteq (v,u)}$ leads to a scalar terms in the sheaf Laplacian. The restriction maps $F_{v \trianglelefteq (v,u)}$ for all node-edge pairs share the weights $\mathbf{W}$ of the linear layer, but will depend on the node features and the order of $\mathbf{x}_v$ and $\mathbf{x}_u$, thus the restriction map of node $u$ to edge $e$ would be computed by

$$F_{u \trianglelefteq (v,u)} = \Phi(\mathbf{x}_u, \mathbf{x}_v) \tag{7.13}$$

This makes it possible to learn non-symmetric mappings for $F_{v \trianglelefteq (v,u)}$ and $F_{u \trianglelefteq (u,v)}$, which has been found to improve the sheaf's representation [27].

### 7.3.1 Sheaf Laplacian Gaussian Processes

We take inspiration from Graph Laplacian-based GPs proposed in Chapter 5 & 6, to define the sheaf Laplacian GP, with an integrated neural network $\Phi$ whose weights $\mathbf{V}$ and $\mathbf{W}$ are optimized as part of the GP training. The sheaf Laplacians in this section are therefore all specified by $\Phi$ with entries determined by the node features as defined in the previous section. Furthermore, we take the designs in Chapter 5 which also shares similarity to [42], as our model of choice because of shared characteristics with the sheaf neural network setting [27] in terms of how the Laplacian is utilized.

Following the analysis of [27], the multiple layers of the sheaf neural network are simulating a diffusion process $\dot{\mathbf{X}}_t = -\mathbf{L}_S \mathbf{X}_t$ for derivatives with respect to $t$, which has the solution $\mathbf{X}_t = \exp\{-t\mathbf{L}_S\}$. The sheaf neural network architecture is based on the Euler discretization of this exponential with unit step size, which becomes $\mathbf{X}_{i+1} = (\mathbf{I} - \mathbf{L}_S)\mathbf{X}_i$. The message passing in each sheaf neural network layer is defined as $\mathbf{X}_{i+1} = \sigma((\mathbf{I} - \mathbf{L}_S)\mathbf{X}_i\mathbf{W})$, where the only difference is in the addition of weights $\mathbf{W}$ and an activation function $\sigma$. Having multiple layers is then replicating the Euler discretization of the diffusion over multiple steps.

Amongst the available graph GPs, we choose the one that has the most parallel with the sheaf neural network setup. We note the sheaf Laplacian is applied to the neural network layer as a linear transform $\mathbf{X}_{i+1} = r(\mathbf{L}_S)\mathbf{X}_i$ where a function of the sheaf Laplacian is represented generally by $r$. The GPs presented in Chapter 5 are therefore the most similar due to the usage of the graph Laplacian, and $r(\mathbf{L}_S)f$ is applied as a linear transform. Thus, we use the following prior on the GP

$$f \sim \mathcal{N}(0, \mathbf{K}) \tag{7.14}$$

$$\implies r(\mathbf{L}_S)f \sim \mathcal{N}(0, r(\mathbf{L}_S)\mathbf{K}r(\mathbf{L}_S)^\top). \tag{7.15}$$

If we were to match the sheaf neural networks, this would correspond to choosing $r(\mathbf{L}_S) = \exp(-\alpha\mathbf{L}_S)$ where we can add a hyperparameter $\alpha$. The Euler discretization $r(\mathbf{L}_S) = \mathbf{I} - \mathbf{L}_S$ does not work however as we do not have multiple layers, and a single layer can only induce linear sheaf information. Using this discretization may be possible if it is used in each layer of a deep GP [11], replicating the diffusion process simulated through the multiple layers of a sheaf neural network [27], but we

will leave this as future work.

A simpler option is first to note that diffusion has smoothing low-pass properties, so we can choose another low-pass function for a similar effect to the smoothing of diffusion, one popular choice being the filtering matrix

$$r(\mathbf{L}_S) = (\mathbf{I} + \alpha \mathbf{L}_S)^{-1}. \tag{7.16}$$

This is less computationally intensive compared to the matrix exponential of diffusion, and utilizes the sheaf information in a non-linear way, making this more flexible than the Euler discretization.

Lastly, for scalability to larger graphs, we can switch to a Chebyshev polynomial $T_k(x)$ to define the function $r(\mathbf{L}_S)$ in order to avoid the costly inversion of (7.16)

$$r(\mathbf{L}_S) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}_S}) \tag{7.17}$$

where $\tilde{\mathbf{L}_S} = \mathbf{L}_S - \mathbf{I}$, and the $\theta_k$'s become hyperparameters to be learnt. The degree $K$ determines the size of neighbourhood information to use, practically we found $K = 4$ leads to a good performance.

The main difference between sheaf Laplacian Gaussian processes and previous graph GPs is in the modification of edges through the sheaf learning, whereas existing graph GPs for semi-supervised learning of [42] and what we presented in Chapter 4, 5 & 6 as well as other applications [15, 47, 60] all assume the given graph is fixed. On heterophilic graphs, differently labelled nodes are likely to be connected, but in current GP kernels they are aggregated in the same way as nodes with the same labels. With a linear layer to compute the restriction maps, this leads to edges that are now weighted, and the optimal weighting is found with respect to the training loss. This makes it possible to infer a graph with generally larger edge weights connecting nodes with the same label, and smaller weights for differently labelled nodes. Meanwhile, in our previous chapters, we view the handling of heterophily as learning a function in the graph spectral domain to capture any patterns as a result of the non-smooth profiles of the data. However, if we wish to operate in the graph spatial domain, modifying graph edges to increase the homophily ratio represents a suitable spatial approach to improve the model's performance.

| Method | Texas | Cornell | Wisconsin | Chameleon | Cora | Citeseer | Squirrel | Actor | PubMed |
|---|---|---|---|---|---|---|---|---|---|
| # Nodes | 183 | 183 | 251 | 2,277 | 2,708 | 3,327 | 5,201 | 7,600 | 18,717 |
| Hom. Rat. | 0.11 | 0.30 | 0.21 | 0.23 | 0.81 | 0.74 | 0.22 | 0.22 | 0.80 |
| **GCN** [44] | 59.5 ±5.3 | 57.0 ±4.7 | 59.8 ±7.0 | 59.8 ±2.6 | 80.5 ±0.8 | 68.1 ±1.3 | 36.9 ±1.3 | 30.3 ±0.8 | 77.8 ±0.7 |
| **GAT** [28] | 58.4 ±4.5 | 58.9 ±3.3 | 55.3 ±8.7 | 54.7 ±2.0 | 82.6 ±0.7 | **72.2 ±0.9** | 30.6 ±2.1 | 26.3 ±1.7 | 76.7 ±0.5 |
| **ChebNet** [76] | 77.3 ±4.1 | 74.3 ±7.5 | 79.4 ±4.5 | 55.2 ±2.8 | 78.0 ±1.2 | 70.1 ±0.8 | 43.9 ±1.6 | 34.1 ±1.1 | 69.8 ±1.1 |
| **MLP** | 69.8 ±5.4 | 73.7 ±3.7 | 78.1 ±3.4 | 46.8 ±2.0 | 43.3 ±3.5 | 41.8 ±3.6 | 30.0 ±0.7 | 32.8 ±0.8 | 32.7 ±0.7 |
| **LP** [116] | 37.8 | 21.6 | 23.5 | 44.5 | 71.3 | 49.9 | 32.7 | 22.4 | 71.2 |
| **GP** [42] | 78.4 | 73.0 | 78.4 | 46.1 | 60.8 | 54.7 | 34.4 | **34.9** | 71.5 |
| **GGP** [42] | 78.4 | 62.1 | 60.8 | **73.5** | 80.9 | 69.7 | **64.8** | 26.3 | 77.1 |
| **ChebGP** [30] | **81.1** | 64.9 | **82.4** | **69.1** | 79.7 | 66.5 | 28.8 | 31.8 | 77.2 |
| **WGGP** [30] | 78.4 | 67.6 | **84.3** | 64.5 | **84.7** | 70.8 | **58.3** | 32.6 | **78.4** |
| **TGGP** [31] | **81.1** | **75.7** | **82.4** | 63.2 | 80.3 | 70.5 | 53.8 | **34.9** | - |
| **SheafGP** (ours) | **81.1** | **75.7** | 80.4 | 64.0 | **83.2** | **72.1** | 48.2 | **35.5** | **80.7** |
| **GGP-X** [42] | 78.4 | 56.8 | 60.8 | **77.6** | 84.7 | 75.6 | **71.9** | 23.0 | 82.4 |
| **WGGP-X** [30] | 81.1 | 75.7 | 84.3 | 65.6 | **87.5** | 76.8 | 61.3 | 32.7 | **90.0** |
| **TGGP-X** [31] | 86.5 | **81.1** | **86.3** | 63.4 | 83.8 | 76.7 | 54.2 | 36.9 | - |
| **SheafGP-X** (ours) | **89.2** | 78.4 | 80.4 | 66.0 | 87.0 | **77.3** | 46.0 | **37.2** | 86.0 |

Table 7.1: Classification percentage accuracy on real world datasets. ***TGGP-X** [31] is not scalable to the biggest dataset PubMed, therefore no values are reported. The two best performing models and the highest **-X** version are highlighted.

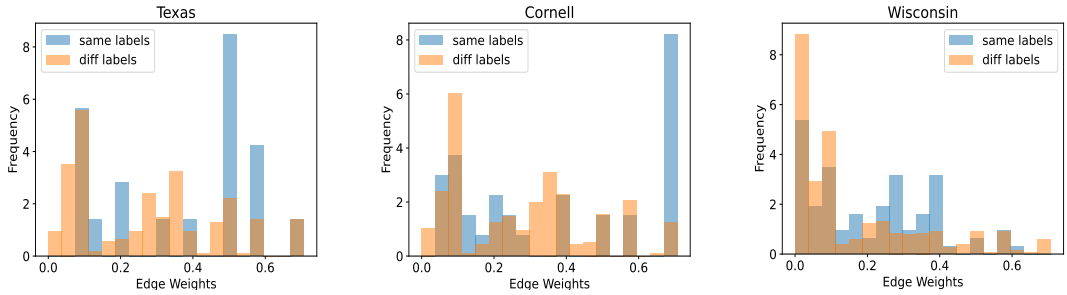
## 7.4 Experiments

### 7.4.1 Graph GPs Setting

We perform semi-supervised classification on a number of real-world graphs, with edge homophily (defined in (3.5)) ranging from 0.11 (very heterophilic) to 0.81 (very homophilic). We use the same splits utilized in Chapter 6, known for using smaller training sets to highlight the advantages of a GP model. Our proposed model is called **SheafGP**, consisting of a single linear layer with tanh activation for learning the sheaf. This makes the neural network hyperparameters determined and will not require tuning, therefore, the validation set is not used unlike other GNN models. We can therefore present **SheafGP-X** that trains on the training and validation set combined replicating the **-X** version of other graph GPs. Table 7.1 shows that **SheafGP** and **SheafGP-X** are always at least competitive compared to the graph GP models of [42, 30, 31], and was the best performing model on multiple datasets.

### 7.4.2 Sheaf Learning & Weighted Homophily Ratio

The sheaf learning element of the model will lead to a graph with modified edge weights. This is the main difference between a sheaf GP and any previous graph GPs, which assume the graph is fixed and the adjacency matrix contains either unit or pre-given edge weights. To understand how the sheaf learning is changing the underlying graph, we examine on the smallest three datasets the distribution of edge weights connecting the same and differently labelled nodes, these are presented in




(a) (b) (c)

Figure 7.3: The distribution of edge weights connecting two nodes of the same labels vs different labels, edges connecting nodes with the same labels are generally larger.

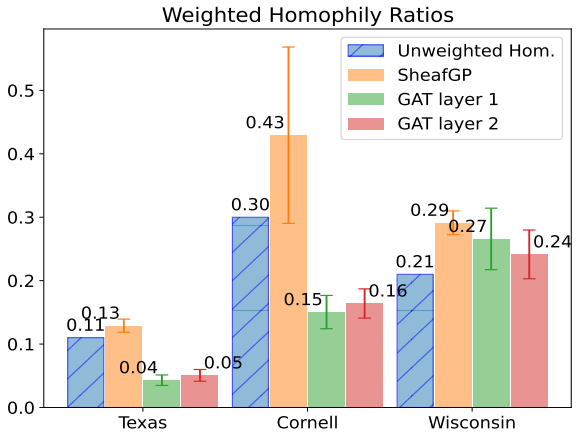



Figure 7.4: Weighted homophily ratios $h_w$ of the edge weights learnt from **SheafGP** and the attention weights from a 2 layer **GAT**, error bars (black lines) are the standard deviations over 10 runs. Unweighted Hom (blue) are the original homophily ratios of the graph introduced in [45].

Fig. 7.3a - 7.3c. In these figures it is noticeable that edges connecting nodes with the same labels are generally given larger weights. A way to measure the quality of these weights is through the homophily ratio, however, the ratio defined in [45] is currently for unit edge weights and thus not suitable for measuring weighted graphs. We can get around this by defining the more general *weighted homophily ratio* as

$$h_w = \frac{\text{sum}\{e : e = \{u, v\} \in \mathcal{E} \text{ and } y_u = y_v\}}{\text{sum}(\{e \in \mathcal{E}\})}. \tag{7.18}$$

The weighted homophily ratio takes into account the edge weight instead of counting the number of edges, thereby providing a measure of not just the number of nodes with the same label, but also how much influence they have; the original graph homophily ratio we will now refer to as the *unweighted homophily ratio*. Meanwhile,

| Method | Texas | Cornell | Wisconsin | Chameleon | Cora | Citeseer | Squirrel | Actor | PubMed |
|---|---|---|---|---|---|---|---|---|---|
| # Nodes | 183 | 183 | 251 | 2,277 | 2,708 | 3,327 | 5,201 | 7,600 | 18,717 |
| Hom. Rat. | 0.11 | 0.30 | 0.21 | 0.23 | 0.81 | 0.74 | 0.22 | 0.22 | 0.80 |
| **Diag-NSD** [27] | 85.7 ±7.0 | 86.5 ±7.4 | 88.6 ±2.8 | 68.7 ±1.7 | 87.1 ±1.1 | 77.1 ±1.9 | 54.8 ±1.8 | 37.8 ±1.0 | 89.4 ±0.4 |
| **O(d)-NSD** [27] | 86.0 ±5.5 | 84.9 ±4.7 | 89.4 ±4.7 | 68.0 ±1.6 | 86.9 ±1.1 | 76.7 ±1.6 | 56.3 ±1.3 | 37.8 ±1.2 | 89.5 ±0.4 |
| **Gen-NSD** [27] | 83.0 ±5.1 | 85.7 ±6.5 | 89.2 ±3.8 | 67.9 ±1.6 | 85.7 ±6.5 | 76.3 ±1.7 | 53.2 ±1.3 | 37.8 ±1.2 | 89.3 ±0.4 |
| **GCN** | 55.1 ±5.2 | 60.5 ±5.3 | 51.8 ±3.1 | 64.8 ±2.2 | 87.0 ±1.3 | 76.5 ±1.4 | 53.4 ±2.0 | 27.3 ±1.1 | 88.4 ±0.5 |
| **GAT** | 52.2 ±6.6 | 61.9 ±5.1 | 49.4 ±4.1 | 60.3 ±2.5 | 86.3 ±0.5 | 76.6 ±1.2 | 40.7 ±1.6 | 27.4 ±0.9 | 87.3 ±1.1 |
| **MLP** | 80.8 ±4.8 | 81.9 ±6.4 | 85.3 ±3.3 | 46.2 ±3.0 | 75.7 ±2.0 | 74.0 ±1.9 | 28.8 ±1.6 | 36.5 ±0.7 | 87.2 ±0.4 |
| **SheafGP** (ours) | 84.3 ±3.8 | 75.7 ±4.7 | 85.7 ±3.7 | 63.4 ±2.0 | 87.3 ±3.5 | 77.7 ±1.6 | 44.1 ±2.1 | 36.6 ±1.0 | 89.3 ±0.8 |

Table 7.2: Classification percentage accuracy on real world datasets training on the 10 splits used in [27, 126].

a natural comparison for our **SheafGP** is against **GAT** [28], from which $h_w$ can be computed on the attention weights. We examine this ratio on the three smallest datasets, comparing **SheafGP** against a simple 2 layer **GAT**, with the ratios presented in Fig. 7.4. From this figure, we can see the superior $h_w$ from **SheafGP** on all three datasets not just against both layers of **GAT**, but they are also higher than the unweighted homophily ratios. This to some extent explains the improvement offered from the sheaf learning as neighbouring nodes with the same label are generally given a higher weighting, which firstly increases the weighted homophily ratio, and also steers the classifier towards predicting the correct label and diminishes the influence of connected nodes from different classes.

### 7.4.3 Neural Sheaf Diffusion Setting

We lastly replicate the setting of neural sheaf diffusion (NSD) [27], which also experimented on the same datasets but over 10 chosen training and testing splits (and validation). Here, the training set is sometimes larger than the split in Chapter 6 that we have used so far. The main models that we compare against are the sheaf neural networks of Neural Sheaf Diffusion (NSD) with three different variants each using a different sheaf learning basis, the results over 10 splits are shown in Table 7.2. The NSD models are slightly superior to our **SheafGP** due to their use of higher dimensional sheaves, often ranging from $d = 3$ to 5, therefore leading to higher separation power. Using a higher dimensional sheaf is feasible in a neural network as the Laplacian only enters the model as matrix multiplication with the weights, while the dimensions can be easily modified in the neural network layer. If we are to construct a GP on a sheaf with $d > 1$, the current challenge is firstly in converting the larger Laplacian of size $nd \times nd$ into that of a covariance matrix of size $n \times n$, as well as dealing with the additional scalability of inverting larger matrices that come with GP inferences. However, with $d = 1$, **SheafGP** was able to produce comparable results to NSD, and even matching them on a small number

of datasets (namely Cora, Citeseer, PubMed). Nonetheless, designing a sheaf GP for higher dimensional sheaves will be an important future work.

## 7.5 Conclusion

Modelling graph data as sheaves has already provided many improvements and flexibility compared to operating on the original graph. We have designed a GP for sheaves by adapting graph Laplacian dependent kernels, using in its place a one dimensional sheaf Laplacian learnt through a linear layer feeding into the GP kernel. Incorporating sheaves has led to superior performances on a number of benchmark datasets compared to other graph GPs, while the sheaf learning aspect also produced well modified graphs comparable to attention on the edges. In particular, it is visible that generally larger weights were assigned to edges connecting nodes with the same label, and this has the overall effect of increasing the weighted homophily ratio of the graph. Lastly, against the sheaf neural networks, the sheaf GP was still able to produce comparable results on a number of datasets, despite the neural networks using higher dimensional sheaf Laplacians and benefitting from generally larger training sets. Nonetheless, building GPs on higher dimensional sheaves remains an important and challenging future work.

# Chapter 8

# Conclusion

## 8.1 Summary

Gaussian processes have always been an important element in the machine learning library. However, as data become more complex, GPs also need to adapt in order to learn the irregular nature of the real world. This thesis looked into designing GPs on graphs to expand the possible applications. This is an important step as graph data are becoming increasingly more common, and we observe that information does not fall in the feature data alone but also in the relational structure between data points. We conclude by first summarizing the work presented in this thesis.

In Chapter 4, we developed a novel multi-output GP model for graph signal prediction to model the evolution of signals on a graph. We established a connection between GP regression and filtering on graphs, to give the model a signal processing interpretation. This helped us identify the recurring use of low-pass filters in GP kernels, a limitation as this ignored the higher frequency information in the data. We thus proceeded to learn the optimal filter based on the data by freely optimizing a polynomial in the spectral domain, giving the model the ability to pick up higher frequency information. In doing so, the predictions resembled the data much more in terms of smoothness on the graph, and this was reflected visually, and empirically with the significantly higher posterior log-likelihoods on the test signals. Equally important was the confidence of the predictions, and here the learning aspect also led to much smaller standard deviations indicating the model is more certain with its predictions.

Our focus shifted towards node classification starting in Chapter 5. As GNNs had demonstrated, node level problems were built on graph convolution, and the infor-

mation utilized needed to be over fixed hop neighbourhoods as a level of locality was desired for the optimal performance. Wavelets however, offered a much more flexible definition of convolution by providing a notion of continuous neighbourhoods, and the ability to adjust the node weighting depending on a scale. This suited graph-based models as multi-scale properties were often observed on larger graphs. Moreover, wavelets offered a trade-off in localization between the spatial and spectral domains, as opposed to the graph convolutions currently defined being localized only in one of the two domains; message passing in particular is strictly limited to spatial operations. We designed a GP that was convoluted with the graph wavelet transform, to allow for localized neighbourhood aggregation on multiple scales. Spectrally, this incorporated higher frequency information, while spatially the neighbouring nodes were weighted in a non-linear way for a more refined aggregation.

Amongst the wide availability of graph data, it was evident that nearly all graphs came with node features, and in Chapter 6 we developed a unified kernel to handle such data for a Gaussian process. We followed the theories in reproducing kernel Hilbert spaces to obtain a kernel from regularization, and this provided a more comprehensive measure of similarity between nodes, accounting for any non-linear relationships between the node features and the graph connectivity information. Inherently, building on the graph also made the kernel transductive, allowing the model access to all data inputs during training instead of just the training data (but only the training labels). Thus, the resulting GP more accurately captured the distribution of the data to better inform the labels of the nodes.

Finally, in Chapter 7 we looked into lifting graphs into sheaves, a higher order structure that also described the topological structure of the data. Through lifting to a sheaf, we equipped the graph with additional structures, of which the restriction maps were used to construct sheaf Laplacians with higher separation power for the Gaussian process kernel. We modified the graph's edge weights through a linear layer taking in the node features, and mapping to embeddings on the edges. As nodes in a graph would be inevitably connected to differently labelled nodes as well as ones in the same class, changing the edge weighting in the Laplacian accordingly helped pick out useful nodes in the neighbourhood aggregation, providing a better handling of heterophilic graphs. The restriction maps were found through a neural network, with the weights learnt during the GP training. Therefore, the sheaf learning aspect led to the modification of the input graph during training, providing a similar framework as attention on graphs.

The graph datasets utilized in this thesis are generally considered sparse, and this has been an important element to the model performances. Sparsity allows for more localized information sharing, leading to more distinguishable embeddings. However, when the graph becomes too sparse, we would expect performance to degrade. Our graph GPs have been relatively robust to sparsity, and the performance improvements compared to non-graph cases (e.g. vanilla GPs, MLPs) are generally dependent on the quality of the graph instead of edge density. However, studying the behaviours in models as the input graph becomes sparser is an important analysis of the robustness of GPs, and we leave this as a topic of investigation in future.

Chapters 5 to 7 were all tested on semi-supervised classification, and the datasets used are described in detail in Appendix A. Each chapter focused on a specific approach to improving the classification accuracy to a level that was competitive against many state-of-the-art models on graphs. We now briefly discuss how these approaches compare with each other: Chapter 5 focused on graphs with multi-scale structure complemented by the use of wavelets, and therefore the improvements were more clear-cut when the graph exhibited such properties; this was also generally more common when the graphs were larger. Chapter 6 presented a more generalized definition of kernels on graphs for which Chapter 5 & 7 fell under. The transductive kernel was the most flexible depending on how the user chose each component in the definition. The kernel design we used in this chapter had additional novel components compared to Chapter 5 & 7, and the improvements came from utilizing the global distribution of the full input data. The limitation was in computational complexity and therefore this model was the slowest to train out of the three. Chapter 7 incorporated additional learning on the edge weights through a neural network architecture, and was debatably the model with the highest learning power. However, training the model required the highest memory complexity of the three due to the additional neural network weights in the model, as a result running on larger graphs was only feasible on processors with sufficient memory.

## 8.2 Potential Future Work

We have developed a diverse series of Gaussian processes on graphs and demonstrated comparable and often superior performances against many state-of-the-art models. However, graphs can represent a wide range of data and this means we have only touched the surface of the possible applications. Social media, financial exchanges and transactions, images, and text are some example application domains, and an immediate future work is to tackle the problems relating to such data

to demonstrate the wider variety of applications of graph GPs.

One element that our technical works have in common is the assumption that a standard graph is available, generally characterized by positive edge weights and each edge connecting only two nodes. However, in some modelling problems, using different types of graphs may be more appropriate, in particular the ones that are of interest to us are directed graphs, signed graphs [127] and hypergraphs [128, 129]. The former offers encoding of different (positive and negative) edge relationships between nodes, while the latter incorporates edges connecting multiple nodes. These provide different information for our graph GPs to handle, and can significantly change the behaviours of spectrally designed functions that are present in many of our models in some form. Although translating to more complex graphs may not be trivial, studies in signal processing such as [130, 131] may provide the initial guide.

As more focused directions, firstly an important aspect to address is the scalability of the transductive kernel we developed in Chapter 6. Evaluation of this kernel at any number of points will involve a large inversion of $\mathcal{O}(N^3)$ complexity for the number of nodes in the graph $N$, this is manageable for graphs of reasonable sizes but will be too slow when $N > 10,000$. $\mathcal{O}(N^3)$ is often encountered when using GPs, but unlike typical GPs however, this complexity is in the prior and not in computing the posterior, and therefore common scalability methods such as sparse variational GPs [90] are not applicable in this situation. One direction we are currently looking into is in adapting the iterative method of [116] to evaluate the kernel without the inversion, but we aim to implement this while still allowing for the hyperparameters to be optimizable.

Another important future work is related to the use of sheaves in Chapter 7. The difference between sheaf neural networks [27, 122] and our sheaf GP is that we made use of sheaves with one-dimensional edge stalks/spaces, this ensured the sheaf Laplacian is the same dimension as the covariance matrix to make it feasible to be part of the GP kernel. The neural networks on the other hand can easily change dimensions using a linear layer, and practically the edge space dimensions ranged from 3 to 5. This currently makes the sheaf neural networks slightly superior due to the higher separation power of the higher dimensional sheaves. However, the difference is small, and our sheaf GP still occasionally outperformed the sheaf neural networks. In adapting graph GPs to higher dimensional sheaves, the challenge is in the covariance matrices, as they need to be of size $n \times n$, while a kernel on a $d$ dimensional sheaf Laplacian will result in a covariance matrix of size $dn \times dn$. Currently, we are still looking for ways to do GP inference that can overcome this

dimension mismatch.

Lastly, we heavily made use of kernels on graphs defined in [25] throughout the thesis. However, this definition is not complete, as it has been established in [53, 55] that the variances on the nodes are non-uniform in this definition, and they depend on the return time of random walks on graphs (which are generally related to the number of neighbours each node has). Thus, there is a need to find a definition of kernels on graphs such that there are uniform variances across the nodes. If the kernel is to continue to be Laplacian based, then some edge scaling is required to offset the variance difference from varied number of neighbours. This is essential for any application where the uncertainty plays an important role to properly extending the use of GPs on graphs. One particular example is Bayesian optimization on graphs where one would rely on the confidence intervals of the GP to inform the search from the acquisition function.

## 8.3 Perspectives

We end with some perspectives relating to the major topics of this thesis.

**Why GPs on graphs?** The advantages of GPs over neural network-based models have been discussed in this thesis, and a graph GP will continue to offer the same qualities such as learning on fewer data and predicting with confidence intervals. While GPs are readily available when the input space is Euclidean, the options are much more limited on graphs, and there is the need for a wider range to choose from in order to handle the irregular graph data that are becoming increasingly common. Indeed, graph data can now be naturally existing, constructed based on the problem, or used to approximate non-Euclidean data such as surfaces and manifolds; and graph GPs can be applied to any of these problem settings. Meanwhile, using kernels on graphs makes the learning transductive, which is an alternative to the inductive assumption of regression-based models. Transductive learning however is a much less explored approach in machine learning, and can often be more suitable in a setting where the training and testing data are readily available from the start. Overall, using graph information is similar to adding a prior to the model, and this we identify more with the Bayesian approach, making GPs more in line with modelling on graphs.

**Why spectral methods on graphs?** A heavily touched element of our work is to look to the graph spectral domain to design our models, as opposed to the commonly used designs that act spatially. Spectrally defined functions have a strong

mathematical foundation, providing a notion of frequency and interpretability from a signal processing perspective. This makes the smoothness quality of the model well defined and easy to control. As people take on more challenging datasets, one property the majority of them have in common is that the data become less smooth. Dealing with this property spectrally can be as straightforward as choosing a suitable filter, while spatially it is much harder to design models that are not prone to problems such as over-smoothing. Although there are times when there may not be any particular patterns in the spectral domain for the model to pick up, one should note that we are not restricted to the Fourier basis, and it is possible that other basis functions, such as wavelets, windowed Fourier, etc., can simplify the problem. Hence, equally important is the openness to consider other basis and their benefits.

**Can graph GPs replace GNNs? And what about GPs vs deep learning?** GPs and deep learning models are both reliable options when it comes to predictive performances, on graph data and more generally. This thesis has demonstrated the superiority of a graph GP in performance when training data is small, the availability of uncertainty with the predictions, as well as the various ways to interpret the model. However, in general when the data is in abundance, GPs have not been able to scale as well as neural networks, and in order to handle the larger amount of data some performance is sacrificed through sparse inferences. Meanwhile, instead of comparing the two types of models, there is also the trend of combining GPs and deep learning architectures, either in the form of deep GPs [11, 132], or employing neural networks within the kernel learning [50, 51]. The advantages of such models are generally more "in-between" what GPs and deep learning offer, instead of making the best of both worlds - increased learning power but higher complexity to train. A debate in-between out of the hands of GP and deep learning developers is how generalizable each model is, which not only depends on the inference methods but also the optimizers. In particular, the recent developments of newer optimizers [133, 134] have claimed to find solutions that are more generalizable than the likes of Adam and L-BFGS-B; so far, it is unclear which type of model the newer optimizers favour. Focusing on the comparison between graph GPs and GNNs, it is also evident that there are more ways to manipulate or change the designs of GNNs for better performances, whereas designing a graph GP is less straightforward as we need to ensure the requirements of a kernel are met. There will always be scenarios when deep learning models outperform a GP, but the Bayesian properties of a GP still cannot be replicated by Bayesian deep learning, and so GPs and graph GPs still provide invaluable qualities for any user today.

# Appendix A

# Dataset Statistics Used in Chapters 5, 6, & 7

## A.1 Datasets

| Data | Type | #Nodes | #Edges | #Classes | Feat. Dim. | Hom. Rat. | Edge Dens. |
|---|---|---|---|---|---|---|---|
| **Texas** | WebKB | 183 | 309 | 5 | 1703 | 0.11 | 0.0186 |
| **Cornell** | WebKB | 183 | 295 | 5 | 1703 | 0.30 | 0.0177 |
| **Wisconsin** | WebKB | 251 | 499 | 5 | 1703 | 0.21 | 0.0159 |
| **Chameleon** | Wikipedia | 2,277 | 36,101 | 5 | 2,325 | 0.23 | 0.0139 |
| **Cora** | Citation | 2,708 | 5,429 | 7 | 1,433 | 0.81 | 0.0015 |
| **Citeseer** | Citation | 3,327 | 4,732 | 6 | 3,703 | 0.74 | 0.0009 |
| **Squirrel** | Wikipedia | 5,201 | 217,073 | 5 | 2,089 | 0.22 | 0.0161 |
| **Actor** | Actor | 7,600 | 33,544 | 5 | 931 | 0.22 | 0.0012 |
| **PubMed** | Citation | 19,717 | 44,338 | 3 | 500 | 0.80 | 0.0001 |

Table A.1: Summary of graphs used for semi-supervised node classification experiments.

## A.2 Description

**WebKB:** Cornell, Texas, and Wisconsin, are webpage datasets collected from computer science departments of various universities by Carnegie Mellon University. The nodes represent web pages, and edges are hyperlinks between them. Node features are the bag-of-words representation of web pages. The web pages are manually classified into five categories: student, project, course, staff, and faculty.

**Wikipedia:** Chameleon and Squirrel are page-page networks on specific topics in

Wikipedia. In those datasets, nodes represent web pages and edges are mutual links between pages, node features correspond to several informative nouns in the Wikipedia pages. We classify the nodes into five categories in terms of the number of the average monthly traffic of the web page.

**Citation:** Cora, Citeseer, and Pubmed are standard citation network benchmark datasets. In these networks, nodes represent papers, and edges denote citations of one paper by another. Node features are the bag-of-words representation of papers, and node label is the academic topic of a paper.

**Actor:** The Actor dataset is the actor-only induced subgraph of the film-director-actor-writer network. Each node corresponds to an actor, and the edge between two nodes denotes co-occurrence on the same Wikipedia page. Node features correspond to some keywords in the Wikipedia pages. We classify the nodes into five categories in terms of words of actor's Wikipedia.

# Appendix B

# Appendix for Chapter 5

## B.1 Visualisation of Wavelet Transform on a Regular Grid

By applying the wavelet transforms to an impulse function centred around a certain node, we can visualize how wavelets of different scales spread around the centre node, capturing different ranges of neighbourhoods (cf. Figure 5.1). When applied to a regular grid graph, the pattern resembles that of the Euclidean domain. This is shown in Figure B.1, where we apply the Mexican Hat wavelet transform with various scales to show the different ranges of neighbourhoods. Neighbours are weighted continuously with intensity becoming zero once beyond a certain proximity. Thus, by using different scales, we can capture different ranges of neighbourhood information.

## B.2 Additional Experimental Results

### B.2.1 Synthetic Scale Recovery Experiments and Implementation Details

We run our synthetic experiments multiple times to show the overall behaviour of the model. We sample the labels for the nodes by Eq. (5.17) 100 times; for each sample, we also randomly select a set of nodes to use for training. The hyperparameters are optimized as part of the training process. For each set of training labels, we test 20 different initializations and use the converged values that lead to the lowest loss.

The selection of nodes for training will have an effect on the scales we recover.

We have presented one particular random split for 50% of nodes used for training in Figure 5.4 of the main text; In Figures B.2, B.3 and B.4, we present the scale recovery results for 10%, 30% and 70% of nodes selected for training for three random splits each. We can see the quality of the recoveries improves as the percentage of training nodes increases. Additionally, the approximate recoveries are consistently very close to the exact recoveries, showing the accuracy of our polynomial approximation.

### B.2.2 Baseline GP Models on Synthetic Data

We also evaluate the baseline GP models from Section 5.4 on synthetic data. The graph neural network models were not compared against as they require a validation set of nodes, which are not assigned and would make an unfair comparison. We use GGP and ChebGP to make predictions on the synthetically generated signals, with the MAEs presented in Figure B.5.

The results in Figure B.5 show that the GGP only improves marginally with additional training data, indicating the model's inability to capture multi-scale information. ChebGP, which uses Chebyshev polynomials for approximations, does approximate a multi-scale spectral wavelet function, but we can see by the means and quantiles of the boxplots that they are less consistent in producing low MAEs compared to our polynomial approximation. As the number of training nodes increases, the model should be able to capture the different scales more accurately; however, the wider quantiles indicate the Chebyshev approximation is less consistent in producing accurate recoveries.

### B.2.3 Performance on Synthetic Data Generated Using Different Ground Truth Wavelets

The synthetic setting described in Section 5.4 uses the Mexican Hat kernel in both the inference GP and the data generating model. We now study the case where there is a mismatch in the mother wavelet between the inference GP and the data generating model. In FigureB.6 we always use a Mexican Hat wavelet for the inference GP and compare the case of using a Mexican Hat wavelet (Figures B.6a and B.6c) versus a Morlet wavelet (Figures B.6b and B.6d) in the data generating GP both in terms of prediction MAE (Figures B.6a and B.6b) and MAE of the reconstructed filter compared to ground truth filter (Figures B.6c and B.6d).

### B.2.4 WGGP without Feature Space Kernel

To measure the importance of the feature space kernel, we repeat experiments with WGGP on Cora and Citeseer with the feature space kernel $K_\Psi(\mathbf{x}_i, \mathbf{x}_j) = \delta_{ij}$ set to the identity. As a result, the model classifies nodes no longer also based on node features but on graph structure alone. We compare the results to those of the full WGGP model in Table B.1. As expected, the performance of the model drops decisively when removing the dependence on the node features, demonstrating the importance of the node feature kernel for the predictive performance.

| **Method** | **Cora** | **Citeseer** |
|---|---|---|
| **WGGP** | 84.7 | 70.8 |
| **WGGP without node features** | 71.9 | 47.7 |

Table B.1: Classification accuracy of the WGGP model with and without the node feature kernel. When removing the node feature kernel, the predictive performance drops by more than 10% for both data sets.

### B.2.5 ELBO Plots

As described in Section 5, early stopping is performed based on the ELBO. To check convergence, we show how the ELBO varies from epoch to epoch in Figure B.7. Note that the ELBO curve for the PubMed data set is non-monotonic as stochastic optimisation is employed during training.

### B.2.6 Robustness Analysis

We perform a robustness analysis examining how the model performance changes as we vary different parts of the model or training setup, while keeping everything else as described in Section 5.

**Data Split** In the first experiment, we use 10 different data splits for Cora and Citeseer that retain the uniform distribution of classes and re-run the model with otherwise equal hyperparameters. The average performance across the 10 data splits is reported in Table B.2 together with one standard deviation. We find that the model performance only varies modestly for both data sets and the performance remains comparable to the one achieved on the public data split.

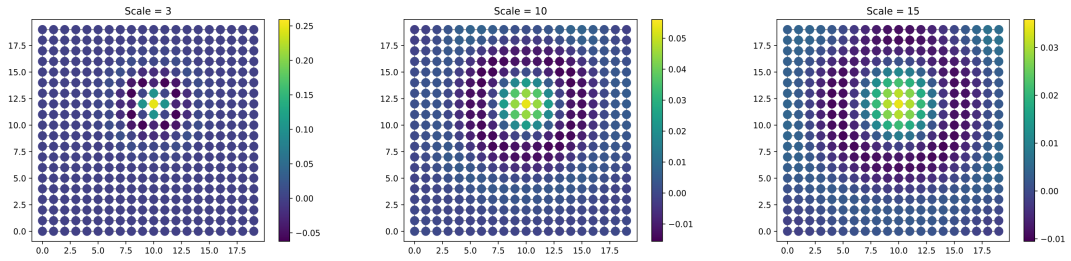
**Number of scales** We also analyze how the model performance varies when using different number of scales in the model, ranging from using only a low-pass filter to also including 4 band-pass filters. The results are again reported in Table B.2 with the standard deviation over the 4 different setups (0-4 scales), showing that the model accuracy varies only slightly when using different number of filters.

**Hyperparameter initialisations** Finally, we repeat the experiments with random initializations of the scale hyperparameters. The results with their standard deviation over 10 different initializations (Table B.2) demonstrate the model's robustness to different hyperparameter initializations.

| Method | Cora | Citeseer |
|---|---|---|
| **WGGP with varying data splits** | $82.4 \pm 1.1$ | $67.8 \pm 2.7$ |
| **WGGP with varying number of scales** | $84.7 \pm 0.2$ | $70.6 \pm 0.2$ |
| **WGGP with varying hyperparameter initalisations** | $84.2 \pm 0.4$ | $71.0 \pm 0.6$ |

Table B.2: Results of the robustness analysis of the WGGP model when varying the data split, the number of scales, or the scale hyperparameter initializations.

## B.3 Figures




(a) (b) (c)

Figure B.1: The Mexican Hat wavelet transform of a $\delta$ signal on a regular grid graph. The grid simulates a Euclidean domain to demonstrate the neighbourhoods more clearly at different scales.

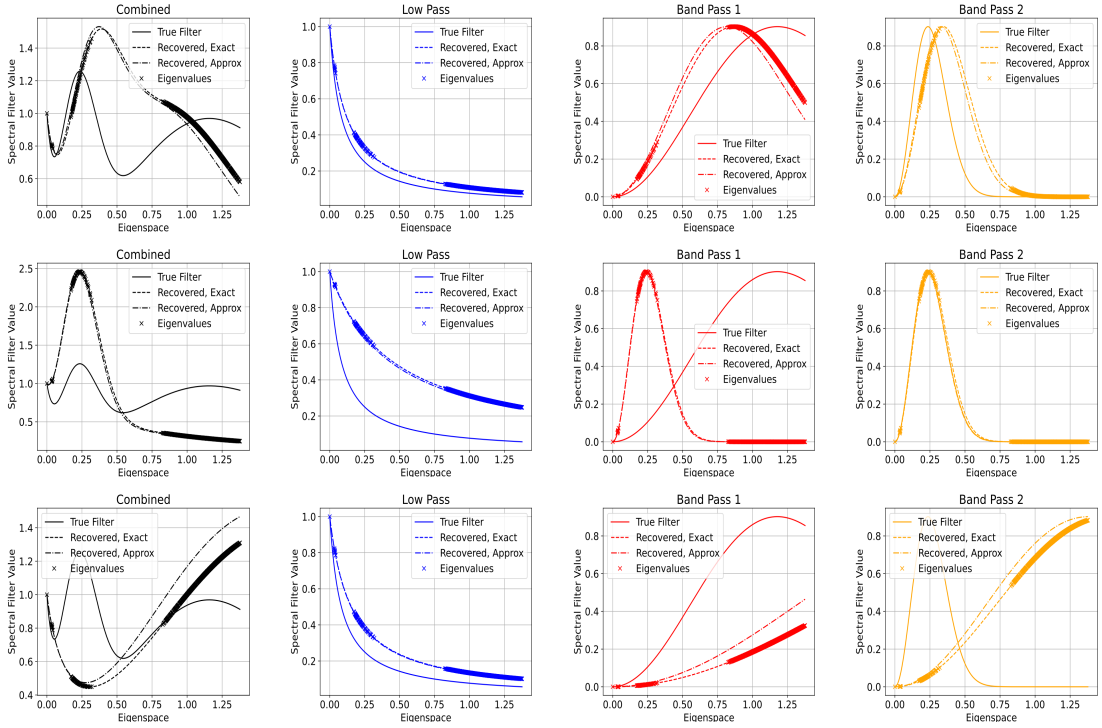



Figure B.2: Scale recoveries from synthetic experiments using 10% of nodes as training. Each row is a different random selection of training nodes.

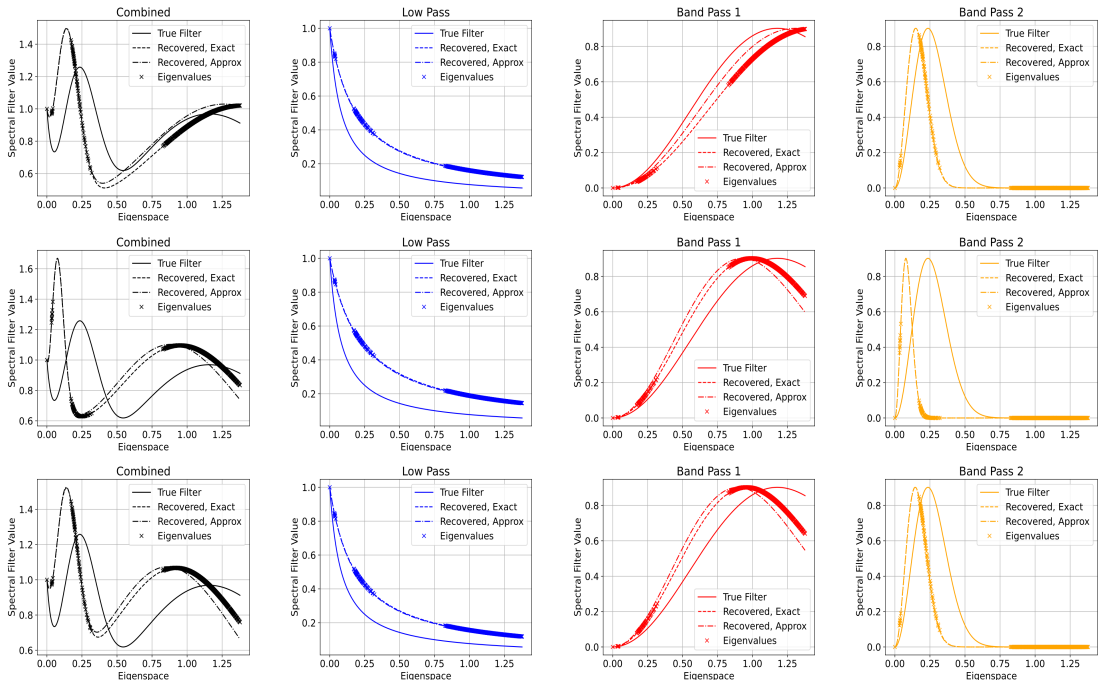



Figure B.3: Scale recoveries from synthetic experiments using 30% of nodes as training. Each row is a different random selection of training nodes.

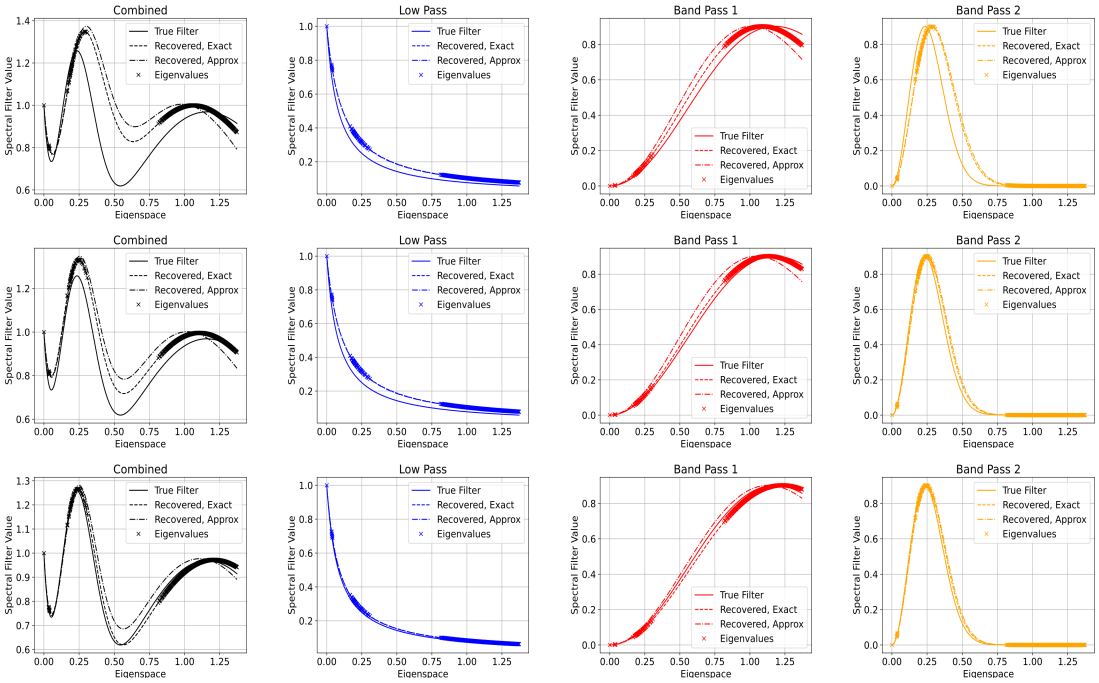
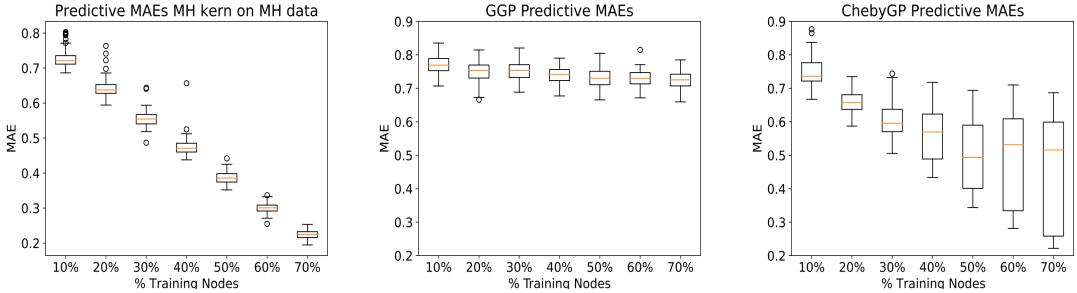
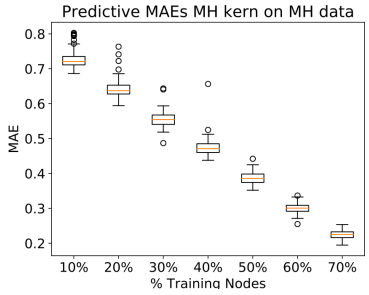



Figure B.4: Scale recoveries from synthetic experiments using 70% of nodes as training. Each row is a different random selection of training nodes.

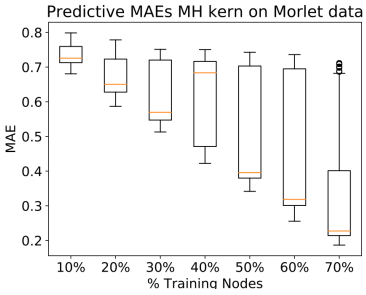



(a) WGGP (b) GGP (c) ChebGP

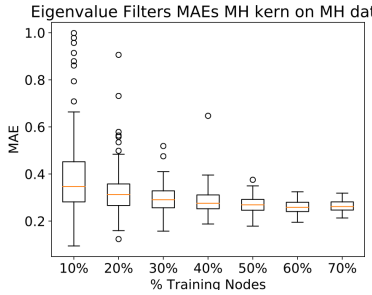
Figure B.5: WGGP prediction MAE on synthetic data (a and identical to Figure 3b in the main text) compared to MAEs of baseline GP models, GGP (b) and ChebGP (c).



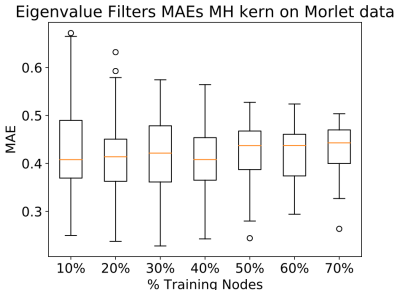

(a) MAE between predicted and ground truth synthetic signal when using a Mexican Hat wavelet in the inference GP and a **Mexican Hat** wavelet in the data generating GP.




(b) MAE between predicted and ground truth synthetic signal when using a Mexican Hat wavelet in the inference GP and a **Morlet** wavelet in the data generating GP.

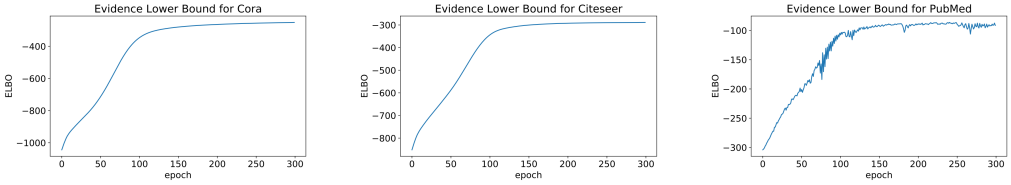



(c) MAE between reconstructed and ground truth filter when using a Mexican Hat wavelet in the inference GP and a **Mexican Hat** wavelet in the data generating GP.




(d) MAE between reconstructed and ground truth filter when using a Mexican Hat wavelet in the inference GP and a **Morlet** wavelet in the data generating GP.

Figure B.6: Comparing the prediction and filter MAEs for different fractions of nodes used for training when the choice of mother wavelet match or does not match between the inference GP and the data generating GP.




(a) Cora (b) Citeseer (c) Pubmed

Figure B.7: Value of the ELBO during training over time.

# Appendix C

# Appendix for Chapter 6

We present a number of examples of regularizers and their corresponding kernel functions as presented in [24].

For regularizers of the form

$$\Omega(||f||^2) = \langle Pf, Pf \rangle = \langle f, r(\Delta)f \rangle, \tag{C.1}$$

the kernel is computed generally by

$$k(\mathbf{x}, \mathbf{x}') = \int_{-\infty}^{\infty} \frac{1}{\hat{g}(\omega)} e^{i\omega(\mathbf{x}-\mathbf{x}')}\, d\omega \tag{C.2}$$

such that $r(\Delta)e^{i\omega x} = \hat{g}(\omega)e^{i\omega x}$.

## C.1 Laplacian kernel:

The Laplacian kernel can also be defined in terms of smoothness operator $P$ as well as $\Delta$, with

$$P = (1, \sigma\nabla)^\top \tag{C.3}$$

$$\implies P^\top P = 1 + \sigma^2\Delta. \tag{C.4}$$

Working in terms of $\Delta$ gives us

$$r(\Delta) = 1 + \sigma^2\Delta. \tag{C.5}$$

$$\implies \hat{g}(\omega) = 1 + \sigma^2\omega^2. \tag{C.6}$$

This lead to the kernel

$$k(\mathbf{x}, \mathbf{x}') = \int_{-\infty}^{\infty} \frac{1}{1 + \sigma^2\omega^2} e^{i\omega(\mathbf{x}-\mathbf{x}')}\, d\omega \tag{C.7}$$

$$\propto \exp\Big\{ -\frac{1}{\sigma}||\mathbf{x} - \mathbf{x}'||\Big\}. \tag{C.8}$$

## C.2 Gaussian RBF kernel:

The RBF kernel regularizer starts with $r(\Delta)$ directly

$$r(\Delta) = \sum_{i=0}^{\infty} \frac{1}{i!}\left(\frac{\sigma^2\Delta}{2}\right)^i \tag{C.9}$$

$$\implies \hat{g}(\omega) = \exp\left\{\frac{\sigma^2\omega^2}{2}\right\}. \tag{C.10}$$

This lead to the kernel

$$k(\mathbf{x}, \mathbf{x}') = \int_{-\infty}^{\infty} \exp\Big\{ -\frac{\sigma^2\omega^2}{2}\Big\} e^{i\omega(\mathbf{x}-\mathbf{x}')}\, d\omega \tag{C.11}$$

$$\propto \exp\Big\{ -\frac{1}{2\sigma^2}(\mathbf{x} - \mathbf{x}')^2\Big\} \tag{C.12}$$

## C.3 Dirichlet kernel:

The Dirichlet kernel of dimension $N$ specifies the reciprocal of $\hat{g}(\omega)$ directly as

$$\frac{1}{\hat{g}(\omega)} = \sum_{i=-N}^{N} \delta(\omega - i). \tag{C.13}$$

The kernel is then computed as

$$k(\mathbf{x}, \mathbf{x}') = \int_{-\infty}^{\infty} \sum_{i=-N}^{N} \delta(\omega - i) e^{i\omega(\mathbf{x}-\mathbf{x}')} d\omega \tag{C.14}$$

$$= \sum_{i=-N}^{N} \int_{-\infty}^{\infty} \delta(\omega - i) e^{i\omega(\mathbf{x}-\mathbf{x}')} d\omega \tag{C.15}$$

$$= \sum_{\omega=-N}^{N} e^{i\omega(\mathbf{x}-\mathbf{x}')} \tag{C.16}$$

$$= \frac{(\sin(N + 1/2)(\mathbf{x} - \mathbf{x}'))}{\sin((\mathbf{x} - \mathbf{x}')/2)}. \tag{C.17}$$

The final density corresponds to the Dirichlet kernel for a given dimension $N$.

## C.4 Gaussian Periodic kernel:

The Gaussian periodic kernel also starts with the following regularizer

$$r(\Delta) = \sum_{i=0}^{\infty} \frac{1}{i!} \left( \frac{\sigma^2 \Delta}{2} \right)^i \tag{C.18}$$

but for functions operating on the domain $[0, 2\pi]^n$, where $n$ is the dimension of $\mathbf{x}$. This leads to the integral in the fixed domain space $\int_{[0,2\pi]^n} \exp\big\{ -\frac{\sigma^2\omega^2}{2} \big\} e^{i\omega(\mathbf{x}-\mathbf{x}')}\, d\omega$ which is difficult to evaluate. Instead we make use of Mercer's Theorem:

**Theorem.** *(Mercer's Theorem): If $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function, and $T_k$, defined as*

$$T_k f(\cdot) = \int_{\mathcal{X}} k(\cdot, x) f(x) dx, \tag{C.19}$$

*is positive semi-definite, then $T_k$ has eigenfunctions $\phi_i$ and eigenvalues $\lambda_i$ such that*

$$k(\mathbf{x}, \mathbf{x}) = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}'). \tag{C.20}$$

We can implicitly express $k(\mathbf{x}, \mathbf{x}') = e^{-\frac{\sigma^2}{2}\Delta}$ for $\mathbf{x}, \mathbf{x}' \in [0, 2\pi]^n$. We use the eigenbasis $\{1/2, \sin(l\mathbf{x}), \cos(l\mathbf{x}), l \in \mathbb{N}\}$ while the regularizer operator has eigenvalues $e^{-l^2\sigma^2/2}$. The integral can then be evaluated using Mercer Theorem to get

$$k(\mathbf{x}, \mathbf{x}') = \sum_{l=1}^{\infty} e^{-l^2\sigma^2} \big( \sin(l\mathbf{x}) \sin(l\mathbf{x}') + \cos(l\mathbf{x}) \cos(l\mathbf{x}') \big) \tag{C.21}$$

$$= \sum_{l=1}^{\infty} e^{-l^2\sigma^2} \cos(l(\mathbf{x} - \mathbf{x}')). \tag{C.22}$$

The periodicity is produced by the cosine function in the kernel.

# Bibliography


[1] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, 2(2):109–127, 2021.

[2] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[3] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[4] Ljubiša Stanković, Miloš Daković, and Ervin Sejdić. Introduction to graph signal processing. In *Vertex-Frequency Analysis of Graph Signals*, pages 3–108. Springer, 2019.

[5] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[6] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

[7] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[8] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[9] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

[10] Pavel Izmailov, Wesley J Maddox, Polina Kirichenko, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Subspace inference for bayesian deep learning. In *Uncertainty in Artificial Intelligence*, pages 1169–1179. PMLR, 2020.

[11] Andreas Damianou and Neil D Lawrence. Deep gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR, 2013.

[12] Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012.

[13] Zhiqian Chen, Fanglan Chen, Lei Zhang, Taoran Ji, Kaiqun Fu, Liang Zhao, Feng Chen, Lingfei Wu, Charu Aggarwal, and Chang-Tien Lu. Bridging the gap between spatial and spectral domains: A survey on graph neural networks. *arXiv preprint arXiv:2002.11867*, 2020.

[14] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[15] Arun Venkitaraman, Saikat Chatterjee, and Peter Handel. Gaussian processes over graphs. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5640–5644. IEEE, 2020.

[16] Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In *International conference on machine learning*, pages 1067–1075. PMLR, 2013.

[17] Gregory Benton, Wesley J Maddox, Jayson Salkey, Julio Albinati, and Andrew Gordon Wilson. Function-space distributions over kernels. *Advances in Neural Information Processing Systems*, 32, 2019.

[18] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[19] R. R. Coifman and M. Maggioni. Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94, 2006.

[20] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[21] Fernando Gama, Alejandro Ribeiro, and Joan Bruna. Diffusion scattering transforms on graphs. In *International Conference on Learning Representations*, 2019.

[22] Bingbing Xu, H. Shen, Qi Cao, Yunqi Qiu, and X. Cheng. Graph wavelet neural network. *ArXiv*, abs/1904.07785, 2019.

[23] Jingyi Wang and Zhidong Deng. A deep graph wavelet convolutional neural network for semi-supervised node classification. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[24] Alex J Smola, Bernhard Schölkopf, and Klaus-Robert Müller. The connection between regularization operators and support vector kernels. *Neural networks*, 11(4):637–649, 1998.

[25] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning theory and kernel machines*, pages 144–158. Springer, 2003.

[26] Justin Michael Curry. *Sheaves, cosheaves and applications*. University of Pennsylvania, 2014.

[27] Cristian Bodnar, Francesco Di Giovanni, Benjamin Paul Chamberlain, Pietro Liò, and Michael M Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. *arXiv preprint arXiv:2202.04579*, 2022.

[28] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[29] Yin-Cong Zhi, Yin Cheng Ng, and Xiaowen Dong. Gaussian processes on graphs via spectral kernel learning. *Transactions on Signal and Information Processing over Networks*, 2022.

[30] *Felix Opolka, *Yin-Cong Zhi, Pietro Liò, and Xiaowen Dong. Adaptive gaussian processes on graphs via spectral graph wavelets. In *International Conference on Artificial Intelligence and Statistics*, pages 4818–4834. PMLR, 2022.

[31] Yin-Cong Zhi, Felix L Opolka, Yin Cheng Ng, Pietro Liò, and Xiaowen Dong. Transductive kernels for gaussian processes on graphs. *arXiv preprint arXiv:2211.15322*, 2022.

[32] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[33] Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656, 2013.

[34] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

[35] Xiaowen Dong, Dorina Thanou, Laura Toni, Michael Bronstein, and Pascal Frossard. Graph signal processing for machine learning: A review and new perspectives. *IEEE Signal processing magazine*, 37(6):117–127, 2020.

[36] Antonio G Marques, Santiago Segarra, Geert Leus, and Alejandro Ribeiro. Stationary graph processes and spectral estimation. *IEEE Transactions on Signal Processing*, 65(22):5911–5926, 2017.

[37] Nathanaël Perraudin and Pierre Vandergheynst. Stationary signal processing on graphs. *IEEE Transactions on Signal Processing*, 65(13):3462–3477, 2017.

[38] David R Burt, Carl Edward Rasmussen, and Mark Van Der Wilk. Convergence of sparse variational inference in gaussian processes regression. *The Journal of Machine Learning Research*, 21(1):5120–5182, 2020.

[39] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

[40] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, 36(3):1171–1220, 2008.

[41] Arun Venkitaraman, Saikat Chatterjee, and Peter Händel. Predicting graph signals using kernel regression where the input signal is agnostic to a graph. *IEEE Transactions on Signal and Information Processing over Networks*, 5(4):698–710, 2019.

[42] Yin Cheng Ng, Nicolò Colombo, and Ricardo Silva. Bayesian semi-supervised learning with graph gaussian processes. *Advances in Neural Information Processing Systems*, 31, 2018.

[43] Zhao-Yang Liu, Shao-Yuan Li, Songcan Chen, Yao Hu, and Sheng-Jun Huang. Uncertainty aware graph gaussian process for semi-supervised learning. In *AAAI Conference on Artificial Intelligence*, 2020.

[44] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

[45] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

[46] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *AAAI Conference on Artificial Intelligence*, 2021.

[47] Felix L Opolka and Pietro Liò. Graph convolutional gaussian processes for link prediction. *arXiv preprint arXiv:2002.04337*, 2020.

[48] Kai Yu and Wei Chu. Gaussian process models for link analysis and transfer learning. *Advances in Neural Information Processing Systems*, 20, 2007.

[49] Xiaowen Dong, Dorina Thanou, Michael Rabbat, and Pascal Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63, 2019.

[50] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.

[51] Andrew G Wilson, Zhiting Hu, Russ R Salakhutdinov, and Eric P Xing. Stochastic variational deep kernel learning. *Advances in neural information processing systems*, 29, 2016.

[52] Jinyuan Fang, Shangsong Liang, Zaiqiao Meng, and Qiang Zhang. Gaussian process with graph convolutional kernel for relational learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 353–363, 2021.

[53] Viacheslav Borovitskiy, Iskander Azangulov, Alexander Terenin, Peter Mostowsky, Marc Deisenroth, and Nicolas Durrande. Matérn gaussian processes on graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 2593–2601. PMLR, 2021.

[54] Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, et al. Matérn gaussian processes on riemannian manifolds. *Advances in Neural Information Processing Systems*, 33:12426–12437, 2020.

[55] Matthew J Urry and Peter Sollich. Random walk kernels and learning curves for gaussian process regression on random graphs. *The Journal of Machine Learning Research*, 14(1):1801–1835, 2013.

[56] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027, 2021.

[57] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[58] Karsten Borgwardt, Elisabetta Ghisu, Felipe Llinares-López, Leslie O'Bray, Bastian Rieck, et al. Graph kernels: State-of-the-art and future challenges. *Foundations and Trends® in Machine Learning*, 13(5-6):531–712, 2020.

[59] Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional gaussian processes. *Advances in Neural Information Processing Systems*, 30, 2017.

[60] Ian Walker and Ben Glocker. Graph convolutional gaussian processes. In *International Conference on Machine Learning*, pages 6495–6504. PMLR, 2019.

[61] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[62] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[63] Ningyuan Teresa Huang and Soledad Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8533–8537. IEEE, 2021.

[64] Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein weisfeiler-lehman graph kernels. *Advances in Neural Information Processing Systems*, 32, 2019.

[65] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

[66] Bin Li, Xingquan Zhu, Lianhua Chi, and Chengqi Zhang. Nested subtree hash kernels for large-scale graph classification over streams. In *2012 IEEE 12th International Conference on Data Mining*, pages 399–408. IEEE, 2012.

[67] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019.

[68] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[69] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[70] Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. *Advances in neural information processing systems*, 29, 2016.

[71] Oliver Hamelijnck, William Wilkinson, Niki Loppi, Arno Solin, and Theodoros Damoulas. Spatio-temporal variational gaussian processes. *Advances in Neural Information Processing Systems*, 34:23621–23633, 2021.

[72] Qin Lu and Georgios B Giannakis. Spatio-temporal inference of dynamical gaussian processes over graphs. In *2021 55th Asilomar Conference on Signals, Systems, and Computers*, pages 1515–1519. IEEE, 2021.

[73] David Blanco Mulero, Markus Heinonen, and Ville Kyrki. Evolving-graph gaussian processes. In *International Conference on Machine Learning: Time Series Workshop*, 2021.

[74] Shweta Dahale and Balasubramaniam Natarajan. Recursive gaussian process over graphs for integrating multi-timescale measurements in low-observable distribution systems. *IEEE Transactions on Power Systems*, 2022.

[75] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[76] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29*, pages 3844–3852, 2016.

[77] Siheng Chen, Aliaksei Sandryhaila, José MF Moura, and Jelena Kovacevic. Signal denoising on graphs via graph filtering. In *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 872–876. IEEE, 2014.

[78] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902, 2021.

[79] Yves-Laurent Kom Samo and Stephen Roberts. Generalized spectral kernels. *arXiv preprint arXiv:1506.02236*, 2015.

[80] Jian Li, Yong Liu, and Weiping Wang. Automated spectral kernel learning. *arXiv preprint arXiv:1909.04894*, 2019.

[81] Lynn H Loomis. *Introduction to abstract harmonic analysis*. Courier Corporation, 2013.

[82] Alex J Smola and Bernhard Schölkopf. From regularization operators to support vector kernels. In *Advances in neural information processing systems*, pages 343–349, 1998.

[83] Alex J Smola, Zoltan L Ovari, and Robert C Williamson. Regularization with dot-product kernels. In *Advances in neural information processing systems*, pages 308–314, 2001.

[84] Enrico Bozzo. The moore–penrose inverse of the normalized graph laplacian. *Linear Algebra and its Applications*, 439(10):3038–3043, 2013.

[85] Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer, 2013.

[86] Rafail N Gasimov. Augmented lagrangian duality and nondifferentiable optimization methods in nonconvex programming. *Journal of Global Optimization*, 24(2):187–203, 2002.

[87] Mokhtar S Bazaraa and Jamie J Goode. A survey of various tactics for generating lagrangian multipliers in the context of lagrangian duality. *European Journal of Operational Research*, 3(4):322–338, 1979.

[88] Xingyue Pu, Siu Lun Chau, Xiaowen Dong, and Dino Sejdinovic. Kernel-based graph learning from smooth signals: A functional viewpoint. *arXiv preprint arXiv:2008.10065*, 2020.

[89] Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. In *International Conference on Machine Learning*, pages 3237–3246. PMLR, 2018.

[90] James Hensman, Alexander Matthews, and Zoubin Ghahramani. Scalable variational gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR, 2015.

[91] Michaël Defferrard, Lionel Martin, Rodrigo Pena, and Nathanaël Perraudin. Pygsp: Graph signal processing in python. *https://github. com/epfl-lts2/pygsp*, 2017.

[92] Hamid Behjat, Ulrike Richter, Dimitri Van De Ville, and Leif Sörnmo. Signal-adapted tight frames on graphs. *IEEE Transactions on Signal Processing*, 64(22):6017–6029, 2016.

[93] Swedish Meteorological and Hydrological Institute (SMHI). `http://opendata-download-metobs.smhi.se/`. [Online]. Accessed: 2019-02-13.

[94] Dorina Thanou, Xiaowen Dong, Daniel Kressner, and Pascal Frossard. Learning heat diffusion graphs. *IEEE Transactions on Signal and Information Processing over Networks*, 3(3):484–499, 2017.

[95] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[96] Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical Review E*, 67(2):026112, 2003.

[97] Aaron Clauset, Cristopher Moore, and Mark E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98–101, 2008.

[98] Janusz Dutkowski, Michael Kramer, Michal A Surma, Rama Balakrishnan, J Michael Cherry, Nevan J Krogan, and Trey Ideker. A gene ontology inferred from molecular networks. *Nature Biotechnology*, 31:38–45, 2013.

[99] David I Shuman, Christoph Wiesmeyr, Nicki Holighaus, and Pierre Vandergheynst. Spectrum-adapted tight graph wavelet and vertex-frequency frames. *IEEE Transactions on Signal Processing*, 2015.

[100] Tiffany Fan, David I. Shuman, Shashanka Ubaru, and Yousef Saad. Spectrum-adapted polynomial approximation for matrix functions with applications in graph signal processing. *Algorithms*, 13(11), 2020.

[101] Piet Van Mieghem. *Graph Spectra for Complex Networks*. Cambridge University Press, USA, 2011.

[102] Lin Lin, Yousef Saad, and Chao Yang. Approximating spectral densities of large matrices. *SIAM Review*, 2016.

[103] Shuni Li, Yan Jin, and David I Shuman. Scalable $M$-Channel Critically Sampled Filter Banks for Graph Signals, 2019.

[104] R. N. Silver and H. Röder. Densities of States of Mega-Dimensional Hamiltonian Matrices. *International Journal of Modern Physics C*, 1994.

[105] RN Silver, H Roeder, AF Voter, and JD Kress. Kernel polynomial approximations for densities of states and spectral functions. *Journal of Computational Physics*, 124(1):115–130, 1996.

[106] Lin-Wang Wang. Calculating the density of states and optical-absorption spectra of large quantum systems by the plane-wave moments method. *Physical Review B*, 1994.

[107] A. Girard. A fast 'monte-carlo cross-validation' procedure for large least squares problems with noisy data. *Numerical Mathematics*, 1989.

[108] Edoardo Di Napoli, Eric Polizzi, and Yousef Saad. Efficient estimation of eigenvalue counts in an interval. *Numerical Linear Algebra with Applications*, 2016.

[109] Gilles Puy and Patrick Pérez. Structured sampling and fast reconstruction of smooth graph signals. *Information and Inference: A Journal of the IMA*, 2018.

[110] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *International Conference on Learning Representations*, 2019.

[111] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 2008.

[112] Louis Guttman. Enlargement methods for computing the inverse matrix. *The annals of mathematical statistics*, pages 336–343, 1946.

[113] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

[114] Hyun-Chul Kim, Jaewook Lee, and Daewon Lee. Transductive gaussian processes with applications to object pose estimation. *The Computer Journal*, 57(3):339–346, 2014.

[115] Alex Smola, Zoltán Ovári, and Robert C Williamson. Regularization with dot-product kernels. *Advances in neural information processing systems*, 13, 2000.

[116] Dengyong Zhou and Bernhard Schölkopf. A regularization framework for learning from graph data. In *ICML 2004 Workshop on Statistical Relational Learning and Its Connections to Other Fields (SRL 2004)*, pages 132–137, 2004.

[117] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[118] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.

[119] Yimeng Min, Frederik Wenkel, and Guy Wolf. Scattering gcn: Overcoming oversmoothness in graph convolutional networks. In *Advances in Neural Information Processing Systems 33*, 2020.

[120] Frederik Wenkel, Yimeng Min, Matthew Hirn, Michael Perlmutter, and Guy Wolf. Overcoming oversmoothness in graph convolutional networks via hybrid scattering networks. *arXiv preprint arXiv:2201.08932*, 2022.

[121] Jakob Hansen and Thomas Gebhart. Sheaf neural networks. *arXiv preprint arXiv:2012.06333*, 2020.

[122] Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković, and Pietro Liò. Sheaf neural networks with connection laplacians. *arXiv preprint arXiv:2206.08702*, 2022.

[123] Jakob Hansen. *Laplacians of Cellular Sheaves: Theory and Applications*. PhD thesis, University of Pennsylvania, 2020.

[124] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Somayeh Sojoudi, Junzhou Huang, and Wenwu Zhu. Spectral graph attention network with fast eigen-approximation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2905–2909, 2021.

[125] Amit Singer and H-T Wu. Vector diffusion maps and the connection laplacian. *Communications on pure and applied mathematics*, 65(8):1067–1144, 2012.

[126] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

[127] Thomas Zaslavsky. Signed graphs. *Discrete Applied Mathematics*, 4(1):47–74, 1982.

[128] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19, 2006.

[129] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2-3):177–201, 1993.

[130] Thomas Dittrich and Gerald Matz. Signal processing on signed graphs: Fundamentals and potentials. *IEEE Signal Processing Magazine*, 37(6):86–98, 2020.

[131] Songyang Zhang, Zhi Ding, and Shuguang Cui. Introducing hypergraph signal processing: Theoretical foundation and practical applications. *IEEE Internet of Things Journal*, 7(1):639–660, 2019.

[132] Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep gaussian processes. *Advances in neural information processing systems*, 30, 2017.

[133] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[134] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in neural information processing systems*, 32, 2019.