Gaussian Processes on Graphs via Spectral Kernel Learning

Yin-Cong Zhi, Yin Cheng Ng, and Xiaowen Dong

***Abstract*—We propose a graph spectrum-based Gaussian process for prediction of signals defined on nodes of the graph. The model is designed to capture various graph signal structures through a highly adaptive kernel that incorporates a flexible polynomial function in the graph spectral domain. Unlike most existing approaches, we propose to learn such a spectral kernel defined on a discrete space. In addition, this kernel has the interpretability of graph filtering achieved by a bespoke maximum likelihood learning algorithm that enforces the positivity of the spectrum. We demonstrate the interpretability of the model through synthetic experiments from which we show various ground truth spectral filters can be accurately recovered, and the adaptability translates to improved predictive performances compared to the baselines on real-world graph data of various characteristics.**

***Index Terms*—Gaussian processes, graph signal prediction, spectral kernel learning**

I. Introduction

Graphs are highly useful data structures that represent relationships and interactions between entities. Such relational structures are commonly observed in the real-world, but can also be artificially constructed from data according to heuristics. The graph structure can be exploited in conjunction with other auxiliary data to build more powerful predictive models. One particular class of models that can be enhanced for graph data is Gaussian processes (GP). As a kernel-based method, GPs can be adapted to incorporate non-Euclidean distance information through kernels derived on graphs. With the kernel defined, the standard Bayesian inference machinery can be directly applied to yield predictions, with the inherent benefit of incorporating uncertainties.

Multi-output Gaussian processes (MOGP) are regression models for vector-valued data. Given a set of input covariates and the corresponding output vectors, the model makes vectorial predictions given a novel input. In graph signal prediction problems, each output signal can be viewed as a vector where the dependency between elements of the signal is encoded in the graph structure. At the same time, the dependency between different graph signals can be modeled 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 co-regionalization model in [1], which makes choosing the overall kernel function straightforward. 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. [2] 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 [3], [4], [5]. One particular low-pass filter defined in [3], commonly used to denoise graph signals, also assumes the form of kernels on graphs. This was subsequently used in [6], [7] to construct a GP model on graphs for predicting low frequency signals. However, the filter as defined in [3], [6], [7] only has a low-pass nature 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 [8], 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 in heterophilic graphs [9], [10]. 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 [11], [12], [13], but the extension of the approach to a discrete graph space has yet to be explored. In this paper, we propose a novel MOGP model for graph-structured data, which uses a kernel on the graph 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 (hence 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 modelers insights on

Yin-Cong Zhi and Xiaowen Dong are with the Oxford-Man Institute and the Department of Engineering Science, University of Oxford, Oxford OX2 6ED, UK (e-mail: yin-cong.zhi@st-annes.ox.ac.uk; xdong@robots.ox.ac.uk).

Yin Cheng Ng is with Man AHL, London EC4R 3AD, UK (e-mail: yincheng.ng@gmail.com).

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.

II. Background

A. Gaussian Processes

A GP f is defined as

$$f(\mathbf{x}) \sim \mathcal{GP}\big(m(\mathbf{x}), \mathcal{K}(\mathbf{x}, \mathbf{x}')\big) \tag{1}$$

for any inputs $\mathbf{x}, \mathbf{x}'$, where $m(\cdot)$ is the mean function, and $\mathcal{K}(\cdot,\cdot)$ is the symmetric and positive definite kernel function. The mean function is often taken to be the zero function, while for training inputs $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and outputs $\mathbf{y} = (y_1, \ldots, y_N)$, and a given novel point $(\mathbf{x}_*, y_*)$, the GP forms the prior of a Bayesian regression model

$$\mathbb{P}\begin{pmatrix}\mathbf{y}\\ y_*\end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}\mathbf{K} & \mathbf{K}_*\\ \mathbf{K}_*^\top & \mathbf{K}_{**}\end{bmatrix}\right) \tag{2}$$

such that $\mathbf{K}_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{K}_* = (\mathcal{K}(\mathbf{x}_1, \mathbf{x}_*), \ldots, \mathcal{K}(\mathbf{x}_N, \mathbf{x}_*)^\top \in \mathbb{R}^N$ and $\mathbf{K}_{**} = \mathcal{K}(x_*, x_*)$. To make predictions we condition on the training data to obtain the posterior

$$\mathbb{P}(y_*|\mathbf{y}) \sim \mathcal{N}(\mathbf{K}_*^\top \mathbf{K}^{-1}\mathbf{y}\ ,\ \mathbf{K}_{**} - \mathbf{K}_*^\top \mathbf{K}^{-1}\mathbf{K}_*). \tag{3}$$

The distribution mean can be used as a point prediction while the covariance allows the construction of confidence intervals to provide a level of uncertainty. We will refer readers to [14] for a more thorough tutorial of the GP models.

B. 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 product of two kernels, one for the inputs and the other on the elements of $\mathbf{f}$. This is described as separable kernels in [1], where between any two inputs the function f will have the following form

$$\text{Cov}(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x}')) = \mathcal{K}(\mathbf{x}, \mathbf{x}')\mathbf{H} \tag{4}$$

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

$$\mathbf{K}(\mathbf{X}, \mathbf{X}) = \mathbf{K} \otimes \mathbf{H} \tag{5}$$

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

C. Spectral Filtering on Graphs

Let $\mathcal{G}$ be a graph with vertex set V such that $|V| = M$, we define the notion of spectral filtering on graphs from the graph Laplacian [15] defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{A}, \tag{6}$$

where $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the diagonal degree matrix. Assuming that $\mathcal{G}$ is undirected, the Laplacian admits the eigen-decomposition $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ where $\mathbf{U}$ contains the eigenvectors and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues. A signal $\mathbf{y} \in \mathbb{R}^M$ on $\mathcal{G}$ can be viewed as a function

$$\mathbf{y} : V \to \mathbb{R}, \tag{7}$$

and the graph Fourier transform of the signal, defined as $\mathbf{U}^\top\mathbf{y}$ in [3], computes the spectrum of $\mathbf{y}$ to produce the amplitude of each eigenvector (frequency) component. Filtering then involves a non-negative function $g(\mathbf{\Lambda})$ in the graph spectral domain that may reduce or amplify each component leading to a filtered signal

$$\mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{y}. \tag{8}$$

The term $\mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^\top$ is therefore referred to as a filtering function on the graph characterized by g.

D. Kernels and Regularization on Graphs

A property of kernel functions is provided by Bochner's theorem [16], 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 [2].

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 [17], [18]. When finding a smooth model $\mathbf{f}$ for graph signal $\mathbf{y}$, it is common to solve for the following regularized problem

$$\min_{\mathbf{f}} ||\mathbf{f} - \mathbf{y}||_2^2 + R(||\mathbf{f}||^2), \tag{9}$$

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. (9) to have a representation in a RKHS, with Moore-Penrose pseudoinverse used if $\mathbf{P}$ is singular [19]. The solution of Eq. (9) then exists by the Representer Theorem [20]. More generally, kernels on graphs assume the following form

$$\sum_{i=1}^{M} r^{-1}(\lambda_i)\mathbf{v}_i\mathbf{v}_i^\top = \mathbf{U}r^{-1}(\mathbf{\Lambda})\mathbf{U}^\top = r^{-1}(\mathbf{L}) \tag{10}$$

for diagonal matrix $\mathbf{\Lambda}$ containing the eigenvalues (i.e., $\{\lambda_i\}_{i=1}^M$) of $\mathbf{L}$ in increasing order and $\mathbf{U}$ containing the corresponding eigenvectors (i.e., $\{\mathbf{v}_i\}_{i=1}^M$). Furthermore, this definition is flexible in that different variations of the Laplacian such as the normalized Laplacian

$$\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} \tag{11}$$

and scaled Laplacian

$$\mathbf{L}_S = \frac{1}{\lambda_{max}(\mathbf{L})}\mathbf{L} \tag{12}$$

will both lead to valid kernels.

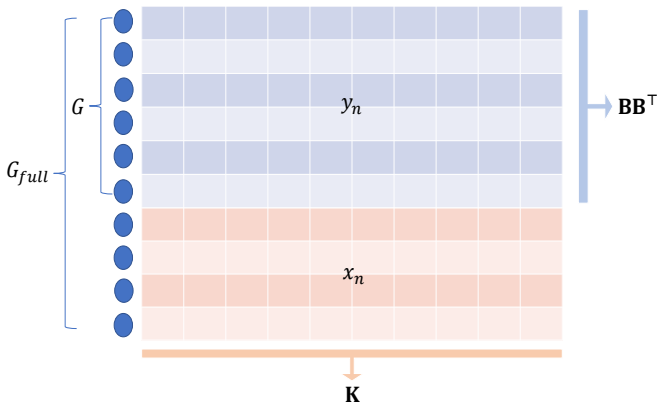


Fig. 1: Illustration of graph data construction into input-output pairs for a GP. Each column in blue is a graph signal that indicates the value on the nodes, with a corresponding column in red of input covariate below.

III. Proposed Model

A. 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 signals 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 (based on $\mathcal{G}$). Fig. 1 visualizes these input-output pairs with associated kernel matrices introduced later this section. Other setups are also possible depending on the problem, for which we will refer readers to our experiments in Section VI-C2 for further 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 $\mathcal{K}$ on any two inputs $\mathbf{x}_n$ and $\mathbf{x}'_n$. This leads to $\text{Cov}(\mathbf{f}(\mathbf{x}_n), \mathbf{f}(\mathbf{x}'_n)) = \mathcal{K}(\mathbf{x}_n, \mathbf{x}'_n)\mathbf{I}_M$, 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 matrix $\mathbf{B}$, giving rise to the following model

$$\mathbf{y}_n = \mathbf{B}\mathbf{f}(\mathbf{x}_n) + \boldsymbol{\epsilon}_n, \tag{13}$$

where $\boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \sigma_\epsilon^2 \mathbf{I}_M)$. The model in Eq. (13) 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 $\text{Cov}(\mathbf{y}_n, \mathbf{y}_m) = \mathbb{E}(\mathbf{y}_n \mathbf{y}_m^\top) = \mathbf{B}\mathbb{E}(\mathbf{f}(\mathbf{x}_n)\mathbf{f}(\mathbf{x}_m)^\top)\mathbf{B}^\top = \mathcal{K}(\mathbf{x}_n, \mathbf{x}_m)\mathbf{B}\mathbf{B}^\top$, and if we let $\tilde{\mathbf{y}} = \text{vec}([\mathbf{y}_1, \ldots, \mathbf{y}_N])$, the covariance of the full data becomes

$$\text{Cov}(\tilde{\mathbf{y}}) = \mathbf{K} \otimes \mathbf{B}\mathbf{B}^\top + \sigma_\epsilon^2 \mathbf{I}_{MN}, \tag{14}$$

where $\mathbf{K}_{nm} = \mathcal{K}(\mathbf{x}_n, \mathbf{x}_m)$, and $\otimes$ denotes the Kronecker product. The $\mathbf{B}\mathbf{B}^\top$ term can be thought of as a kernel between elements of each outputs $\mathbf{y}_n$, 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

$$\mathbb{P}\left(\begin{bmatrix}\tilde{\mathbf{y}} \\ \mathbf{y}_*\end{bmatrix}\right) \sim \mathcal{N}\Bigg(0\,,\, \begin{bmatrix}\mathbf{K} \otimes \mathbf{B}\mathbf{B}^\top & \mathbf{K}_* \otimes \mathbf{B}\mathbf{B}^\top \\ \mathbf{K}_*^\top \otimes \mathbf{B}\mathbf{B}^\top & \mathbf{K}_{**} \otimes \mathbf{B}\mathbf{B}^\top\end{bmatrix} + \sigma_\epsilon^2 \mathbf{I}_{MN}\Bigg), \tag{15}$$

where $\mathbf{K}_* = (\mathcal{K}(\mathbf{x}_1, \mathbf{x}_*), \ldots, \mathcal{K}(\mathbf{x}_N, \mathbf{x}_*))^\top \in \mathbb{R}^N$ and $\mathbf{K}_{**} = \mathcal{K}(\mathbf{x}_*, \mathbf{x}_*)$. For the inputs, the kernel $\mathcal{K}$ 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. (12), and follows the general form in Eq. (10) 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, λ_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 [1]. From the filtering perspective, we identify that all kernels on graphs defined in [2] are of low-pass nature, and suggest that this is restrictive and less suitable to data that does not exhibit smoothness or a low-frequency characteristic. In order 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.

B. 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

$$g(\lambda) = \beta_0 + \beta_1\lambda + \cdots + \beta_P\lambda^P \implies \mathbf{B} = \sum_{i=0}^P \beta_i \mathbf{L}_S^i, \tag{16}$$

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 IV.

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 [2] 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) [21].
- 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 [3] 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 graph neural networks (e.g., [21] suggested that information propagation on graphs might not benefit from going beyond the 3-hop neighbourhood).

In addition, our setting is a more generalized version of that in Section II.B of [6], where the terms in the regularizer $\mathbf{J}_p$ can in theory be learned, but it does not have the advantage of the proposed polynomial methods: first, it does not correspond to a localised 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.

C. Equivalence to the Co-regionalization Model

The prior model in (15) follows the form of separable kernels similar to the co-regionalization model in the literature of kernels for vector-valued functions [1]. 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) [1], we have

$$\mathbf{f}(\mathbf{x}_n) = \sum_{i=1}^{Q} \mathbf{b}_i u^i(\mathbf{x}_n) \tag{17}$$

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

$$\begin{aligned}\text{Cov}(\mathbf{f}(\mathbf{x}_n), \mathbf{f}(\mathbf{x}_m)) &= \sum_{i=1}^{Q}\sum_{j=1}^{Q} \mathbf{b}_i \mathbf{b}_j^\top \mathbb{E}(u^i(\mathbf{x}_n)u^j(\mathbf{x}_m)) \\ &= \sum_{i=1}^{Q} \mathbf{b}_i \mathbf{b}_i^\top \mathbb{E}(u^i(\mathbf{x}_n)u^i(\mathbf{x}_m)) \\ &= \mathcal{K}(\mathbf{x}_n, \mathbf{x}_m) \sum_{i=1}^{Q} \mathbf{b}_i \mathbf{b}_i^\top. \end{aligned} \tag{18}$$

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 $\text{Cov}(\mathbf{f}(\mathbf{x}_n), \mathbf{f}(\mathbf{x}_m)) = \mathcal{K}(\mathbf{x}_n, \mathbf{x}_m)\mathbf{B}\mathbf{B}^\top$. When we have N input-output data pairs, the full covariance of $\tilde{\mathbf{f}} = \text{vec}(\mathbf{f}(\mathbf{x}_1), \ldots, \mathbf{f}(\mathbf{x}_N))$ will follow the separable form $\text{Cov}(\tilde{\mathbf{f}}) = \mathbf{K} \otimes \mathbf{B}\mathbf{B}^\top$. 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. (10).

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 $\tilde{\mathbf{f}} = \text{vec}((\mathbf{f}(\mathbf{x}_1), \ldots, \mathbf{f}(\mathbf{x}_N))^\top)$ 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.

IV. Optimizing GP Log-Marginal Likelihood

The polynomial coefficients β_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 Ω contain $\boldsymbol{\beta}$ and all other hyperparameters, the GP log-marginal likelihood is

$$\begin{aligned} L(\Omega) &= \log \mathbb{P}(\tilde{\mathbf{y}}|\Omega) \\ &= -\frac{1}{2}\log|\boldsymbol{\Sigma}_\Omega| - \frac{1}{2}\tilde{\mathbf{y}}^\top \boldsymbol{\Sigma}_\Omega^{-1}\tilde{\mathbf{y}} - \frac{NM}{2}\log(2\pi), \end{aligned} \tag{19}$$

where $\boldsymbol{\Sigma}_\Omega$ is the covariance of (14). As described in Eq. (13), 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 $\mathbf{B}$ to be a valid filter, 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 [3], [1]. 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

$$\begin{aligned} &\min_{\boldsymbol{\beta}} -L(\boldsymbol{\beta}) \\ &\text{subject to } -\mathbf{B}_v\boldsymbol{\beta} \leq 0, \end{aligned} \tag{20}$$

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:** $\boldsymbol{\beta}$

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

$$\mathbf{B}_v = \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \ldots & \lambda_1^P \\ 1 & \lambda_2 & \lambda_2^2 & \ldots & \lambda_2^P \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_M & \lambda_M^2 & \ldots & \lambda_M^P \end{pmatrix}. \tag{21}$$

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

$$\mathbb{L}(\boldsymbol{\beta}, \mathcal{L}) = -L(\boldsymbol{\beta}) + \mathcal{L}^\top(-\mathbf{B}_v\boldsymbol{\beta}) = -L(\boldsymbol{\beta}) - \mathcal{L}^\top\mathbf{B}_v\boldsymbol{\beta} \tag{22}$$

where $\mathcal{L} \in \mathbb{R}^M$ is a vector of Lagrange multipliers. The solution to this problem is guided by the Karush–Kuhn–Tucker (KKT) conditions [22], which specifies that $\boldsymbol{\beta}^*$ is the optimal solution to Eq. (20) if $(\boldsymbol{\beta}^*, \mathcal{L})$ is the solution to

$$\min_{\boldsymbol{\beta}} \max_{\mathcal{L}\geq 0} \mathbb{L}(\boldsymbol{\beta}, \mathcal{L}). \tag{23}$$

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

$$\max_{\mathcal{L}\geq 0} \min_{\boldsymbol{\beta}} \mathbb{L}(\boldsymbol{\beta}, \mathcal{L}) \tag{24}$$

as this makes the function concave with respect to $\mathcal{L}$ [23], [24] 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 softplus function on $\mathcal{L}$ defined as

$$S_m(\mathcal{L}') = \log(1 + e^{\mathcal{L}'}) \tag{25}$$

to keep the Lagrange multipliers positive during the optimization. Due to the non-convexity of Eq. (24), Algorithm 1 may only find a local optimum depending on the initialization. A simple strategy to obtain a sensible initialization is to maximize for the log-marginal likelihood (problem (20) 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.

A. Scalability

By exploiting the Kronecker product structure of the covariance matrix, inversion of Eq. (14) 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 $\mathcal{O}(N^3M^3)$. We manipulate the covariance matrix in a similar fashion to [25], first re-writing it as follows

$$\begin{aligned} \boldsymbol{\Sigma} &= \mathbf{B}\mathbf{B}^\top \otimes \mathbf{K} + \sigma_\epsilon^2\mathbf{I} \\ &= (\mathbf{I}\otimes\mathbf{K})(\mathbf{B}\mathbf{B}^\top\otimes\mathbf{I}) + \sigma_\epsilon^2(\mathbf{B}\mathbf{B}^\top\otimes\mathbf{I})^{-1}\mathbf{B}\mathbf{B}^\top\otimes\mathbf{I} \\ &= \big[\mathbf{I}\otimes\mathbf{K} + \sigma_\epsilon^2((\mathbf{B}\mathbf{B}^\top)^{-1}\otimes\mathbf{I})\big]\mathbf{B}\mathbf{B}^\top\otimes\mathbf{I} \\ &= \big[\sigma_\epsilon^2(\mathbf{B}\mathbf{B}^\top)^{-1}\oplus\mathbf{K}\big]\mathbf{B}\mathbf{B}^\top\otimes\mathbf{I} \end{aligned} \tag{26}$$

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

$$\begin{aligned} \boldsymbol{\Sigma} &= \big[\sigma_\epsilon^2\mathbf{U}_B\boldsymbol{\Lambda}_B^{-1}\mathbf{U}_B^\top \oplus \mathbf{U}_K\boldsymbol{\Lambda}_K\mathbf{U}_K^\top\big]\mathbf{U}_B\boldsymbol{\Lambda}_B\mathbf{U}_B^\top\otimes\mathbf{I} \\ &= \sigma_\epsilon^2(\mathbf{U}_B\otimes\mathbf{U}_K)(\boldsymbol{\Lambda}_B^{-1}\oplus\boldsymbol{\Lambda}_K) \\ &\quad\times(\mathbf{U}_B^\top\otimes\mathbf{U}_K^\top)(\mathbf{U}_B\boldsymbol{\Lambda}_B\mathbf{U}_B^\top\otimes\mathbf{I}). \end{aligned} \tag{27}$$

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

$$\begin{aligned} \boldsymbol{\Sigma}^{-1} =& \frac{1}{\sigma_\epsilon^2}(\mathbf{U}_B\boldsymbol{\Lambda}_B^{-1}\mathbf{U}_B^\top\otimes\mathbf{I})(\mathbf{U}_B\otimes\mathbf{U}_K) \\ &\times(\boldsymbol{\Lambda}_B\oplus\boldsymbol{\Lambda}_K^{-1})(\mathbf{U}_B^\top\otimes\mathbf{U}_K^\top). \end{aligned} \tag{28}$$

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 [26], as well as sparse GP variational inference [27]. We will leave these as future work.

V. Related Work

Learning on graph-structured data has been studied from both machine learning and signal processing perspectives such as [3], [4], [28], [29]. Our model is unique in that it makes use of tools from both fields to achieve interpretations of filtering and kernel learning in the graph spectral domain.

Laplacian-based functions in graph signal processing such as graph filters have been applied to data with certain smoothness assumptions, thus transforming data into one of low-, band- or high-pass profiles [3], [4]. In contrast, our algorithm learns the filter based on the data to exempt the need for choosing the filter profile a priori. This extends the non-probabilistic approach in [30], [31] with the added benefit of producing a measure of uncertainty. From the kernel methods perspective, our model is based on Smola et al [2] where kernels are derived by regularization; the dependence on graphs is achieved by using the graph Laplacian in place of the Laplace operator, thus graph smoothness is enforced on the function based on the connectivity of the graph. The same concept is used in [32] to derive the graph equivalence of the Matern class of kernels. We however take the more flexible form of these graph Laplacian based kernels by using

a polynomial and adapting the shape to the data for better performances.

The node-level kernel we have defined also bears similarity to spectral designs of graph neural networks (GNNs) such as [33], [34], [21], [35]. The work in [33] proposes to learn a free-form graph filter, which does not guarantee its spatial localization. The models in [34], [21] and more recently [36], [35] do offer such localization property by controlling the degree of the polynomial that is modelling the filter, while the use of larger neighbourhoods while avoiding over-smoothing on graphs was achieved through various additional tools such as [37], [38], [39]. However, as neural network models they typically require a large amount of training data, while being a Bayesian model, the predictions our model makes also provides a level of uncertainty, a property the neural network based models lack.

In previous works relating to GP on graphs, our model resembles that of [6], but with the distinction that the kernel on the output space is learned instead of a chosen low-pass filter. More importantly, Algorithm 1 demonstrates that spectral kernel learning is possible on a discrete graph space. Previously this concept has only been applied to learning continuous kernels such as [11], and if one wishes to apply a similar concept on graphs the method will not translate trivially.

Other graph-based GP models are predominantly applied to scalar output (instead of vector output or multiple output as called in the GP literature) problems. The way the graph is utilized follows a similar framework to graph neural network models such as [21], with one representative approach being local averaging of neighbourhood information for node level classification [8]. More complicated aggregation functions have since been applied as a linear function to the GP covariance in [40], [41], [42]. Although these models may be extended to vector outputs, they generally involve averaging or smoothing of the data, and the resulting effect is similar to a low-pass filter on the graph. Hence, these models are likely to perform less well on data that are not customarily smooth. Our model overcomes this issue through spectral learning of the kernel on graphs to adapt to the data more effectively. More recently, the use of spectral graph wavelets are proposed in [43] to capture multi-scale information, while models have expanded to using deep GPs [44] to effectively predict vectorial graph signals, but this relies more on the inference step and does not learn a kernel on the graph. Finally, the convolutional patch-based technique in [45] has also been extended to graph data [46]. This method can be viewed as an extension to the approach in [8], but it is still based on pre-defined kernel functions in the graph domain.

VI. 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. (15) and we consider baseline GP models from [6], [8], and kernels on graphs defined in [2] in Eq. (17-20):

- Standard GP: $\mathbf{B} = \mathbf{I}$
- Global filtering: $\mathbf{B} = (\mathbf{I} + \alpha\mathbf{L})^{-1}$ [6]
- Local averaging: $\mathbf{B} = (\mathbf{I} + \alpha\mathbf{D})^{-1}(\mathbf{I} + \alpha\mathbf{A})$ [8] where we also added a weighting parameter α.
- Graph Laplacian regularization: $\mathbf{B}\mathbf{B}^\top = \mathbf{L}^\dagger$ (pseudo-inverse of the Laplacian) [1]
- Regularized Laplacian: $\mathbf{B}\mathbf{B}^\top = (\mathbf{I} + \alpha\tilde{\mathbf{L}})^{-1}$ [2]
- Diffusion: $\mathbf{B}\mathbf{B}^\top = \exp\{(-\alpha/2)\tilde{\mathbf{L}}\}$ [2]
- p-step random walk: $\mathbf{B}\mathbf{B}^\top = (\alpha\mathbf{I} - \tilde{\mathbf{L}})^p$ [2]
- Cosine: $\mathbf{B}\mathbf{B}^\top = \cos(\tilde{\mathbf{L}}\pi/4)$ [2]

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 meaning $\boldsymbol{\mu}_*$ and covariance $\boldsymbol{\Sigma}_*$ defined in Eq. (3). To provide a measure of robustness, the test set is split into 10 subsets for which we compute the batch posterior log-likelihoods on each subset, and we 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.

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. (20) 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.

We provide the code for producing the results in this paper in the following: https://github.com/yincong-zhi/Polynomial_Kernel_Learning.

A. 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

$$l = \text{Mean}(\{||\mathbf{y}_1||_2^2, \ldots, ||\mathbf{y}_N||_2^2\}) \tag{29}$$

$$\sigma_w^2 = \text{Var}([\mathbf{y}_1; \ldots; \mathbf{y}_N)] \tag{30}$$

with σ_w^2 is set as the variance of the flattened vector from the response data matrix $[\mathbf{y}_1; \ldots; \mathbf{y}_N]$. 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:

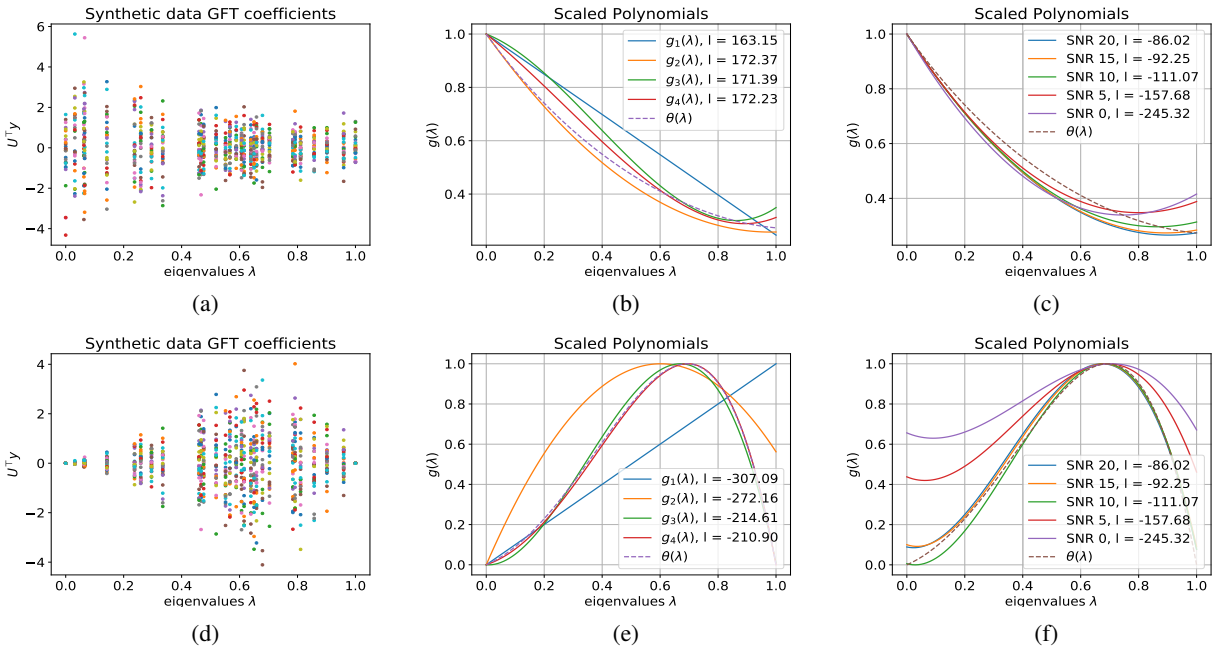


Fig. 2: Spectral kernel learning on synthetic Sensor graphs. We show the synthetic graph Fourier coefficients in (a) and (d) (each colour represents a signal), and the scaled polynomials learned with their log-marginal likelihoods, for data with low- (first row) and band-pass (second row) spectra. Ground 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 (b) and (c) compare the degrees of the polynomial, (c) and (f) compare the SNR noise levels.

1) Find the optimal $\boldsymbol{\beta}$ and σ_ϵ^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 elements β_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 σ_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 (σ_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.

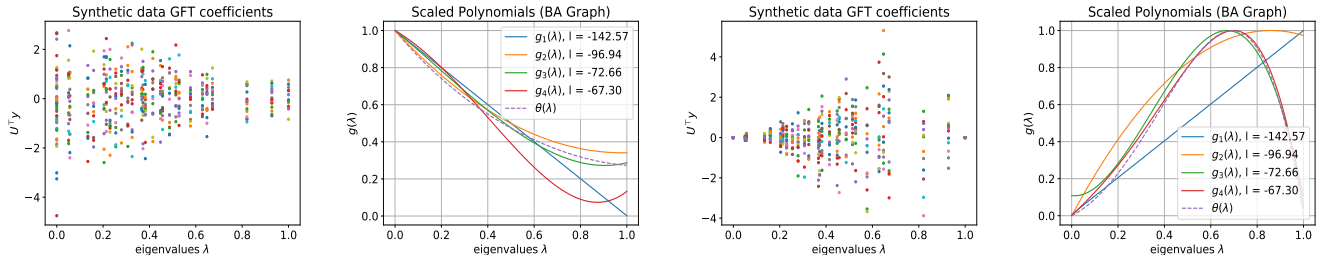
As a final note, we follow a general rule for selecting the learning rate for each parameter in the gradient optimization as: choosing the largest $r \in \mathbb{Z}$ such that $\gamma_p = 10^r$ for hyperparameter p, that leads to a consistent increase/decrease in the objective function. This will require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

B. Synthetic Experiments

1) Synthetic Filter Reconstruction: For the first experiment we use synthetic signals which are generated following Eq. (13) 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 [47]. 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 σ_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 σ_w^2.

In Fig. 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. 2a and Fig. 2d we plot the graph Fourier coefficients $\mathbf{U}^\top\mathbf{y}$ of the generated signals (using the ground truth filters). The learned polynomials with different degrees can be found in Fig. 2b and 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 truth 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



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

Fig. 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. 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 $e^{-1.5}$) for (a) and (b), and $\theta = (0, 1, 4, 1, -6)$ for the band-pass (c) and (d).

TABLE I: Synthetic test log-likelihoods (standard error), higher the better. **Low**, **Band**, **High** are pre-chosen ground truth profiles, while **Filtering** is set to $(\mathbf{I}+\mathbf{L})^{-1}$ to match the model of [6].

Model	Low	Band	High	Filtering
Degree 1	-13.79 (0.13)	-81.02 (0.24)	-132.05 (0.29)	-38.18 (0.17)
Degree 2	-13.48 (0.14)	-76.39 (0.22)	-127.70 (0.30)	**-37.80** (0.17)
Degree 3	**-13.23 (0.14)**	**-65.54 (0.21)**	**-127.24 (0.30)**	-37.95 (0.17)
Degree 4	-13.88 (0.13)	-67.43 (0.22)	**-127.24 (0.31)**	-37.90 (0.17)
Standard GP	-33.41 (0.13)	-82.41 (0.19)	-171.50 (0.37)	-48.28 (0.19)
Laplacian [1]	-156.44 (0.12)	-96.94 (0.25)	-205.54 (0.92)	-190.52 (0.14)
Global Filtering [6]	-15.90 (0.15)	-83.03 (0.19)	-204.43 (0.86)	-38.80 (0.17)
Local Averaging [8]	-30.23 (0.18)	-99.26 (0.29)	-424.96 (17.58)	-70.79 (0.24)
Regularized Lap [2]	-16.38 (0.15)	-83.08 (0.19)	-206.02 (0.95)	-40.59 (0.18)
Diffusion [2]	-15.81 (0.15)	-82.42 (0.19)	-205.91 (0.94)	-38.76 (0.17)
1-Step RW [2]	-17.74 (0.15)	-83.13 (0.19)	-356.29 (0.32)	-39.85 (0.17)
3-Step RW [2]	-19.00 (0.15)	-88.03 (0.24)	-173.31 (0.36)	-45.14 (0.19)
Cosine [2]	-19.56 (0.15)	-86.35 (0.19)	-185.03 (0.29)	-38.97 (0.17)

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. 2c and 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 decrease as SNR decreases when data becomes noisier.

2) 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. 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. 3 show the low- and band-pass filter shapes are still recovered well using Algorithm 1.

3) 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 [7] 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 Δ of size $M \times N$. Each column in Δ is of dimension M, forming a signal on the graph. We use a Sensor graph again with $M = 25$ nodes with Laplacian $\mathbf{L}_S$. Next, let $\mathbf{r}_i$ denote the ith row of Δ, we filter this signal by

$$\mathbf{y}_i = \theta(\mathbf{L}_S)\mathbf{r}_i. \tag{31}$$

We test on 4 different ground truths θ, this includes firstly the same low- and band-pass polynomials as the previous section, as well as the addition of a high-pass $\theta = (0, 1.5, 1.5^2/2, 1.5^2 3/6, 1.5^4/24)$ (first 5 terms of $e^{1.5} - 1$), and finally we also test on $\theta(\mathbf{L}) = (\mathbf{I} + \alpha\mathbf{L})^{-1}$ for $\alpha = 1$ which is also low-pass but matches the generative model of [6]. 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

$$\mathbf{C} \otimes \mathbf{B}\mathbf{B}^\top + \sigma_\epsilon^2 \mathbf{I}. \tag{32}$$

After running Algorithm 1 of log-marginal likelihood maximization, we compute the posterior by conditioning on the first 20 signals $\mathbb{P}(\mathbf{y}_*|\mathbf{y}_1, \ldots, \mathbf{y}_{20})$, to get the posterior log-likelihood on the remaining test signals. We present these performances against the baselines in Table I, with the **Filtering** column corresponding to $\theta(\mathbf{L}) = (\mathbf{I} + \mathbf{L})^{-1}$. Where there are significant improvements are on band- and high-pass data, which demonstrates our model's ability to capture the higher frequency elements in the data, on low-pass signals the baseline profiles now match the data, leading to a number of models producing similar test log-likelihoods to that of the polynomials. The difference is expectedly the smallest between the polynomials and global filtering [6] on the **Filtering**

TABLE II: Real world test log-likelihoods (standard error), higher the better.

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 [1]	-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 [8]	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 [6]	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 [2]	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 [2]	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 [2]	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 [2]	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 [2]	10.09 (0.11)	10.55 (0.14)	-54.99 (4.60)	-53.83 (4.01)	-27.64 (1.18)	-26.64 (1.44)

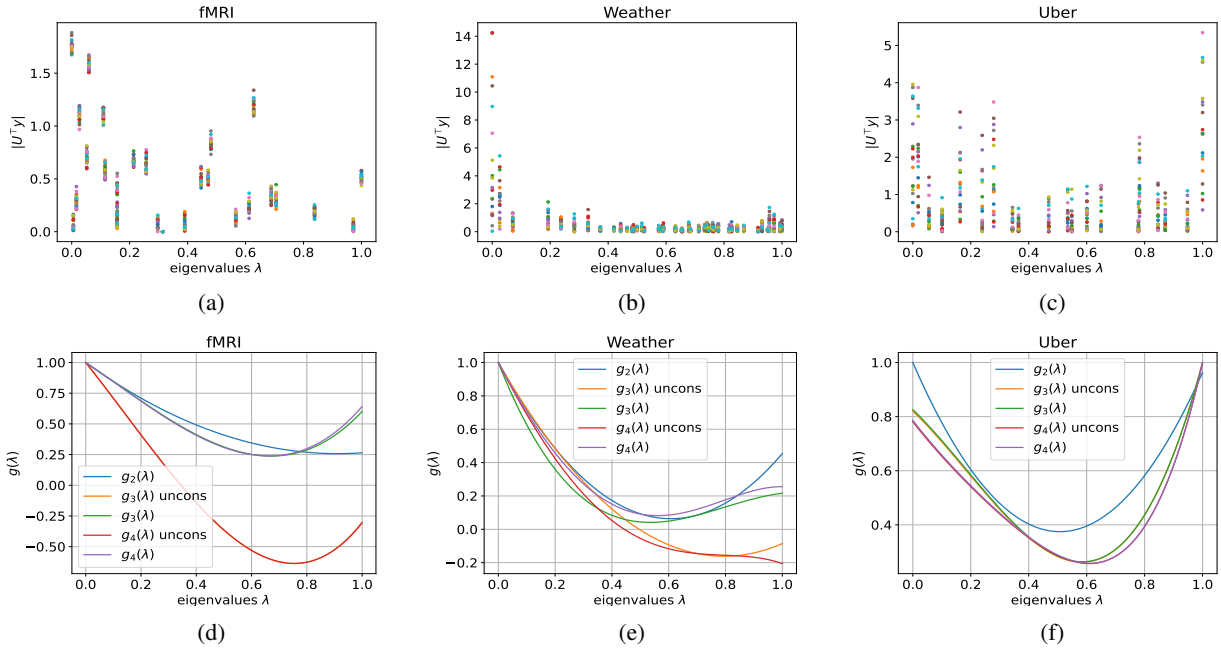


Fig. 4: (a) - (c) Real world data magnitude of graph Fourier transform coefficients of the training signals $|\mathbf{U}^\top y|$. (d) - (f) Polynomial filters learned on the corresponding datasets (on the larger training set). The polynomials generally picked up non-low-pass elements, leading to the predictions in Fig. 5 varying in a more similar manner to the test signals.

ground truth. Overall, using a polynomial still offers marginal improvements on low-pass data, but is the most advantageous when there are higher frequency elements.

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.

C. 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.

1) 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 [6], [48] 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 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

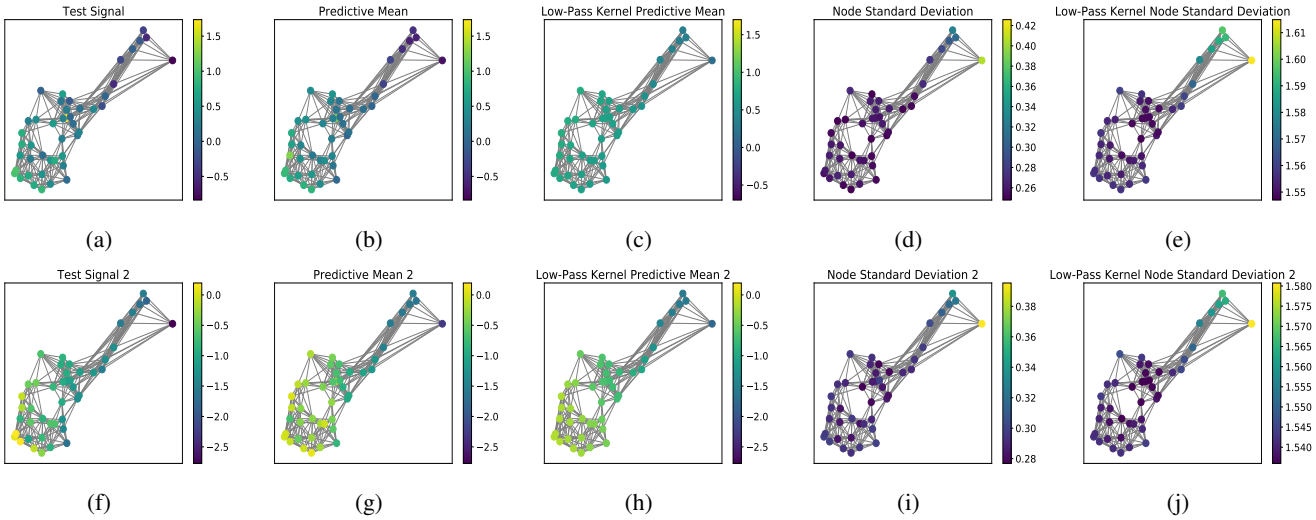


Fig. 5: Test signals $\mathbf{z}_t$ from Weather dataset (a) $\mathbf{z}_t^\top \mathbf{L}\mathbf{z}_t = 19.84$ & (f) $\mathbf{z}_t^\top \mathbf{L}\mathbf{z}_t = 36.35$; GP predictions from degree 3 polynomial $\mathbf{z}_p$ (b) $\mathbf{z}_p^\top \mathbf{L}\mathbf{z}_p = 16.28$ (g) $\mathbf{z}_p^\top \mathbf{L}\mathbf{z}_p = 29.56$ (d) and (i); from a low-pass of [6] $\mathbf{z}_l$ (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.

to 42 signals to learn the hyperparameters, we then compute the posterior to predict the remaining 250 test signals. The model's polynomial filters are presented in Fig. 4d while the GFT coefficients are displayed in Fig. 4a to show how the polynomials resembled the data. The posterior log-likelihoods are presented in the first two columns of Table II, where we see our polynomial learning was significantly better than any of the baseline models. The polynomial results did have slightly higher standard error, which may be due to the model not consistently producing equally high likelihoods for every test signal, but all polynomials produced higher test log-likelihoods than all baselines. The spectrum of the data has a relatively smooth shape as shown in Fig. 4a and so a low degree polynomial was able to capture the spectrum well and using a higher degree has little improvements on the 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 that our model predicts with.

2) Weather Dataset: We now consider the temperature measurement in 45 cities in Sweden available from SMHI [49]. Using the cities' longitude and latitude, we construct a k-nearest neighbour graph for $k = 10$ using the function from PyGSP [47]. 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. 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 II middle two columns and the polynomial filters are shown in Fig. 4e, where again, we can see the significant difference between the polynomial models and the baselines. Furthermore, by doing next-day prediction, our signals are on the whole graph, allowing us to visualize them in Fig. 5. Here, we compared the predictions of a degree 3 polynomial, and that of [6] which represents a typical low-pass filter, something all baseline models have in common. We also reported the graph smoothness of each signal in the figure, 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. 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 from the polynomials meaning our model predicts with much more certainty, this is one of the main reasons the polynomial log-likelihoods are much higher.

3) 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 [50]). 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 II, where the mean and standard errors are over

10 splits like previous experiments. The data is of low- and high-pass nature as shown by the GFT coefficients 4c and reflected by the filters learned in Fig. 4f, thus all our models offered improvements compared to the baselines due to the high frequency information picked up in polynomial filter. Similar to the fMRI dataset, the standard errors of our models are slightly higher than the baselines, this again is due to the model not predicting some test signals as well as others, but all polynomial predictions still had higher log-likelihoods than all baselines.

VII. 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 better 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 scalability of the model.

References

[1] 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.

[2] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings*, pages 144–158. Springer, 2003.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Yin Cheng Ng, Nicolò Colombo, and Ricardo Silva. Bayesian semi-supervised learning with graph gaussian processes. In *Advances in Neural Information Processing Systems*, pages 1683–1694, 2018.

[9] 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.

[10] 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.

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

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

[13] Jian Li, Yong Liu, and Weiping Wang. Automated spectral kernel learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4618–4625, 2020.

[14] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[15] Fan R. K. Chung. *Spectral graph theory*. Regional conference series in mathematics. no. 92. Published for the Conference Board of the mathematical sciences by the American Mathematical Society, Providence, R.I., 1997.

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

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

[18] 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.

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

[20] Sun Yuan Kung. *Kernel methods and machine learning*. Cambridge University Press, 2014.

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[22] Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer, 2014.

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

[24] 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.

[25] Xingyue Pu, Siu Lun Chau, Xiaowen Dong, and Dino Sejdinovic. Kernel-based graph learning from smooth signals: A functional viewpoint. *IEEE Transactions on Signal and Information Processing over Networks*, 7:192–207, 2021.

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

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

[28] 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.

[29] 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.

[30] Xuan Zhang, Xiaowen Dong, and Pascal Frossard. Learning of structured graph dictionaries. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3373–3376. IEEE, 2012.

[31] Dorina Thanou, David I Shuman, and Pascal Frossard. Parametric dictionary learning for graph signals. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 487–490. IEEE, 2013.

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

[33] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations*, 2014.

[34] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[35] Mingguo He, Zhewei Wei, Hongteng Xu, et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in Neural Information Processing Systems*, 34:14239–14251, 2021.

[36] Yushun Dong, Kaize Ding, Brian Jalaian, Shuiwang Ji, and Jundong Li. Adagnn: Graph neural networks with adaptive frequency response filter. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 392–401, 2021.

[37] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3496–3507, 2021.

[38] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019.

[39] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

[40] Felix Opolka and Pietro Lio. Bayesian link prediction with deep graph convolutional gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 4835–4852. PMLR, 2022.

[41] Pengyu Cheng, Yitong Li, Xinyuan Zhang, Liqun Chen, David Carlson, and Lawrence Carin. Dynamic embedding on textual networks via a gaussian process. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7562–7569, 2020.

[42] Zhaoyang Liu, ShaoYuan Li, Song can Chen, Yao Hu, and Sheng-Jun Huang. Uncertainty aware graph gaussian process for semi-supervised learning. In *AAAI*, 2020.

[43] Felix Opolka, Yin-Cong Zhi, Pietro Lio, 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.

[44] Naiqi Li, Wenjie Li, Jifeng Sun, Yinghua Gao, Yong Jiang, and Shu-Tao Xia. Stochastic deep gaussian processes over graphs. *Advances in Neural Information Processing Systems*, 33, 2020.

[45] Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional gaussian processes. In *Advances in Neural Information Processing Systems*, pages 2849–2858, 2017.

[46] Ian Walker and Ben Glocker. Graph convolutional gaussian processes. In *ICML*, 2019.

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

[48] 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.

[49] Swedish Meteorological and Hydrological Institute (SMHI). http://opendata-download-metobs.smhi.se/, 2013. [Online]. Accessed: 2019-02-13.

[50] 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.