General linear-time inference for Gaussian Processes on one dimension

Jackson Loper [1] **David Blei** [1] **John P. Cunningham** [1] **Liam Paninski** [1]

Abstract

Gaussian Processes (GPs) provide a powerful probabilistic framework for interpolation, forecasting, and smoothing, but have been hampered by computational scaling issues. Here we prove that for data sampled on one dimension (e.g., a time series sampled at arbitrarily-spaced intervals), approximate GP inference at any desired level of accuracy requires computational effort that scales *linearly* with the number of observations; this new theorem enables inference on much larger datasets than was previously feasible. To achieve this improved scaling we propose a new family of stationary covariance kernels: the Latent Exponentially Generated (LEG) family, which admits a convenient stable state-space representation that allows linear-time inference. We prove that any continuous integrable stationary kernel can be approximated arbitrarily well by some member of the LEG family. The proof draws connections to Spectral Mixture Kernels, providing new insight about the flexibility of this popular family of kernels. We propose 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. However, this expressiveness comes at a cost: if implemented naively, inference in a GP given m observed data points will require $O(m^3)$ 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 that do not rely on special properties of the covariance kernel. Some well-known examples of this approach include (Quiñonero-Candela & Rasmussen, 2005; Snelson & 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), 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 & Nickisch, 2015).

When the GP has one-dimensional input – e.g., a scalar or vector time-series sampled at arbitrary time points – the most 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 & 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 $O(m)$ inference via message passing.

Is approximating a GP with a state-space model a generally viable strategy? In several special cases it has been shown that state-space models provide excellent approximations of specific covariance kernels (Karvonen & Sarkkä, 2016; Benavoli & Zaffalon, 2016). Discrete-time processes on a finite interval can also be approximated this way (Lindgren et al., 2011). Practically, (Gilboa et al., 2013) shows it is straightforward to learn many GP models using a state-space model.

In this work we establish the full generality of this strategy: we offer a new theorem proving that *any* GP on one dimension with a Lebesgue-integrable continuous kernel can be arbitrarily well approximated by a specifically-chosen state-space model. By doing so, we effectively reduce the

[1]Columbia University, New York, New York, USA. Correspondence to: Jackson Loper <jl5116@columbia.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020.

run-time burden of GPs on one dimension from cubic to linear.

We first develop a new class of Gaussian hidden Markov models on one dimension: the Latent Exponentially Generated (LEG) process. This model family is a generalization of the Celerite family of Gaussian Processes (Foreman-Mackey et al., 2017). Unlike some popular state-space models such as the ARMA, LEG processes do not require that the observations are equally spaced. These models define a distribution on vector-valued functions on the entire real line, $X : \mathbb{R} \to \mathbb{R}^n$. By construction, LEG processes are stable and stationary, with a kernel that can be evaluated easily, and inference requires linear time. In addition, we here show that inference for these models can be parallelized efficiently via a technique known as Cyclic Reduction (Sweet, 1974), leading to significant runtime improvements. Furthermore these 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 ε, the covariance of X can be matched within ε by a LEG covariance kernel.

LEG kernels generalize the Celerite kernel (Foreman-Mackey et al., 2017) by allowing more model flexibility and permitting vector-valued observations. Every Celerite kernel can be understood as a special case of a LEG kernel.

The remainder of this paper defines the LEG family, derives its essential properties and 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. Preamble: Gaussian process generalities

A Gaussian Process on one dimension is a random function $X : \mathbb{R} \to \mathbb{R}^n$ such that for any finite collections of times $t_1, t_2 \cdots t_m$ the joint distribution of $(X(t_1), \cdots X(t_m)) \in \mathbb{R}^{m\times n}$ is jointly Gaussian. The covariance kernel of X is a matrix-valued function defined by

$$\Sigma(s,t) = \mathrm{Cov}(X(s), X(t)) \in \mathbb{R}^{n\times n}.$$

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 τ for the time-lag $t-s$, i.e $C = C(\tau)$.

We will focus on two critical computational tasks here: inference and learning. "Inference" refers to using a GP model to compute the conditional densities of X given a finite collection of observations $D = (X(t_1), \cdots X(t_m))$. "Learning" refers to estimating the covariance of X from the data D. Both of these tasks can be computationally intensive: naive evaluation of the likelihood of D requires computing the determinant of an $m\times m$ matrix and solving an m-dimensional linear system. In general these tasks require $O(m^3)$ operations. Here we circumvent this scaling law by restricting ourselves to a parametric family of kernels which admit linear-time (i.e., $O(m)$) algorithms. We further show that this restriction is without loss of generality, since this family of kernels is capable of approximating any integrable continuous stationary kernel on the real line.

3. The LEG kernel

We introduce a parametric family of random processes on one dimension: the Latent Exponentially Generated (LEG) process. This process will achieve both goals of this work: linear-time inference and arbitrary approximation quality to any GP. For clarity of exposition, what follows assumes stationarity and zero mean; generalizations are discussed in Section 4. We first define the latent GP, after which we define the observation model; taken together these objects will form the LEG family.

In designing a family of latent GP models, our first goal is to enable fast computation: the models should be stationary, with an easily-computed kernel. In addition, it is convenient to focus on Markovian models, since the Markov property will enable efficient inference.

A general and classic family of Markovian models are given by linear Langevin equations (Coffey et al., 2004), i.e. processes defined by

$$z(t) = z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s)),$$

where w is an ℓ-dimensional Brownian motion, and G, σ are square matrices. To ensure this process doesn't grow without bound, we need the real part of the eigenvalues of G to be positive. Guaranteeing this nontrivial constraint is challenging (Buesing et al., 2012; Choudhary et al., 2019). To remedy this problem we developed a closely-related family of models which are always stable and stationary:

Definition 1. Let $z(0) \sim \mathcal{N}(0, I)$, let w denote a Brownian motion, let N, R be any $\ell \times \ell$ matrices, and let $G = NN^\top + R - R^\top$. Let z satisfy

$$z(t) = z(0) + \int_0^t \left(-\frac{1}{2}Gz(s)ds + Ndw(s)\right).$$

Then we will say z is a Purely Exponentially Generated process, $z \sim \mathrm{PEG}(N, R)$.

This family has another advantage: the covariance kernel is easy to compute. The covariance kernel for linear Langevin models usually involves an integral (Vatiwutipong & Phewchean, 2019), but for PEG models we can compute this integral in closed form:

Lemma 1. *$z \sim \mathrm{PEG}(N, R)$ is stationary, with covariance kernel given by*

$$C_{\mathrm{PEG}}(\tau; N, R) \triangleq \exp\left(-\frac{\tau}{2}\left(NN^\top + R - R^\top\right)\right).$$

Proof: See supplementary material.

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 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 rapid oscillations in the process, while the eigenvectors control how these oscillations are mixed across the dimensions of Z. As an illustration, Figure 1 shows the first dimension of samples from an $\ell = 2$ dimensional PEG process with various values of N, R.

We now turn to the observed process:

Definition 2. Let $z \sim \mathrm{PEG}(N, R)$. Fix any $n \times \ell$ matrix B and any $\ell \times \ell$ matrix Λ. For each t independently, define the conditional observation model:

$$x(t)|z(t) \sim \mathcal{N}(Bz(t), \Lambda\Lambda^\top).$$

We define a **Latent Exponentially Generated (LEG)** process to be the Gaussian Process $x : \mathbb{R} \to \mathbb{R}^n$ generated by a PEG prior and the above observation model. We write $x \sim \mathrm{LEG}(N, R, B, \Lambda)$. X has a LEG kernel:

$$C_{\mathrm{LEG}}(\tau; N, R, B, \Lambda) \triangleq B\left(C_{\mathrm{PEG}}(\tau; N, R)\right)B^\top + \delta_{\tau=0}\Lambda\Lambda^\top.$$

Here δ is the indicator function, and again $\tau > 0$. We will refer to the latent dimension ℓ as the **rank** of the LEG kernel.

3.1. Computation with LEG processes

The LEG model is a Gaussian hidden Markov model. As usual in such models, it follows that problems of evaluation, interpolation, smoothing, and sampling reduce to operations with block-tridiagonal matrices (De Jong, 1988). This block-tridiagonal structure is what enables linear-time inference.

While the obvious choice for processing these block-tridiagonal matrices might be a Kalman filter, it is not ideally suited for modern hardware: the naïve Kalman filter requires a single sequential sweep through the data. If the latent process has a quick mixing time this requirement can be relaxed (Gonzalez et al., 2009), but we seek an algorithm that parallelizes efficiently regardless of the parameters of the model.

We here propose to use Cyclic Reduction (CR) techniques instead. These offer a convenient parallelizable approach to computation with block-tridiagonal matrices (Sweet, 1974). To our knowledge, the CR approach has not previously been applied in the Gaussian Process literature. Like the Kalman filter, CR can be understood as a linear-time Cholesky decomposition algorithm for block-tridiagonal matrices (Eubank & Wang, 2002). Linear-time Cholesky decompositions lead directly to linear-time algorithms for solving linear systems and computing the determinant, which, in turn, allows us to compute all quantities required for inference in LEG processes. The difference between the Kalman filter and CR is "pivoting"; CR computes the Cholesky decomposition of a carefully permuted version of a block-tridagonal matrix. This pivoting allows the CR algorithm to proceed in $\log_2 m$ parallelizable stages, each stage concerning a matrix half the size of the matrix from the previous stage[1]. Unlike the Kalman Filter, CR can be completed with k processors on the order of m/k time (as long as $k < m$). Implementing parallel versions of CR in modern software libraries (TensorFlow2 in this case) was straightforward, making it easy to take advantage of modern hardware.

Exact linear-time algorithms for likelihood (and gradient) evaluation, smoothing, forecasting, and interpolation are given in the supplement. TensorFlow2-based Python code, tutorial notebooks, and API documentation can be found at `https://github.com/jacksonloper/leg-gps`.

3.2. Generality of the LEG family

The LEG family is useful only in so much as it is able to accurately approximate other GP kernels. Here, we prove that in fact the LEG family is general: *any* stationary Lebesgue-integrable stationary continuous kernel can be approximated to arbitrary accuracy with a LEG family of a certain rank ℓ.

Intuitively the argument is as follows: first, the PEG family provides a general and well-behaved (stable, stationary, correlated) collection of ℓ state-space components. Second, the LEG observation model creates *mixture* of those underlying PEG components. Third, we show that the LEG family has nonzero intersection with spectral mixture kernels (a popular class of kernels defined more carefully below), thus drawing a novel and useful connection between spectral mixtures and state space models. Fourth, we extend known facts about the generality of spectral mixtures to the multidimensional case. As a result, finally, we conclude that the LEG family is general (all without sacrificing its linear

[1] Those familiar with the multigrid technique (Terzopoulos, 1986; Hackbusch, 2013) – which has been used for Gaussian inference in other contexts (Papandreou & Yuille, 2010; Mukadam et al., 2016; Zanella & Roberts, 2017) – will note similarities between multigrid and CR.

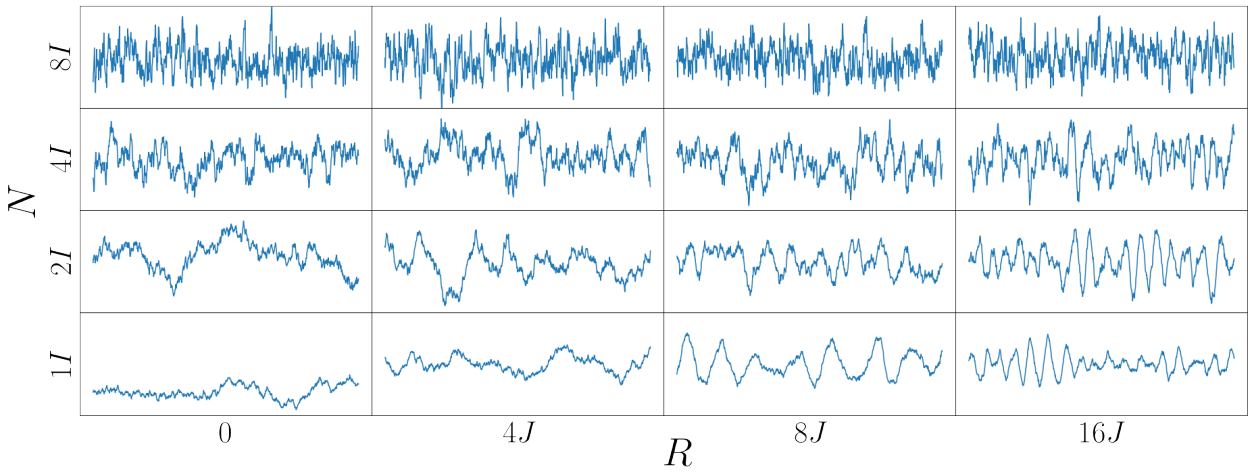

Figure 1. **PEG process samples**. The plots above show representative samples from the model $\mathrm{PEG}(N, R)$ as we vary N and R. Here we consider rank-2 PEG models (only one element of the 2d vector is plotted), so N, R are both 2×2 matrices. We vary N by taking it to be various multiples of the identity. We vary R by taking various multiples of J, the antisymmetric 2×2 matrix with zeros on the diagonal and ± 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.

runtime).

To begin, we study the spectral representation of the LEG kernel. If a kernel is stationary and continuous, Bochner's theorem guarantees it has a spectrum (Bhatia, 2015): a unique matrix-valued measure F such that

$$C(\tau) = \int e^{-i\tau\omega} dF(\omega).$$

Spectral Mixture (SM) methods offer a direct way to approximate any stationary kernel through its spectrum. For the purposes of this article we will define SM kernels as follows:

Definition 3. Let p denote a probability density on $\mathbb{R}$, let $b_1, b_2 \cdots b_\ell \in \mathbb{C}^n$, let $\mu \in \mathbb{R}^\ell$, and let $\gamma > 0$. The **Spectral Mixture** kernel, $C_{\mathrm{SM}}(t; p, b, \mu, \gamma)$, is given by

$$\sum_{k=1}^{\ell} \int e^{-i\xi x} b_k b_k^* \gamma p(\gamma(\xi - \mu_k)) d\xi.$$

We will say that C is **based on** p, since its spectrum is a sum of scaled and shifted versions of p.

Spectral Mixtures were first introduced in machine learning in (Wilson & Adams, 2013), where it was noted that any kernel which is the covariance of a weakly stationary mean square continuous random process $X : \mathbb{R} \to \mathbb{R}$ (or indeed $X : \mathbb{R}^n \to \mathbb{R}$) can be well-approximated using SM kernels. However, that result does not hold for our case, i.e. kernels for processes of the form $X : \mathbb{R} \to \mathbb{R}^n$. In this situation the spectrum of the kernel becomes a complex-matrix-valued measure (instead of an ordinary probability measure).

These mixture kernels have an interesting connection to LEG kernels: all Cauchy-based spectral mixture kernels are actually also LEG kernels. These kernels thus fall at the intriguing intersection of Gaussian Hidden Markov models (which are linear run-time) and Spectral Mixture models (which have not previously been considered linear run-time). Every Cauchy-based SM kernel can be understood as a LEG kernel. There is also another generalization of Cauchy-based SM kernels, known as the Celerite kernels (Foreman-Mackey et al., 2017); these are built by linear combinations of kernels called Celerite terms. Below we summarize the relationship between these three families of kernels:

Lemma 2 (SM kernels, Celerite kernels, LEG kernels)**.**

1. *Every Cauchy-based real-valued SM kernel* $C_{\mathrm{SM}} : \mathbb{R} \to \mathbb{R}$ *can be understood as a Celerite kernel.*
2. *Every positive-definite Celerite term can be understood as a LEG kernel.*
3. *Every Cauchy-based real-valued SM kernel* $C_{\mathrm{SM}} : \mathbb{R} \to \mathbb{R}^{n \times n}$ *can be understood as a LEG kernel.*

Proof: See supplementary material.

Thus, to prove that the family of LEG kernels is general, it suffices to show that SM kernels are general. The key idea is to generalize a classic result from kernel density estimation. We achieve this generalization in the following theorem:

Theorem 1 (Total variation convergence for weighted kernel density estimation)**.** *Let* K, p *denote bounded densities*

on $\mathbb{R}^d$. Let $g : \mathbb{R} \to [-M, M]$. 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

$$\mathbb{P}\left(\lim_{\ell\to\infty} \int |h_\ell(\xi) - p(\xi)g(\xi)|d\xi = 0\right) = 1.$$

Proof: See supplementary material.

With this theorem in place, we next show that spectral mixture kernels can approximate any integrable continuous kernel for a stationary Gaussian process on one dimension:

Corollary 1 (Flexibility of Spectral Mixture kernels)**.** *Fix p, a bounded probability density on $\mathbb{R}^n$, $\varepsilon > 0$, and any Lebesgue-integrable continuous positive definite[2] stationary kernel $\Sigma : \mathbb{R} \to \mathbb{C}^{n\times n}$. There exists a real valued kernel $C = C_{\mathrm{SM}}(p, b, \mu, \gamma)$ such that $\|C(\tau)z - \Sigma(\tau)z\| < \varepsilon\|z\|$ for every $\tau \in \mathbb{R}, z \in \mathbb{C}^n$.*

Proof: See supplementary material.

This corollary can be used to establish our main mathematical result, i.e., that LEG models enjoy the same flexibility guarantee.

Theorem 2 (Flexibility of LEG and Celerite kernels)**.** *For every $\varepsilon > 0$ and every Lebesgue-integrable continuous positive definite stationary kernel $\Sigma : \mathbb{R} \to \mathbb{R}^{n\times n}$ there exists a Celerite kernel C such that $\|C(\tau)z - \Sigma(\tau)z\| < \varepsilon\|z\|$ for every $\tau > 0, z \in \mathbb{C}^n$. Moreover, there exists a LEG kernel with the same guarantee.*

Proof: See supplementary material.

In conclusion, we have proven that any stationary Gaussian Process on one dimension can be well approximated using LEG processes, and further that the computational effort for LEG processes scales linearly with the number of observations. Thus, putting these two pieces together, approximate inference for any stationary Gaussian Processes on one dimension, at any desired level of accuracy, requires computational effort that scales linearly with the number of observations.

4. Extensions

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

[2]The requirements of continuity and integrability are slightly too strong. For example, the sinc kernel is not Lebesgue integrable, but it is easy to approximate with a spectral mixture kernel. In the future we hope to refine these conditions.

4.1. Non-stationary processes

We have focused on stationary processes here for simplicity. A number of potential extensions to non-stationary processes are possible while retaining linear-time scaling. As one example, starting with LEG processes as a base, non-stationary models can be developed using the techniques from (Benavoli & Zaffalon, 2016).

4.2. 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 & 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 & Brown, 2003; Paninski et al., 2010; Fahrmeir & Tutz, 2013; Polson et al., 2013; Khan & Lin, 2017; Nickisch et al., 2018).

4.3. Non-linear domains

Just as Gaussian Markov models in discrete time can be easily extended to Gaussian graphical models on general graphs, we can extend the Gaussian Markov PEG and LEG processes to stochastic processes on more general domains. In the simplest case the domain of the process could be a tree, with the PEG kernel defined in terms of distance along the tree, rather than distance on the line. Inference in the resulting tree-structured Gaussian graphical model can proceed via message passing in $O(m)$ time.

4.4. Multi-dimensional domains

We can also use LEG kernels to model processes of the form $x : \mathbb{R}^d \to \mathbb{R}^n$. Let B, Λ be matrices, let N, R be collections of matrices, and let $C_{\mathrm{KLEG}}(\tau; N, R, B, \Lambda) \triangleq \delta_\tau \Lambda\Lambda^\top + \sum_{r=1}^{\zeta} \prod_{k=1}^{d} BC_{\mathrm{PEG}}(\tau_k; N_{rk}, R_{rk})B^\top$.

Theorem 3. *Let $\Sigma : \mathbb{R}^d \to \mathbb{R}^{n\times n}$ any positive-definite integrable continuous stationary kernel, and fix $\varepsilon > 0$. There exists a KLEG kernel such that $\|C(\tau)z - C_{\mathrm{KLEG}}(\tau)z\| < \varepsilon z$.*

Proof: See supplementary material.

Efficient computation is possible for observations from a KLEG process along a (potentially irreguarly-spaced) grid. The covariance matrix of these observations has structure which can be leveraged for efficient computation. For example, in the supplement we give an algorithm for multiplying by this matrix, and show that the computational cost of this

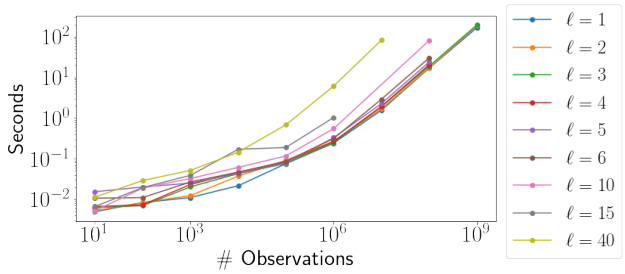

Figure 2. **Walltime for evaluating LEG likelihoods**. How long does it take to compute the likelihood of observations from a LEG model on an m5.24xlarge machine on the Amazon AWS service? We compare times for differently-ranked models and different numbers of observations. For example, the likelihood for one billion observations under a rank-3 LEG model can be computed in roughly three and a half minutes.

algorithm scales linearly with the number of points in the grid. Combined with GPyTorch (cf. (Gardner et al., 2018)), this algorithm should yield efficient inference algorithms for KLEG processes.

5. Experiments

Here we are interested in testing the theoretical results described above. In practice, how fast is inference with the LEG process? How well can the LEG model approximate popular kernels? How well does the LEG model extrapolate and interpolate? How well can it smooth?

5.1. Computational complexity

Throughout what follows, we will perform inference on LEG processes using the Cyclic Reduction algorithm outlined in 3.1. We wanted to check if there are practical difficulties that could negate the theoretically linear computational cost of this method. We measured how long it took to compute the likelihood of single contiguous chains of observations from LEG processes of various ranks. In each case we used an m5-24xlarge machine on Amazon Web Services (AWS).

Overall, the empirical scaling appeared consistent with the theoretical predictions. 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 2. 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 approximately linear for larger values of m.

5.2. Matching one-dimensional kernels

Theorem 2 shows that LEG kernels can represent any stationary Gaussian process arbitrarily well *if* the latent dimension ℓ is sufficiently high. How high does this dimension actually need to be in order to get a good fit? We investigate this question by examining several popular one-dimensional kernels. In each case we draw fifty thousand observations, each taken .1 units apart from the next. We fit LEG models of various ranks by optimizing the log likelihood using the BroydenFletcherGoldfarbShanno (BFGS) algorithm (as implemented in SciPy). Gradients were computed by TensorFlow2 using backpropogation through the Cyclic Reduction algorithm. We found this approach to be simple, scalable, and robust across datasets. The likelihood could also be optimized using Expectation Maximization, but we found it was not as fast (Dempster et al., 1977). The model could also by fit by moment-matching instead of likelihood; this is a common practice for ARMA models (Brockwell & Davis, 2013) and we hope to explore this possibility in the future.

How well do LEG kernels approximate the Rational Quadratic (RQ) kernel,

$$C_{\mathrm{RQ}}(\tau) = 2/(1+\tau^2)?$$

On the one hand, the Rational Quadratic kernel is profoundly different from every LEG kernel. The spectrum of the RQ kernel decays exponentially, whereas the spectrum of of a LEG process is asymptotically an inverse polynomial (recall from section 3.2 that PEG processes contain Cauchy spectral mixture models as a special case). On the other hand, Theorem 2 guarantees a Rational Quadratic kernel can be matched uniformly well by a LEG kernel. Figure 3 shows a rank-4 LEG kernel $C(\tau)$ appears to do an excellent job of matching the RQ kernel for $t < 2$ and an adequate job of matching for $t > 2$.

This apparent contradiction – RQ is profoundly different (on the tails of the spectrum) from every LEG kernel, yet every RQ kernel can be matched arbitrarily well by a LEG kernel – is resolved by considering the different timescales involved in any Gaussian Process. LEG kernels can uniformly approximate any stationary covariance, which means that we can use them to get uniformly accurate forecasts and interpolations at any fixed timescale. If a LEG kernel is trained on observations at a particular timescale, the kernel will attempt to match smoothness *at that timescale*. For example, these LEG kernels were trained on observations at a timescale of .1, so they will attempt to match the covariance at that scale and larger.

The case of the (RBF) kernel, given by $C_{\mathrm{RBF}}(\tau) = \exp(-\tau^2/2)$, is even more extreme. This kernel's spectrum decays log-quadratically – even faster than the spectrum of the RQ kernel. For any RBF process, any RQ process, and any PEG process, one can always find a small enough scale

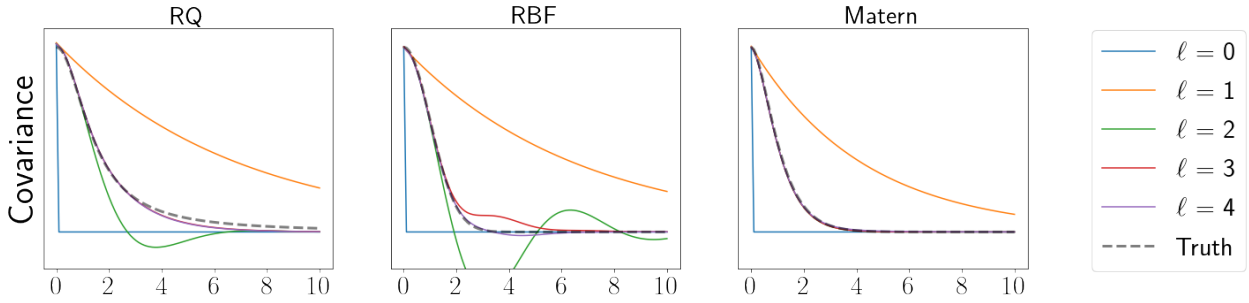

Figure 3. **LEG kernel approximation of some popular specific kernels**. By taking the rank ℓ sufficiently high we can achieve arbitrarily good approximations to any kernel. In the three examples shown here, $\ell = 4$ already provides adequate approximation quality. Note that in some cases some lines aren't visible because they are superimposed on each other; for example, when approximating Matern kernels we find nearly identical models for for $\ell \in 2, 3, 4$.

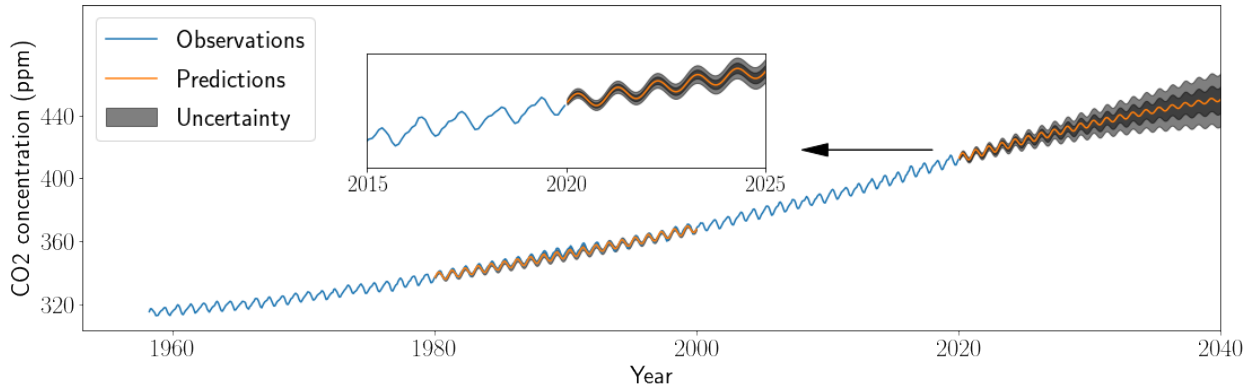

Figure 4. **LEG processes interpolate and extrapolate well across long timescales**. It appears that a rank-5 LEG model is sufficient to capture the linear and periodic trends in the Mauna Loa CO_2 dataset. Above we compare the true observations with interpolations made by the LEG model. The gray areas 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.

so that the RBF process will look smoother than the RQ process and the RQ process is smoother than the PEG processes. Modeling this smoothness is difficult for lower-rank LEG models. For example, the best rank-2 LEG model includes a large oscillation that is not found in the ground-truth RBF kernel. It is not until rank 4 that the LEG process is able to match the kernel well.

Finally, the Matern kernel with $\nu = 1.5$ turns out to be an easy case. This kernel is given by

$$C_{\text{Matern}}(\tau) = (1 + \sqrt{3}\tau)\exp(-\sqrt{3}\tau).$$

Like PEG kernels, the spectrum of the Matern kernel decays slower than exponentially. In fact, this Matern kernel lies inside the rank-2 LEG family. Let

$$N = 3^{1/4}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \qquad R = \sqrt{3}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

Then the Matern kernel is given by $\text{LEG}(N, R, \mathbf{1}/\sqrt{2}, 0)$.

5.3. Mauna Loa CO_2

To find out whether LEG models can offer a practical tool for extrapolation and interpolation, we turn 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 & 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 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.

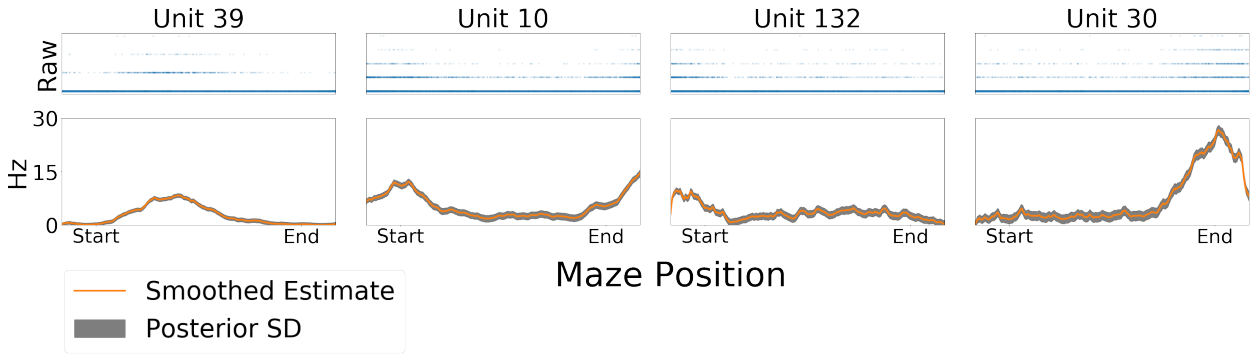

Figure 5. **LEG processes can smooth irregularly spaced neural data**. Ten thousand irregularly spaced observations suggest that the firing rates of hippocampal neurons are modulated by the rat's position in a maze. However, the modulation strength visible in the raw data is weak. By smoothing this data 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.

The results are shown in Figure 4. It is encouraging that the LEG predictions interpolate adequately from 1980 to 1920. 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 1920. This example shows that the LEG model is sufficiently flexible to learn unanticipated structures in the data.

5.4. Hippocampal place-cells

Smoothing is another common application of GPs. Here we see whether LEG models can be used to smooth irregularly spaced observations from neural spiking data (Grosmark & 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 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 z. When the rat is at 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 & Brown, 2003; Rahnama Rad & Paninski, 2010; Savin & Tkacik, 2016; Gao et al., 2016), and references therein; as noted in section 4, 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 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 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.

6. Conclusion

We here make two advances in speeding up inference 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, critically enabling linear runtime scaling in the number of observations. Second, we make this theoretical result practical by developing Cyclic Reduction-based algorithms to parallelize this computation, and sharing TensorFlow2-based implementations of these algorithms. 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.

References

Ambikasaran, S., Foreman-Mackey, D., Greengard, L., Hogg, D. W., and ONeil, M. Fast direct methods for gaussian processes. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):252–265, 2015.

Benavoli, A. and Zaffalon, M. State space representation of non-stationary gaussian processes. *arXiv preprint arXiv:1601.01544*, 2016.

Bhatia, R. *Positive Definite Matrices*. Princeton Series in Applied Mathematics. Princeton University Press, 2015. ISBN 9780691168258.

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

Buesing, L., Macke, J. H., and Sahani, M. Learning stable, regularised latent models of neural population dynamics. *Network: Computation in Neural Systems*, 23(1-2):24–47, 2012.

Cheung, B. L. P., Riedner, B. A., Tononi, G., and Van Veen, B. D. Estimation of cortical connectivity from eeg using state-space models. *IEEE Transactions on Biomedical engineering*, 57(9):2122–2134, 2010.

Choudhary, N., Gillis, N., and Sharma, P. On approximating the nearest ω-stable matrix. *arXiv preprint arXiv:1901.03069*, 2019.

Coffey, W., Kalmykov, Y., and Waldron, J. *The Langevin Equation: With Applications to Stochastic Problems in Physics, Chemistry, and Electrical Engineering*. Series in contemporary chemical physics. World Scientific, 2004. ISBN 9789812384621.

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

Cutajar, K., Osborne, M., Cunningham, J., and Filippone, M. Preconditioning kernel matrices. In *Proceedings of the International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2016.

De G. Matthews, A. G., Van Der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. Gpflow: A gaussian process library using tensorflow. *The Journal of Machine Learning Research*, 18(1):1299–1304, 2017.

De Jong, P. The likelihood for a state space model. *Biometrika*, 75(1):165–169, 1988.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Eubank, R. and Wang, S. The equivalence between the cholesky decomposition and the kalman filter. *The American Statistician*, 56(1):39–43, 2002.

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

Foreman-Mackey, D., Agol, E., Ambikasaran, S., and Angus, R. Fast and scalable gaussian process modeling with applications to astronomical time series. *The Astronomical Journal*, 154(6):220, 2017.

Gao, Y., Archer, E. W., Paninski, L., and Cunningham, J. P. Linear dynamical neural population models through nonlinear embeddings. In *Advances in neural information processing systems*, pp. 163–171, 2016.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, pp. 7576–7586, 2018.

Gilboa, E., Saatçi, Y., and Cunningham, J. P. Scaling multidimensional inference for structured gaussian processes. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):424–436, 2013.

Gonzalez, J., Low, Y., and Guestrin, C. Residual splash for optimally parallelizing belief propagation. In *Proceedings of Artificial Intelligence and Statistics*, 2009.

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

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

Hartikainen, J., Riihimäki, J., and Särkkä, S. Sparse spatiotemporal gaussian processes with general likelihoods. In *International Conference on Artificial Neural Networks*, pp. 193–200. Springer, 2011.

Hensman, J., Fusi, N., and Lawrence, N. D. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, 2013.

Karvonen, T. and Sarkkä, S. Approximate state-space gaussian processes via spectral transformation. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing*, pp. 1–6. IEEE, 2016.

Keeling, C. D. and Whorf, T. Atmospheric carbon dioxide record from mauna loa. *Carbon Dioxide Research Group, Scripps Institution of Oceanography, University of California La Jolla, California*, pp. 92093–0444, 2005.

Khan, M. E. and Lin, W. Conjugate-computation variational inference. In *Proceedings of Artificial Intelligence and Statistics*, 2017.

Lindgren, F., Rue, H., and Lindström, J. 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.

Low, K. H., Yu, J., Chen, J., and Jaillet, P. Parallel gaussian process regression for big data: Low-rank representation meets markov approximation. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Mergner, S. *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.

Mukadam, M., Yan, X., and Boots, B. Gaussian process motion planning. In *2016 IEEE international conference on robotics and automation*, pp. 9–15. IEEE, 2016.

Nguyen, T. V. and Bonilla, E. V. Automated variational inference for gaussian process models. In *Advances in Neural Information Processing Systems*, pp. 1404–1412, 2014.

Nickisch, H., Solin, A., and Grigorevskiy, A. State space Gaussian processes with non-Gaussian likelihood. In Dy, J. and Krause, A. (eds.), *Proceedings of the International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3789–3798, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

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

Paninski, L., Ahmadian, Y., Ferreira, D. G., Koyama, S., Rad, K. R., Vidne, M., Vogelstein, J., and Wu, W. A new look at state-space models for neural data. *Journal of computational neuroscience*, 29(1-2):107–126, 2010.

Papandreou, G. and Yuille, A. L. Gaussian sampling by local perturbations. In *Advances in Neural Information Processing Systems*, pp. 1858–1866, 2010.

Polson, N. G., Scott, J. G., and Windle, J. Bayesian inference for logistic models using pólya–gamma latent variables. *Journal of the American statistical Association*, 108(504):1339–1349, 2013.

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

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

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

Riihimäki, J., Vehtari, A., et al. Laplace approximation for logistic gaussian process density estimation and regression. *Bayesian analysis*, 9(2):425–448, 2014.

Savin, C. and Tkacik, G. Estimating nonlinear neural response functions using gp priors and kronecker methods. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3603–3611. 2016.

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

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

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

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

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

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K. Q., and Wilson, A. G. Exact gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, pp. 14622–14632, 2019.

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

Wilson, A. and Nickisch, H. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *Proceedings of the International Conference on Machine Learning*, pp. 1775–1784, 2015.

Zanella, G. and Roberts, G. Analysis of the gibbs sampler for gaussian hierarchical models via multigrid decomposition. *arXiv preprint arXiv:1703.06098*, 2017.

Zhang, Y., Leithead, W. E., and Leith, D. J. 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*, pp. 3711–3716. IEEE, 2005.

General linear-time inference for Gaussian Processes on one dimension: Supplementary Material

February 21, 2020

In this document we detail the theory and practice for working with Latent Exponentially Generated (LEG) Gaussian Processes.

- Section 1 is a review about Gaussian Processes of the form $x : \mathbb{R} \to \mathbb{R}^n$ for $n > 1$. This may be helpful for readers which are not familiar with the peculiar matrix-valued spectra of such processes.
- Section 2 is about the theory of the PEG and LEG models. This section includes proofs for all of the results in the main text. It shows how we arrive at the expression for the covariance of the LEG model. It also details the connection between Spectral Mixture kernels and LEG kernels. This section will be interesting to those seeking to understand why LEG kernels can approximate any Lebesgue-integrable continuous kernel. We conjecture that the conditions of continuity and Lebesgue-integrability are too strong; we hope that an interested reader may be able to figure out how to loosen these conditions.
- Section 3 is about using Cyclic Reductions to do fast inference with PEG and LEG models. We show how inference with these models is easy as long as we can work efficiently with block-tridiagonal matrices. We define the Cyclic Reduction algorithm for block-tridiagonal matrices and show how it enables us to efficiently compute what we need. (N.B. some notation in this section differs slightly from Section 2; in particular, the symbol $\tilde{B}$ is repurposed)
- Section 4 is about the leggps python package which implements the algorithms from Section 3. This section may be helpful for users of this python code. The leggps package exposes a numpy-based API for learning, finding the posterior, smoothing, interpolating, and forecasting with LEG models (note however that this code uses TensorFlow2 as a backend, so TensorFlow2 must be installed for this code to work). This package also exposes a TensorFlow2-based API for Cyclic Reduction algorithms, which could be used for unrelated applications involving block-tridiagonal matrices.
- Section 5 includes extensions we would like to implement in the future. For example, we show that LEG kernels can be used to accelerate inference for processes of the form $z : \mathbb{R}^d \to \mathbb{R}^n$; in future we would like to write code to make this vision a reality. If any extensions in this section are important for your work, don't hesitate to raise an issue on the Github Repo at `https://github.com/jacksonloper/leg-gps` and start a conversation.

1 Some known facts about GPs

Here we collect some important definitions and facts – already known in the literature – which will be useful in the theory that follows.

1.1 Covariance kernels

Definition. Let $z : \mathbb{R} \to \mathbb{R}^\ell$ a Gaussian Process. The covariance kernel of z is given by

$$K(t,s) \triangleq \mathrm{Cov}(z(t), z(s))$$

Note that $K(s,t) = K(t,s)^\top$ (by the definition of a covariance matrix).

Sometimes the covariance kernel only depends on the difference between t and s. Say there exists a matrix-valued function C such that $K(t,s) = C(t-s)$, then K is said to be stationary. In this case, by a slight abuse of terminology, we will call C the covariance kernel of z. Note that $C(-\tau) = C(\tau)^\top$ (again by the definition of a covariance matrix).

For any $C : \mathbb{R} \to \mathbb{R}^{n\times n}$, we will say that C is "nonnegative-definite" if it is the covariance kernel of some Gaussian Process.

1.2 The spectrum of a covariance kernel

Here we consider the spectrum of matrix-valued functions. This spectrum is slightly more involved than the spectra of ordinary functions. To even define it we need to be slightly careful with the complex values involved.

We use the following notation for complex variables in this supplement. Unless it is clear from context that i is being used as an index, we will take $i = \sqrt{-1}$. For any b let $\overline{b}$ denote the **complex conjugate** of b, i.e. if $b = x + iy$ then $\overline{b} = x - iy$. Let the same definition hold elementwise for vectors and matrices, e.g. if $B_{ij} = x + iy$ then $\overline{B}_i j = x - iy$. For any matrix B let B^* denote the conjugate transpose, i.e. $B^*_{ij} = \overline{B}_{ji}$. Let $\Re(b) \triangleq (b + \overline{b})/2$ denote the "real part" of b and let $\Im(b) = i(\overline{b} - b)/2$ denote the "imaginary part" of b.

The following Lemma summarizes the results we will need in what follows. These results are already known in the literature.

Proposition 1 (The spectrum of continuous integrable covariance kernels)**.** *Assume $C : \mathbb{R} \to \mathbb{R}^{\ell\times\ell}$ satisfies three properties:*

- *C is continuous.*
- *C is Lebesgue integrable. That is, for each τ let $|C(\tau)| = \sup_x \|C(\tau)x\|/\|x\|$ denote the operator norm of the matrix $C(\tau)$. We require that $\int_0^\infty |C(\tau)|d\tau < \infty$.*
- *C is the covariance kernel of some Gaussian Process $z : \mathbb{R} \to \mathbb{R}^\ell$ (i.e. C is nonnegative-definite).*

Then there is an (almost-surely) unique Hermitian-nonnegative-definite-matrix-valued function $M : \mathbb{R} \to \mathbb{C}^{\ell\times\ell}$ such that

$$C(\tau) = \int e^{-i\tau\omega} M(\omega) d\omega$$

Proof. The continuity of C and the fact that it is a covariance kernel allow us to apply Bochner's Theorem to write

$$C(\tau) = \int e^{-i\tau\omega} dF(\omega)$$

for some unique Hermitian-positive-definite-matrix-valued-measure F (cf. [1]). In order to avoid the peculiarities of matrix-valued measures, we can write this in a more familiar way. Let

$$d\mu(\omega) = \mathrm{tr}\,(dF(\omega))$$

where tr denotes the trace. Note that

- μ is a finite positive measure: $\mu(\mathbb{R}) = \mathrm{tr}\,(C(0)) = \mathbb{E}[\|z(0)\|^2] < \infty$. Here we have used that the variance of a Gaussian random variable is finite.

- $\mathrm{tr}A = 0 \Leftrightarrow A = 0$ for all Hermitian positive definite matrices A, and so we have that F is absolutely continuous with respect to μ.

It follows that there exists a Radon-Nikodym derivative, $\tilde{M} : \mathbb{R} \to \mathbb{C}^{\ell \times \ell}$, such that $\tilde{M}(\omega)$ is a positive-definite Hermitian matrix for each ω and

$$F(S) = \int_S \tilde{M}(\omega) d\mu(\omega)$$

Thus the spectrum of a covariance kernel can also be written in terms of this M and a regular positive measure μ:

$$C(\tau) = \int e^{-i\tau\omega} \tilde{M}(\omega) d\mu(\omega)$$

Now we apply the Lebesgue-integrability conditions. These imply that the integral of the absolute value of each entry of C is also finite. The usual properties of Fourier Transforms thus yield that μ is actually absolutely continuous with respect to the Lebesgue measure. Thus $\mu(d\omega) = f(\omega)d\omega$ for some positive function f. We can thus write

$$C(\tau) = \int e^{-i\tau\omega} M(\omega) d\omega$$

where $M(\omega) = f(\omega)M(\omega)$. □

This leads to the following definition:

Definition. If M is a Hermitian-nonnegative-definite-matrix-valued function such that

$$C(\tau) = \int e^{-i\tau\omega} M(\omega) d\omega$$

then M is called the spectrum of C.

2 Theory of PEG and LEG models

We now turn to the definition and theory of the LEG models introduced in the main text. We also study the PEG models upon which the LEG model is built.

2.1 Model definitions

The PEG and LEG models are defined through the following generative story:

1. The PEG model.
   - N, R are $\ell \times \ell$ square matrices.
   - $G = NN^\top + R - R^\top$.
   - $z : \mathbb{R} \to \mathbb{R}^\ell$ is defined by the fact that $Z(0) \sim \mathcal{N}(0, I)$ and
   $$z(t) = z(0) + \int_0^t \left(-\frac{1}{2} G z(s) dt + N dw(s) \right)$$
   for some Brownian motion, w. In this case we say that $z \sim \mathrm{PEG}(N, R)$.
2. The LEG model, formed through an observation model on top of the PEG model:
   - B is an $n \times \ell$ matrix.
   - Λ is an $n \times n$ matrix.
   - For each t independently,
   $$x(t)|z \sim \mathcal{N}(Bz(t), \Lambda\Lambda^T)$$
   where $z \sim \mathrm{PEG}(N, R)$. In this case we say that $x \sim \mathrm{LEG}(N, R, B, \Lambda)$. The dimension of the latent PEG process, ℓ, is called the rank of the LEG process.

2.2 First properties of the PEG and LEG models

The covariance of the PEG model can be written in closed form.

Lemma 1. *$z \sim \mathrm{PEG}(N, R)$ is stationary, with covariance kernel given by*

$$C_{\mathrm{PEG}}(\tau; N, R) \triangleq \exp\left(-\frac{\tau}{2}\left(NN^\top + R - R^\top\right)\right).$$

for $\tau \geq 0$.

Proof. Let $z \sim \mathrm{PEG}(N, R)$, $G = NN^\top + R - R^\top$.

We start by looking at conditional distributions across time. Fix any $t > s$. Per [2], we have that

$$\begin{aligned}
\mathbb{E}[z(t)|z(s)] &= e^{-G(t-s)/2} z(s) \\
\mathrm{Cov}(z(t)|z(s)) &= e^{-G(t-s)/2}\left(\int_0^{t-s} e^{G\tau/2} NN^\top e^{G^\top \tau/2} d\tau\right) e^{-G^\top (t-s)/2}
\end{aligned}$$

To compute this integral, let us consider

$$M(\tau) = \exp(G\tau/2)\exp(G^\top \tau/2)$$

Using the fact that G commutes with $\exp(G)$, we have that

$$\begin{aligned}
M'(\tau) &= \frac{1}{2}\exp(G\tau/2)(G + G^\top)\exp(G^\top \tau/2) \\
&= \exp(G\tau/2) NN^T \exp(G^\top \tau/2)
\end{aligned}$$

This is precisely the object we were integrating before. The fundamental theorem of calculus therefore gives that

$$\begin{aligned}
\mathrm{Cov}(z(t)|z(s)) &= e^{-G(t-s)/2}(M(t-s) - M(0))e^{-G^\top (t-s)/2} \\
&= e^{-G(t-s)/2}(e^{G(t-s)/2} e^{G^\top (t-s)/2} - I)e^{-G^\top (t-s)/2} \\
&= I - e^{-G(t-s)/2} e^{-G^\top (t-s)/2}
\end{aligned}$$

Now we will look at unconditional marginal distributions. In particular, fix any any $t > 0$. Using the law of total expectation and the law of total covariance we find that the marginal distribution of $z(t)$ is given by:

$$\begin{aligned}
\mathbb{E}[z(t)] &= \mathbb{E}[\mathbb{E}[z(t)|z(0)]] = e^{-G(t-s)/2}\mathbb{E}[z(0)] = 0 \\
\mathrm{Cov}(z(t)) &= \mathbb{E}\left[\mathrm{Cov}(z(t)|z(0))\right] + \mathrm{Cov}(\mathbb{E}[z(t)|z(0)]) \\
&= (I - e^{-G(t-s)/2} e^{-G^\top (t-s)/2}) + (e^{-G(t-s)/2} e^{-G^\top (t-s)/2}) = I
\end{aligned}$$

Thus $z(t) \sim \mathcal{N}(0, I)$ for every $t \geq 0$. The same arguments can be applied to show that $z(t) \sim (0, I)$ for $t < 0$.

Finally, we turn to unconditional covariances across time. Since we just showed that the means are all zero it suffices to look at the second-order expectations. Fix $t > s$. We calculate that

$$\begin{aligned}
\mathbb{E}[z(t)z(s)^\top] &= \mathbb{E}[\mathbb{E}[z(t)|z(s)]z(s)^\top] \\
&= e^{-G(t-s)/2}\mathbb{E}[\mathbb{E}[z(s)z(s)^\top]] \\
&= e^{-G(t-s)/2}
\end{aligned}$$

Note that this depends only upon $t - s$. Together with the fact that the marginals are also the same for every t, this shows that z is stationary. The formula above gives the covariance kernel: $C(\tau) = e^{-G\tau/2}$, as desired. □

A final remark is warranted here about the case $\tau < 0$. In this case, recall that the definition of the covariance matrix gives that $C_{\mathrm{PEG}}(\tau; N, R) = C_{\mathrm{PEG}}(-\tau; N, R)^\top$. Thus a more complete definition of the kernel might be given by

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

Now that the covariance of the PEG model is understood, the covariance of the LEG model follows immediately: Let $x \sim \mathrm{LEG}(N, R, B, \Lambda)$. The usual rules for the covariances of Gaussian random variables yield that the covariance of x is given by

$$C_{\mathrm{LEG}}(\tau; N, R, B, \Lambda) \triangleq B\left(C_{\mathrm{PEG}}(\tau; N, R)\right) B^\top + \delta_{\tau=0}\Lambda\Lambda^\top.$$

Here δ is the indicator function.

This representation makes it straightforward to see that the sum of two LEG kernels is itself a LEG kernel. This will be helpful later as we explore connections to Spectral Mixture kernels, which can be understood as a sum of relatively simple kernels.

Proposition 2 (The sum of two LEG kernels is a LEG kernel)**.** *Let* $C(\tau) = C_{\mathrm{LEG}}(\tau; N_1, R_1, B_1, \Lambda_1) + C_{\mathrm{LEG}}(\tau; N_2, R_2, B_2, \Lambda_2)$*. Then there exists* N, R, B, Λ *such that* $C(\tau) = C_{\mathrm{LEG}}(\tau; N, R, B, \Lambda)$ *and the rank of* $C_{\mathrm{LEG}}(\tau; N, R, B, \Lambda)$ *is equal to the the rank of* $C_{\mathrm{LEG}}(\tau; N_1, R_1, B_1, \Lambda_1)$ *plus the rank of* $C_{\mathrm{LEG}}(\tau; N_2, R_2, B_2, \Lambda_2)$*.*

Proof. We can construct it directly:

- N can be constructed as the a direct sum, $N = N_1 \oplus N_2$, i.e.

$$N = \begin{pmatrix} N_1 & 0 \\ 0 & N_2 \end{pmatrix}$$

  Note that $\oplus$ is also sometimes used to indicate the Kronecker sum; in this supplement we will always use it to signify the direct sum.

- $R = R_1 \oplus R_2$, i.e.

$$R = \begin{pmatrix} R_1 & 0 \\ 0 & R_2 \end{pmatrix}$$

- $B = (B_1, B_2)$
- Take Λ to be the Cholesky decomposition of $\Lambda_1\Lambda_1^\top + \Lambda_2\Lambda_2^\top$

□

Note that in some cases the sum of two LEG kernels can be written as a LEG kernel whose rank is less than the combined rank of the two constituent LEG kernels. For example, let $C_{\mathrm{LEG}}(N, R, B, \Lambda)$ be a LEG kernel of rank ℓ. The obviously $C_{\mathrm{LEG}}(N, R, B, \Lambda) + C_{\mathrm{LEG}}(N, R, B, \Lambda)$ can be written as a LEG kernel of rank ℓ.

2.3 Spectrum of the PEG and LEG models

Here we study the spectrum of $C_{\mathrm{PEG}}(\tau; N, R)$. It is is straightforward to write down the spectrum of C in terms of the spectrum of the underlying matrix $G = NN^\top + R - R^\top$. For technical reasons we will here assume that $NN^\top$ is strictly positive definite. When $NN^\top$ is merely nonnegative definite, with some zero eigenvalues, it is no longer possible to represent the spectrum with a matrix-valued function M and things become a bit more complicated. Essentially the same ideas go through, but for simplicity we focus on the positive-definite case here.

Proposition 3. *Let $NN^\top$ be strictly positive definite. Then the spectrum of $C_{\mathrm{PEG}}(N,R)$ is given by*

$$M_{\mathrm{PEG}}(\omega;N,R) \triangleq \frac{1}{2\pi}\left(\left(\frac{G}{2}-\omega i I\right)^{-1}+\left(\frac{G^\top}{2}+\omega i I\right)^{-1}\right)$$

where $G = NN^\top + R - R^\top$. That is,

$$C_{\mathrm{PEG}}(\tau,N,R) = \int_{-\infty}^{\infty} e^{i\omega\tau} M_{\mathrm{PEG}}(\omega;N,R)d\omega$$

for all $\tau \geq 0$.

Proof. Since $NN^\top$ is strictly positive definite, the real parts of the eigenvalues of G are also strictly positive, and so $C(\tau)$ decays exponentially as $\tau \to 0$. It follows that $C(\tau)$ is Lebesgue integrable. We can therefore apply the Fourier Inversion formula to argue that the spectrum of C is given by the formula

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

This integral splits into two parts: positive and negative. Let's look at the positive half first.

$$\begin{aligned}\int_0^\infty e^{i\omega\tau}C(\tau)d\tau &= \int_0^\infty e^{\tau(i\omega I - G/2)}d\tau \\ &= -(i\omega I - G/2)^{-1} = (G/2 - i\omega I)^{-1}\end{aligned}$$

Here we have used the fact that the derivative of matrix exponentials behaves essentially the same as regular scalar exponentials. Together with the fundamental theorem of calculus, this allows us to get the integral in closed form. Now the negative half:

$$\begin{aligned}\int_{-\infty}^0 e^{i\omega\tau}C(\tau)d\tau &= \int_{-\infty}^0 e^{\tau(i\omega I + G^\top/2)}d\tau \\ &= (i\omega I + G^\top/2)^{-1}\end{aligned}$$

Adding the two halves together, we obtain our result. □

Now that we have understood the spectrum of PEG kernels, we turn to the spectrum of LEG kernels. If $\Lambda\Lambda^\top \neq 0$, the LEG kernel is discontinuous and the spectrum becomes technically involved. However, when $\Lambda = 0$ the spectrum will be continuous and is easy to write down:

$$M_{\mathrm{LEG}}(\omega;N,R,B,0) \triangleq BM_{\mathrm{PEG}}(\omega;N,R)B^T$$

It is straightforward to verify that this is indeed the spectrum i.e.

$$\int e^{-i\tau\omega}M_{\mathrm{LEG}}(\omega;N,R,B,0)d\omega = C_{\mathrm{LEG}}(\tau;N,R,B,0)$$

The expression for M_{LEG} suggests that the spectrum of any LEG process decays like an inverse polynomial in ω. This is a relatively slow rate of decay when compared with Radial Basis Function kernels (whose spectrum decays like $\exp(-\omega^2)$) and Rational Quadratic kernels (whose spectrum decays like $\exp(-|\omega|)$). This slow rate of decay in the spectrum allows LEG processes to have "rough" sample paths. This, in turn, permits statistically efficient smoothing even when the underlying function does not have infinitely-many derivatives [3].

2.4 Connections to Celerite and Spectral Mixture kernels

Here we investigate how LEG kernels are related to two model families already known in the literature: Spectral Mixture kernels and Celerite kernels. We'll start by defining these two families of kernels:

2.4.1 Spectral Mixture Kernels

Spectral Mixture (SM) kernels are a family of kernels for Gaussian Processes of the form $z : \mathbb{R}^n \to \mathbb{R}$, introduced in 2013 by Wilson and Adams [4]. Here we extend their idea to the kinds of processes we are interested in, namely $z : \mathbb{R} \to \mathbb{R}^n$. For such processes, we define Spectral Mixture (SM) kernels as follows:

Definition 1 (Spectral Mixture Kernels)**.** Let p denote a probability density on $\mathbb{R}$, let $b_1, b_2 \cdots b_\ell \in \mathbb{C}^n$, let $\mu \in \mathbb{R}^\ell$, and let $\gamma > 0$. The **Spectral Mixture kernel with ℓ components** parameterized by p, b, μ, γ is defined by

$$C_{\mathrm{SM}}(\tau; p, b, \mu, \gamma) \triangleq \sum_{k=1}^{\ell} \int e^{-i\omega\tau} b_k b_k^* \gamma p(\gamma(\omega - \mu_k)) d\omega.$$

We will say that a kernel $C_{\mathrm{SM}}(p, b, \mu, \gamma)$ is **based on** p, since it is designed by combining shifted scaled versions of p.

Note that SM kernels are, in general, complex-valued (we refer the reader back to Section 1.2 for the notation we use for such values, e.g. $b^*, \bar{b}, \Re(b), \Im(b)$). For Gaussian Processes we are generally interested in real-valued kernels. In this regards, the following definition and proposition may be helpful:

Definition 2. Let p a probability distribution on $\mathbb{R}$. Let $b \in \mathbb{C}^n$, $\mu \in \mathbb{R}$, and $\gamma > 0$. Let $\tilde{b}_1 = b/2, \tilde{b}_2 = \bar{b}/2, \tilde{\mu}_1 = \mu, \tilde{\mu}_2 = -\mu$. The two-component SM kernel $C_{\mathrm{SM}}(p, \tilde{b}, \tilde{\mu}, \gamma)$ is said to be the **Simple Real** Spectral Mixture kernel arising from p, b, μ, γ.

Proposition 4 (All real SM kernels are sums of Simple Real SM kernels)**.**

1. *Let $C_{\mathrm{SM}}(p, b, \mu, \gamma)$ denote an SM kernel with one component. Let $C_{\mathrm{SM}}(p, \tilde{b}, \tilde{\mu}, \gamma)$ denote the Simple Real SM kernel arising from p, b, μ, γ. Then*

$$C_{\mathrm{SM}}(p, \tilde{b}, \tilde{\mu}, \gamma) = \Re(C_{\mathrm{SM}}(p, b, \mu, \gamma))$$

2. *Let $C_{\mathrm{SM}}(p, b, \mu, \gamma)$ denote any real-valued SM kernel. Then it can be written as the sum of Simple Real SM kernels.*

Proof. The first point follows by observing that that $\Re(x) = (x + \bar{x})/2$.

For the second point. Since $C_{\mathrm{SM}}(p, b, \mu, \gamma)$ is real-valued, it follows that $C_{\mathrm{SM}}(p, b, \mu, \gamma) = \Re(C_{\mathrm{SM}}(p, b, \mu, \gamma))$. Note that $C_{\mathrm{SM}}(p, b, \mu, \gamma)$ is the sum of SM kernels with one component. We can apply the first point to each of these one-component kernels. This yields that $\Re(C_{\mathrm{SM}}(p, b, \mu, \gamma))$ must be the sum of Simple Real SM kernels. □

2.4.2 Celerite kernels

Celerite is a family of kernels for Gaussian Processes of the form $z : \mathbb{R} \to \mathbb{R}$, introduced in 2017 by Foreman-Mackey, Agol, Ambikasaran, and Angu [5].

Definition 3. Let $a, b, c, d \in \mathbb{R}^\ell$. The **Celerite kernel with ℓ components** parameterized by a, b, c, d is defined by

$$C_{\mathrm{CEL}}(\tau; a, b, c, d) = \sum_k a_k e^{-c_k\tau} \cos(d_k\tau) + b_k e^{-c_k\tau} \sin(d_k\tau)$$

Note that C_{CEL} is not necessarily positive definite. A **Celerite term** is a Celerite kernel with exactly one component, i.e. $\ell = 1$. As shown in the original paper, a Celerite term $C_{\mathrm{CEL}}(a, b, c, d)$ is positive definite if and only if $|bd| < ac$ and $a, c \geq 0$.

We conjecture that Celerite kernels can also be generalized to Gaussian Processes of the form $z : \mathbb{R} \to \mathbb{R}^n$ for $n > 1$. We leave this for future work.

2.4.3 Connections between the families

How are SM kernels, Celerite kernels, and LEG kernels related?

Lemma 2 (SM kernels, Celerite kernels, LEG kernels)**.**

1. *Every Cauchy-based real-valued SM kernel* $C_{\mathrm{SM}} : \mathbb{R} \to \mathbb{R}$ *can be understood as a Celerite kernel.*
2. *Every positive-definite Celerite term can be understood as a LEG kernel.*
3. *Every Cauchy-based real-valued SM kernel* $C_{\mathrm{SM}} : \mathbb{R} \to \mathbb{R}^{n \times n}$ *can be understood as a LEG kernel.*

Proof. We take each point separately.

1. In the original paper [5] it is shown that each Simple Real SM kernel based on the Cauchy distribution can be understood as a Celerite term. Proposition 4 thus yields that all Cauchy-based real-valued SM kernels can be understood as Celerite kernels.

2. Let $C_{\mathrm{CEL}}(a, b, c, d)$ denote a positive definite Celerite term. Let

$$\begin{aligned} N_1 &= \sqrt{2c - 2bd/a} \\ R_1 &= \sqrt{2c^2 + 4d^2 + 2b^2d^2/a^2} \\ N_2 &= \sqrt{c + bd/a} \end{aligned}$$

   The original Celerite paper shows that positive-definiteness implies $|bd| < ac$ and $a, c \geq 0$, thus $N_1, R_1, N_2 \in \mathbb{R}$. Let

$$N = \begin{pmatrix} N_1 & 0 \\ N_2 & N_2 \end{pmatrix} \qquad R = \begin{pmatrix} 0 & R_1 \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} \sqrt{a} & 1 \end{pmatrix}$$

   One can then use a symbolic algebra package (we used sympy) to prove that that the spectrum of $C_{\mathrm{LEG}}(\tau; N, R, B, 0)$ is the same as the spectrum of $C_{\mathrm{CEL}}(\tau; a, b, c, d)$. The formula for the spectrum of the Celerite kernel is given in the original paper and the formula for the spectrum of the LEG kernel follows from Proposition 3.

3. Applying Proposition 2 and 4, we see that it suffices to show that every Simple Real SM kernel based on a Cauchy distribution can be understood as a LEG kernel. Let $C_{\mathrm{SM}}(p, b, \mu, \gamma)$ denote a Simple Real SM kernel. Now define

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

   and let $\tilde{B} \in \mathbb{R}^{n \times 2}$ be given by $\tilde{B} = \left(\tilde{B}_1, \tilde{B}_2\right)$ such that $b = \tilde{B}_1 + i\tilde{B}_2$. Take $N = I\sqrt{2/\gamma}, R = \mu J$. One can then use a symbolic algebra package (we used sympy) to prove that that the spectrum of $C_{\mathrm{LEG}}(N, R, \tilde{B}, 0)$ is the same as the spectrum of $C_{\mathrm{SM}}(p, b, \mu, \gamma)$. The formula for the spectrum of the SM kernel is given by definition and the formula for the spectrum of the LEG kernel follows from Proposition 3.

□

This Lemma leads to an open problem. We now know that every positive-definite Celerite term can be understood as a LEG kernel. By Proposition 2 it follows that any sum of positive-definite Celerite terms can be understood as a LEG kernel. However, is it true that every positive-definite Celerite kernel can be understood as a LEG kernel? In general, there exist positive-definite Celerite kernels which are the sum of Celerite terms which are *not all nonnegative-definite*. The negativity of some of the kernels is cancelled out by the positivity in others so that the overall result is positive. Can LEG kernels represent these kinds of Celerite kernels? We leave this question for future work.

2.5 Flexibility of Spectral Mixture Kernels

Just as mixture models can approximate any distribution, it seems reasonable to hope that Spectral Mixture kernels could approximate any stationary kernel. Wilson and Adams already argued this point for SM kernels for processes $z : \mathbb{R}^n \to \mathbb{R}$ [4]. Here we generalize their flexibility result to processes of the form $z : \mathbb{R} \to \mathbb{R}^n$.

The key point is the following Theorem, a slight generalization of the usual results for kernel density estimation:

Theorem 1 (Total variation convergence for weighted kernel density estimation)**.** *Let K, p denote bounded densities on $\mathbb{R}^d$. Let $g : \mathbb{R} \to [-M, M]$. Let $\gamma_\ell = \ell^{1/2d}$. Let $\mu_1, \mu_2 \cdots \sim p$, independently. For each $\ell \in 1, 2, \cdots$, define*

$$h_\ell(\omega) = \frac{1}{\ell} \sum_{k=1}^{\ell} g(\mu_k) \gamma_\ell^d K(\gamma_\ell(\omega - \mu_k)).$$

Then

$$\mathbb{P}\left(\lim_{\ell \to \infty} \int |h_\ell(\omega) - p(\omega)g(\omega)| d\omega = 0 \right) = 1.$$

Proof. We largely imitate the proof of Devroye and Wagner [6], which handles the special case that $g(x) = 1$.

For almost any fixed ω, 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 δ. 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 δ, 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 γ_ℓ. Since K is a density we can always find δ so that this is arbitrarily small.

Therefore, for any fixed ε we can always find a δ 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 ℓ. In short, $|\bar{h}_\ell(\omega) - p(\omega)g(\omega))| \to 0$ for each ω.

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 ω, the sequence $h_1(\omega), h_2(\omega), \cdots$ converges almost surely to $p(\omega)g(\omega)$.

To complete the proof we must 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. Indeed, we have that

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

Since g is bounded, the law of large numbers gives us that this converges almost surely to $|p(\omega)g(\omega)|d\omega$. This allows us to extend our pointwise result to the desired $\mathscr{L}^1$ result via Lemma 3, below. □

Lemma 3 (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 π-almost-every x and π-almost-every ω, assume that $\lim_{\ell\to\infty} h_\ell(x,\omega), h_2(x,\omega) = h_\infty(x)$. For π-almost-every ω, assume that $\lim_{\ell\to\infty} \int |h_1(x,\omega)|dx = \int |h_\infty(x,\omega)|dx$, π-almost-surely. Then, for π-almost-every ω, we have that*

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

Proof. Apply Scheffe's lemma for each ω. This observation is generally credited to Glick [7]. □

Theorem 1 makes it straightforward to show that SM kernels (and, by extension LEG kernels) can be used to approximate any integrable continuous kernel:

Corollary 1 (Flexibility of Spectral Mixture kernels)**.** *Fix p, a bounded probability density on $\mathbb{R}^n$, $\varepsilon > 0$, and any Lebesgue-integrable continuous positive definite stationary kernel $\Sigma : \mathbb{R} \to \mathbb{R}^{n\times n}$. There exists a real valued kernel $C = C_{\mathrm{SM}}(p, b, \mu, \gamma)$ such that $\|C(\tau)z - \Sigma(\tau)z\| < \varepsilon\|z\|$ for every $\tau \in \mathbb{R}, z \in \mathbb{C}^n$.*

Proof. Apply Proposition 1 to argue that

$$\Sigma(\tau) = \int e^{i\tau\omega} M(\omega)d\omega$$

where $M(\tau)$ is Hermitian for each τ. Apply a unitary eigendecomposition to each M, i.e. write

$$\Sigma(\tau) = \sum_k^\ell \int e^{i\tau\omega} b_k(\omega) b_k^*(\omega)d\omega$$

Thus, applying Theorem 1, we can match each entry of the matrix-valued spectrum in an $\mathscr{L}^1$ sense with the corresponding entry of the spectrum of an SM kernel, i.e. we can find p, b, μ, γ to ensure that

$$\int |(M(\omega))_{jj'} - (M_{\mathrm{SM}}(\omega; p, b, \mu, \gamma))_{jj'}| d\omega$$

is arbitrarily small for each j, j'. It follows that we can ensure

$$\sup_z \frac{1}{\|z\|} \int \|M(\omega)z - M_{\mathrm{SM}}(\omega; p, b, \mu, \gamma)z\| d\omega$$

is arbitrarily small.

So far, we have showed that we can match the spectrum of any Lebesgue-integrable continuous positive definite stationary kernel with the spectrum of an SM kernel. Now we will use this fact to show that we can match the kernel itself. We have that

$$\begin{aligned}\sup_{z,\tau} \frac{1}{\|z\|} \|\Sigma(\tau)z - C_{\mathrm{SM}}(\tau)z\| &\leq \sup_{z,\tau} \frac{1}{\|z\|} \int \|e^{i\tau\omega}(M(\omega)z - M_{\mathrm{SM}}(\omega; p, b, \mu, \gamma)z)\| d\omega \\ &= \sup_z \frac{1}{\|z\|} \int \|M(\omega)z - M_{\mathrm{SM}}(\omega; p, b, \mu, \gamma)z\| d\omega\end{aligned}$$

But as we just described, we can always ensure that this last expression is as small as we like. Finally, note that the kernel generated in this fashion may not be perfectly real-valued; however, if the kernel is arbitrarily close to the correct kernel it will be arbitrarily close to real already; summing the resulting kernel with its complex conjugate yields a kernel which is close to the target and real-valued. □

Theorem 2 (Flexibility of LEG kernels)**.** *For every $\varepsilon > 0$ and every Lebesgue-integrable continuous positive definite stationary kernel $\Sigma : \mathbb{R} \to \mathbb{R}^{n \times n}$ there exists $C = C_{\mathrm{LEG}}(N, R, B, \Lambda)$ such that $\|C(\tau)z - \Sigma(\tau)z\| < \varepsilon\|z\|$ for every $\tau > 0, z \in \mathbb{C}^n$.*

Proof. Combine Corollary 1 and Lemma 2. □

3 Algorithms for LEG processes

3.1 The importance of block-tridiagonal matrices

There are a number of tasks related to the LEG model which we would like to be able to solve efficiently. It turns out that the computationally intensive part of all of these tasks involves working with block-tridiagonal matrices. Here we will look at some common tasks and see how this plays out.

Throughout what follows, we will assume

- $z \sim \mathrm{PEG}(N, R)$
- $t_1 \leq t_2 \leq \cdots \leq t_m$
- $\vec{x}_i \sim \mathcal{N}\left(Bz(t_i), \Lambda\Lambda^T\right)$, independently for each i
- $\vec{z} = (z(t_1) \cdots z(t_n))$

There is a slight subtlety that occurs when $t_i = t_{i+1}$ for some i. However, when this happens we can effectively reduce the problem to a case where the times are distinct by "combining" multiple observations into one. To keep the exposition clear, we will here assume that $t_1 < t_2 < t_3 \cdots t_m$, though the the leggps package can cope with the more general case.

We may be interested in...

- Computing the covariance of $\vec{z}$ and the inverse of that covariance. As shown by Lemma 1, the covariance of $\vec{z}$ can be expressed in terms of $\ell \times \ell$ blocks as follows:

$$\Sigma = \begin{pmatrix} I & e^{-\frac{1}{2}|t_1-t_2|G^T} & e^{-\frac{1}{2}|t_1-t_3|G^T} & \cdots \\ e^{-\frac{1}{2}|t_1-t_2|G} & I & e^{-\frac{1}{2}|t_1-t_2|G^T} & \\ e^{-\frac{1}{2}|t_1-t_3|G} & e^{-\frac{1}{2}|t_1-t_2|G^T} & I & \\ \vdots & & & \ddots \end{pmatrix}$$

One can readily verify that the the inverse is given by the block-tridiagonal matrix

$$\Sigma^{-1} = \begin{pmatrix} R_1 & O_1^T & 0 & \cdots \\ O_1 & R_2 & O_1^T & \\ 0 & O_2 & R_3 & \\ \vdots & & & \ddots \end{pmatrix}$$

where

$$d_i = \begin{cases} \infty & i = 0 \\ t_{i+1} - t_i & i \in \{1 \cdots m\} \\ \infty & i = m+1 \end{cases}$$

$$R_i = -(I - e^{-\frac{1}{2}d_i G^T} e^{-\frac{1}{2}d_i G})^{-1} e^{-\frac{1}{2}d_i G^T}$$

$$O_i = I + e^{-\frac{1}{2}d_{i-1}G}(I - e^{-\frac{1}{2}d_{i-1}G^T} e^{-\frac{1}{2}d_{i-1}G})^{-1} e^{-\frac{1}{2}d_{i-1}G^T} + e^{-\frac{1}{2}d_i G^T}(I - e^{-\frac{1}{2}d_i G} e^{-\frac{1}{2}d_i G^T})^{-1} e^{-\frac{1}{2}d_i G}$$

- Computing the likelihood, $\log p(\vec{x})$. Let

$$\tilde{B} = \bigoplus_{i=1}^{m} B$$

$$\tilde{\Lambda} = \bigoplus_{i=1}^{m} (\Lambda\Lambda^T)$$

Here by $\oplus$ we signify the direct sum (not the Kronecker sum). For example, $\tilde{B}$ is a block-diagonal matrix with m diagonal blocks, each of which is identically equal to B:

$$\tilde{B} = \begin{pmatrix} B & 0 & 0 & \cdots \\ 0 & B & 0 & \cdots \\ 0 & 0 & B & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Note that $\tilde{B}$ is not necessarily a square matrix, since B is not necessarily a square matrix. $\tilde{\Lambda} = \oplus(\Lambda\Lambda^\top)$ is constructed similarly, and it is always a square matrix because $\Lambda\Lambda^\top$ is a square matrix.

In terms of these objects, the covariance of $\vec{x}$ is given by

$$\mathrm{Cov}(\vec{x}) = \tilde{B}\Sigma\tilde{B}^T + \tilde{\Lambda}$$

And so the likelihood is given by

$$\log p(x) = -\frac{1}{2} x^T \left(\tilde{B}\Sigma\tilde{B}^T + \tilde{\Lambda}\right)^{-1} x - \frac{1}{2}\log\left|2\pi\left(\tilde{B}\Sigma\tilde{B}^T + \tilde{\Lambda}\right)\right|$$

where $|\cdot|$ here denotes the determinant. Let's take one term at a time:

– The Mahalanobis term. The Sherman-Morrison formula gives that

$$x^T\left(\tilde{B}\Sigma\tilde{B}^T+\tilde{\Lambda}\right)^{-1}x = x^T\left(\tilde{\Lambda}^{-1}-\tilde{\Lambda}^{-1}B^T\left(\Sigma^{-1}+B^T\tilde{\Lambda}^{-1}B\right)^{-1}B^T\tilde{\Lambda}^{-1}\right)x$$
$$= x^T\tilde{\Lambda}^{-1}x - x^T\tilde{\Lambda}^{-1}B^T\left(\Sigma^{-1}+B^T\tilde{\Lambda}^{-1}B\right)^{-1}B^T\tilde{\Lambda}^{-1}x$$

The hard part in computing this is solving the linear system

$$\left(\Sigma^{-1}+B^T\tilde{\Lambda}^{-1}B\right)^{-1}\left(B^T\tilde{\Lambda}^{-1}x\right) =?$$

Fortunately, the matrix $\Sigma^{-1}+B^T\tilde{\Lambda}^{-1}B$ is block-tridiagonal. So if we can compute solves with block-tridiagonal matrices this is not a problem.

– The determinant term. The Matrix Determinant Lemma gives that

$$\left|\tilde{B}\Sigma\tilde{B}^T+\tilde{\Lambda}\right| = \frac{\left|\tilde{\Lambda}\right|}{\left|\Sigma^{-1}\right|}\left|\Sigma^{-1}+B^T\tilde{\Lambda}^{-1}B\right|$$

The hard part here is computing the determinant of $\left|\Sigma^{-1}\right|$ and $\left|\Sigma^{-1}+B^T\tilde{\Lambda}^{-1}B\right|$. Again, these are block-tridiagonal matrices. So if we can do that we have no problem.

- In-sample posterior estimates. Here we are interested in computing $\mathbb{E}\left[\vec{z}|\vec{x}\right], \mathrm{Cov}\left(\vec{z}|\vec{x}\right)$. We calculate that

$$\mathbb{E}\left[\vec{z}|\vec{x}\right] = \left(\Sigma^{-1}+B^T\tilde{\Lambda}^{-1}B\right)^{-1}\left(B^T\tilde{\Lambda}^{-1}x\right)$$
$$\mathrm{Cov}\left(\vec{z}|\vec{x}\right) = \left(\Sigma^{-1}+B^T\tilde{\Lambda}^{-1}B\right)^{-1}$$

Thus to compute the first object we need to be able to compute solves with block-tridiagonal matrices. To compute the second object we need to be able to invert block-tridiagonal matrices. As we shall see, computing every entry in this inverse is intractable, but computing the diagonal and off-diagonal blocks can be done efficiently. This is sufficient to find the marginal in-sample posterior distributions for each $\vec{z}_k$.

- Out-of-sample posterior estimates. Here we are interested in computing $\mathbb{E}\left[\vec{z}(t)|\vec{x}\right], \mathrm{Cov}\left(\vec{z}(t)|\vec{x}\right)$ for arbitrary t. There are four different cases here:

– If $t = t_i$ for some i, then we should use the in-sample posterior estimates.

– If $t > t_m$, then we have a forecasting problem. The Markov structure gives that

$$p(z(t)|\vec{x}) = \int p(z(t)|\vec{z}_m)p(\vec{z}_m|\vec{x})d\vec{z}_m$$

The distribution of $p(z(t)|z(t_m))$ is found in closed-form by using the covariance formulas for the PEG process. Thus we can compute the marginal distribution of $z(t)|\vec{x}$ as long as we know the marginal distribution of $\vec{z}_m|\vec{x}$. This requires knowing the in-sample posterior mean and final diagonal block of the in-sample posterior covariance.

– If $t_i < t < t_{i+1}$ we have an interpolation problem. In this case the Markov structure gives that

$$p(z(t)|\vec{x}) = \iint p(z(t)|\vec{z}_i,\vec{z}_{i+1})p(\vec{z}_i,\vec{z}_{i+1}|\vec{x})d\vec{z}_i d\vec{z}_{i+1}$$

The distribution of $p(z(t)|\vec{z}_i,\vec{z}_{i+1})$ is found in closed-form by using the covariance formulas for the PEG process. Thus we can compute the marginal distribution of $z(t)|\vec{x}$ as long as we know the marginal distribution of $\vec{z}_i,\vec{z}_{i+1}|\vec{x}$. This requires knowing the in-sample posterior mean and the diagonal and off-diagonal blocks of the in-sample posterior covariance.

- If $t < t_1$, we have a backwards forecasting problem. This is essentially the same as the forward forecasting problem, but the PEG process covariance formulas are slightly different.

- Smoothing/forecasting. Here we are interested in a few related things, all of which follow immediately from the out-of-sample posterior estimates.
  - The posterior predictive means:

$$\mathbb{E}\left[B\vec{z}(t)|\vec{x}\right] = B\mathbb{E}\left[\vec{z}(t)|\vec{x}\right]$$

  - The posterior predictive uncertainty:

$$B^T \mathrm{Cov}\left[z(t)|\vec{x}\right] B^T$$

    This represents the uncertainty we have about the posterior predictive means.
  - The posterior predictive variances:

$$B^T \mathrm{Cov}\left[z(t)|\vec{x}\right] B^T + \Lambda\Lambda^T$$

    This is the conditional variance of a new sample taken at position t.

In conclusion, we see that all of the things we need to do can be achieved efficiently as long as we can...

- Solve $J^{-1}x$ when J is block-tridiagonal.
- Compute the determinant $|J|$ when J is block-tridiagonal.
- Compute the diagonal and off-diagonal blocks of the inverse of J^{-1} when J is block-tridiagonal.

3.2 Computations with block-tridiagonal matrices using Cyclic Reduction (CR)

Above we saw that most common tasks with LEG processes amount to computations with block-tridiagonal matrices. Cyclic Reduction (CR) is a classic technique for efficient parallel computations with such matrices [8]. These techniques appear to be relatively unknown in the Machine Learning literature, and the original text is a bit dense. We here describe the algorithms involved in 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 start by decomposing J using what is called a "Cyclic Reduction." This gives us a convenient representation of J which is easy to work with. Here's how it works.

Definition 4 (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

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

What are these P, Q matrices doing? They are simply selecting 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 5. *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 as we go. 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.

3.2.1 CyclicReduction

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

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

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

3 **else**

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

5 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;

6 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);

7 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;

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

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

10 **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 blazing fast algorithms exist for achieving these goals on multiple CPU cores. There also exist fast algorithms for achieving these on the GPU. Unfortunately, TensorFlow2 is quite slow at computing many small Cholesky decompositions on the GPU. Their code uses the default CUDA libraries; as of January 2020 it is much slower than the corresponding pytorch code. NVidia is currently in the process of trying to develop a better batch-CUDA platform which will make these algorithms quite fast. For the moment we recommend running the leggps package (see below) on CPU devices.

3.2.2 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.

3.2.3 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.

3.2.4 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.

3.2.5 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.

3.2.6 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 $\leftarrow$ 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 ΣUD^{-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 Σ. 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

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

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

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

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. Luckily, 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.

3.3 Learning

We are now positioned to design an algorithm to learn a LEG model from data via maximum likelihood. We need three ingredients. We describe them here:

- The ability to efficiently compute the likelihood and the gradient of the likelihood. This is facilitated by the algorithms above.
- An optimization algorithm that uses those gradients. We use BFGS, as implemented by scipy.optimize.
- Initial conditions. In the leggps package (described below) the user can provide their own initial conditions. If none are provided, we use the following simple initialization which seemed to work in practice for all the problems we looked at:
  - $N = I$. This assumes that the smoothness lengthscale of the time series is roughly on the order of one unit. If this is not the case, one can either scale the input times or provide a correspondingly different N. In general, the smoothness timescale is inversely proportional to $NN^\top$.
  - Each entry of R is sampled from $\mathcal{N}(0, \sqrt{.2})$. This assumes that the oscillations of the timeseries are roughly at a frequency of .2 oscillations per unit of time (i.e. one oscillation for every five units of time).
  - $\Lambda = .1I$. This assumes that the independent noise has a standard deviation of roughly .1.

  Even when the true smoothness lengthscale was hundreds of times different from unity, we found that BFGS was able to detect this and adjust quickly to a better regime.

This optimization problem does not appear to be convex. There were cases where we found that multiple restarts (each initialized with the same random initialization routine, described above) resulted in superior fits. The user may wish to try this if the result is unsatisfactory the first time.

When the observations are regularly spaced, a learned AR model could also be used to initialize the parameters of the LEG model, though we have not worked out exactly how this would be done. There is also a literature on using Hankel matrices to guess the dynamics of state-space models like the LEG model. In the future we hope to explore new methods for initializing. If you have ideas, don't hesitate to raise an issue on the GitHub repo.

4 The leggps python package

The leggps python package can be found at `https://github.com/jacksonloper/leg-gps`. It provides functionality for working with LEG models and also exposes some of the cyclic reduction algorithms.

4.1 Working with LEG processes

leggps follows a few conventions:

- All vectors and matrices are represented as numpy objects.
- A LEG model is represented by a matrix-valued dictionary with keys for N, R, B, and Lambda.
- m observations from a LEG model comprise a sequence of times and a sequence of values.

- The times $t_1 \leq t_2 \leq t_3 \cdots t_m$ are represented as a vector of length m. We assume these times are sorted.
- The corresponding observations $\vec{x}$ are represented as a matrix. If the LEG process has the form $z : \mathbb{R} \to \mathbb{R}^n$, this matrix will have dimensions $m \times n$. To avoid mistakes, even if the LEG process has the form $z : \mathbb{R} \to \mathbb{R}^n$, we will expect a matrix with dimensions $m \times 1$.

We assume that the observations of x arise from a LEG model, i.e. $z \sim \mathrm{PEG}(N, R)$ and

$$\vec{x}_i \sim \mathcal{N}(Bz(t_i), \Lambda\Lambda^\top)$$

Note that if $t_i = t_i + 1$ we assume that $\vec{x}_i, \vec{x}_{i+1}$ are sampled independently.

- Sometimes we will wish to represent several independent samples from a LEG model. For example, perhaps each independent sample represents a neural recording on a different day. We will represent this using a list of time-vectors and a corresponding list of observation matrices. That is, we will have that
  - ts[i][j] indicates the time of the jth observation in the ith independent sample
  - xs[i][j,k] indicates the kth dimension of the jth observation in the ith independent sample

leggps provide the following functions:

leggps.C_LEG Calculates the covariance of a LEG process for various values of τ. Inputs:

taus A vector of m times, $\tau_1 \leq \tau_2 \leq \tau_3 \cdots$

N,R,B,Lambda Model parameters

Outputs an $m \times n \times n$ tensor. The ith element of this indicates indicates $C_{\mathrm{LEG}}(\tau_i; N, R, B, \Lambda)$.

leggps.leg_log_likelihood Calculates the log likelihood of an observation under a LEG model.

Inputs:

ts A vector of m times, $t_1 \leq t_2 \leq t_3 \cdots t_m$

xs A matrix of observations at those times, i.e. $\vec{x} = x(t_1), x(t_2), \cdots$.

N,R,B,Lambda Model parameters

Outputs the log likelihood.

leggps.leg_log_likelihood_tensorflow Essentially the same as the function above, but inputs and outputs are assumed to be TensorFlow2 objects. This may be helpful if you would like to efficiently calculate gradients, write your own optimization routines, or use the LEG family as a building block in a larger model (e.g. one can put a prior on the parameters and use the gradient together with Hamiltonian Monte-Carlo to sample from the posterior on the parameters N, R, B, Λ). To make this fast, we recommend enclosing your code with a tf.function(autograph=False) decorator, compiling the operations to a graph which can be run quite efficiently.

Inputs:

ts A vector of $\tilde{m}$ times, $t_0 < t_1 < t_2 \cdots t_{\tilde{m}-1}$. Note that these times *must* be distinct (compare with the leg_log_likelihood which allows non-distinct values).

xs A matrix of m observations, each of which was taken at one of those times. In particular, $\vec{x} = x(t_{\texttt{idxs}_0}), x(t_{\texttt{idxs}_1}), \cdots$, where...

idxs is a vector of length m indicating which observations came from which times. Each entry of idxs should be an integer in the set $\{0, 1, \cdots \tilde{m} - 1\}$.

N,R,B,Lambda Model parameters

This function does enable computation with multiple observations taken at the same time (and assumes they are independently sampled) – but the user must ensure that the vector of times contains no duplicates and the user must figure out which observations correspond to what entry of that time-vector. Compare with the function leg_log_likelihood: the non-TensorFlow2 function isn't designed for speed, so it does this deduplication automatically at every invocation.

Outputs the log likelihood.

leggps.posterior_predictive Indicates interpolations/forecasts, i.e $BE[z(t)|\vec{x}]$ and $B\text{Cov}(z(t)|\vec{x})B^\top$ for various values of t.

Inputs:

ts A vector of m times, $t_1 \le t_2 \le t_3 \cdots t_m$

x A matrix of observations at those times, i.e. $\vec{x} = x(t_1), x(t_2), \cdots$.

targets A vector of times at which to evaluate the interpolations/forecasts, based on the data $\vec{t}, \vec{x}$.

N,R,B,Lambda Model parameters

Outputs a tuple with two elements:

1. Means, a matrix indicating $BE[z(\texttt{targets}_i)|\vec{x}]$
2. Variances, n $m \times n \times n$ tensor. The ith element of this indicates $B\text{Cov}(z(\texttt{targets}_i)|\vec{x})B^\top$

leggps.posterior Indicates posterior moments, i.e $E[z(t)|\vec{x}]$ and $\text{Cov}(z(t)|\vec{x})$ for various values of t.

Inputs:

ts A vector of m times, $t_1 \le t_2 \le t_3 \cdots t_m$

x A matrix of observations at those times, i.e. $\vec{x} = x(t_1), x(t_2), \cdots$.

targets A vector of times at which to evaluate the interpolations/forecasts, based on the data $\vec{t}, \vec{x}$.

N,R,B,Lambda Model parameters

Outputs a tuple with two elements:

1. Means, a matrix indicating $E[z(\texttt{targets}_i)|\vec{x}]$
2. Variances, n $m \times n \times n$ tensor. The ith element of this indicates $\text{Cov}(z(\texttt{targets}_i)|\vec{x})$

leggps.fit uses a collection of independent samples to learn a LEG model. For each sample i we assume that

$$t_{i1} \le t_{i2} \le t_{i2} \cdots \le t_{i,m_i}$$
$$z_i \sim \text{PEG}(N, R) \qquad x_i(t)|z \sim \mathcal{N}(Bz_i(t), \Lambda\Lambda^\top)$$
$$\vec{x}_i = (x_i(t_{i1}), x_i(t_{i2}), x_i(t_{i3}), \cdots)$$

If the process is of the form $x : \mathbb{R} \to \mathbb{R}^n$ we expect each $\vec{x}_i$ to be an $m_i \times n$ matrix. Even if x is a process $x : \mathbb{R} \to \mathbb{R}^1$, we expect each $\vec{x}_i$ to be a $m_i \times 1$ matrix.

We attempt to maximize the likelihood of this data with respect to N, R, B, Λ.

Inputs:

ts a list of vectors of timepoints. The ith entry in this list is itself a vector of times, $t_{i1} \le t_{i2} \le t_{i2} \cdots$.

xs a list of observation matrices. The ith entry in this list corresponds to the matrix $\vec{x}_i \in \mathbb{R}^{m_i \times n}$.

ell the rank of the LEG model to be learned

N,R,B,Lambda initial conditions (otherwise will be randomly initialized)

maxiter=200 maximum number of iterations of BFGS to use

use_tqdm_notebook=False if True, uses tqdm.notebook to display training progress

diag_Lambda=False if True, enforces that Lambda is a diagonal matrix (can be essential if n is large)

Output is a dictionary with many keys. Perhaps the most important keys is "params," which corresponds to the learned parameters, but there are many others which give useful information about the optimization process used to find those parameters:

params A dictionary indicating the learned parameters. It contains four elements: N, R, B, and Lambda, corresponding to N, R, B, Λ.

fun The nats at completion of the optimization

jac The gradient at completion of the optimization

hess_inv The estimate of the inverse of the hessian at completion of the optimization

nfev The number of times the likelihood was evaluated

njev The number of times the gradient of the likelihood was evaluated

status A status code (see scipy.optimize.minimize for details)

success Whether we were able to find a local optimum (up to machine precision). In practice, this usually is not true; nonetheless the learned parameter values perform well.

message A message indicating under what conditions the optimization terminated

nit The number of iterations used by the optimization algorithm

losses A list of the nats discovered along the optimization process. Note that the final loss may not be the same as the nats for the learned params – BFGS always picks the best parameter values that it found, which may not be the last parameter values it looked at.

4.2 Working with Cyclic Reductions

The leggps package also exposes an API for cyclic reduction algorithms.

This API is more low-level. All inputs are assumed to be TensorFlow2 objects and the outputs are likewise TensorFlow2 objects. Most of the functions in this package should be run on the CPU not the GPU, because the CUDA implementations for Cholesky decomposition and triangular_solve are currently still quite bad for handling many decompositions at once (as of this writing, 2020). The m5 line of machines in the AWS platform is fairly cost-effective for running these functions on the CPU.

This package follows the convention that block-tridiagonal matrices are represented as (Rs,Os) where Rs indicates the block diagonal components and Os indicates the lower off-diagonal blocks. Thus Rs will be an $m \times \ell \times \ell$ tensor and Os will be an $m-1 \times \ell \times \ell$ tensor. We assume all the blocks are the same size.

leggps provides the following functions:

leggps.cr.decompose(Rs,Os) Let J denote the block-tridiagonal matrix represented by Rs,Os. This function returns an opaque representation of the CR decomposition of the block-tridiagonal matrix. This can be used in other functions.

leggps.cr.mahal(decomp,x) Let J denote the block-tridiagonal matrix whose CR decomposition is given by decomp. Evaluates $x^T J^{-1} x$.

leggps.cr.det(decomp) Let J denote the block-tridiagonal matrix whose CR decomposition is given by decomp. Evaluates the log determinant of J.

leggps.cr.mahal_and_det(Rs,Os,x) Let J denote the block-tridiagonal matrix represented by Rs,Os. Uses CR algorithms to to evaluate $x^T J^{-1} x$ and compute the log determinant of J. This function may require half as much RAM than using the functions above – it never stores the entire decomposition at once.

leggps.cr.solve(decomp,y) Let J denote the block-tridiagonal matrix whose CR decomposition is given by decomp. Returns $J^{-1}x$.

leggps.cr.sample(decomp) Let J denote the block-tridiagonal matrix whose CR decomposition is given by decomp. Samples from $\mathcal{N}(0, J^{-1})$.

leggps.cr.inverse_blocks(decomp) Let J denote the block-tridiagonal matrix whose CR decomposition is given by decomp. Returns the diagonal and off-diagonal blocks of J^{-1}.

5 Extensions

There are a number of ways this package could be extended if there was sufficient interest. Raise an issue on the GitHub repo if these or other extensions would be important to your work.

Some extensions we have considered:

- We assume each observation is always fully observed, i.e. if we observe $x(t_i) \in \mathbb{R}^n$ then we observe all n numbers. This restriction could be lifted.
- The CR algorithms assume all blocks are the same size. If you need the blocks need to be of different sizes the TensorFlow2 raggedtensor API could conceivably be used to lift this limitation.
- We assume the noise variance, Λ, is the same for each observation. It would be fairly straightforward to lift this restriction.
- The observations do not need to lie along a line – in general, they could lie along any one-dimensional tree-structured topology.
- The model does not need to be stationary. If there are specific nonstationarities you would like to capture, the authors would be interested in discussing them with you and figuring out which (of many possible) options would be most useful in incorporating nonstationarity.
- We would like to think about better initialization strategies. If you are having difficulty initializing the LEG model for a scientific problem, the authors would be interested in discussing your problem and thinking about what might work.

There are also two extensions we have considered in some depth. We detail these below.

5.1 Multi-dimensional GPs

Let $z: \mathbb{R}^d \to \mathbb{R}^n$ denote a Gaussian Process with covariance kernel

$$\Sigma: \mathbb{R}^d \to \mathbb{R}^{n \times n}$$

How can we use LEG kernels could be used to model Σ? One technique to build one-dimensional models into multi-dimensional models is by using Kronecker products [9]. Here we generalize this to processes with $n > 1$.

Definition 5. For each $k \in 1 \cdots d$, let $C_k : \mathbb{R} \to \mathbb{R}^{\ell \times \ell}$ denote integrable continuous kernels. If $C : \mathbb{R}^d \to \mathbb{R}^{n \times n}$ is given by

$$C_{ij}(\tau) = \prod_k C_{kij}(\tau_k)$$

then we will say C is the **Kronecker-Hadamard product** of $C_1, C_2 \cdots C_d$, and write $C = \circledast_{k=1}^d C_k$.

Proposition 6. *Let $C_1, C_2 \cdots C_d$ denote integrable continuous positive-definite kernels. Then $C = \circledast_k^d C_k$ is also positive definite.*

Proof. Let M_k denote the spectrum of C_k. Recall that M_k is a positive-definite-matrix-valued function. Let $M_{ij}(\omega) = \prod_k M_{kij}(\omega)$. Observe that M is the spectrum of C. On the other hand, the Schur product theorem shows that $M(\omega)$ is a positive definite matrix. Bochner's theorem then yields that C is positive definite. □

We can combine PEG kernels via these Kronecker-Hadamard products, leading to a multidimensional extension to LEG models.

Definition 6. Fix ℓ, ζ. For each $r \in \{1 \cdots \zeta\}, k \in \{1 \cdots d\}$, let N_{rk}, R_{rk} be $\ell \times \ell$ matrices. Consider the kernel defined by

$$C_{\mathrm{KPEG}}(N, R, B, \Lambda) \triangleq \sum_{r=1}^{\zeta} \circledast_{k=1}^d C_{\mathrm{PEG}}(N_{rk}, R_{rk})$$

We will call this the Kroneckered Purely Exponentially Generated (KPEG) kernel. If z is a zero-mean Gaussian Process whose covariance is a KPEG kernel we will say it is a KPEG process. Furthermore, let $B \in \mathbb{R}^{n \times \ell}, \Lambda \in \mathbb{R}^{n \times n}$. If z is a KPEG process and $x(\tau)|z \sim \mathcal{N}(Bz(\tau), \Lambda\Lambda^\top)$ we will say x is a KLEG process. We will call the covariance kernel of x a KLEG kernel, denoted $C_{\mathrm{KLEG}}(N, R, B, \Lambda)$.

Theorem 3. *Let Σ any positive-definite integrable continuous kernel, and fix $\varepsilon > 0$. There exists a KLEG kernel such that $\|C(\tau)z - C_{\mathrm{KLEG}}(\tau)z\| < \varepsilon z$.*

Proof. The proof is essentially the same as that of Theorem 2. Note the spectrum of a C_{KLEG} kernel can be understood as a mixture of matrix-valued densities on $\mathbb{R}^d$, and the N, R parameters can be used to arbitrarily shift and scale these spectral densities. Applying Theorem 1, it follows that we can match every element of any spectrum arbitrarily well in an integrated-absolute-value-sense using KPEG kernels. It follows that we can match any integrable continuous positive-definite kernel in a uniform sense. □

We can compute efficiently with such kernels if the observations are taken on a d-dimensional grid. For example, GPyTorch offers algorithms for Gaussian processes where the runtime is limited only by the speed with which one can multiply by the covariance matrix [10]. If we have observations from a KPEG model on a grid, this can be done efficiently:

Theorem 4. *Let $\Omega = \prod_k^d \{\tau_{k1}, \tau_{k2} \cdots \tau_{km}\} \subset \mathbb{R}^d$ denote a grid. Let x denote a KLEG process and let $\vec{x}$ denote observations of x on this grid. Let Σ denote the covariance matrix of $\vec{x}$ under the KLEG model. For any y the time required to compute Σy scales like m^d. In particular, in the limiting case where we have an observation at each grid-point, we have m^d observations and the computation scales linearly in the number of observations.*

Proof. It suffices to show that we can perform matrix-multiplications in m^d time for matrices which are the Kronecker-Hadamard product of d matrices when the inverses of those matrices are block-tridiagonal with m blocks of size ℓ. It suffices to show for the case $d = 2$ and apply induction.

Let C, D denote block-tridiagonal matrices. We will index them by $C_{i,j}(\tau_{1s}, \tau_{1s'})$ and $D_{i,j}(\tau_{2u}, \tau_{2u'})$. Because D^{-1} is block-tridiagonal, we can compute Dy in linear time. That is, we can compute

$$(Dy)_{i,u} = \sum_{j=1}^{\ell} \sum_{u'=1}^{m} D_{i,j}(\tau_{2u}, \tau_{2u'}) y_j(\tau_{2u'})$$

in $O(m)$ steps. It follows we can also compute

$$\xi_{i,j,u} = \sum_{u'} D_{i,j}(\tau_{2u}, \tau_{2u'}) y_j(\tau_{2u'})$$

in $O(m)$ steps (for each j, define $\tilde{y}$ by $\tilde{y}_{j'} = \delta_{j,j'} y_{j'}$, then $\xi_{i,j,u} = D\tilde{y}$).

We are interested in the Kronecker, i.e.

$$F_{i,j}(\tau_{1s}, \tau_{2u}, \tau_{1s'}, \tau_{2u'}) = C_{i,j}(\tau_{1s}, \tau_{1s'}) D_{i,j}(\tau_{2u}, \tau_{2u'})$$

Given an observation y we need to compute

$$\begin{aligned}(Fy)_i(\tau_{1s}, \tau_{2u}) &= \sum_{j,s',u'} F_{i,j}(\tau_{1s}, \tau_{2u}, \tau_{1s'}, \tau_{2u'}) y_j(\tau_{1s'}, \tau_{2u'}) \\ &= \sum_{j,s'} C_{i,j}(\tau_{1s}, \tau_{1s'}) \underbrace{\sum_{u'} D_{i,j}(\tau_{2u}, \tau_{2u'}) y_j(\tau_{1s'}, \tau_{2u'})}_{\triangleq \xi_{ijus'}}\end{aligned}$$

As discussed earlier, we can compute $\xi_{ijus'}$ independently for each s' in $O(m)$ time; the total computation time will be $O(m^2)$. Once this is computed, we can compute $(Fy)(\cdot, \tau_{2u})$ in $O(m)$ time for each u – an overall cost of $O(m^2)$. The result is that the total computation requires $O(m^2)$ operations, as desired.

□

Combining the previous two propositions, we see that arbitrarily accurate linear-time inference is possible for any Gaussian Process as long as we observe the process on a multidimensional grid. Note that this grid does not need to be regularly spaced. Moreover, it is straightforward to see that we do not need to have observations from every point in the grid. However, note that the computational cost will scale with the number of gridpoints (not the number of observations). In particular, if our observations occur on a very small proportion of the gridpoints, then different methods may be required.

5.2 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 [11, 12, 13, 14]. 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 [15, 16, 17, 18, 19, 20].

Here we sketch one way this can be achieved.

Definition 7. Let $p(x; \theta, \gamma)$ denote a family of densities indexed by $\theta \in \mathbb{R}^n$ and additional hyperparameters γ. Let $B \in \mathbb{R}^{n \times \ell}$ and $z \sim \mathrm{PEG}(N, R)$. Let $x(t)|z \sim p(Bz(t))$, independently for each t. Then we will say that x is the **Non-Gaussian Latent Exponentially Generated** (NGLEG) model parameterized by p, N, R, B.

How can we learn N, R, B, θ, γ from data? Let us say we have $t_1 < t_2 \cdots t_m$ and we have observed $\vec{x} = (x(t_1) \cdots x(t_m)$ from a NGLEG model. We adopt a Variational Inference point of view. Let $\vec{z} = (z(t_1), z(t_2), \cdots z(t_m))$. Let $\Sigma(N, R)$ denote the prior covariance of $\vec{z}$ when $z \sim \mathrm{PEG}(N, R)$. We posit a family of possible posterior distributions for $\vec{z}$, namely

$$\vec{z} \sim q(z; \mu, J) \triangleq \mathcal{N}(z; \mu, J^{-1})$$

where J is a block-tridiagonal matrix. We then seek to maximize a lower bound on the likelihood of the observations, namely

$$\begin{aligned}\mathcal{L}(N, R, B, \gamma, J, \mu) &= \mathcal{L}_{\text{data}}(B, \gamma, J, \mu) + \mathcal{L}_{\text{KL}}(N, R, J, \mu) \\ \mathcal{L}_{\text{data}}(B, \gamma, J, \mu) &= \sum_i \mathbb{E}_{\vec{z}\sim q}\left[\log p(\vec{x}_i; B\vec{z}_i, \gamma)\right] \\ \mathcal{L}_{\text{KL}}(N, R, J, \mu) &= \frac{1}{2}\mathbb{E}_{\vec{z}\sim q}\left[-\vec{z}^\top \Sigma(N, R)^{-1}\vec{z} - \log|\Sigma(N, R)||J| + (\vec{z}-\mu)^\top J(\vec{z}-\mu)\right]\end{aligned}$$

We refer the reader to [21] for a proof that this is indeed a lower-bound on the likelihood of $\vec{x}$. Note that $\mathcal{L}_{\text{KL}}$ and its gradients can be computed in $O(m)$ time using Cyclic Reduction algorithms. For the data term we can either use Monte-Carlo samples from $\vec{z} \sim q$ or reduce the expectation to something which can be computed in terms of the mean and variance of $\vec{z}$. The Laplace method is an example of the latter approach, approximating the log likelihoods by taking a second order taylor expansion of $z_i \mapsto \log p(\vec{x}_i; B\vec{z}_i, \gamma)$ around the μ_i. The Polya-Gamma trick is another example of this method which works for Bernoulli and Negative Binomial likelihoods; this trick yields a lower-bound which can be computed in terms of the mean and variance of $\vec{z}_i$ [18]. Regardless of which approach we use for the data term, we ultimately obtain approximate gradients of $\mathcal{L}$ with respect to N, R, B, γ, J, μ and use these gradients to optimize the parameters.

References

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

[2] 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.

[3] Aad van der Vaart and Harry van Zanten. Information rates of nonparametric gaussian process methods. *Journal of Machine Learning Research*, 12(Jun):2095–2119, 2011.

[4] 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.

[5] 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.

[6] LP Devroye and TJ Wagner. The l1 convergence of kernel density estimates. *The Annals of Statistics*, pages 1136–1139, 1979.

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

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

[9] Theodoros Tsiligkaridis and Alfred O Hero. Covariance estimation in high dimensions via kronecker product expansions. *IEEE Transactions on Signal Processing*, 61(21):5347–5360, 2013.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

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

[18] 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.

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

[20] 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.

[21] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.