Journal of Machine Learning Research 22 (2021) 1-36 Submitted 1/21; Revised 8/21; Published 9/21

# A general linear-time inference method for Gaussian Processes on one dimension

**Jackson Loper** JL5116@COLUMBIA.EDU
*Data Science Institute*
*Columbia University*
*New York, New York 10027*

**David Blei** DAVID.BLEI@COLUMBIA.EDU
*Data Science Institute*
*Departments of Statistics and Computer Science*
*Columbia University*
*New York, New York 10027*

**John P. Cunningham** JPC2181@COLUMBIA.EDU
*Department of Statistics*
*Mortimer B. Zuckerman Mind Brain Behavior Institute*
*Grossman Center for the Statistics of Mind*
*Columbia University*
*New York, New York 10027*

**Liam Paninski** LMP2107@COLUMBIA.EDU
*Departments of Statistics and Neuroscience*
*Mortimer B. Zuckerman Mind Brain Behavior Institute*
*Grossman Center for the Statistics of Mind*
*Columbia University*
*New York, New York 10027*

**Editor:** Philipp Hennig

## Abstract

Gaussian Processes (GPs) provide powerful probabilistic frameworks for interpolation, forecasting, and smoothing, but have been hampered by computational scaling issues. Here we investigate data sampled on one dimension (e.g., a scalar or vector time series sampled at arbitrarily-spaced intervals), for which state-space models are popular due to their linearly-scaling computational costs. It has long been conjectured that state-space models are general, able to approximate any one-dimensional GP. We provide the first general proof of this conjecture, showing that any stationary GP on one dimension with vector-valued observations governed by a Lebesgue-integrable continuous kernel can be approximated to any desired precision using a specifically-chosen state-space model: the Latent Exponentially Generated (LEG) family. This new family offers several advantages compared to the general state-space model: it is always stable (no unbounded growth), the covariance can be computed in closed form, and its parameter space is unconstrained (allowing straightforward estimation via gradient descent). The theorem's proof also draws connections to Spectral Mixture Kernels, providing insight about this popular family of kernels. We develop parallelized algorithms for performing inference and learning in the

LEG model, test the algorithm on real and synthetic data, and demonstrate scaling to datasets with billions of samples.

## 1. Introduction

Gaussian Process (GP) methods are a powerful and expressive class of nonparametric techniques for interpolation, forecasting, and smoothing (Rasmussen and Williams, 2005). However, this expressiveness comes at a cost: if implemented naively, inference in a GP given $n$ observed data points will require $O(n^3)$ operations. A large body of work has devised various means to circumvent this cubic run-time; briefly, this literature can be broken down into several threads. A first approach is to attempt to perform exact inference without imposing any restrictions on the covariance kernel, using careful numerical methods typically including preconditioned conjugate gradients (Cutajar et al., 2016). Wang et al. (2019) represents the state of the art: inference and learning can be performed on $\sim 10^6$ datapoints on an 8-GPU machine and a few days of processing time. A second approach searches for good approximations to the posterior (e.g., low-rank approximations to the posterior covariance) that do not rely on special properties of the covariance kernel. Some well-known examples of this approach include (Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2007; Hensman et al., 2013; Low et al., 2015; De G. Matthews et al., 2017). In a third approach, several techniques exploit special kernel structure. Examples include matrices with Kronecker product structure (Gilboa et al., 2013), Toeplitz structure (Zhang et al., 2005; Cunningham et al., 2008), approximate bandedness (Gonzalez et al., 2009), matrices that can be well-approximated with hierarchical factorizations (Ambikasaran et al., 2015), or matrices which are sufficiently smooth to allow for interpolation-based approximations (Wilson and Nickisch, 2015).

When the GP has one-dimensional input, one popular method for scaling learning and inference is to approximate the GP with some form of Gaussian hidden Markov model (that is, a state-space model) (Reinsel, 2003; Mergner, 2009; Cheung et al., 2010; Brockwell and Davis, 2013). This model class has considerable virtue: it is a particular case of a GP, it includes popular models like the auto-regressive moving average process (ARMA, when on an evenly spaced grid), and perhaps most importantly it admits linear-time $O(n)$ inference via message passing.

It has long been conjectured that state-space models are not only fast, but general. In several special cases it has been shown that state-space models provide arbitarily-good approximations of specific covariance kernels (Karvonen and Sarkkä, 2016; Benavoli and Zaffalon, 2016; Särkkä and Solin, 2019). Discrete-time processes on a finite interval can also be approximated this way (Lindgren et al., 2011). The idea is simple: by taking a sufficiently high-dimensional latent space, one can produce processes whose observed covariance is quite intricate. Much progress has been made in understanding how quickly such approximations converge, i.e. how many latent dimensions are required (Hartikainen and Särkkä, 2010; Särkkä and Piché, 2014; Karvonen and Sarkkä, 2016). Practically, Gilboa et al. (2013) suggests it is straightforward to learn many GP models using state-space techniques.

However, no general proof of this conjecture exists, to our knowledge. Karvonen and Sarkkä (2016) provide a good example of the results in this vein that have already been achieved: theoretical tools for proving that particular Gaussian Process models can be

approximated by state-space models. These tools represent a promising advance from what came before, insofar as they can be applied to arbitrary kernels, allowing one to transform problems of kernel approximation into more elegant questions about Taylor series convergence. However, they do have some disadvantages. First, the proofs only show convergence for a compact subset of the Fourier transform of the kernels; thus the approximations might have arbitrarily poor high-frequency behavior. Second, the methods do not apply if one measures a vector of observations at each time-point.

Here we present a new theorem which lays the "generality" question to rest for time-series data with vector-valued observations. It is based on a connection between state-space models and Spectral Mixture kernels (Wilson and Adams, 2013). The new theorem shows that any stationary GP on one dimension with vector-valued observations and a Lebesgue-integrable continuous kernel can be uniformly approximated by a state-space model. Along the way, we also prove a similar generality result for Spectral Mixture kernels of the form $\Sigma : \mathbb{R} \to \mathbb{R}^{D \times D}$.

To prove this theorem, we develop a convenient family of Gaussian hidden Markov models on one dimension: the Latent Exponentially Generated (LEG) process. This model has several advantages over the general state-space models: its covariance can be computed in closed form; it is stable (with no unbounded growth); and its parameter space is unconstrained (allowing for simple gradient descent methods to be applied to learn the model parameters). The LEG model generalizes the Celerité family of Gaussian Processes (Foreman-Mackey et al., 2017) to allow more model flexibility and permit vector-valued observations. Unlike some popular state-space models such as the ARMA, LEG processes do not require that the observations are equally spaced. LEG kernels have an intersection with spectral mixture kernels (Wilson and Adams, 2013), which enables a proof that these LEG kernels can approximate any integrable continuous kernel. Thus LEG models are general: our main mathematical result is to prove that for any stationary Gaussian Process $x$ on one dimension with integrable continuous covariance, for any $\varepsilon$, the covariance of $x$ can be uniformly matched within $\varepsilon$ by a LEG covariance kernel. Finally, inference for the LEG processes can be parallelized efficiently via a technique known as Cyclic Reduction (Sweet, 1974), leading to significant runtime improvements.

To summarize, this paper contains three contributions. First, it is shown for the first time that any vector-valued GP on one dimension with a stationary integrable kernel can be approximated to arbitrary accuracy by a state-space model. Second, an unconstrained parameterization for state-space models is developed (the LEG family). Finally, based on that parameterization, this paper introduces an open-source, parallel-hardware, linear-time package for learning, interpolating, and smoothing vector-valued continuous-time state-space models with irregularly-timed observations.

The remainder of this paper defines the LEG family, explores its connections to other families of kernels, derives its generality, and finally empirically backs up these claims across real and synthetic data. In particular, we show that the LEG family enables inference on datasets with billions of samples with runtimes that scale in minutes, not days.

## 2. Latent Exponentially Generated (LEG) kernels

Consider a zero-mean Gaussian Process on one dimension, i.e. a random function $\mathbf{x} : \mathbb{R} \to \mathbb{R}^D$ such that for any finite collections of times $t_1, t_2 \cdots t_n$ the distribution of $(\mathbf{x}(t_1), \cdots \mathbf{x}(t_n)) \in$

$\mathbb{R}^{D\times n}$ is jointly a zero-mean Gaussian. The covariance kernel of $\mathbf{x}$ is a matrix-valued function defined by $\Sigma(s,t) = \mathbb{E}[\mathbf{x}(s)\mathbf{x}(t)^\top] \in \mathbb{R}^{D\times D}$. A process is said to be stationary if $\Sigma(s,t) = C(s-t)$ for some matrix-valued function $C$ and $\mathbb{E}[x(t)]$ is the same for all values of $t$. In this case we write $\tau$ for the time-lag $s-t$, i.e $C = C(\tau)$. For clarity of exposition, what follows assumes stationarity and zero mean; generalizations are discussed in Section 6.

Some stationary Gaussian Processes are also *linear state-space models*. For the purposes of this article, we will say a process is a state-space model if there is a Markovian Gaussian Process $\mathbf{z} : \mathbb{R} \to \mathbb{R}^Q$ and at each timepoint $t$ we independently sample $\mathbf{x}(t) \sim \mathcal{N}(B\mathbf{z}(t), \Lambda\Lambda^\top)$ with some matrices $B, \Lambda$. The process $\mathbf{z}$ is called the "latent" process: it is used to define a distribution on the observations, $\mathbf{x}$, but it may not actually refer to any observable property of the physical world. The matrices $B, \Lambda$ are referred to as the "observation model."

This paper was born from the following question: can *any* stationary covariance kernel be approximated to arbitrary accuracy by a state-space model?

To answer this question, we found it convenient to design a family of linear state-space models which we call the "Latent Exponentially-Generated" family. This family has several convenient features which enabled simpler analysis: the family can be indexed with an unconstrained parameter space, every member of the family is stationary, and it is easy to compute marginal covariances. To define this family, we will start by defining a latent Markovian GP, then we will define the observation model. Taken together these objects will form the LEG family of state-space models.

**Definition 1** *Let $\mathbf{z}(0) \sim \mathcal{N}(0, I)$, let $\mathbf{w}$ denote a standard Brownian motion, let $N, R$ be any $Q \times Q$ matrices, and let $G = NN^\top + R - R^\top$. Let $\mathbf{z}$ satisfy $\mathbf{z}(t) = \mathbf{z}(0) + \int_0^t \left(-\frac{1}{2}G\mathbf{z}(s)ds + Nd\mathbf{w}(s)\right)$. Then we will say $\mathbf{z}$ is a Purely Exponentially Generated process, $\mathbf{z} \sim \mathrm{PEG}(N, R)$. Applying the standard formulae for the covariances of linear stochastic differential equations, we obtain that the covariance kernel of $\mathbf{z}$ is given by the formula below (we refer the reader to Appendix C for a brief review).*

$$C_{\mathrm{PEG}}(\tau; N, R) = \begin{cases} \exp\left(-\frac{|\tau|}{2}G\right) & \text{if } \tau \geq 0 \\ \exp\left(-\frac{|\tau|}{2}G^\top\right) & \text{if } \tau \leq 0 \end{cases}$$

*If the real parts of the eigenvalues of $G$ are strictly positive, the spectral representation of the PEG kernel may be calculated as*

$$C_{\mathrm{PEG}}(\tau; N, R) = \frac{1}{2\pi}\int \exp(-i\omega\tau)\left[(G/2 - i\omega I)^{-1} + (G^\top/2 + i\omega I)^{-1}\right] d\omega$$

*We refer the reader to Cramér (1940) for a collection of first properties on such spectral representations; we also briefly review these properties in Appendix C.*

The matrices $N, R$ can be interpreted intuitively. The positive definite diffusion $NN^\top$ controls the predictability of the process: when an eigenvalue of $NN^\top$ becomes larger, the process $\mathbf{z}$ becomes less predictable along the direction of the corresponding eigenvector. The antisymmetric $R - R^\top$ term affects the process by applying an infinitesimal deterministic rotation at each point in time. The eigenvalues of $R - R^\top$ are purely imaginary, and when they are large they lead to strong rapid oscillations in the process, while the eigenvectors

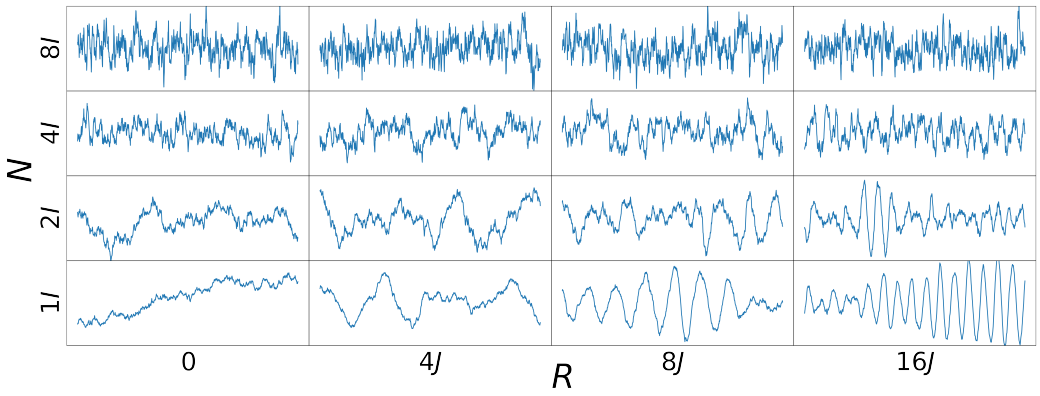


Figure 1: **PEG process samples**. The plots above show representative samples from the model $\mathrm{PEG}(N, R)$ as we vary $N$ and $R$. Here we consider rank-2 PEG models (only one element of the 2d vector is plotted), so $N, R$ are both $2 \times 2$ matrices. We vary $N$ by taking it to be various multiples of the identity. We vary $R$ by taking various multiples of $J$, the antisymmetric $2 \times 2$ matrix with zeros on the diagonal and $\pm 1$ on the off-diagonal. In this simple rank-2 case, increasing $N$ leads to a less predictable process and increasing $R$ leads to faster oscillations.

control how these oscillations are mixed across the dimensions of $\mathbf{z}$. As an illustration, Figure 1 shows the first dimension of samples from a $Q = 2$ PEG process with various values of $N, R$.

We now turn to the observed process:

**Definition 2** *Let $\mathbf{z} \sim \mathrm{PEG}(N, R)$. Fix any $D \times Q$ matrix $B$ and any $D \times D$ matrix $\Lambda$. For each $t$ independently, define the conditional observation model: $\mathbf{x}(t)|\mathbf{z}(t) \sim \mathcal{N}(B\mathbf{z}(t), \Lambda\Lambda^\top)$. We define a **Latent Exponentially Generated (LEG)** process to be the Gaussian Process $\mathbf{x} : \mathbb{R} \to \mathbb{R}^D$ generated by a PEG prior and the above observation model. We write $\mathbf{x} \sim \mathrm{LEG}(N, R, B, \Lambda)$. The formula for the covariance of $\mathbf{x}$ is given below.*

$$C_{\mathrm{LEG}}(\tau; N, R, B, \Lambda) = \begin{cases} B\left(C_{\mathrm{PEG}}(\tau; N, R)\right) B^\top & \text{if } \tau = 0 \\ B\left(C_{\mathrm{PEG}}(\tau; N, R)\right) B^\top + \Lambda\Lambda^\top & \text{otherwise} \end{cases}$$

*When $\Lambda = 0$, we may suppress that parameter by the abuse of notation $C_{\mathrm{LEG}}(N, R, B) = C_{\mathrm{LEG}}(N, R, B, 0)$. We will refer to the latent dimension $Q$ as the **rank** of the LEG kernel and $D$ as the **dimension** of the kernel.*

In the next section we will see how the LEG family of kernels is connected to other popular families. Before we do so, let us collect a few elementary properties about the LEG family itself.

**Lemma 3 (Connections among LEG kernels)** *Let $\Sigma = C_{\mathrm{LEG}}(N, R, B, \Lambda)$ denote a LEG kernel of dimension $D$ and rank $Q$. Let $\tilde{\Sigma} = C_{\mathrm{LEG}}(\tilde{N}, \tilde{R}, \tilde{B}, \tilde{\Lambda})$ denote a LEG kernel of dimension $\tilde{D}$ and rank $\tilde{Q}$. Let $\otimes$ denote the Kronecker product and $\oplus$ denote the direct sum.*

1. *If $D = \tilde{D}$, then the kernel $\tau \mapsto \Sigma(\tau) + \tilde{\Sigma}(\tau)$ lies in the family of LEG kernels of rank $Q + \tilde{Q}$.*
2. *The kernel $\tau \mapsto \Sigma(\tau) \otimes \tilde{\Sigma}(\tau)$ lies in the family of LEG kernels of rank $Q\tilde{Q}$ and dimension $D\tilde{D}$.*
3. *Let $\gamma > 0$. The kernel $\tau \mapsto \Sigma(\tau/\gamma)$ lies in the family of LEG kernels of rank $Q$ and dimension $D$.*

**Proof** We can construct each new kernel explicitly in the LEG form.

1. Take $C_{\text{LEG}}(N \oplus \tilde{N}, R \oplus \tilde{R}, (B\ \tilde{B}), \Lambda + \tilde{\Lambda})$.
2. Take $C_{\text{LEG}}((NN^\top \otimes I + I \otimes \tilde{N}\tilde{N}^\top)^{1/2}, R \otimes I + I \otimes \tilde{R}, B \otimes \tilde{B}, (\Lambda\Lambda^\top \otimes \tilde{\Lambda}\tilde{\Lambda}^\top)^{1/2})$. To see that this kernel matches the desired Kronecker product, we use the fact that $\exp(A \otimes I + I \otimes \tilde{A}) = \exp(A) \otimes \exp(\tilde{A})$; we refer the reader to Neudecker (1969) for exposition on this subject.
3. Take $C_{\text{LEG}}(N/\sqrt{\gamma}, R/\gamma, B, \Lambda)$.

■

## 3. Connections between LEG kernels and other model families

We here summarize the relationship between the LEG family and several popular families of stationary positive-definite kernels on one dimension. We first define each family in turn and then present a Lemma describing some connections between these families and the LEG family.

**Spectral mixtures.** Spectral mixture kernels, as introduced by Wilson and Adams (2013), define a covariance kernel by taking the Fourier transform of a weighted sum of different shifts of a probability density. It was originally designed for GPs of the form $\mathbf{x} : \mathbb{R}^d \to \mathbb{R}$; here we generalize this to ones of the form $\mathbf{x} : \mathbb{R}^d \to \mathbb{R}^D$. Let $\hat{K} : \mathbb{R}^d \to \mathbb{R}$ denote any positive definite kernel. Let $\gamma > 0$, and $b \in \mathbb{C}^D$. We define a Spectral Mixture Term by

$$C_{\text{SMT}}(\tau; \hat{K}, b, \mu, \gamma) = \Re(\exp(-i\tau\mu)bb^*)\hat{K}(\tau/\gamma),$$

where $\Re(x)$ takes the real part of a matrix and $b^*$ denotes the conjugate transpose of $b$. We will say such a term is **based on** $\hat{K}$. Spectral mixture terms are the result of applying six transformations to $\hat{K}$: taking the Fourier transform, scaling the domain by $\gamma$, shifting the domain by $\mu$, multiplying by $bb^*/\gamma$, taking the inverse Fourier transform, and taking the real part. We define a Spectral Mixture kernel $C_{\text{SM}} : \mathbb{R}^d \to \mathbb{R}^D$ with $Q$ components to be any sum of $Q$ spectral mixture terms each carrying the same choice for $\hat{K}, \gamma$.

**Matern.** Let $K_\nu$ denote the modified bessel function of the second kind. The standard Matern kernel of order $\nu$ is given by $C_{\text{MTN}}(\tau; \nu) = 2^{1-\nu}(\tau\sqrt{2\nu})^\nu K_\nu(\tau\sqrt{2\nu})/\Gamma(\nu)$. We refer the reader to Särkkä and Solin (2019) for details on this popular family of kernels.

**Celerité terms.** Developed by Foreman-Mackey et al. (2017), a Celerité term is given by $C_{\mathrm{CLR}}(\tau; a, b, c, d) = a\exp(-c|\tau|)\cos(d|\tau|) + b\exp(-c|\tau|)\sin(d|\tau|)$. Such terms are not always valid kernels, i.e. there are parameters $a, b, c, d$ such that $C(\cdot; a, b, c, d)$ is not the kernel of *any* Gaussian Process. If a Celerité term is the kernel of a Gaussian Process, it is said to be "positive-definite."

**Stationary Ornstein-Uhlenbeck.** Let $G \in \mathbb{R}^{Q\times Q}$ such that that the real part of the eigenvalues of $G$ are strictly positive. Let $N \in \mathbb{R}^{Q\times Q}$ denote any other matrix. Then there exists a unique symmetric nonnegative-definite matrix $P_\infty$ satisfying the continuous Lyapunov equation $GP_\infty + P_\infty G^\top = 2NN^\top$ (cf. Chapter 2 of Artemiev and Averina (1997)). Let $\mathbf{z}(0) \sim \mathcal{N}(0, P_\infty)$ and $\mathbf{z}(t) = \mathbf{z}(0) + \int_0^t \left(-\frac{1}{2}G\mathbf{z}(s)ds + Nd\mathbf{w}(s)\right)$, where $\mathbf{w}$ is a standard Brownian motion. Then $\mathbf{z}$ is a stationary Gaussian process, known as the Ornstein-Uhlenbeck process parameterized by $G, N$. The covariance kernel of $\mathbf{z}$ is given below (we refer the reader to Appendix C for a brief review of such covariances).

$$C_{\mathrm{OU}}(\tau; G, N) = \begin{cases} \exp(-G|\tau|/2)P_\infty & \text{if } \tau \geq 0 \\ P_\infty \exp(-G^\top|\tau|/2) & \text{if } \tau \leq 0 \end{cases}$$

We will refer to $Q$ as the **dimension** of the kernel. Note that if $G + G^\top = 2NN^\top$ then $P_\infty = I$ and this is exactly equal to a PEG kernel.

**Stationary linear state-space.** Let $C_{\mathrm{OU}}(G, N)$ denote a stationary Ornstein-Uhlenbeck kernel of dimension $Q$. Fix any matrix $B \in \mathbb{R}^{D\times Q}$. We define the stationary state-space model kernel family as follows:

$$C_{\mathrm{SSM}}(\tau; G, N, B) = BC_{\mathrm{OU}}(\tau; G, N)B^\top.$$

We will refer to $Q$ as the **rank** of the kernel and $D$ as the **dimension** of the kernel. Such kernels are always nonnegative definite, though they are not guaranteed to be strictly positive definite; for example, if $Q < D$ the kernel will fail to be strictly positive definite.

Below we summmarize a few connections between these families and LEG kernels.

**Lemma 4 (Connections to other model families)**

1. *Let $G \in \mathbb{R}^{Q\times Q}$ such that the real part of all eigenvalues is strictly positive. Let $N \in \mathbb{R}^{Q\times Q}, B \in \mathbb{R}^{D\times D}$. Then $C_{\mathrm{SSM}}(G, N, B)$ lies in the family of LEG kernels of rank $Q$.*

2. *Let $\nu \in \{1/2, 3/2, \cdots\}$. Then the Matern kernel of order $\nu$ lies in the family of LEG kernels of rank $\nu + 1/2$.*

3. *Let $C_{\mathrm{LEG}}(N, R, B)$ denote a LEG kernel of dimension 1 and rank $Q$. Then every spectral mixture kernel with $\tilde{Q}$ components based on $C_{\mathrm{LEG}}(N, R, B)$ lies in the family of LEG kernels with rank $2\tilde{Q}Q$.*

4. *Every positive-definite Celerité term lies in the family of LEG kernels with rank 2.*

**Proof** We treat each point in turn.

1. Let $P_\infty = C_{\mathrm{OU}}(0; G, N)$. Without loss of generality, we assume $P_\infty$ is strictly positive definite (if any eigenvalue of $P_\infty$ is zero, it follows that $\alpha^\top \mathbf{z}(t) = 0$ for all $t$ for the corresponding eigenvector $\alpha$; we may then adopt a change of variables to a new family of state-space processes without this degeneracy). Fix any invertible matrix $\Psi$ such that $\Psi P_\infty \Psi^\top = I$ (for example, the principal square root of the inverse of $P_\infty$ will always suffice). Let $\tilde{G} = \Psi G \Psi^{-1}$, $\tilde{N} = \Psi N$, $\tilde{B} = \Psi^{-1} B$, and $R = (\tilde{G} - \tilde{G}^\top)/4$. The continuous Lyapunov equation $GP_\infty + P_\infty G^\top = 2NN^\top$ yields that $C_{\mathrm{SSM}}(\tau; G, N, B) = C_{\mathrm{LEG}}(\tau; \tilde{N}, R, \tilde{B})$.

2. Combine point 1 of this lemma with the corresponding result for state-space models (cf. Chapter 12 of Särkkä and Solin (2019)).

3. Let $\mu \geq 0$ and $b \in \mathbb{C}^D$. In light of Lemma 3, it suffices to show that $\Re(\exp(-i\tau\mu)bb^*)$ lies in the LEG family. We start by defining a skew-symmetric matrix $J$, below.

$$J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

   Let $\Sigma(\tau) = C_{\mathrm{LEG}}(\tau; 0, \mu J, (\Re(b)\ \Im(b)))$. We calculate.

$$\Sigma(\tau) = \cos(\mu\tau)\left(\Re(b)\Re(b)^\top + \Im(b)\Im(b)^\top\right) + \sin(\mu\tau)\left(-\Re(b)\Im(b)^\top + \Im(b)\Re(b)^\top\right)$$

   Applying Euler's formula, we obtain $\Sigma(\tau) = \Re(\exp(-i\tau\mu)bb^*)$, as desired.

4. Let $N_1 = \sqrt{2c - 2bd/a}$, $R_1 = \sqrt{2c^2 + 4d^2 + 2b^2d^2/a^2}$ and $N_2 = \sqrt{c + bd/a}$. Foreman-Mackey et al. (2017) show that that $|bd| < ac$ and $a, c \geq 0$ for any positive-definite Celerité term, thus $N_1, R_1, N_2 \in \mathbb{R}$. Let

$$N = \begin{pmatrix} N_1 & 0 \\ N_2 & N_2 \end{pmatrix} \quad R = \begin{pmatrix} 0 & R_1 \\ 0 & 0 \end{pmatrix} \quad B = \begin{pmatrix} \sqrt{a} & 0 \end{pmatrix}$$

   Then observe that $C_{\mathrm{CLR}}(a, b, c, d) = C_{\mathrm{LEG}}(N, R, B)$.

■

## 4. Generality of LEG kernels and Spectral Mixtures for vector-valued time-series

Here we show that spectral mixture and LEG kernels are dense in the uniform norm in the space of continuous integrable positive definite kernels on one dimension. We additionally show that for each $\xi \in \mathbb{N}$, the subset of LEG kernels associated with $\xi$-times-differentiable processes is also dense in this space.

To obtain these results we develop a generalization of a classic theorem from the kernel density estimation literature.

**Theorem 5** *Let $K, p$ denote bounded densities on $\mathbb{R}^d$. Let $g : \mathbb{R}^d \to [-M, M]^D$. Let $\gamma_\ell = \ell^{1/2d}$. Let $\mu_1, \mu_2 \cdots \sim p$, independently. For each $\ell \in 1, 2, \cdots$, define $h_\ell(\xi) = \frac{1}{\ell} \sum_{k=1}^{\ell} g(\mu_k) \gamma_\ell^d K(\gamma_\ell(\xi - \mu_k))$. Then*

$$\mathbb{P}\left( \lim_{\ell \to \infty} \sum_{i=1}^{D} \int |h_{\ell,i}(\xi) - p(\xi) g_i(\xi)| d\xi = 0 \right) = 1.$$

**Proof** We largely imitate the proof of Devroye and Wagner (Devroye and Wagner, 1979), which handles the special case that $g(x) = 1$.

Let us start by assuming $g$ takes the form $g : \mathbb{R}^d \to [0, M]$.

For almost any fixed $\omega$, we have that $\lim_{\ell \to \infty} |h_\ell(\omega) - p(\omega) g(\omega)| = 0$ almost surely. This follows from two steps:

1. *Controlling the bias.* Let

$$\begin{aligned} \bar{h}_\ell(\omega) &= \mathbb{E}\left[h_\ell(\omega)\right] \\ &= \int g(x) \gamma_\ell^d K(\gamma_\ell(\omega - x)) p(x) dx \end{aligned}$$

Now fix any $\delta > 0$. We have that

$$\begin{aligned} |\bar{h}_\ell(\omega) - p(\omega) g(\omega)| \leq & \int_{\|x-\omega\| < \delta/\gamma_\ell} |p(x) g(x) - p(\omega) g(\omega)| \gamma_\ell^d K(\gamma_\ell(\omega - x)) dx \\ & + \int_{\|x-\omega\| \geq \delta/\gamma_\ell} |p(x) g(x) - p(\omega) g(\omega)| \gamma_\ell^d K(\gamma_\ell(\omega - x)) dx \end{aligned}$$

We look at each term separately:

- For $x \approx \omega$ we apply the Lebesgue differentiation theorem. Let $c = \sup K(x)$ and let $\lambda(\delta)$ denote the volume of the ball of radius $\delta$. Noting that $\gamma_\ell^d = \lambda(\delta)/\lambda(\delta/\gamma_\ell)$, we see that the integral of the error over $\|x - \omega\| < \delta/\gamma_\ell$ is bounded by

$$c\lambda(\delta) \frac{1}{\lambda(\delta/\gamma_\ell)} \int_{\|x-\omega\| < \delta/\gamma_\ell} |p(x) g(x) - p(\omega) g(\omega)| dx$$

  For any fixed $\delta$, the Lebesgue differentiation theorem shows that this goes to zero almost everywhere because $\gamma \to \infty$.

- For $\|x - \omega\| > \delta/\gamma_\ell$. Let $c = M \sup_x p(x)$. Then the integral of the error over this domain is bounded by

$$2c \int_{\|x-\omega\| \geq \delta} K(\omega - x) dx$$

  Note that we used a change of variables to drop any dependency on $\gamma_\ell$. Since $K$ is a density we can always find $\delta$ so that this is arbitrarily small.

Therefore, for any fixed $\varepsilon$ we can always find a $\delta$ which ensures that the second term is less than $\varepsilon/2$, and then ensure that the first term is less than $\varepsilon/2$ for all sufficiently large $\ell$. In short, $|\bar{h}_\ell(\omega) - p(\omega)g(\omega))| \to 0$ for each $\omega$.

2. *Controlling the variation.* Now we would like to bound $\bar{h}_\ell(\omega) - h_\ell(\omega)$. To do this we note that it is a sum of independent random variables of the form $g(\mu_k)\gamma_\ell^d K(\gamma_\ell(\omega - \mu_k))/\ell$. Letting $c = M \sup_x K(x)$ we observe that the absolute value of each random variable is bounded by $\gamma^d c/\ell$. Hoeffding's inequality then gives that

$$\mathbb{P}\left(\left|\bar{h}_\ell(\omega) - h_\ell(\omega)\right| > \varepsilon\right) \leq 2\exp\left(-\frac{2\ell t^2}{\gamma_\ell^d c}\right) = 2\exp\left(-\sqrt{\ell}\frac{2t^2}{c}\right)$$

The right-hand-side is always summable for any $t > 0$. Indeed, one may readily verify that if $f(x) = -2\exp(-c\sqrt{x})(c\sqrt{x}+1)/c^2$, then $f'(x) = \exp(-c\sqrt{x})$. For any $c > 0$ it follows that $\int_1^\infty \exp\left(-c\sqrt{x}\right) dx = 2(c+1)e^{-c}/c^2 < \infty$ and

$$\sum_\ell \mathbb{P}\left(\left|\bar{h}_\ell(\omega) - h_\ell(\omega)\right| > \varepsilon\right) < \infty$$

Applying Borel-Cantelli we find that $|\bar{h}_{y,\ell}(\omega) - h_{y,\ell}(\omega)|$ converges almost surely to zero.

Combining these steps together, we obtain a pointwise result: for almost every $\omega$, the sequence $h_1(\omega), h_2(\omega), \cdots$ converges almost surely to $p(\omega)g(\omega)$.

We must now extend this pointwise result to $\mathscr{L}^1$. To do so we start by noting that $\int |h_\ell(\omega)|d\omega \to \int |p(\omega)g(\omega)|d\omega$ almost surely. Since $g(\omega) \geq 0$, we have that

$$\int |h_\ell(\omega)|d\omega = \int h_\ell(\omega)d\omega = \frac{1}{\ell}\sum_{k=1}^{\ell} g(\mu_k)$$

Note that this is the average of independent and identically distributed bounded objects. The law of large numbers thus gives us that $\int |h_\ell(\omega)|d\omega$ converges almost surely to $\mathbb{E}_{\mu_1 \sim p}[g(\mu_1)] = \int p(\omega)g(\omega)d\omega = \int |p(\omega)g(\omega)|d\omega$. This allows us to extend our pointwise result to the desired $\mathscr{L}^1$ result via Glick's Lemma, which is stated for the benefit of the reader in Lemma 12 in Appendix C.

Finally, let us consider the case of $g : \mathbb{R}^d \to [-M, M]^D$. For each $i \in \{1 \cdots D\}$ apply the above arguments twice (once for the positive part of $g_i$ and once for the negative part of $g_i$) together with the triangle inequality to yield a proof for full statement of this theorem. ■

With this in hand, the results we seek are straightforwad.

**Theorem 6** *Fix any positive-definite Lebesgue-integrable kernel $\hat{K} : \mathbb{R}^d \to \mathbb{R}$ with $K(0) = 1$, any $\varepsilon > 0$, and any continuous Lebesgue-integrable covariance kernel of the form $\Sigma : \mathbb{R}^d \to \mathbb{R}^{D\times D}$. One may find a spectral mixture kernel based on $\hat{K}$ which uniformly approximates $\Sigma$ to within $\varepsilon$.*

**Proof** Apply Bochner's theorem to find the bounded density $K$ such that $\hat{K}(\tau) = \int \exp(-i\tau \cdot \omega)K(\tau)d\tau$. Apply Bochner's theorem again to find a bounded density $f$ and bounded Hermitian nonnegative-definite matrix-valued function $M$ such that

$$\Sigma(\tau) = \int \exp(i\tau \cdot \omega)f(\omega)M(\omega)d\omega$$

We refer the reader to Appendix C for a review of these matrix-valued spectra. Apply a unitary eigendecomposition to each $M(\omega)$, obtain $b_i(\omega)$ such that $M(\omega) = \sum_i^D b_i(\omega)b_i^*(\omega)$. Sample $\mu_1, \mu_2, \mu_3, \cdots$ independently and identically from $f$. For any fixed value of $Q$, we can select a value for $\gamma$ and define a mixture of $QD$ components as follows.

$$\hat{M}(\omega; Q) = \frac{1}{Q}\sum_i^D \sum_k^Q b_i(\mu_k)b_i^*(\mu_k)\gamma_Q K(\gamma_Q(\omega - \mu_k))$$

Applying Theorem 5, we may choose $\gamma_Q$ such that $\hat{M}$ almost surely converges in total variation to $fM$ as $Q \to \infty$. Let $C(\tau) = \int \exp(-i\tau\omega)M(\omega)d\omega$. The total variation convergence of $\hat{M}$ yields uniform convergence of $C$ to $\Sigma$. One can thus achieve any degree of accuracy by increasing $Q$ until the uniform error drops below the desired level. Finally, to obtain a (real-valued) spectral mixture kernel which uniformly approximates $\Sigma$, take the real part of $C$. ■

The theorem above covers kernels of the form $\mathbb{R}^d \to \mathbb{R}^{D\times D}$, though our focus here is on kernels of the form $\mathbb{R} \to \mathbb{R}^{D\times D}$. In particular, combining Theorem 6 with Lemma 4 we obtain the main result we sought.

**Theorem 7 (Flexibility of LEG kernels)** *For every $\xi \in \mathbb{N}, \varepsilon > 0$ and every Lebesgue-integrable continuous positive definite stationary kernel $\Sigma : \mathbb{R} \to \mathbb{R}^{n\times n}$ there exists a LEG kernel $C$ which is associated with a $\xi$-times differentiable process and differs from $\Sigma$ by at most $\varepsilon$.*

**Proof** Use Theorem 6 to obtain a spectral mixture kernel, based on the Matern kernel of order $\xi + 1/2$ and achieving the desired accuracy. Recall that processes with such Matern-based kernels are $\xi$-times-differentiable. Finally, use Lemma 4 to obtain a LEG representation of this spectral mixture. ■

## 5. Black-box Gaussian Processes on one dimension

The theoretical results above suggest it may be possible to produce a black-box solution for efficient parameter estimation, interpolation/extrapolation, and smoothing for Gaussian Processes on one dimension. That is, it should be possible to design algorithms which can robustly perform all of these tasks without requiring a user to specify anything more than the rank $Q$ of the LEG kernel to be learned.

To test this hypothesis, we developed a collection of algorithms to enact such a black-box solution. We implemented these algorithms in the pure-python `leggps` package (`https://github.com/jacksonloper/leg-gps`). We sketch the main ideas below.

- Perform parameter estimation by gradient descent on the likelihood. This is straightforward to implement because the space of LEG parameters is unconstrained. Although the likelihood is not convex, we found in our experiments (see below) that random initialization was sufficient to find good parameter estimates.

- Use dynamic programming to enable linear-time computations for likelihood, interpolation/extrapolation, and smoothing. Dynamic programming algorithms are the standard tool for analyzing state-space models (Li, 2009). When properly designed, the total computational cost of such algorithms scales linearly with the number of observations.

- Use the Cyclic Reduction (CR) variable-ordering for this dynamic programming, in order to obtain a parallelizable algorithm. Recall that any dynamic programming algorithm requires the specification of the order in which variables are processed. If one uses the time-points to order the variables (processing them one at a time along the state-space chain) one obtains the popular Kalman Filter algorithm. Unfortunately, this algorithm cannot be easily parallelized to take advantage of modern hardware such as GPUs. The CR ordering, well-known in the matrix algorithms literature (Sweet, 1974), yields parallelizable algorithms with state-space models. The CR ordering proceeds in roughly $\log_2 m$ stages. At each stage all the relevant dynamic programming computations can be performed entirely in parallel. The first stage handles every other variable (e.g. all the variables with even-valued indices). The second stage then applies the same idea recursively, processing every other variable of that which remains. For example, given 14 variables, the CR ordering proceeds in 5 stages, given by

$$\overbrace{2,4,6,8,10,12,14}^{1},\overbrace{3,7,11}^{2},\overbrace{5,13}^{3},\overbrace{9}^{4},\overbrace{1}^{5}.$$

  This ordering yields parallelizable dynamic programming algorithms for all of the matrix operations necessary for computation of likelihoods, interpolation/extrapolation, and smoothing. Details can be found in Appendix C, but here we summarize the key operations. Let $J$ denote a block tridiagonal matrix with $n$ blocks of size $Q$. We need to be able to compute the determinant of $J$, solve linear systems $Jx = y$, compute $y^\top J^{-1} y$ (this can in fact be done in half the time it takes to solve $Jx = y$), and compute the diagonal and off-diagonal blocks of $J^{-1}$. Given $k < n$ processors, each of these algorithms can be completed in $O(Q^3 n/k)$ clock cycles using dynamic programming with a CR ordering.

  CR has similarities to other techniques. It is intuitively similar to multigrid techniques (Terzopoulos, 1986; Hackbusch, 2013) which have been used for Gaussian inference in other contexts (Papandreou and Yuille, 2010; Mukadam et al., 2016; Zanella and Roberts, 2017), but CR is a direct exact method whereas multigrid is iterative. CR is quite similar to the associative-scan method developed by Särkkä and García-Fernández (2019). Like CR, associative scans use parallel hardware to get exact calculations for state-space models. In practice, we used CR because we found it easy to write the code to calculate all the relevant quantities (e.g. posterior variances). Implementing CR in modern software libraries (TensorFlow2 in this case) was straightforward, making it easy to take advantage of modern hardware.

- Compute the gradients of matrix exponentials efficiently using the work of Najfeld and Havel (1995). Computation of $\exp(-G\tau/2)$ for many different values of $\tau$ can be achieved quickly using an eigendecomposition, but the same idea cannot be directly applied to compute the gradient of $\exp(-G\tau/2)$ with respect to $G$. Fortunately, Najfeld and Havel (1995) derive an explicit expression for this gradient in terms of the eigendecomposition of $G$. Details can be found in Appendix A.7.

## 6. Extensions

Before moving on to illustrate these results with experiments on simulated and real data, we pause to note several useful extensions:

*Nonstationary processes.* We have focused on stationary processes here for simplicity. A number of potential extensions to nonstationary processes are possible while retaining linear-time scaling. For example, time-warped versions of a stationary processes could be accurately modeled by a time-warped LEG model, assuming the time-warping is Lipschitz-continuous. Benavoli and Zaffalon (2016) suggests another direction for nonstationarity processes with state-space models, achieved by introducing determinism into the relevant stochastic differential equations. Finally, switching linear dynamical systems and the recurrent versions thereof offer a promising direction for moving beyond the stationary Gaussian model (Linderman et al., 2017); the ideas presented here fit without alteration into those frameworks.

*Non-Gaussian observations.* Many approaches have been developed to adapt GP inference methods to non-Gaussian observations, including Laplace approximations, expectation propagation, variational inference, and a variety of specialized Monte Carlo methods (Hartikainen et al., 2011; Riihimäki et al., 2014; Nguyen and Bonilla, 2014; Nishihara et al., 2014). Many of these can be easily adapted to the LEG model, using the fact that the sum of a block-tridiagonal matrix (from the precision matrix of the LEG prior evaluated at the sampled data points) plus a diagonal matrix (contributed by the likelihood term of each observed data point) is again block-tridiagonal, leading to linear-time updates (Smith and Brown, 2003; Paninski et al., 2010; Fahrmeir and Tutz, 2013; Polson et al., 2013; Khan and Lin, 2017; Nickisch et al., 2018).

*Tree-structured domains.* Just as Gaussian Markov models in discrete time can be easily extended to Gaussian graphical models on tree-structured graphs, it is straightforward to extend the Gaussian Markov PEG and LEG processes to stochastic processes on more general domains with a tree topology, where message-passing can still be applied.

*Multidimensional domains.* We may also be able to adapt state-space models for multi-dimensional Gaussian processes of the form $x : \mathbb{R}^d \to \mathbb{R}^n$, with $d > 1$. Given collections of matrices $N, R, B$, consider the "sum-of-separable terms" kernel $\tau \mapsto \sum_{r=1}^{\zeta} \prod_{k=1}^{d} C_{\mathrm{LEG}}(\tau_k; N_{rk}, R_{rk}; B_{rk}))$ where $\prod$ denotes a Hadamard product. This family of kernels has a number of nice properties. First, the Schur product theorem readily shows that such kernels are nonnegative definite. Second, this family of kernels includes spectral mixtures of the form required by Theorem 6 (so it is dense among stationary continuous Lebesgue-integrable kernels of the form $\Sigma : \mathbb{R}^d \to \mathbb{R}^{n \times n}$). Finally, as shown in Appendix B, efficient computation with such kernels is possible if our observations all lie within a (potentially irregularly-spaced) grid: the cost of matrix-multiplication by the covariance will scale linearly with the number of points in the grid. Preconditioned conjugate gradient methods such as those implemented in GPyTorch (cf.

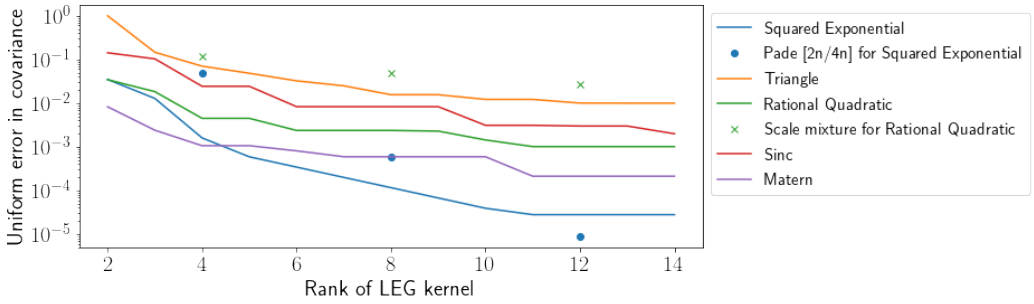


Figure 2: **Popular kernels can be approximated to within 1% of their maximum value using a rank 7 LEG model**. We used gradient descent to find LEG kernels of various ranks to match a few popular one-dimensional kernels. We also examined two proposals from the literature for approximating these kernels with state-space models (the Pade [2n/4n] proposal for squared exponential kernels and the scale mixture proposal for rational quadratic kernels); in most cases we found that gradient descent was able to find a better state-space model.

Gardner et al. (2018)) require nothing more than efficient covariance matrix-mutliplications; this kind of approach could thus yield efficient inference algorithms for such processes.

## 7. Experiments

### 7.1 Model rank versus approximation error

To achieve an accurate approximation to a given target kernel, what rank of LEG model is necessary? Unfortunately, we have been unable to obtain the convergence rates for Theorem 5 that would enable a firm answer to this question; rates for total variation convergence of kernel density estimators can be challenging to obtain without additional assumptions. To get some idea for what convergence rates can be expected, we turn to numerical experiments.

We considered several kernels: a squared exponential kernel, a triangle kernel, a rational quadratic kernel with $\alpha = 2$, a sinc kernel, and a Matern kernel of order 1. Note that Matern kernels of half-integer order can be represented exactly with LEG kernels, but we do not have the same guarantee for integer-order Matern kernels such as the one considered here. Indeed, as far as we are aware, none of these kernels can be represented exactly with any state-space model, but we will see that all of them can be approximated with reasonable accuracy using LEG kernels.

For each target kernel, we wanted to minimize the maximum absolute difference between the target kernel and a LEG kernels of various ranks. However, this objective is difficult to optimize. Fortunately, the total variation ("L1") distance between the two kernels in Fourier space is an upper bound for the uniform error (indeed, if $M, \hat{M}$ are the spectra for $C, \hat{C}$, then observe that $\sup_\tau |C_{ij}(\tau) - \hat{C}_{ij}(\tau)| \leq \sup_\tau \int |\exp(-i\omega\tau)||M_{ij}(\omega) - \hat{M}_{ij}(\omega)|d\omega)$. To obtain an approximating kernel, we thus started from a random initial condition and applied gradient descent on the total variation loss in the Fourier space.

In addition to this gradient-descent approach, we also used two other proposals from the literature. Särkkä and Solin (2019) give a proposal based on Padé approximants to approximate squared exponential kernels using state space models; per Lemma 4, this proposal can be interpreted as a particular choice of LEG kernel designed to approximate the squared exponential kernel. Similarly, Särkkä and Solin (2019) give a proposal for a state-space representation of rational quadratic kernels using a mixture of approximate squared exponential kernels.

The results are summarized in Figure 2. With the exception of the triangle kernel, all kernels could be uniformly matched within .01 by a LEG kernel of rank 7; the triangle kernel could be uniformly matched within .01 by a LEG kernel of rank 13. Given that the ground-truth kernel is rarely known exactly in any case, this suggests a rank-7 LEG kernel may often be sufficient in practice. The triangle kernel was the most difficult to match, perhaps because of its discontinuous derivative. In most cases the proposals from the literature yielded higher error than the LEG kernels found by gradient descent, though for rank-12 kernels gradient descent was unable to find a model with less error than the Padé approximant. A cleverer optimization strategy might yield numerically stable low-rank LEG kernels that are even more accurate than the ones presented here. In future we hope to investigate better strategies for initialization and optimization.

### 7.2 Parameter estimation from simulated data

Given a set of data drawn from a stationary Gaussian Process, Section 5 proposes to estimate the covariance of this process by using gradient descent to seek a maximum-likelihood LEG model for that data. We will call this the "maximum-likelihood-based LEG estimator," although the likelihood is not convex and gradient descent may not find the global maximum-likelihood estimator. Will this estimator be accurate? Theorem 7 suggests it might be, especially given unlimited data and sufficiently high choice of rank. However, we have no theorem on this subject. Any such theorem would require us to specify how the rank $Q$ should grow as the amount of data grows; in future we hope to explore this question more thoroughly. For now, we turn to numerical experiments.

We examine six kernels: the sum of a Radial Basis Function (RBF) and Rational Quadratic (RQ) kernels, an RBF kernel multiplied by cosine, the sum of an RBF kernel with one-fifth of an RQ kernel, the product of a cosine with the previous kernel, a triangle kernel, and the product of an RBF kernel with a high-frequency cosine. In each case we draw eighteen-thousand observations, each taken .1 units apart from the next. We then attempt to estimate the true covariance of the model from these observations.

We compare the maximum-likelihood-based LEG estimator with two other approaches: the Celerité approach and an approach based on spectral mixtures of squared-exponential kernels. We estimated Celerité parameters using the `celerite2` package and estimated the SM parameters using the `gpytorch` package. All three families include a "rank" hyperparameter (i.e. the rank of LEG models, the number of terms in a Celerité kernel, and the number of mixtures in an SM kernel); in order to give all models the best chance, we exhaustively searched over values of this rank hyperparameter and selected the model with the best likelihood on six thousand held-out samples (the best LEG kernels were of rank 7-11, the best Celerité kernels were of rank 6-7, and the best SM kernels were of rank 4-8). The Spectral

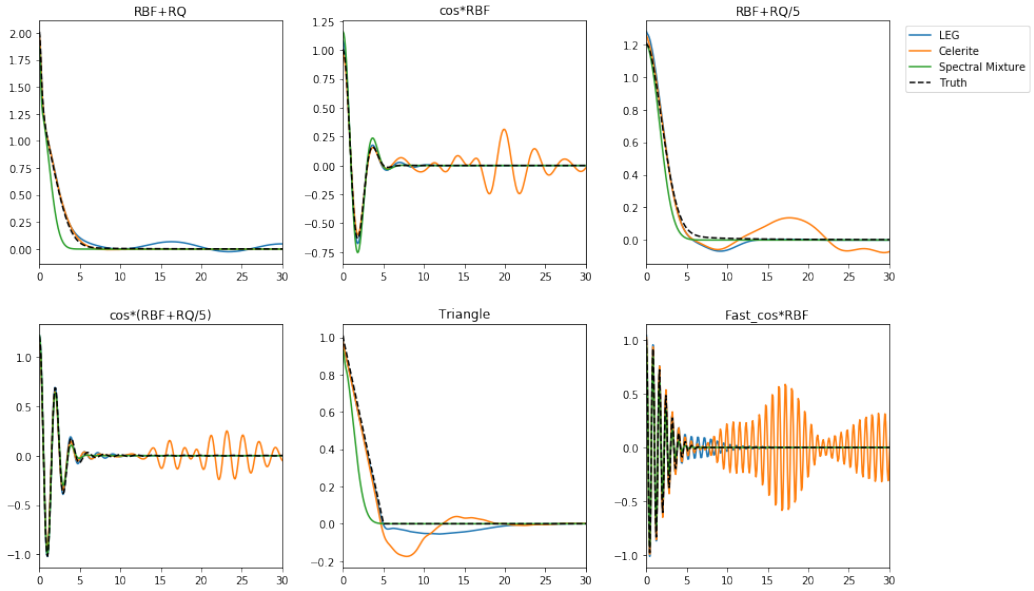


Figure 3: **The maximum-likelihood-based LEG estimator is accurate in simulations.** We tried to learn covariance kernels from data, using maximum likelihood with three different model families: LEG, Celerité, and spectral mixtures of squared exponential kernels. We found the LEG and spectral mixture families yielded comparable accuracy (though the spectral mixture methods are slower as they have no state-space representation); the Celerité estimates performed a bit worse, exhibiting some spurious cyclic activity.

Mixture kernels and the LEG kernels appear to perform comparably; the Celerité family is designed for periodic astronomical phenomena, and has a corresponding inductive bias that leads it to estimate overlarge periodic components. These results are summarized in Figure 3. The LEG and spectral mixture kernels perform comparably (though the spectral mixture kernels do not have a state-space representation and took longer to train), and the Celerité approach (designed for periodic astronomical phenomena) exhibited some spurious cyclic activity. In brief, we see no sign that the LEG family carries a problematic inductive bias for parameter estimation.

### 7.3 Learning, smoothing, and interpolating on real-world data

#### 7.3.1 Interpolation for the Mauna Loa $CO_2$ dataset

To check whether LEG parameterization can capture periodic covariances, we turned to the Mauna Loa $CO_2$ dataset. For the last sixty years, the monthly average atmosphere $CO_2$ concentrations at the the Mauna Loa Observatory in Hawaii have been recorded (Keeling and Whorf, 2005). This dataset is interesting because it features two different kinds of structures: an overall upward trend and a yearly cycle. To test the ability of the LEG model to learn these kinds of structures from data, we trained a rank-5 LEG kernel on all the data

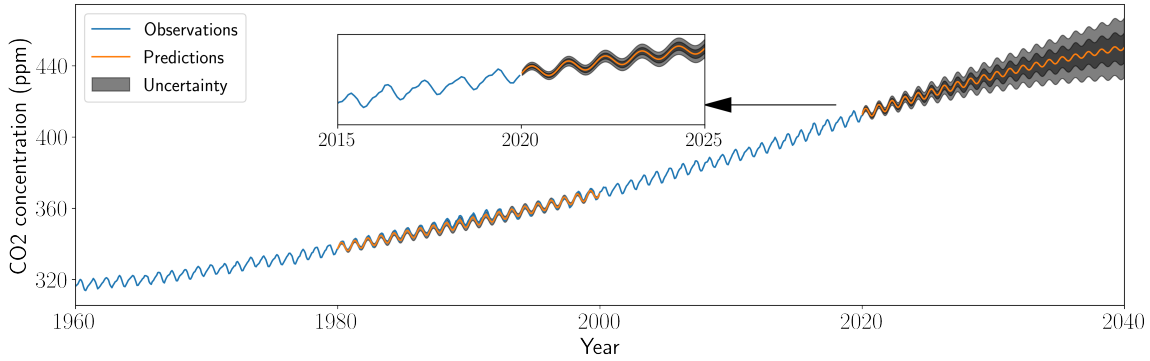


Figure 4: **LEG processes interpolate and extrapolate well across long timescales**. It appears that a rank-5 LEG model is sufficient to capture the linear and periodic trends in the Mauna Loa $CO_2$ dataset. Above we compare the true observations with interpolations made by the LEG model. The gray areas encompass one and two predictive standard deviations, i.e. the LEG model's uncertainty in forecasting and extrapolating what out-of-sample observations would look like.

before 1980 and all the data after 2000. We then asked the LEG model to interpolate what happened in the middle and forecast what the concentration might look like in the next twenty years.

The results are shown in Figure 4. It is encouraging that the LEG predictions interpolate adequately from 1980 to 2000. Even though the LEG process is given no exogenous information about "years" or "seasons," it correctly infers the number of number of bumps between 1980 and 2000. This example shows that the LEG model is sufficiently flexible to learn unanticipated structures in the data.

### 7.3.2 Smoothing for a Hippocampal place-cell dataset

Finally, we wanted to check whether the LEG approach for covariance estimation might introduce any strange artifacts when applied to irregularly-spaced data.

We looked at observations from neural spiking data (Grosmark and Buzsáki, 2016). In this data a rat's position in a one-dimensional maze is reported on a regular schedule, around 40 times per second. At each time-step, each neuron may be silent or may fire ("spike") some number of times. For each neuron, we would like to estimate the "tuning curve" – a function which takes in positions and returns the expected number of spikes as a function of the rat's position. With no smoothness assumptions on this function, the problem is impossible; the rat is never observed at exactly the same place twice. However, it is unclear how much smoothness should be assumed. Gaussian Processes offer a natural way to automatically learn an appropriate level of smoothness from the data itself. Note that the observed positions do not fall into a regularly spaced grid, so classical approaches such as the ARMA model cannot be directly applied.

Here we model this tuning curve using a PEG process, $z \sim \mathrm{PEG}(N, R)$. In this view, each data-point from the experiment constitutes a noisy observation of $\mathbf{z}$. When the rat is at

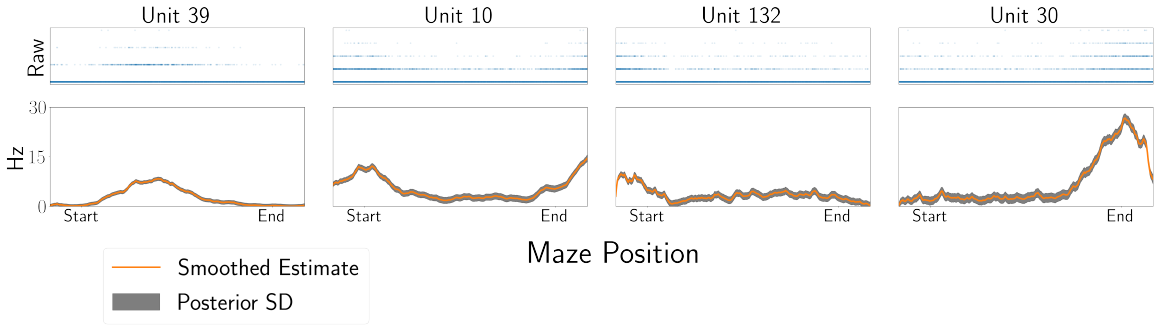


Figure 5: **LEG processes can smooth irregularly spaced neural data**. Ten thousand irregularly spaced observations suggest that the firing rates of Hippocampal neurons are modulated by the rat's position in a maze. However, the modulation strength visible in the raw data is weak. By smoothing this data with a LEG process we can see the trend more clearly. How much should we smooth? By training a rank-5 LEG process we can determine a smoothness level automatically. The gray areas indicate the LEG model's posterior uncertainty about the estimated tuning curve.

position $t$ we model the distribution on the number of spikes observed in a small timebin as a Gaussian, with mean $Bz(t)$ and variance $\Lambda\Lambda^\top$. (It would be interesting to apply a non-Gaussian observation model here, as in, e.g., (Smith and Brown, 2003; Rahnama Rad and Paninski, 2010; Savin and Tkacik, 2016; Gao et al., 2016), and references therein; as noted in section 6, linear-time approximate inference is feasible in this setting and is an important direction for future work.) For each neuron we train the parameters of a separate LEG model. We can then look at the posterior distribution on the underlying tuning curve $\mathbf{z}$. The posterior mean of this process for various neurons is shown in Figure 5. We also represent one standard-deviation of the posterior variance of $\mathbf{z}$ with gray shading. Fitting the LEG model and looking at the posterior under the learned model appears to yield an effective Empirical Bayes approach for this kind of data.

### 7.4 Speed

In theory, the algorithms proposed in Section 5 should scale linearly with the number of samples. However, we wanted to check if there might be constant computational costs that might make this result irrelevant in practice. To do so, we measured how long it took to compute the likelihood for a PEG model. We varied both the number of observations and the rank. In each case we used an m5-24xlarge machine on Amazon Web Services (AWS).

Overall, the empirical scaling appeared consistent with the theoretical predictions of linear scaling. The likelihood of one million observations from a rank-3 model could be computed in 0.25 seconds, and one billion observations could be computed in 195 seconds. We saw similar trends across models of other ranks; the results are summarized in Figure 6. Note that for smaller datasets we actually observed a sublinear scaling (i.e. a slope of less than one on the log-log plot) that turns linear for larger values of $m$.

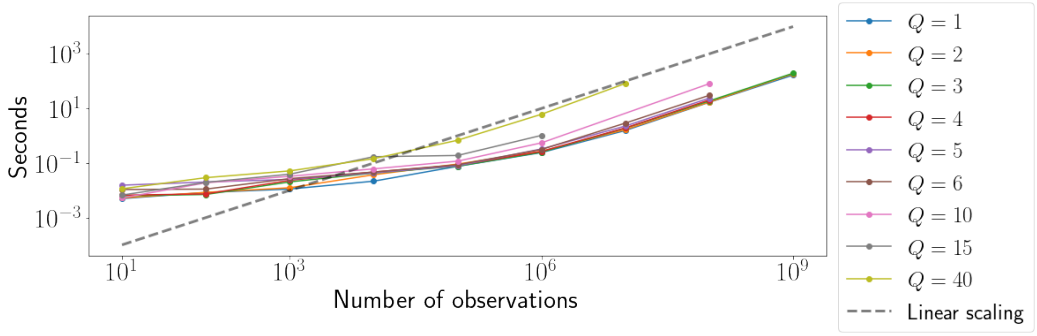


Figure 6: **Walltime for evaluating PEG likelihoods**. How long does it take to compute the likelihood for a PEG model on an m5.24xlarge machine on the Amazon AWS service? We compare times for different ranks ($Q$) and different numbers of blocks (# Observations). For example, the likelihood for one billion observations with $Q = 3$ can be computed in three and a half minutes.

We also wanted to check whether the algorithms proposed in Section 5 would make adequate use of parallel hardware. We compared the `leggps` package with two alternatives: `pylds` (a single-threaded C package for state-space models) and `pyro.distributions.GaussianHMM` (a parallel associative-scan package for state-space models). We also investigated how performance varied between CPU-only hardware and GPU hardware. These experiments yielded three main findings:

- GPU hardware is faster than any number of CPU cores – but with enough samples it becomes impossible to keep everything in GPU memory, so it is not easy to use GPUs to directly compute likelihoods on billions of observations. If the process is expected to mix well within a few million observations, this difficulty can be circumvented by using batches of a few million observations at a time; it may even be possible to learn the parameters using a subset of the data. Once the model is learned on the GPU using this batch-approach, exact smoothing for a much larger dataset can be performed on the CPU. However, if the process has extremely long-range dependencies, new approaches would be necessary. One option would be to break the data into tiles and move the relevant tensors in and out of GPU memory as necessary to compute the relevant quantities. We hope to explore these possibilities in the future.

- CR and associative-scan methods seem to have roughly the same runtimes. Likely as not, any difference in runtime reflects a difference in implementation details (e.g. the `GHMM` package was written in PyTorch, but `leggps` was written in TensorFlow).

- When the number of observations is less than 200, the single-threaded `pylds` package can be slightly faster.

To obtain these results, we used two different measures of speed. To compare CR with associative-scans, we evaluated the time required to compute the gradient of the likelihood of a state-space model with respect to its parameters. To compare with `pylds`, we evaluated

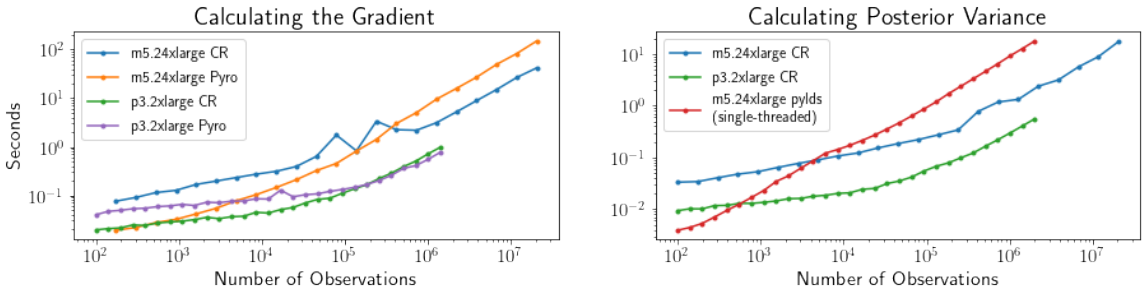


Figure 7: **`leggps` computes with state-of-the-art speed**. Here we evaluate time required to perform two key tasks for inference and learning with state-space models. In each case we assume $Q = 5, D = 4$; we found similar results for other dimensionalities. We look at three different packages: `pyro.distributions.GaussianHMM` (Pyro), `pylds` (pylds), and `leggps` (CR). We also look at two different types of hardware on Amazon ec2: m5.24xlarge machines (96 cores with 384 gigabytes of RAM) and p3.2xlarge (one Nvidia V100 GPU). These plots suggest three overall findings: Pyro and CR perform comparably, a single GPU is much faster than any number of CPUs, and pylds can be faster when the number of observations is small.

the time required to compute posterior variances. We did not use the same benchmark for both packages because the `pylds` package does not compute gradients and the `GaussianHMM` package does not compute posterior variances.

The times are summarized in Figure 7. Several remarks are in order about these plots:

- When the number of observations is large, the GPU architecture is unable to finish the computations due to RAM limitations.
- We did not run pylds on very large datasets due to the prohibitive time costs.
- We used Amazon hardware to perform our tests. We used m5.24xlarge machines to represent CPU computation (96 cores from Intel Xeon Platinum 8000 processors, 384 gigabytes of memory), and p3.2xlarge machines to represent GPU computation (one Nvidia Tesla V100 processor with 16 gigabytes of memory). Note that the `pylds` package is single-threaded in nature, and unable to take advantage of the 96 cores offered by the m5.24xlarge; this package achieves essentially the same times even with much more modest machines such as the m5.large machine.

## 8. Conclusions

We here provide several advances for Gaussian Processes on one dimension. First, we show that the LEG model (a particularly tractable continuous-time Gaussian hidden Markov process) can be used to approximate any GP with a stationary integrable continuous kernel, enabling linear runtime scaling in the number of observations. Second, we develop a new unconstrained parameterization of continuous-time state-space models, making it easy to

use simple gradient-descent methods. Finally, we develop a simple package for learning, interpolation, and smoothing with such models; this package uses Cyclic Reduction to achieve rapid runtimes on modern hardware. We believe these advances will open up a wide variety of new applications for GP modeling in highly data-intensive areas involving data sampled at high rates and/or over long intervals, including geophysics, astronomy, high-frequency trading, molecular biology, neuroscience, and more.

## Acknowledgements

Thanks to Jake Soloff for resolving a thorny point about matrix-valued measures.

We gratefully acknowledge support from NIH U19NS107613 (JL+LP), IARPA D16PC00003 (JL+LP), NSF NeuroNex (JL+JC+LP), the Simons Foundation (JC+LP) and the Gatsby Charitable Foundation (JL+JC+LP). David Blei was supported by ONR N00014-17-1-2131, ONR N00014-15-1-2209, NSF CCF-1740833, DARPA SD2 FA8750-18-C-0130, 2Sigma, Amazon, and NVIDIA. John P. Cunningham was additionally supported by the McKnight Foundation.

## References

Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W Hogg, and Michael O'Neil. Fast direct methods for Gaussian Processes. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):252–265, 2015.

S.S. Artemiev and T.A. Averina. *Numerical Analysis of Systems of Ordinary and Stochastic Differential Equations*. VSP, 1997. ISBN 9789067642507.

Alessio Benavoli and Marco Zaffalon. State space representation of non-stationary Gaussian Processes. *arXiv preprint arXiv:1601.01544*, 2016.

P.J. Brockwell and R.A. Davis. *Time Series: Theory and Methods*. Springer Series in Statistics. Springer New York, 2013. ISBN 9781489900043.

Bing Leung Patrick Cheung, Brady Alexander Riedner, Giulio Tononi, and Barry D Van Veen. Estimation of cortical connectivity from EEG using state-space models. *IEEE Transactions on Biomedical engineering*, 57(9):2122–2134, 2010.

Harald Cramér. On the theory of stationary random processes. *Annals of Mathematics*, pages 215–230, 1940.

John Cunningham, K Shenoy, and Maneesh Sahani. Fast gaussian process methods for point process estimation. In *Proceedings of the International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2008.

Kurt Cutajar, Michael Osborne, Johnn Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. In *Proceedings of the International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2016.

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. *The Journal of Machine Learning Research*, 18(1):1299–1304, 2017.

LP Devroye and TJ Wagner. The L1 convergence of kernel density estimates. *The Annals of Statistics*, pages 1136–1139, 1979.

Ludwig Fahrmeir and Gerhard Tutz. *Multivariate statistical modelling based on generalized linear models*. Springer Science & Business Media, 2013.

Daniel Foreman-Mackey, Eric Agol, Sivaram Ambikasaran, and Ruth Angus. Fast and scalable Gaussian Process modeling with applications to astronomical time series. *The Astronomical Journal*, 154(6):220, 2017.

Yuanjun Gao, Evan W Archer, Liam Paninski, and John P Cunningham. Linear dynamical neural population models through nonlinear embeddings. In *Advances in neural information processing systems*, pages 163–171, 2016.

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix Gaussian Process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586, 2018.

Elad Gilboa, Yunus Saatçi, and John P Cunningham. Scaling multidimensional inference for structured Gaussian Processes. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):424–436, 2013.

N Glick. Consistency conditions for probability estimators and integrals of density estimators. *Utilitas Mathematica*, 6:61–74, 1974.

Joseph Gonzalez, Yucheng Low, and Carlos Guestrin. Residual splash for optimally parallelizing belief propagation. In *Proceedings of Artificial Intelligence and Statistics*, 2009.

Andres D Grosmark and György Buzsáki. Diversity in neural firing dynamics supports both rigid and learned hippocampal sequences. *Science*, 351(6280):1440–1443, 2016.

W. Hackbusch. *Multi-Grid Methods and Applications*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2013. ISBN 9783662024270.

Jouni Hartikainen and Simo Särkkä. Kalman filtering and smoothing solutions to temporal Gaussian Process regression models. In *2010 IEEE international workshop on machine learning for signal processing*, pages 379–384. IEEE, 2010.

Jouni Hartikainen, Jaakko Riihimäki, and Simo Särkkä. Sparse spatio-temporal Gaussian Processes with general likelihoods. In *International Conference on Artificial Neural Networks*, pages 193–200. Springer, 2011.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian Processes for big data. In *Uncertainty in Artificial Intelligence*, 2013.

Toni Karvonen and Simo Sarkkä. Approximate state-space Gaussian Processes via spectral transformation. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing*, pages 1–6. IEEE, 2016.

Charles D Keeling and TP Whorf. Atmospheric carbon dioxide record from Mauna Loa. *Carbon Dioxide Research Group, Scripps Institution of Oceanography, University of California La Jolla, California*, pages 92093–0444, 2005.

Mohammad Emtiyaz Khan and Wu Lin. Conjugate-computation variational inference. In *Proceedings of Artificial Intelligence and Statistics*, 2017.

Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

Scott Linderman, Matthew Johnson, Andrew Miller, Ryan Adams, David Blei, and Liam Paninski. Bayesian learning and inference in recurrent switching linear dynamical systems. In *Artificial Intelligence and Statistics*, pages 914–922. PMLR, 2017.

Finn Lindgren, Håvard Rue, and Johan Lindström. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(4):423–498, 2011.

Kian Hsiang Low, Jiangbo Yu, Jie Chen, and Patrick Jaillet. Parallel Gaussian Process regression for big data: Low-rank representation meets Markov approximation. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

S. Mergner. *Applications of State Space Models in Finance: An Empirical Analysis of the Time-varying Relationship Between Macroeconomics, Fundamentals and Pan-European Industry Portfolios*. Univ.-Verlag Göttingen, 2009. ISBN 9783941875227.

Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian Process motion planning. In *2016 IEEE international conference on robotics and automation*, pages 9–15. IEEE, 2016.

Igor Najfeld and Timothy F Havel. Derivatives of the matrix exponential and their computation. *Advances in applied mathematics*, 16(3):321–375, 1995.

H Neudecker. A note on Kronecker matrix products and matrix equation systems. *SIAM Journal on Applied Mathematics*, 17(3):603–606, 1969.

Trung V Nguyen and Edwin V Bonilla. Automated variational inference for Gaussian Process models. In *Advances in Neural Information Processing Systems*, pages 1404–1412, 2014.

Hannes Nickisch, Arno Solin, and Alexander Grigorevskiy. State space Gaussian Processes with non-Gaussian likelihood. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3789–3798, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Robert Nishihara, Iain Murray, and Ryan P Adams. Parallel MCMC with generalized elliptical slice sampling. *The Journal of Machine Learning Research*, 15(1):2087–2112, 2014.

Liam Paninski, Yashar Ahmadian, Daniel Gil Ferreira, Shinsuke Koyama, Kamiar Rahnama Rad, Michael Vidne, Joshua Vogelstein, and Wei Wu. A new look at state-space models for neural data. *Journal of computational neuroscience*, 29(1-2):107–126, 2010.

George Papandreou and Alan L Yuille. Gaussian sampling by local perturbations. In *Advances in Neural Information Processing Systems*, pages 1858–1866, 2010.

Nicholas G Polson, James G Scott, and Jesse Windle. Bayesian inference for logistic models using Pólya–Gamma latent variables. *Journal of the American statistical Association*, 108 (504):1339–1349, 2013.

Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec): 1939–1959, 2005.

Kamiar Rahnama Rad and Liam Paninski. Efficient, adaptive estimation of two-dimensional firing rate surfaces via Gaussian Process methods. *Network: Computation in Neural Systems*, 21(3-4):142–168, 2010.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

G.C. Reinsel. *Elements of Multivariate Time Series Analysis*. Springer Series in Statistics. Springer New York, 2003. ISBN 9780387406190.

Jaakko Riihimäki, Aki Vehtari, et al. Laplace approximation for logistic Gaussian Process density estimation and regression. *Bayesian analysis*, 9(2):425–448, 2014.

S. Särkkä and A. Solin. *Applied Stochastic Differential Equations*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2019. ISBN 9781316510087. URL `https://books.google.com/books?id=g5SODwAAQBAJ`.

Simo Särkkä and Ángel F García-Fernández. Temporal parallelization of bayesian filters and smoothers. *arXiv preprint arXiv:1905.13002*, 2019.

Simo Särkkä and Robert Piché. On convergence and accuracy of state-space approximations of squared exponential covariance functions. In *2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2014.

Cristina Savin and Gasper Tkacik. Estimating nonlinear neural response functions using GP priors and Kronecker methods. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3603–3611. 2016.

Anne C Smith and Emery N Brown. Estimating a state-space model from point process observations. *Neural computation*, 15(5):965–991, 2003.

Edward Snelson and Zoubin Ghahramani. Local and global sparse gaussian process approximations. In *Proceedings of Artificial Intelligence and Statistics*, 2007.

Roland A Sweet. A generalized cyclic reduction algorithm. *SIAM Journal on Numerical Analysis*, 11(3):506–520, 1974.

Demetri Terzopoulos. Image analysis using multigrid relaxation methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2):129–139, 1986.

P Vatiwutipong and N Phewchean. Alternative way to derive the distribution of the multivariate Ornstein–Uhlenbeck process. *Advances in Difference Equations*, 2019(1):1–7, 2019.

Ke Wang, Geoff Pleiss, Jacob Gardner, Stephen Tyree, Kilian Q Weinberger, and Andrew Gordon Wilson. Exact Gaussian Processes on a million data points. In *Advances in Neural Information Processing Systems*, pages 14622–14632, 2019.

Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of the International Conference on Machine Learning*, pages 1067–1075, 2013.

Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian Processes (kiss-gp). In *Proceedings of the International Conference on Machine Learning*, pages 1775–1784, 2015.

Giacomo Zanella and Gareth Roberts. Analysis of the Gibbs sampler for Gaussian hierarchical models via multigrid decomposition. *arXiv preprint arXiv:1703.06098*, 2017.

Yunong Zhang, William E Leithead, and Douglas J Leith. Time-series Gaussian Process regression based on Toeplitz computation of $O(n^2)$ operations and $O(n)$-level storage. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 3711–3716. IEEE, 2005.

## Appendix A. Computations with block-tridiagonal matrices using Cyclic Reduction (CR)

Let $J$ be the symmetric positive-definite block-tridiagonal matrix, defined blockwise by

$$J = \begin{pmatrix} R_0 & O_0^T & 0 & \cdots \\ O_0 & R_1 & O_1^T & \\ 0 & O_1 & R_2 & \\ \vdots & & & \ddots \end{pmatrix}$$

We would like to be able to compute efficiently with $J$. To do so, we can decompose $J$ using what is called a "Cyclic Reduction" ordering. This ordering is well-known among those well-acquainted with parallel matrix algorithms Sweet (1974). However, as far as we are aware the literature currently lacks a precise specification of how this ordering can be used for the particular algorithms needed in our application. For the benefit of the reader, we present a self-contained summary here.

**Definition 8 (Cyclic Reduction)** *For each $m$, let $P_m$*

$$P_m = \begin{pmatrix} I & 0 & 0 & 0 & 0 & \\ 0 & 0 & I & 0 & 0 & \\ 0 & 0 & 0 & 0 & I & \\ & & & & & \ddots \end{pmatrix}$$

*denote the permutation matrix which selects every other block of a matrix with $m$ blocks. Let*

$$Q_m = \begin{pmatrix} 0 & I & 0 & 0 & 0 & \\ 0 & 0 & 0 & I & 0 & \\ 0 & 0 & 0 & 0 & 0 & \\ & & & & & \ddots \end{pmatrix}$$

*denote the complementary matrix, which takes the other half of the blocks.*

*The **Cyclic Reduction** of a block-tridiagonal matrix $J$ with $m$ blocks is defined recursively by*

$$\begin{aligned} L = \text{CyclicReduction}\,(J, m) = \begin{pmatrix} P_m^T & Q_m^T \end{pmatrix} \begin{pmatrix} D & 0 \\ U & \tilde{L} \end{pmatrix} \\ D = \text{Cholesky}\left(P_m J P_m^\top\right) \\ U = Q_m J P_m^T D^{-\top} \\ \tilde{J} = Q_m J Q_m^T - U^\top U \\ \tilde{L} = \text{CyclicReduction}\left(\tilde{J}, \lceil m/2 \rceil\right) \end{aligned}$$

*Note that the $P, Q$ matrices select subsets of blocks of the matrices involved. For example, it is straightforward to show that $P_m J P_m^\top$ is block-diagonal, with blocks given by $R_0, R_2, R_4, \cdots$. The matrix $Q_m J Q_m^\top$ is also block-tridiagonal, with blocks given by $R_1, R_3, R_5, \cdots$. The matrix $Q_m J P_m^\top$ is upper block-didiagonal (i.e. it has diagonal blocks and one set of upper off-diagonal blocks); the diagonal blocks are given by $O_0, O_2, O_4, \cdots$ and the upper off-diagonal blocks are given by $O_1, O_3, O_5, \cdots$.*

*For this recursive algorithm to make sense, we need that $\tilde{J}$ is also block-tridiagonal – but this is always true if $J$ is block-tridiagonal. The recursion terminates when $J$ has exactly one block. For this we define the base-case*

$$\text{CyclicReduction}\,(J, 1) = \text{Cholesky}(J)$$

**Proposition 1** *Let $L = \text{CyclicReduction}\,(J, n)$. Then $LL^\top = J$.*

**Proof** By induction. For the case $n = 1$ the algorithm works because the Cholesky decomposition works.

Now let us assume the algorithm works for all $\tilde{n} < n$. We will show it works for $m$. Let

$$
\begin{aligned}
L &= \begin{pmatrix} P_m^T & Q_m^T \end{pmatrix} \begin{pmatrix} D & 0 \\ U & \tilde{L} \end{pmatrix} \\
D &= \text{Cholesky}\left(P_m J P_m^\top\right) \\
U &= Q_m J P_m^T D^{-\top} \\
\tilde{J} &= Q_m J Q_m^T - U^\top U \\
\tilde{L} &= \text{CyclicReduction}\left(\tilde{J}, \lceil m/2 \rceil\right)
\end{aligned}
$$

By induction $\tilde{L}\tilde{L}^\top = \tilde{J}$. Thus

$$
\begin{aligned}
LL^T &= \begin{pmatrix} P_m^T & Q_m^T \end{pmatrix} \begin{pmatrix} D & 0 \\ U & \tilde{L} \end{pmatrix} \begin{pmatrix} D^T & U^T \\ 0 & \tilde{L}^T \end{pmatrix} \begin{pmatrix} P_m \\ Q_m \end{pmatrix} \\
&= \begin{pmatrix} P_m^T & Q_m^T \end{pmatrix} \begin{pmatrix} DD^T & DU^T \\ UD^T & UU^T + \tilde{L}\tilde{L}^T \end{pmatrix} \begin{pmatrix} P_m \\ Q_m \end{pmatrix} \\
&= \begin{pmatrix} P_m^T & Q_m^T \end{pmatrix} \begin{pmatrix} P_m J P_m^T & \cancel{DD^{-1}} P_m J Q_m^T \\ Q_m J P_m^T \cancel{D^{-T} D^T} & \cancel{UU^T} + Q_m J Q_m^T - \cancel{UU^T} \end{pmatrix} \begin{pmatrix} P_m \\ Q_m \end{pmatrix} \\
&= J
\end{aligned}
$$

■

This decomposition enables efficient computations with $J$. Below we describe all of the relevant algorithms (including the CR decomposition algorithm itself) from an algorithms point of view, giving runtimes for each. We will see that all operation counts scale linearly in the number of blocks. We will also discuss parallelization; as we shall see, almost all of the work of a Cyclic Reduction iteration can be done in parallel across the $m$ blocks of $J$.

### A.1 Cyclic Reduction

**Algorithm 1:** decompose

**input :** rblocks,oblocks,$m$ – the diagonal and lower off-diagonal blocks of a block-tridiagonal matrix $J$ which has $m$ blocks

**output :** dlist,flist,glist– a representation of the CR decomposition of $J$

**if** $m = 1$ **then**

  **return** [Cholesky($R_0$)], [], []

**else**

  Adopt the notation $R_i$ =rblocks[i] and $O_i$ =oblocks[i];

  Let

$$D \triangleq \begin{pmatrix} D_0 & 0 & 0 \\ 0 & D_1 & \\ & & \ddots \end{pmatrix} \triangleq \begin{pmatrix} \text{Cholesky}\,(R_0) & 0 & 0 \\ 0 & \text{Cholesky}\,(R_2) & \\ & & \ddots \end{pmatrix}$$

  and store the diagonal blocks of $D$ in dblocks;

  Let

$$U \leftarrow \begin{pmatrix} O_0 D_0^{-T} & O_1 D_1^{-T} & 0 & \cdots & 0 \\ 0 & O_2 D_1^{-T} & O_3 D_2^{-T} & & \\ 0 & 0 & O_4 D_2^{-T} & \ddots & \\ \vdots & & & \ddots & \end{pmatrix}$$

  and store diagonal and upper-off-diagonal blocks of $U$ in (fblocks,gblocks);

  Let

$$\tilde{J} = \begin{pmatrix} R_1 & 0 & 0 \\ 0 & R_3 & \\ & & \ddots \end{pmatrix} - UU^\top$$

  and store the diagonal and lower-offdiagonal blocks of $\tilde{J}$ in newrblocks,newoblocks;

  newdlist,newflist,newglist ← decompose(newrblocks,newoblocks,len(newrblocks));

  **return** concat([dblocks],newdlist),concat([fblocks],newflist),concat([gblocks],newglist);

**end**

Observe that the dlist,flist,glist returned by this algorithm stores everything we would need to reconstruct the CyclicReduction($J$).

How long does this algorithm take?

- Step 5 requires we compute $m$ Cholesky decompositions
- Step 6 requires $m - 1$ triangular solves
- Step 7 has two components. First we must compute the diagonal and lower-off-diagonal blocks of $UU^\top$ (which requires about $m$ matrix-multiplies and $m$ matrix additions). Second we must compute $\lfloor m/2 \rfloor$ matrix subtractions.

- Step 8 requires we run the CR algorithm on a problem with $\lfloor m/2 \rfloor$ blocks.

Let $C(m)$ denote overall number of operations for a Cyclic Reduction on an $m$-block matrix. Since steps 7, 8, and 9 require $O(m)$ operations, we have that there exists some $c$ such that

$$C(m) \leq cm + C(\lfloor n/2 \rfloor) \qquad C(1) \leq c$$

from which we see that $C(m) < 2cm$,[1] i.e. the computation scales linearly in $m$.

What about parallelization? To compute steps 5-7 we need to compute many small Cholesky decompositions, compute many small triangular solves, compute many small matrix multiplies. These are all common problems, and fast algorithms exist for achieving these goals on multiple CPU cores or using GPUs.

### A.2 Solving $Lx = b$

This algorithm uses the tuple (dlist,flist,glist) representing a Cyclic Reduction $L$ on a matrix with $m$ blocks to compute $L^{-1}b$.

**Algorithm 2:** halfsolve

**input :** dlist,flist,glist,$b$,$m$
**output :** $x = L^{-1}b$

1 Adopt the notation $D$ is the block-diagonal matrix whose diagonal blocks are given by dlist[0] ;
2 Adopt the notation that $U$ is the upper didiagonal matrix whose diagonal blocks are given by flist[0] and whose upper off-diagonal blocks are given by glist[0];
3 **if** $m = 1$ **then**
4 | **return** $x = D^{-1}b$
5 **else**
6 | $x_1 \leftarrow D^{-1}P_m b$;
7 | $x_2 \leftarrow$ halfsolve(dlist[1:],flist[1:],glist[1:],$Q_m b - Ux_1$,$\lfloor m/2 \rfloor$);
8 | **return**
$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$
9 **end**

Note that step 6 requires $O(m)$ operations, the base case requires $O(1)$ operations, and step 7 is a recursion on a problem of half-size. The overall computation thus scales linearly in $m$. Moreover, step 6 can be understood as $m$ independent triangular solves, all of which can be solved completely independently (this algorithm is thus easy to parallelize across many cores).

1. One way to see this is by induction. For the base case, we have $C(1) \leq c$. Then, under the inductive hypothesis, we have that $C(m) < c(m + 2m/2) = 2cm$. In general for all the recursive algorithms that follow, to prove linear-time it will suffice to show that the non-recursive steps require $O(m)$ time.

### A.3 Solving $L^\top x = b$

This algorithm uses the tuple (dlist,flist,glist) representing a Cyclic Reduction $L$ on a matrix with $m$ blocks to compute $L^{-\top}b$.

**Algorithm 3:** backhalfsolve

**input** : dlist,flist,glist,$b$,$m$
**output** : $x = L^{-\top}b$

Adopt the notation $D$ is the block-diagonal matrix whose diagonal blocks are given by dlist[-1] (i.e. the last entry in dlist);
Adopt the notation that $U$ is the upper didiagonal matrix whose diagonal blocks are given by flist[-1] and whose upper off-diagonal blocks are given by glist[-1];
**if** $m = 1$ **then**
  **return** $x = D^{-T}b$
**else**
  $\tilde{x}_2 \leftarrow$ backhalfsolve(dlist[:-1],flist[:-1],glist[:-1],$b$,$\lfloor m/2 \rfloor$);
  $\tilde{x}_1 \leftarrow D^{-\top}(P_n b - U^\top \tilde{x}_2)$;
  **return**

$$x = \begin{pmatrix} P_n^\top & Q_n^\top \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix}$$

**end**

Just like the halfsolve algorithm, the cost of backhalfsolve scales linearly in $m$ and is easy to parallelize across the $m$ blocks.

### A.4 Solving $Jx = b$

1. First solve $Ly = b$ using halfsolve.
2. Then solve $L^T x = y$ using backhalfsolve.

Then

$$Jx = LL^T x = Ly = b$$

as desired.

### A.5 Computing determinants

The determinant of a block-Cholesky decomposition is just the square of the product of the determinants of the diagonal blocks. Thus if (dlist,flist,glist) represents the CR decomposition of $J$ we have that the determinant of $J$ is given by the square of the products of the determinants of all the matrices in dlist. This can be done in parallel across all of the $m$ blocks, requiring $O(m)$ operations in total.

### A.6 Computing the diagonal and off-diagonal blocks of the inverse

**Algorithm 4:** invblocks

**input :** dlist,flist,glist

**output :** diags,offdiags – the diagonal and off-diagonal blocks of $J$

1 Adopt the notation $D$ is the block-diagonal matrix whose diagonal blocks are given by dlist[-1] (i.e. the last entry in dlist);

2 Adopt the notation that $U$ is the upper didiagonal matrix whose diagonal blocks are given by flist[-1] and whose upper off-diagonal blocks are given by glist[-1];

3 **if** $m = 1$ **then**

4 | **return** $[D^{-T}D^{-1}]$,[]

5 **else**

6 | subd,suboff ← invblocks(dlist[:-1],flist[:-1],glist[:-1]);

7 | Adopt the notation that $\tilde{\Sigma}$ is a matrix whose diagonal blocks are given by subd and whose lower off-diagonal blocks are given by suboff;

8 | Let SUDid store the diagonal blocks of $\tilde{\Sigma}UD^{-1}$;

9 | Let DitUtSo store the upper-off-diagonal blocks of $D^{-\top}U^{\top}\tilde{\Sigma}$;

10 | Let DitUtSUDid store the diagonal blocks of $D^{-\top}U^{\top}\tilde{\Sigma}UD^{-1}$;

11 | Let

$$\text{diags} \leftarrow \begin{pmatrix} P_n^{\top} & Q_n^{\top} \end{pmatrix} \begin{pmatrix} \text{DitUtSUDid} \\ \text{subd} \end{pmatrix}$$

| where DitUtSUid and subd are understood as tall columns of matrices. For example, if each block is $\ell \times \ell$, we understand DitUtSUDid as a $\lceil m/2 \rceil \times \ell$ matrix;

12 | Let

$$\text{offdiags} \leftarrow \begin{pmatrix} P_n^{\top} & Q_n^{\top} \end{pmatrix} \begin{pmatrix} \text{SUDid} \\ \text{DitUtSo} \end{pmatrix}$$

| where SUDid and DitUtSo are understood as tall columns of matrices;

13 | **return** diags,offdiags;

14 **end**

The cost of this algorithm scales linearly in $m$ because steps 7-11 require $O(m)$ operations. To see how this can be so, recall that $U$ is block didiagonal and $D$ is block diagonal. Thus, for example, step 8 involves $\Sigma UD^{-1}$. Computing this entire matrix would be quite expensive. However, $U$ is block didiagonal and we only need the diagonal blocks of the result, so we can get what we need in linear time and we only need to know the diagonal and off-diagonal blocks of $\Sigma$. As in the other algorithms, note that all of the steps can be done in parallel across the $m$ blocks.

Why does this algorithm work? As usual, let

$$
\begin{aligned}
L = \text{CyclicReduction}\,(J, m) = \begin{pmatrix} P_m^T & Q_m^T \end{pmatrix} \begin{pmatrix} D & 0 \\ U & \tilde{L} \end{pmatrix} \\
D = \text{Cholesky}\left(P_m J P_m^\top\right) \\
U = Q_m J P_m^\top D^{-\top} \\
\tilde{J} = Q_m J Q_m^\top - U^\top U \\
\tilde{L} = \text{CyclicReduction}\left(\tilde{J}, \lceil m/2 \rceil\right)
\end{aligned}
$$

Now let $\tilde{\Sigma} = \tilde{J}^{-1}$. It follows that

$$
J^{-1} = \begin{pmatrix} P_n \\ Q_n \end{pmatrix} \begin{pmatrix} D^{-T}D^{-1} + D^{-\top}U^\top \tilde{\Sigma} U D^{-1} & -D^{-\top}U^\top \tilde{\Sigma} \\ -\tilde{\Sigma} U D^{-1} & \tilde{\Sigma} \end{pmatrix} \begin{pmatrix} P_n^T & Q_n^\top \end{pmatrix}
$$

So to compute the diagonal and off-diagonal blocks of $J^{-1}$ we just need to collect all the relevant blocks from the the inner matrix on the RHS of the equation above. However, all of the relevant blocks can be calculated using only the diagonal and off-diagonal blocks of $\tilde{\Sigma}$. This is what the invblocks algorithm does.

### A.7 Gradients of matrix exponentials

In order to learn LEG models via gradient descent we will need to be able to compute gradients with respect to the parameters. Most of these gradients can be computed automatically through a package such as TensorFlow. There is one gradient which standard packages will compute inefficiently. For many different values of $t$, we need to compute the value of $\exp(-tG)$. By default, most packages will simply compute all the matrix exponentials for each $t$ separately, which is quite expensive. However, all these matrix exponentials can be computed much more efficiently if $G$ is first diagonalized. On the other hand, differentiating through diagonalization isn't very stable. To get the gradients we need in a fast and stable

manner we need to special-case the gradient-computation algorithm using Theorem 4.8 of Najfeld and Havel (1995). We outline the relevant algorithm, below.

**Algorithm 5:** expm

**input** : $G$ (a square matrix), tlist (a list of scalars)
**output**: rez (the value of $\exp(tG)$ for each $t$ in tlist), grad (a corresponding gradient-evaluating function)

$\Lambda, U \leftarrow$ eigendecomposition$(G)$;
$\mathrm{Ui} \leftarrow U^{-1}$;
$\mathrm{rez}_{\mathrm{mjk}} \leftarrow \sum_r U_{jr}\mathrm{Ui}_r k \exp(\Lambda_r \mathrm{tlist}_m)$;
grad $\leftarrow (M \mapsto \mathrm{expmgrad}(\Lambda, U, \mathrm{Ui}, \mathrm{rez}, M))$

**Algorithm 6:** expmgrad

**input** : $G, \Lambda, U$,Ui,rez,$M$,tlist
**output**: gradG $(mjk \mapsto \sum_{jk} M_{kr}(\partial \exp(\mathrm{tlist_m} G)_{kr}/\partial G_{ij})$
and gradt $(m \mapsto \sum_{jk} M_{kr}(\partial \exp(tG)_{kr}/\partial t|_{t=\mathrm{tlist_m}})$
$\Phi_{mij} \leftarrow \lim_{\lambda\to\Lambda_i}(\exp(\lambda \mathrm{tlist_m}) - \exp(\Lambda_j \mathrm{tlist_m})/(\lambda - \Lambda_j)$;
$H_{ij} \leftarrow \sum_m \Phi_{mij}(U^\top M_m \mathrm{Ui}^\top)_{ij}$;
sand $\leftarrow \mathrm{Ui}^\top H U^\top$;
**return** $\Re$(sand);

## Appendix B. Matrix-multiplication by Kronecker-Hadamard product of PEG kernels on a grid

The ideas introduced in this paper may also facilitate inference for GPs of the form $\mathbb{R}^d \to \mathbb{R}^D$. For example, efficient computation is possible for kernels of the separable form $\tau \mapsto \prod_{k=1}^d C_{\mathrm{LEG}}(\tau_k; N_k, R_k, B_k)$; efficient matrix multiplication by the covariance matrices induced by sums of such kernels can be achieved by taking advantage of the special structure of the constituent LEG kernels, as shown in the following theorem.

**Theorem 9** *Let $\Omega = \prod_k^d \{\tau_{k1}, \tau_{k2} \cdots \tau_{km}\} \subset \mathbb{R}^d$ denote a grid and $\{N_1 \cdots N_d\}, \{R_1 \cdots R_d\}, \{B_1 \cdots B_d\}$ denote collections of $Q \times Q$ matrices. For each $k$ let $K_k$ denote the $m \times Q \times m \times Q$ tensor such that $K_k(u, i, v, j)$ is equal to the value at row $i$ and column $j$ of $C_{\mathrm{LEG}}(\tau_{ku} - \tau_{kv}; N_k, R_k, B_k)$. Let $\Sigma$ denote the $m \times m \cdots \times Q \times m \times m \cdots \times Q$ tensor such that*

$$\Sigma(u_1, u_2, \cdots u_d, i, v_1, v_2 \cdots v_d, j) = \prod_{k=1}^{d} K_k(u_k, i, v_k, j).$$

*Let $x$ denote any $m \times m \cdots \times Q$ tensor. Let $y$ denote the $m \times m \cdots \times Q$ tensor defined by*

$$y(u, i) = \sum_{v,j} \Sigma(u, i, v, j) x(v, j).$$

*For any fixed $N, R, B$, the cost of evaluating $y$ is $O(m^d)$ as a function of the width $m$ of the grid $\Omega$.*

**Proof** We compute $y$ by applying the operators $K_1, \cdots, K_d$ one at a time.

$$\begin{aligned} y(u,i) &= \sum_j^Q \sum_{v_1}^m \cdots \sum_{v_d}^m \left( \prod_k K_k(u_k, i, v_k, j) \right) x(v,j) \\ &= \sum_j^Q \sum_{v_1}^m \cdots \sum_{v_{d-1}}^m \left( \prod_k^{d-1} K_d(u_k, i, v_k, j) \right) \underbrace{\sum_{v_d}^m K_k(u_d, i, v_d, j) x(v,j)}_{\triangleq \xi_d(v_1 \cdots v_{d-1}, u_d, i, j)} \end{aligned}$$

If $K_d$ had no special structure, computing the $m \times m \times \cdots Q \times Q$ tensor $\xi$ would require $m^{d-1}$ matrix-vector multiplications, each costing $O(m^2)$. However, as shown above, multiplications with with LEG kernels can be achieved with cost $O(m)$. It follows that $\xi$ can in fact be computed with $O(m^d)$ cost. Once this is computed, we proceed as follows.

$$y(u,i) = \sum_j^Q \sum_{v_1}^m \cdots \sum_{v_{d-2}}^m \left( \prod_k^{d-2} K_k(u_k, i, v_k, j) \right) \underbrace{\sum_{v_{d-1}}^m K_{d-1}(u_{d-1}, i, v_{d-1}, j) \xi_d(v_1 \cdots v_{d-1}, u_d, i, j))}_{\triangleq \xi_{d-1}(v_1, v_2 \cdots u_{d-1}, u_d, i, j}$$

As before, computing the tensor $\xi_{d-1}$ requires only a $O(m^d)$ cost.

We can continue in this way for $d$ steps, finally arriving at $y$ with $O(m^d)$ cost, as desired. ■

Therefore, matrix-vector multiplication with these separable multi-dimensional LEG kernels scales linearly with the number of grid points; it thus follows trivially that matrix-vector multiplication with finite sums of these separable multi-dimensional LEG kernels will also scale linearly with the number of grid points.

## Appendix C. Known results

### C.1 Spectra of Gaussian Processes of the form $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$

Let $\Sigma : \mathbb{R}^d \to \mathbb{R}^{n \times n}$ denote the covariance kernel of some Gaussian Process. The spectrum of $\Sigma$ admits a few properties which we use in our results.

**Lemma 10 (Spectra of Gaussian Processes)**

- *There exists a complex matrix-valued measure $\tilde{M}(d\omega)$ such that*

$$\Sigma(\tau) = \int \exp(-i\tau \cdot \omega) \tilde{M}(d\omega)$$

- *For any interval $(a, b)$, $\tilde{M}(a, b)$ is Hermitian and nonnegative-definite.*
- *If $\Sigma$ is continuous and Lebesgue-integrable, $\tilde{M}$ is absolutely continuous with respect to Lebesgue measure on $\omega$. In particular, it admits the representation $\tilde{M}(d\omega) = f(\omega)M(\omega)d\omega$ for a bounded probability density $f$ and a bounded Hermitian nonnegative-definite-valued function $M$.*

**Proof** See Cramér (1940). ■

### C.2 Covariance and Spectra of Ornstein-Uhlenbeck Processes

**Lemma 11** *Let* $\mathbf{z}$ *denote a stationary zero-mean process satisfying* $\mathbf{z}(t) = \mathbf{z}(0) + \int_0^t \left(-\frac{1}{2}G\mathbf{z}(s)ds + Nd\mathbf{w}(s)\right)$, *where* $\mathbf{w}$ *is a standard Brownian motion. Then*

1. *The stationary covariance,* $P_\infty = \mathbb{E}[\mathbf{z}(0)\mathbf{z}(0)^\top]$, *satisfies*

$$\frac{1}{2}\left(GP_\infty + P_\infty G^\top\right) = NN^\top.$$

2. *The covariance* $C(\tau) = \mathbb{E}[\mathbf{z}(\tau)\mathbf{z}(0)^\top]$ *is given below.*

$$C(\tau) = \begin{cases} \exp\left(-\frac{1}{2}G\left|\tau\right|\right)P_\infty & \text{if } \tau \geq 0 \\ P_\infty \exp\left(-\frac{1}{2}G^\top\left|\tau\right|\right) & \text{if } \tau \leq 0 \end{cases}$$

3. *Define the spectrum* $S$ *as the unique function satisfying* $C(\tau) = \int_{-\infty}^{\infty} S(\omega)\exp\left(-i\omega\tau\right)$. *If the real part of the eigenvalues of* $G$ *are strictly positive, the spectrum satisfies the following.*

$$\begin{aligned} S(\omega) &= \frac{1}{2\pi}\left[\left(\frac{1}{2}G - i\omega I\right)^{-1}P_\infty + P_\infty\left(\frac{1}{2}G^\top + i\omega I\right)^{-1}\right] \\ &= \frac{1}{2\pi}\left(\frac{1}{2}G - i\omega I\right)^{-1}NN^\top\left(\frac{1}{2}G^\top + i\omega I\right)^{-1} \end{aligned}$$

**Proof** In turn.

1. Observe that this matrix equation corresponds to the fixed point equation for the Fokker-Planck equation corresponding to this stochastic differential equation (Artemiev and Averina, 1997).

2. Vatiwutipong and Phewchean (2019) give that $\mathbb{E}[\mathbf{z}(t)|\mathbf{z}(0)] = \exp(-Gt/2)\mathbf{z}(0)$. Thus

$$\mathbb{E}[\mathbf{z}(t)\mathbf{z}(0)^\top] = \mathbb{E}[\mathbb{E}[\mathbf{z}(t)|\mathbf{z}(0)]\mathbf{z}(0)] = \exp(-G^\top t/2)P_\infty$$

   as desired.

3. Applying the Fourier inversion formula, we obtain that

$$S(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\exp(i\omega\tau)C(\tau)d\tau$$

   as long as such an integral converges. Since $C$ is itself defined differently for $t \geq 0$ and $t \leq 0$, it is convenient to split this into parts. First the positive part.

$$\int_0^\infty e^{i\omega\tau}C(\tau)d\tau = \left(\int_0^\infty e^{\tau(i\omega I - G/2)}d\tau\right)P_\infty = (G/2 - i\omega I)^{-1}P_\infty$$

Note that this integral does indeed converge, since we have assumed the real part of the eigenvalues of $G$ are strictly positive. By the same reasoning, the negative part takes the form $P_\infty(G^\top/2 + i\omega I)^{-1}$. This yields the first expression for $S(\omega)$. The second expression follows by using part 1 of this Lemma.

■

### C.3 Glick's Lemma

**Lemma 12 (Glick's extension of Scheffe's lemma)** *Let $(\Omega, \mathcal{F}, \pi)$ a probability measure space. Let $h_1, h_2, h_3 \cdots$ denote a sequence of $\mathcal{F}$-measurable functions of the form $h_\ell : \mathbb{R}^d \times \Omega \to \mathbb{R}$. Let $h_\infty : \mathbb{R}^d \to \mathbb{R}$ another function. For Lebesgue-almost-every $x$ and $\pi$-almost-every $\omega$, assume that $\lim_{\ell\to\infty} h_\ell(x,\omega), h_2(x,\omega) = h_\infty(x)$. For $\pi$-almost-every $\omega$, assume that $\lim_{\ell\to\infty} \int |h_1(x,\omega)|dx = \int |h_\infty(x,\omega)|dx$. Then, for $\pi$-almost-every $\omega$, we have that*

$$\lim_{\ell\to\infty} \int |h_\ell(x,\omega) - h_\infty(x)|dx = 0$$

**Proof** Apply Scheffe's lemma for each $\omega$. This observation is generally credited to Glick (Glick, 1974). ■