Fast hierarchical Gaussian processes

Seth Flaxman CMU
Andrew Gelman Columbia
Daniel Neill CMU
Alex Smola CMU
Aki Vehtari Aalto University

Andrew Gordon Wilson CMU

Abstract

While the framework of Gaussian process priors for functions is very flexible and has a number of advantages, its use within a fully Bayesian hierarchical modeling framework has been limited due to computational constraints. Most often, simple models are fit, with hyperparameters learned by maximum likelihood. But this approach understates the posterior uncertainty in inference. We consider priors over kernel hyperparameters, thus inducing a very flexible Bayesian hierarchical modeling framework in which we perform inference using MCMC not just for the posterior function but also for the posterior over the kernel hyperparameters. We address the central challenge of computational efficiency with MCMC by exploiting separable structure in the covariance matrix corresponding to the kernel, yielding significant gains in time and memory efficiency. Our method can be conveniently implemented in a probabilistic programming language (Stan), is widely applicable to any setting involving structured kernels, and immediately enables a number of useful features, including kernel learning through novel prior specifications, learning nonparametric priors over categorical variables, clustering through a factor analysis approach, and missing observations. We demonstrate our methods on real and synthetic spatiotemporal datasets.

1 Introduction

Gaussian process modeling has a long history in statistics and machine learning [21, 33, 20, 22]. The central modeling choice with GPs is the specification of a kernel. As revealed by the extensive spatial statistics literature on this topic (where a stationary kernel is presented as the mathematically equivalent "covariogram") this is by no means a trivial problem, and it can be very hard to estimate a kernel from data, even with scientific knowledge. For this reason, we stress the importance of placing prior distributions on kernel hyperparameters. This stands in contrast to most applications of GPs, where kernel hyperparameters are learned using a maximum likelihood approach.

We address the computational challenges that stand in the way of a fully Bayesian approach to Gaussian process modeling. Our approach works in the case of separable kernels and grid-structured inputs, which together induce structure in the design matrix. Our approach is applicable to any Markov Chain Monte Carlo (MCMC) scheme for Bayesian inference, and we demonstrate its use within Hamiltonian Monte Carlo as implemented in the probabilistic programming language Stan. Our implementation is much faster and uses much less memory than standard approaches, thus opening up many new avenues for scalable Bayesian modeling on orders of magnitude larger datasets (see Figure 1). By placing priors on kernel hyperparameters, our model becomes more general than standard GP regression or classification. To demonstrate, we show how to efficiently learn structured kernels (e.g. those with a multiplicative structure) and non-parametric priors over categorical variables. We propose a clustering through factor analysis method, address the limitation of separable structure, show how to automatically handle missing observations (i.e. for incomplete grids), and extend our methods for non-Gaussian observation models.

Efficiency gains from structured covariance functions within GP models have been exploited previously with MCMC (e.g. [8]). There is much recent work on approximate methods for Gaussian processes and kernels (e.g. [26, 16, 14, 36]). Our work builds on previous work exploiting structured kernels and especially Kronecker methods for GP learning and inference (e.g. [29, 27, 11, 23, 35, 13, 9]).

In Section 2 we provide background on MCMC, GPs, and Kronecker inference. In Section 3 we develop our scalable learning and inference methods. In Section 4 we demonstrate the novel modeling approaches that our methods enable. In Section 5 we compare our model's performance on synthetic data to standard implementations of elliptical slice sampling and HMC, and we demonstrate our novel modeling approaches on a real dataset. Implementations in Stan are provided in the Appendix.

2 Background

2.1 Markov Chain Monte Carlo sampling

Markov Chain Monte Carlo sampling schemes are methods for numerical inference in Bayesian models in which random samples are iteratively generated, where the limiting distribution of the samples is the posterior over the parameters in the model. For non-conjugate models, MCMC is the default inference method. In the hierarchical GP models we consider, with priors over kernel hyperparameters, the posterior is not a Gaussian process, which is why we use MCMC. A critical subroutine, executed each time a new draw is generated, is the evaluation of the log of the probability density of the posterior at the current values of the parameters, which can be very costly in a GP model as described below.

Another reason that MCMC inference for GP models is challenging is that the parameters in Gaussian process models are tightly correlated, so off-the-shelf methods like Metropolis-Hastings and Gibbs sampling, which are known to have slow convergence in the case of correlated parameters, have not proved effective[1]. Early work by Neal focused on Hamiltonian Monte Carlo (HMC) [20], a method drawn from the physics literature that uses gradient information to make sampling more efficient. More recent work has focused on variants of slice sampling, especially elliptical slice sampling [1, 19, 18] which provides a variant on Metropolis-Hastings without tuning parameters by adaptively selecting the step size. Our methods could be used in either of these schemes; to demonstrate the modeling advantages that our approach enables, we implement them using a probabilistic programming language (Stan), which uses HMC.

2.2 Gaussian processes

Given observations $(X, Y) = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, let $k_{\boldsymbol{\theta}}(\cdot, \cdot)$ be a kernel with hyperparameters $\boldsymbol{\theta}$ and corresponding covariance matrix $K_{\boldsymbol{\theta}}$. Placing a prior on $\boldsymbol{\theta}$, the hierarchical specification is:

$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta}) \tag{1}$$

$$\boldsymbol{f} \mid X, \boldsymbol{\theta} \sim \mathcal{N}(\mu, K_{\boldsymbol{\theta}}) \tag{2}$$

$$y_i \mid f(x_i) \sim \mathcal{N}(f(x_i), \sigma^2), \;\; \forall i \tag{3}$$

The computational difficulty of an MCMC scheme for this model arises from Eq. (2) which requires the computation of a multivariate Gaussian pdf:

$$\mathcal{N}(\mu, K_{\boldsymbol{\theta}}) = (2\pi)^{-n/2} |K_{\boldsymbol{\theta}}|^{-1/2} e^{-\frac{1}{2}\mu^\top K_{\boldsymbol{\theta}}^{-1}\mu} \tag{4}$$

Forming $K_{\boldsymbol{\theta}}$ takes storage $\mathcal{O}(n^2)$ and it takes time $\mathcal{O}(n^3)$ to calculate its inverse and log-determinant, using e.g. the Cholesky decomposition [22]. This costly operation occurs for each draw of a sampler, and HMC, it occurs for each "leapfrog" step, many of which are taken per draw.

We will primarily focus on the case of a Gaussian observation model. This will conveniently allow us to sidestep the issues that arise from trying to sample $\boldsymbol{f}$. For fixed hyperparameters, a GP prior with Gaussian observation model is conjugate. Usually this is used to find the posterior distribution over $\boldsymbol{f}$ in closed form. Our setting is even simpler: we only need to have a way of calculating the likelihood of our observations y, integrating out $\boldsymbol{f}$. We have the following standard result [22]:

$$y \mid X, \boldsymbol{\theta}, \boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}, K_{\boldsymbol{\theta}} + \sigma^2 I) \tag{5}$$

[1] In a medium data spatial statistics setting, Diggle et al. [2013] report fitting a GP model with MCMC: after running for 18 million iterations, they retained every 18,000th iteration to yield a sample size of 1,000.

2.3 Kronecker inference

A series of recent papers [27, 23, 11, 35, 13, 9] has developed a set of inference techniques for the case of GP inference and prediction with separable covariance structures and inputs with a Cartesian product structure. We extend this line of work to an MCMC framework.

Assume that the covariance function $k(\cdot,\cdot)$ decomposes with Kronecker structure so that $k = k_1 \otimes k_2 \otimes \cdots \otimes k_d$ meaning that $k(x,x') = k_1(x_1,x_1')k_2(x_2,x_2')\cdots k(x_d,x_d')$ for $x \in \mathcal{R}^d$. Further assume that we have a grid of input locations given by the Cartesian product $(x_1^1,\ldots,x_1^N) \times (x_2^1,\ldots,x_2^N)\cdots\times (x_d^1,\ldots,x_d^N)$ where for notational convenience we assume the grid has the same size N in each dimension. Then the covariance matrix K corresponding to $k(\cdot,\cdot)$ has $N^d \times N^d$ entries, but it can be calculated by first calculating the smaller $N \times N$ covariance matrices $K_1,\ldots,K_d$ and then calculating the Kronecker product $K = K_1 \otimes K_2 \otimes \cdots \otimes K_d$. The assumption that our data lies on a grid occurs naturally in various settings: images [35], spatiotemporal point patterns [9], and as we will illustrate below, time series and categorical data (where grid cells correspond to cells in a contingency table.)

We rely on standard Kronecker algebra results [27], specifically efficient Kronecker matrix-vector multiplication to calculate expressions like $(K_1 \otimes K_2 \otimes \cdots \otimes K_d)v$, efficient eigendecomposition of Kronecker matrices, and efficient Cholesky factorization of Kronecker matrices.

3 Theory

3.1 Inference

A key subroutine in any MCMC scheme is the evaluation of the log of the probability density function (pdf) of the posterior. In a GP model, this means calculating the pdf of a Gaussian distribution shown in Eq. (4). Following [27], we show how to efficiently evaluate the pdf in the case of a Kronecker-structured covariance matrix $K_{\boldsymbol{\theta}}$ and also in the case of a covariance matrix $K_{\boldsymbol{\theta}} + \sigma^2 I$, which will arise when we analytically integrate out $\boldsymbol{f}$.

Working with the log of the pdf, and considering the case of $K = K_1 \otimes K_2$ (the extension to higher dimensions follows as in [27]), the log of Eq. (4) is:

$$-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|K_1 \otimes K_2| - \frac{1}{2}y^\top (K_1 \otimes K_2)^{-1} y \tag{6}$$

for observations y, where we have assumed for simplicity that $\boldsymbol{\mu} = 0$. Applying standard Kronecker algebra results, we calculate:

$$\log|K_1 \otimes K_2| = N_2 \log|K_1| + N_1 \log|K_2| \tag{7}$$

where K_1 is $N_1 \times N_1$ and K_2 is $N_2 \times N_2$. If we let Y be the reshaped column-major $N_2 \times N_1$ matrix corresponding to y, so that $\mathrm{vec}(Y) = y$ (where vec stacks columns of a matrix), then we have:

$$(K_1 \otimes K_2)^{-1} y = K_1^{-1}(Y K_2^{-1})^\top \tag{8}$$

And we apply any standard linear solver to evaluate these matrix products. In general, for n training points on a d-dimensional grid, the time complexity is $\mathcal{O}(dn^{\frac{d+1}{d}})$ where $n = N^d$ and $N_1 = N_2 = \cdots = N_d = N$ [27] .

For a Gaussian observation model after integrating out $\boldsymbol{f}$, our sampler needs to be to evaluate the log of the pdf in Eq. (5). We can use eigendecomposition where $K_1 = Q_1^\top \Lambda_1 Q_1$, $K_2 = Q_2^\top \Lambda_2 Q_2$ gives:

$$K_1 \otimes K_2 = (Q_1^\top \otimes Q_2^\top)(\Lambda_1 \otimes \Lambda_2)(Q_1 \otimes Q_2) \tag{9}$$

and since the Q matrices are orthonormal:

$$K_1 \otimes K_2 + \sigma^2 I = (Q_1^\top \otimes Q_2^\top)(\Lambda_1 \otimes \Lambda_2 + \sigma^2 I)(Q_1 \otimes Q_2) \tag{10}$$

Thus we can calculate:

$$\log|K_1 \otimes K_2 + \sigma^2 I| = N_1 N_2 \sum_{ij} \log(\Lambda_{1ii}\Lambda_{2jj} + \sigma^2) \tag{11}$$

$$(K_1 \otimes K_2 + \sigma^2 I)^{-1} y = \left((Q_1^\top \otimes Q_2^\top)(\Lambda_1 \otimes \Lambda_2 + \sigma^2 I)^{-1}(Q_1 \otimes Q_2)\right) y \tag{12}$$

Eq. (11) follows because the eigenvalues of K_1 and K_2 are given by the diagonal of Λ_1 and Λ_2 and the computation is $\mathcal{O}(N_1 N_2)$ or in general $\mathcal{O}(n)$. For Eq. (12), we use a Kronecker matrix-vector product to calculate $(Q_1 \otimes Q_2)y$. The middle term $(\Lambda_1 \otimes \Lambda_2 + \sigma^2 I)^{-1}$ is diagonal, and multiplying a diagonal matrix by a vector is just the elementwise product. Finally, we have one more Kronecker matrix-vector product to calculate, as above. Eigendecomposition is $\mathcal{O}(N^3)$ for an $N \times N$ matrix. Additional speed improvements are available for the calculation of the smaller K_i matrices: e.g., FFT works well for stationary kernels and regular gridded input [24] and Toeplitz methods work well for stationary kernels in one-dimension with regularly spaced inputs [6].

3.2 Prediction

We can extend the ideas introduced above to efficiently infer the posterior $p(\boldsymbol{f}^*|y, X, x^*)$ at a new location x^*. For a fixed $K_{\boldsymbol{\theta}}$ we have the following standard result [22]:

$$p(\boldsymbol{f}^*|y, X, x^*, \boldsymbol{\theta}) = \mathcal{N}(K^*_{\boldsymbol{\theta}}(K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1} y, K^{**}_{\boldsymbol{\theta}} - K^*_{\boldsymbol{\theta}}(K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1} K^{*\top}_{\boldsymbol{\theta}}) \tag{13}$$

where $K^*_{\boldsymbol{\theta}} = [k_{\boldsymbol{\theta}}(x^*, x_1), \ldots, k_{\boldsymbol{\theta}}(x^*, x_n)]$ and $K^{**}_{\boldsymbol{\theta}} = k_{\boldsymbol{\theta}}(x^*, x^*)$. A naive implementation would have time complexity $\mathcal{O}(n^3)$. To calculate the mean in Eq (13), we can again exploit Kronecker structure with the eigendecompositions in Eqs. (9) and (12):

$$K^*_{\boldsymbol{\theta}}(K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1} y = K^*_{\boldsymbol{\theta}}(Q_1^\top \otimes Q_2^\top)(\Lambda_1 \otimes \Lambda_2 + \sigma^2 I)^{-1}(Q_1 \otimes Q_2)y \tag{14}$$

We now apply Kronecker-matrix vector multiplication to the first two terms and the last two terms, and we are left with a vector times a diagonal matrix times a vector, which we can calculate efficiently through elementwise vector multiplication. Thus, the overall complexity is the same as for Eq. (12). For the variance term we have:

$$K^{**}_{\boldsymbol{\theta}} - K^*_{\boldsymbol{\theta}}(K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1} K^{*\top}_{\boldsymbol{\theta}} = K^{**}_{\boldsymbol{\theta}} - K^*_{\boldsymbol{\theta}}(Q_1^\top \otimes Q_2^\top)(\Lambda_1 \otimes \Lambda_2 + \sigma^2 I)^{-1}(Q_1 \otimes Q_2) K^{*\top}_{\boldsymbol{\theta}} \tag{15}$$

We use Kronecker matrix-vector multiplication twice to efficiently calculate the variance.

4 Modeling approaches

In this section, we demonstrate the advantages and flexibility of a fully Bayesian approach to GP modeling. We explore priors for categorical data and a low-rank factor analysis style model for clustering, demonstrate novel priors over hyperparameters for structured kernels, show how to infer missing data, and close with extensions for non-Gaussian observation models.

4.1 Kernel choice and priors over hyperparameters

The spatial statistics and machine learning literature provides a rich palette of scientifically and statistically motivated kernels from which to choose. As with any modeling choice, this could be informed by prior information, continuous model expansion [10], or a nonparametric [34] approach could be taken. Because the problem of kernel learning is so difficult, including priors over the kernel hyperparameters to accurately characterize posterior uncertainty is very important.

Another advantage of placing priors over kernel hyperparameters is that the posterior distribution, which integrates out these parameters, is a *heavy tailed non-Gaussian process*. Following [1], we adopt weakly informative priors for inverse length-scales and variances which concentrate their support on reasonable values. We face an issue similar to the problem of applying HMC to neural networks [3]: small changes in the inverse length-scales can result in orders-of-magnitude changes in the posterior. Experience suggests that more informative priors can contribute to sampling efficiency with HMC. Our provisional suggestions for priors are given in Section A.1 for standard stationary kernels.

4.2 Categorical data

Consider a multitask / co-kriging model where our dataset is structured as a real-valued observation y_i from category (task) c_i occuring at time t_i. We propose the following GP model:

$$y_i \sim \mathcal{N}(f(t_i, c_i), \sigma^2) \tag{16}$$

$$f(t, c) \sim \mathcal{GP}(0, K_t \otimes K_c) \tag{17}$$

We can immediately apply the methods developed in the previous sections to this setting, which is similar to the multitask approach in [2]. It remains to choose the kernel over categories. Let us express c_i as an indicator vector, where, e.g. for 3 categories we have the vectors $[1\ 0\ 0]^\top$, $[0\ 1\ 0]^\top$ and $[0\ 0\ 1]^\top$. Then we can use the covariance matrix K_c as the kernel:

$$k(c, c') = c^\top K_c c \tag{18}$$

The default choice of prior for a covariance matrix is the inverse Wishart distribution, which is conjugate, but has well-known drawbacks; various alternatives have been proposed and analyzed [30]. As an alternative, and because we only care about learning correlations, not covariances, between tasks, we use a recently proposed prior [17] which was termed the LKJ prior in [28]. This prior has the advantage that for precision parameter $\alpha = 2$, each partial correlation has distribution Uniform$(-1, 1)$.

4.3 Factor analysis

Another natural extension if we want to cluster our m categories into p clusters is to use a low-rank factorization where $K_s = LL^\top + \sigma^2 I$ with $K_s \in \mathbb{R}^{m\times m}$ and $L \in \mathbb{R}^{m\times p}$ for $p \ll m$. This kernel has been called a "factor analysis" kernel [22]. We propose constraining each row of L to sum to 1 to represent a soft clustering model, where L_{ij} gives the degree of membership of x_i in group j. A natural choice of prior is the Dirichlet distribution. Denoting row i of L as L_i we have:

$$L_i \overset{iid}{\sim} \text{Dirichlet}(\alpha, \ldots, \alpha) \tag{19}$$

for a concentration parameter α which may be fixed or have its own prior.

If we have, for example, time series observations for each category and we want to cluster categories then we can use the Kronecker formulation: $K = K_t \otimes K_s$. The factorization above is convenient because we can readily obtain the eigendecomposition of Eq. (12) through singular value decomposition (SVD) of L. We can efficiently find the p singular values of L, which we denote $e_1, \ldots, e_p$. Then the m eigenvalues of K_s are $e_1^2 + \sigma^2, \ldots, e_p^2 + \sigma^2, \sigma^2, \ldots, \sigma^2$. Similarly, we can use the left-singular vectors from SVD to obtain the eigenvectors of K_s.

4.4 Additive models

Another structured model worth considering is an additive covariance model $K_s + K_t$, which is equivalent to $f \sim \mathcal{GP}(0, K_s) + \mathcal{GP}(0, K_t)$. We propose a very convenient way of emulating this model using a mixture modeling approach. We introduce a latent variable $z \sim \text{Bernoulli}(\pi)$. Then we have:

$$f|z = 0 \sim \mathcal{GP}(0, K_s) \tag{20}$$

$$f|z = 1 \sim \mathcal{GP}(0, K_t) \tag{21}$$

The goal is to obtain a very flexible model, without adding much computational burden. As shown in the Appendix, the implementation is very straightforward after integrating out z.

4.5 Non-separable covariances

In this section we propose a way of relaxing the assumption of separability. Standard machine learning kernels like RBF (with or without Automatic Relevance Determination) have a separable product structure. While these kernels have proven quite successful for machine learning approaches, in spatiotemporal settings separable structure in a space/time covariance function implies the potentially undesirable property that for a space/time covariance function $K((s, t), (s', t'))$ the temporal covariance structure does not vary in space, and the spatial covariance structure does not vary in time [25]. But in a variety of applications, it is precisely the second order space/time interaction that we are interested in modeling.

While there has been vigorous work in recent decades on proposing classes of non-separable covariance functions [5, 12], we propose a much simpler approach for modeling non-separability which preserves the computational benefits of the Kronecker structure. We consider placing a joint prior on the hyperparameters. Let K_s have lengthscale λ_s and K_t have lengthscale λ_t. Then the β parameter models the "interaction" between dimensions s and t:

$$\begin{pmatrix} \lambda_s \\ \lambda_t \end{pmatrix} \sim \mathcal{N}\left(\mu_{st}, \begin{pmatrix} \sigma_s^2 & \beta \\ \beta & \sigma_t^2 \end{pmatrix}\right) \tag{22}$$

4.6 Missing observations

Incomplete grids due to missing observations can be straightforwardly handled in the fully Bayesian framework (especially when using a probabilistic programming language): for any observation location x_i where we do not observe y_i, we treat y_i as a latent parameter. The key likelihood calculation in Eq. (6) remains the same, only now we mix together observed (and thus for the purposes of our sampler fixed) y_i's with missing y_i's which we must learn by sampling. Code is in the Appendix.

4.7 Extensions to non-Gaussian observation models

Non-Gaussian observation models are very useful for generalized regression and classification. For example, classification uses the Bernoulli likelihood with logistic link function:

$$y \sim \text{Bernoulli}(\text{logit}(f(x))) \tag{23}$$
$$f \sim \mathcal{GP}(0, K_{\boldsymbol{\theta}}) \tag{24}$$

This model is no longer conjugate, so we cannot integrate out $\boldsymbol{f}$, but we can handle it in an MCMC framework. This has the effect of increasing the number of parameters in our model, and these parameters have strong correlations. To attempt to mitigate these correlations, we adopt the formulation of [4] which has been used in HMC in [31], introducing latent variables $z_1, \ldots, z_n$ based on the weight-space view of GPs:

$$z_1, \ldots, z_n \overset{iid}{\sim} \mathcal{N}(0, 1) \tag{25}$$

Now we calculate $K_{\boldsymbol{\theta}}$ and its Cholesky decomposition L where $LL^T = K_{\boldsymbol{\theta}}$. Then we have:

$$\boldsymbol{f} := Lz \tag{26}$$
$$y_i \sim \text{Bernoulli}(\text{logit}(f(x_i))) \tag{27}$$

By introducing z we have avoided some computational difficulty as we no longer need to calculate the determinant of $K_{\boldsymbol{\theta}}$. But we still need to calculate K itself and its Cholesky decomposition, which is $O(n^3)$ time and $O(n^2)$ memory. Once again we can exploit two Kronecker algebra results. We calculate the $N \times N$ Kronecker matrices K_s and K_t and find their Cholesky decompositions L_s and L_t in $O(N^3)$ time. Since $K = LL^\top$ we have:

$$K = K_s \otimes K_t = (L_s \otimes L_t)(L_s^\top \otimes L_t^\top) \tag{28}$$

Now we need to calculate $\boldsymbol{f} = Lz = (L_s \otimes L_t)z$ for Eq. (26). Once again, we use efficient Kronecker matrix-vector multiplication. Now, rather than a costly multivariate Gaussian pdf, we only have to evaluate the much cheaper univariate Gaussian distribution in Eq. (25).

In practice, we have had difficulties using HMC to update both $z_1, \ldots, z_n$ and the hyperparameters simultaneously. A reasonable solution might be to follow the blocked approach of [1, 32], wherein the hyperparameters are sampled with, e.g. HMC and then conditional on these hyperparameters the z_i are sampled, with HMC or another algorithm.

5 Experiments

We implemented our models using HMC in the probabilistic programming language Stan [28]. This allowed us to easily try out different choices of priors and different modeling approaches. All source code is provided in the Appendix.

5.1 Synthetic data

We simulate from a Gaussian process on an $n \times n$ regular grid using a product of RBF kernels: $k((s,t),(s',t')) = e^{-4|s-s'|^2} e^{-|t-t'|^2}$ with spherical Gaussian noise $\sigma = 0.1$. A sample is shown in Figure A1. We compare our proposed Kronecker-based inference to non-Kronecker inference, both with HMC, and to elliptical slice sampling. As shown in Figure 1 our approach is much more efficient than the alternatives. We are not in a regime in which the $\mathcal{O}(n^3)$ asymptoptic time of the Cholesky decomposition dominates the computation, and there are many other factors which come into play in determining

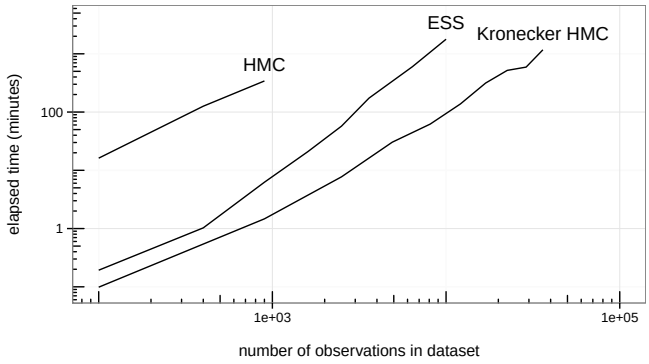

Figure 1: Our method ("Kronecker HMC") was implemented in Stan, standard HMC and elliptical slice sampling were implemented in GPstuff. HMC was run for 200 iterations, with 100 iterations of warm-up, and elliptical slice sampling (ESS) for 30 iterations. Each method was compared on the same simulated datasets (Section A.2).

how long MCMC takes to run, but it is clear that our HMC approach is much faster, especially considering that we implemented it in a general purpose probabilistic programming language (Stan) rather than relying on custom code. Furthermore, the memory requirements for the non-Kronecker methods became prohibitively large in practice. As another comparison, we calculated the effective sample size [15] for a dataset of size $n = 2{,}500$. Our model generated an effective sample of 203 draws in 296 seconds or 0.69 samples per second. Elliptical slice sampling generated an effective sample of 221 draws in 3,438 seconds or 0.06 samples per second. Standard HMC generated an effective sample size of 51 samples in 31,364 seconds or 0.002 samples per second.

5.2 Real data

We obtained time series of monthly population-adjusted incidence of hepatitis A, measles, mumps, pertussis, and rubella for the 48 lower United States plus DC from Project Tycho[2].

We used our factor analysis formulation to cluster US states based on the time series of measles incidence. Our separable covariance function was $K_t \otimes K_s$ where K_t was an RBF kernel and $K_s = \Lambda\Lambda^\top + \sigma^2 I$, with a Dirichlet$(0.1, 0.1, 0.1)$ on each row of $\Lambda \in \mathcal{R}^{49\times 3}$. The clustering of states is shown in Figure 2 (left): a geographic pattern is recovered, despite the model not using geographic information. In Figure 2 (right) we show the posterior mean time series averaged for each cluster. Different dynamics are evident.

We used the centroids of each state in an additive model with covariance $K_s + K_t$ for RBF kernels and joint prior on the inverse length-scales as proposed in Section 4.5. The posterior over the space/time interaction β was 0.47 with 95% UI: (0.02,0.96), indicating nonseparability in the covariance structure of the data, at least as compared to an additive model. The mixture component weights were 0.67 for K_s and 0.33 for K_t. Finally, we considered the national time series for 4 different diseases from the Project Tycho dataset with a separable covariance $K_t \otimes K_c$ where K_t is as above and K_c is the categorical kernel in Eq. (18). We assign an LKJ prior with $\alpha = 2$ over the cross-type correlation matrix K_c. In Table 1 we show the posterior cross-type correlation matrix K_c. The lengthscale for K_t was 2 months $(1.9, 2.3)$ which corresponds to short-scale variation. Posterior plots are shown in Figure 3.

	Hepatitis A	Mumps	Pertussis	Rubella
Hepatitis A	1	0.6 (0.4,0.8)	-0.3 (-0.6,-0.1)	0.4 (0.1,0.6)
Mumps	0.6 (0.4,0.8)	1	-0.2 (-0.4,0.0)	0.6 (0.4,0.7)
Pertussis	-0.3 (-0.6,-0.1)	-0.2 (-0.4,0.0)	1	-0.2 (-0.5,-0.0)
Rubella	0.4 (0.1,0.6)	0.6 (0.4,0.7)	-0.2 (-0.5,-0.0)	1

Table 1: For the multitask model with covariance $K_t \otimes K_c$, we learned the posterior over a cross-task correlation matrix K_c. Medians and 95% UI intervals are stated. The corresponding lengthscale for K_t was 2 months $(1.9, 2.4)$ which corresponds to short-scale variation.

[2] www.tycho.pitt.edu

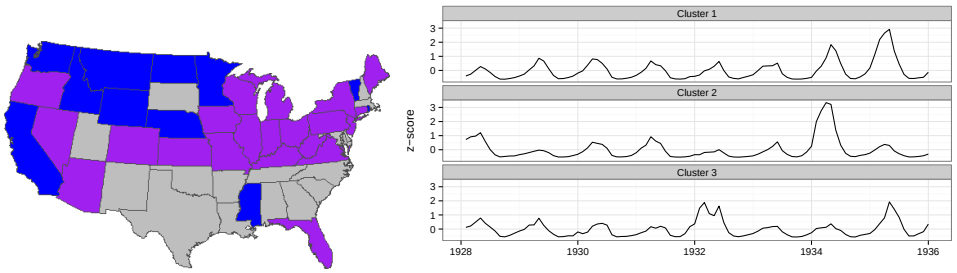

Figure 2: Left: clustering US states based on their time series of measles incidence. For each state we learned a 3-dimensional vector giving cluster assignment probabilities. We assign each state to its most probable cluster of the three, and shade it accordingly. Despite not using any geographic information in our model, we find a clear geographic pattern based on the similarities in time series. Right: mean posterior time series are shown for each cluster with evident differences in dynamics.

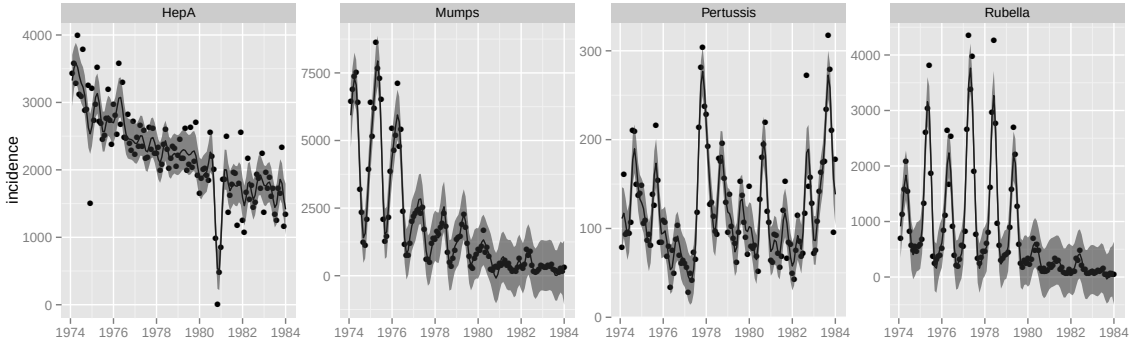

Figure 3: Raw incidence across the United States of 4 types of infectious disease are shown as points, along with our model's estimates and 95% uncertainty intervals.

6 Conclusion

We presented a Bayesian hierarchical modeling framework based on GPs, and demonstrated the efficiency gains possible in the case of structured kernels as compared to standard approaches. This modeling approach enabled a variety of interesting and novel models and approaches to kernel learning. Further work is needed on the challenges of a non-Gaussian observation model. Inducing points might prove useful for further speedups and a relaxation of the grid requirement.

References

[1] Deepak K Agarwal and Alan E Gelfand. Slice sampling for simulation based fitting of spatial data models. *Statistics and Computing*, 15(1):61–69, 2005.

[2] Edwin V Bonilla, Kian M Chai, and Christopher Williams. Multi-task gaussian process prediction. In *Advances in Neural Information Processing Systems*, pages 153–160, 2007.

[3] Kiam Choo. *Learning hyperparameters for neural network models using Hamiltonian dynamics*. PhD thesis, Citeseer, 2000.

[4] Ole F Christensen, Gareth O Roberts, and Martin Sköld. Robust markov chain monte carlo methods for spatial generalized linear mixed models. *Journal of Computational and Graphical Statistics*, 15(1):1–17, 2006.

[5] Noel Cressie and Hsin-Cheng Huang. Classes of nonseparable, spatio-temporal stationary covariance functions. *Journal of the American Statistical Association*, 94(448):1330–1339, 1999.

[6] John P Cunningham, Krishna V Shenoy, and Maneesh Sahani. Fast gaussian process methods for point process intensity estimation. In *Proceedings of the 25th ICML*, pages 192–199. ACM, 2008.

[7] Peter J Diggle, Paula Moraga, Barry Rowlingson, Benjamin M Taylor, et al. Spatial and spatio-temporal Log-Gaussian Cox processes: extending the geostatistical paradigm. *Statistical Science*, 28(4):542–563, 2013.

[8] Andrew O Finley, Sudipto Banerjee, Patrik Waldmann, and Tore Ericsson. Hierarchical spatial modeling of additive and dominance genetic variance for large spatial trial datasets. *Biometrics*, 65(2):441–451, 2009.

[9] Seth R Flaxman, Andrew G Wilson, Daniel B Neill, Hannes Nickisch, and Alexander J Smola. Fast Kronecker Inference in Gaussian Processes with non-Gaussian Likelihoods. *International Conference on Machine Learning 2015*, 2015.

[10] Andrew Gelman and Cosma Rohilla Shalizi. Philosophy and the practice of bayesian statistics. *British Journal of Mathematical and Statistical Psychology*, 66(1):8–38, 2013.

[11] E. Gilboa, Y. Saatci, and J. Cunningham. Scaling multidimensional inference for structured gaussian processes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PP(99):1–1, 2013.

[12] Tilmann Gneiting. Nonseparable, stationary covariance functions for space–time data. *Journal of the American Statistical Association*, 97(458):590–600, 2002.

[13] Perry Groot, Markus Peters, Tom Heskes, and Wolfgang Ketter. Fast laplace approximation for gaussian processes with a tensor product kernel. In *BNAIC*, 2014.

[14] James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. 2013.

[15] Robert E Kass, Bradley P Carlin, Andrew Gelman, and Radford M Neal. Markov chain monte carlo in practice: a roundtable discussion. *The American Statistician*, 52(2):93–100, 1998.

[16] Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum Gaussian process regression. *JMLR*, 11:1865–1881, 2010.

[17] Daniel Lewandowski, Dorota Kurowicka, and Harry Joe. Generating random correlation matrices based on vines and extended onion method. *Journal of multivariate analysis*, 100(9):1989–2001, 2009.

[18] Iain Murray and Ryan P Adams. Slice sampling covariance hyperparameters of latent gaussian models. In *Advances in Neural Information Processing Systems 23*, pages 1732–1740. 2010.

[19] Iain Murray, Ryan Prescott Adams, and David J.C. MacKay. Elliptical slice sampling. *JMLR: W&CP*, 9: 541–548, 2010.

[20] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996. ISBN 0387947248.

[21] Anthony O'Hagan. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society*, B(40):1–42, 1978.

[22] Carl Edward Rasmussen and Christopher KI Williams. Gaussian processes for machine learning, 2006.

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

[24] Brian D Ripley. *Stochastic simulation*, volume 316. John Wiley & Sons, 2009.

[25] Jonathan Rougier. A representation theorem for stochastic processes with separable covariance functions. 2011.

[26] Håvard Rue, Sara Martino, and Nicolas Chopin. Approximate bayesian inference for latent gaussian models by using integrated nested laplace approximations. *Journal of the royal statistical society: Series b (statistical methodology)*, 71(2):319–392, 2009.

[27] Yunus Saatçi. *Scalable inference for structured Gaussian process models*. PhD thesis, University of Cambridge, 2011.

[28] Stan Development Team. Stan: A c++ library for probability and sampling, version 2.5.0, 2014.

[29] Oliver Stegle, Christoph Lippert, Joris M Mooij, Neil D Lawrence, and Karsten M Borgwardt. Efficient inference in matrix-variate gaussian models with iid observation noise. In *Advances in neural information processing systems*, pages 630–638, 2011.

[30] Tomoki Tokuda, Ben Goodrich, I Van Mechelen, Andrew Gelman, and F Tuerlinckx. Visualizing distributions of covariance matrices. *Columbia Univ., New York, NY, USA, Tech. Rep*, 2011.

[31] Jarno Vanhatalo and Aki Vehtari. Sparse log gaussian processes via mcmc for spatial epidemiology. In *Gaussian Processes in Practice*, pages 73–89, 2007.

[32] Jarno Vanhatalo, Jaakko Riihimäki, Jouni Hartikainen, Pasi Jylänki, Ville Tolvanen, and Aki Vehtari. GPstuff: Bayesian modeling with Gaussian processes. *JMLR*, 14(1):1175–1179, 2013.

[33] Grace Wahba. *Spline models for observational data*, volume 59. Siam, 1990.

[34] Andrew G Wilson and Ryan P Adams. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1067–1075, 2013.

[35] Andrew Gordon Wilson, Elad Gilboa, Arye Nehorai, and John P. Cunningham. Fast kernel learning for multidimensional pattern extrapolation. In *NIPS*. MIT Press, 2014.

[36] Z. Yang, A.J. Smola, L. Song, and A.G. Wilson. A la carte - learning fast kernels. *Artificial Intelligence and Statistics*, 2015.

A Appendix

A.1 Priors for kernel hyperparameters

For a stationary kernel $k(x, x') = v^2\kappa(|x - x'|_2\lambda) + \sigma^2 I(x = x')$ where, e.g. $\kappa(d) = exp(-d^2)$ is the RBF kernel or $\kappa(d) = \left(1 + d\sqrt{3}\right)\exp\left(-d\sqrt{3}\right)$ is the Matérn-$\frac{3}{2}$ kernel we suggest priors as follows:

- The inverse length-scale λ should have a weakly informative prior, like Cauchy$(0, 2.5)$ constrained to be positive (also known as the half-Cauchy).[3] We usually standardize our input locations to have standard deviation 1, but it is very important to untransform the learned length-scale to check that it is a reasonable value.
- The signal-variance v^2 should be about the scale of the data. In practice, we standardize our observed y to have standard deviation 1 so that we can place a log-normal$(0, 1)$ prior on v^2.
- For computational reasons (equivalent to the jitter that is often added to the main diagonal) we constrain the nugget σ^2 to be greater than $\epsilon = 10^{-6}$ and use a log-normal$(0, 1)$ prior.

A.2 Synthetic Data

To generate large synthetic datasets, we use the following R code:

```
library(kernlab)

eps = 1e-8
n = 200
space = seq(-2,2,length.out=n)
time = space

K1 = kernelMatrix(rbfdot(4), as.matrix(space))
K2 = kernelMatrix(rbfdot(1), as.matrix(time))
L1 = chol(K1 + eps * diag(n))
L2 = chol(K2 + eps * diag(n))

v = rnorm(n*n)
y = as.numeric(matrix(t(t(L2) %*% matrix(t(t(L1) %*% matrix(v,n,n)),n,n)),n*n,1))
y = y + rnorm(n*n,sd=.1)  # Add spherical noise

data = list(n1=length(space),n2=length(time), x1=space, x2=time, y=as.numeric(y))
```

A sample dataset is shown in Figure A1.

[3] An alternative worth considering is placing a Student-t with $\nu = 4$ centered at 0 with scale parameter 1 constrained to be positive on the *length-scale*. This is a more informative choice than the half-Cauchy.

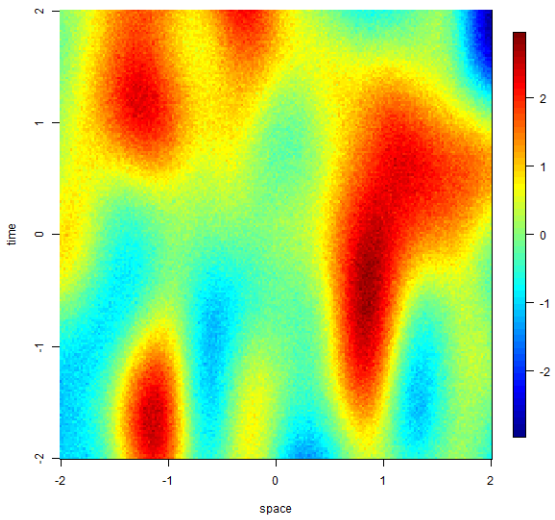

Figure A1: A synthetic dataset with $n = 40,000$ observations generated by a GP with a separable covariance function $k((s,t),(s',t')) = e^{-4|s-s'|^2} e^{-|t-t'|^2}$.

A.3 Stan source code for categorical data model

```
functions {
  // return (A \otimes B) v where:
  // A is n1 x n1, B = n2 x n2, V = n2 x n1 = reshape(v,n2,n1)
  matrix kron_mvprod(matrix A, matrix B, matrix V) {
    return transpose(A * transpose(B * V));
  }

  // A is a length n1 vector, B is a length n2 vector.
  // Treating them as diagonal matrices, this calculates:
  // v = (A \otimes B + sigma2)^{-1}
  // and returns the n1 x n2 matrix V = reshape(v,n1,n2)
  matrix calculate_eigenvalues(vector A, vector B, int n1, int n2, real sigma2) {
    matrix[n1,n2] e;
    for(i in 1:n1) {
      for(j in 1:n2) {
        e[i,j] <- (A[i] * B[j] + sigma2);
      }
    }
    return(e);
  }
}

data {
  int<lower=1> n1;
  int<lower=1> n2; // categories for learning cross-type correlations
  vector[n2] x1; // observation locations (e.g. timestamps)
  matrix[n2,n1] y; // NB: this should be reshape(y, n2, n1),
                //  where y corresponds to expand.grid(x2,x1).
                //  To double-check, make sure that y[i,j] is
                //  the observation from category x2[i]
                //  at location x1[j]
}

transformed data {
  matrix[n1, n1] xd;
  vector[2] one;
```

```
  one[1] <- 1;
  one[2] <- 1;

  for (i in 1:n1) {
    xd[i, i] <- 0;
    for (j in (i+1):n1) {
      xd[i, j] <- -(x1[i]-x1[j])^2;
      xd[j, i] <- xd[i, j];
    }
  }
}
parameters {
  real<lower=0> var1; // signal variance
  real<lower=0> bw1; // this is equivalent to 1/sqrt(length-scale)
  corr_matrix[n2] L;
  real<lower=0.00001> sigma1;
}

model {
  matrix[n1, n1] Sigma1;
  matrix[n1, n1] Q1;
  vector[n1] R1;
  matrix[n2, n2] Q2;
  vector[n2] R2;
  matrix[n2,n1] eigenvalues;

  Sigma1 <- var1 * exp(xd * bw1);
  for(i in 1:n1)
    Sigma1[i,i] <- Sigma1[i,i] + .00001;

  L ~ lkj_corr(2.0);

  Q1 <- eigenvectors_sym(Sigma1);
  R1 <- eigenvalues_sym(Sigma1);

  Q2 <- eigenvectors_sym(L);
  R2 <- eigenvalues_sym(L);

  eigenvalues <- calculate_eigenvalues(R2,R1,n2,n1,sigma1);

  var1 ~ lognormal(0,1);
  bw1 ~ cauchy(0,2.5);
  sigma1 ~ lognormal(0,1);
  increment_log_prob(
     -0.5 * sum(y .* kron_mvprod(Q1,Q2, // calculates -0.5 * y' (K1 \otimes K2) y
          kron_mvprod(transpose(Q1),transpose(Q2),y) ./ eigenvalues))
     -0.5 * sum(log(eigenvalues))); // calculates logdet(K1 \otimes K2)
}
```

A.4 Stan source code for low-rank factorization model

```
functions {
  ... see first model above ...
}
data {
  int<lower=1> n1; // categories for clustering

  int<lower=1> n2;
  vector[n2] x2; // observation locations (e.g. timestamps)

  matrix[n2,n1] y; // NB: this should be reshape(y, n2, n1),
                   //  where y corresponds to expand.grid(x2,x1).
                   //  To double-check, make sure that y[i,j] is
                   //  the observation from category x1[j] at location
                   //  x2[i]
```

```
  int K;
}
transformed data {
  vector[K] alpha;
  matrix[n2,n2] xd;
  for(i in 1:n2) {
    xd[i,i] <- 0;
    for (j in (i+1):n2) {
      xd[i, j] <- -(x2[i]-x2[j])^2;
      xd[j, i] <- xd[i, j];
    }
  }

  for(i in 1:K)
    alpha[i] <- .1;
}
parameters {
  real<lower=0> var1;
  real<lower=0> bw2;
  real<lower=0.0001> sigma1;
  real<lower=0.0001> sigma2;
  simplex[K] Lambda1[n1];
}
transformed parameters {
  matrix[n1,K] Lambda1m;
  for(i in 1:n1) {
    Lambda1m[i] <- to_row_vector(Lambda1[i]);
  }
}

model {
  matrix[n1, n1] Sigma1;
  matrix[n2, n2] Sigma2;
  matrix[n1, n1] Q1;
  matrix[n2, n2] Q2;
  vector[n1] L1;
  vector[n2] L2;
  matrix[n2,n1] eigenvalues;

  for(i in 1:n1)
    to_vector(Lambda1[i]) ~ dirichlet(alpha);
  Sigma1 <- var1 * Lambda1m * transpose(Lambda1m);
  for (i in 1:n1)
    Sigma1[i] <- Sigma1[i] + sigma1;

  Sigma2 <- exp(xd * bw2);
  for (i in 1:n2) {
    Sigma2[i, i] <- Sigma2[i,i] + 0.000001;
  }

  Q1 <- eigenvectors_sym(Sigma1);
  Q2 <- eigenvectors_sym(Sigma2);
  L1 <- eigenvalues_sym(Sigma1);
  L2 <- eigenvalues_sym(Sigma2);

  eigenvalues <- calculate_eigenvalues(L2,L1,n2,n1,sigma2);
  bw2 ~ cauchy(0,2.5);
  var1 ~ lognormal(0,1);
  sigma1 ~ lognormal(0,1);
  sigma2 ~ lognormal(0,1);
  increment_log_prob(
     -0.5 * sum(y .* kron_mvprod(Q1,Q2, // calculates -0.5 * y' (K1 \otimes K2) y
          kron_mvprod(transpose(Q1),transpose(Q2),y) ./ eigenvalues))
     -0.5 * sum(log(eigenvalues))); // calculates logdet(K1 \otimes K2)
}
```

A.5 Stan source code for synthetic data

```
functions {
  ... see first model above ...
}

data {
  int<lower=1> n1;
  int<lower=1> n2;
  vector[n1] x1;
  vector[n2] x2;
  matrix[n1,n2] y;
  real sigma2;
}

parameters {
  real<lower=0> bw1;
  real<lower=0> bw2;
  real<lower=0> var1;
}

model {
  matrix[n1, n1] Sigma1;
  matrix[n2, n2] Sigma2;
  matrix[n1, n1] Q1;
  matrix[n2, n2] Q2;
  vector[n1] L1;
  vector[n2] L2;
  matrix[n1,n2] eigenvalues;

  // these loops can be moved to the transformed data
  // block for efficiency, as in the source code in
  // the next section
  for (i in 1:n1) {
    Sigma1[i, i] <- var1;
    for (j in (i+1):n1) {
      Sigma1[i, j] <- var1 * exp(-(x1[i]-x1[j])^2*bw1);
      Sigma1[j, i] <- Sigma1[i, j];
    }
  }
  for (i in 1:n2) {
    Sigma2[i, i] <- 1;
    for (j in (i+1):n2) {
      Sigma2[i, j] <- exp(-(x2[i]-x2[j])^2*bw2);
      Sigma2[j, i] <- Sigma2[i, j];
    }
  }

  Q1 <- eigenvectors_sym(Sigma1);
  Q2 <- eigenvectors_sym(Sigma2);
  L1 <- eigenvalues_sym(Sigma1);
  L2 <- eigenvalues_sym(Sigma2);

  eigenvalues <- calculate_eigenvalues(L1,L2,n1,n2,sigma2);
  var1 ~ lognormal(0,1);
  bw1 ~ cauchy(0,2.5);
  bw2 ~ cauchy(0,2.5);
  sigma2 ~ lognormal(0,1);
  increment_log_prob( -0.5 * sum(y .* kron_mvprod(Q1,Q2,
       kron_mvprod(transpose(Q1),transpose(Q2),y) ./ eigenvalues))
       - .5 * sum(log(eigenvalues)));
}
```

A.6 Stan source code for incomplete grids / missing observations

```
functions {
   ... see first model above ...
}
data {
  int<lower=1> n1;
  int<lower=1> n2;
  int<lower=0> nmissing;
  vector[n1] x1;
  vector[n2] x2;
  matrix[n2,n1] y;
  int x1missing[nmissing];
  int x2missing[nmissing];
}

transformed data {
  matrix[n1, n1] xd1;
  matrix[n2, n2] xd2;

  for(i in 1:n1) {
    xd1[i, i] <- 0;
    for (j in (i+1):n1) {
      xd1[i, j] <- -(x1[i]-x1[j])^2;
      xd1[j, i] <- xd1[i, j];
    }
  }
  for (i in 1:n2) {
    xd2[i, i] <- 0;
    for (j in (i+1):n2) {
      xd2[i, j] <- -(x2[i]-x2[j])^2;
      xd2[j, i] <- xd2[i, j];
    }
  }
}
parameters {
  real<lower=0> var1; // signal variance
  real<lower=0> bw1; // bandwidth
  real<lower=0> bw2;
  real<lower=0.00001> sigma1;
  vector[nmissing] ymissing;
}
transformed parameters {
  matrix[n2,n1] ystar;
  ystar <- y;
  for(i in 1:nmissing) {
    ystar[x2missing[i],x1missing[i]] <- ymissing[i];
  }
}

model {
  matrix[n1, n1] Sigma1;
  matrix[n2, n2] Sigma2;
  matrix[n1, n1] Q1;
  vector[n1] R1;
  matrix[n2, n2] Q2;
  vector[n2] R2;
  matrix[n2,n1] eigenvalues;

  Sigma1 <- var1 * exp(xd1 * bw1);
  for(i in 1:n1)
    Sigma1[i,i] <- Sigma1[i,i] + .00001;
  Sigma2 <- exp(xd2 * bw2);
  for(i in 1:n2)
    Sigma2[i,i] <- Sigma2[i,i] + .00001;

  Q1 <- eigenvectors_sym(Sigma1);
```

```
  R1 <- eigenvalues_sym(Sigma1);

  Q2 <- eigenvectors_sym(Sigma2);
  R2 <- eigenvalues_sym(Sigma2);

  eigenvalues <- calculate_eigenvalues(R2,R1,n2,n1,sigma1);

  var1 ~ lognormal(0,1);
  bw1 ~ cauchy(0,2.5);
  bw2 ~ cauchy(0,2.5);
  sigma1 ~ lognormal(0,1);
  increment_log_prob(
     -0.5 * sum(ystar .* kron_mvprod(Q1,Q2,
       kron_mvprod(transpose(Q1),transpose(Q2),ystar) ./ eigenvalues)) // calculates -0.5 * y' (
     -0.5 * sum(log(eigenvalues))); // calculates logdet(K1 \otimes K2)
}
```

A.7 Stan source code for mixture model

```
data {
  int<lower=1> n1;
  int<lower=1> n2;
  vector[2] x1[n1];
  vector[n2] x2;
  matrix[n2,n1] y;
}
transformed data{
  vector[2] one;
  matrix[n1,n2] yt;
  vector[n1] zero1;
  vector[n2] zero2;
  one[1] <- 1;
  one[2] <- 1;
  for(i in 1:n1)
    zero1[i] <- 0;
  for(i in 1:n2)
    zero2[i] <- 0;
  yt <- transpose(y);
}
parameters {
  real<lower=0> var1;
  real<lower=0> var2;
  vector<lower=0>[2] bw;
  real<lower=0.00001> sigma1;
  real<lower=0.00001> sigma2;
  real<lower=0> spacetime;
}

model {
  matrix[n1, n1] Sigma1;
  matrix[n2, n2] Sigma2;
  matrix[2,2] SpaceTime;
  real lp1;
  real lp2;

  SpaceTime[1,1] <- 10;
  SpaceTime[1,2] <- spacetime;
  SpaceTime[2,1] <- spacetime;
  SpaceTime[2,2] <- 10;

  for (i in 1:n1) {
    Sigma1[i, i] <- var1 + sigma1;
    for (j in (i+1):n1) {
      Sigma1[i, j] <- var1 * exp(-dot_self(x1[i]-x1[j])*bw[1]);
```

```
    Sigma1[j, i] <- Sigma1[i, j];
   }
  }
  for (i in 1:n2) {
   Sigma2[i, i] <- var2 + sigma2;
   for (j in (i+1):n2) {
    Sigma2[i, j] <- var2 * exp(-(x2[i]-x2[j])^2*bw[2]);
    Sigma2[j, i] <- Sigma2[i, j];
   }
  }

  bw ~ multi_normal(one,SpaceTime);
  spacetime ~ uniform(-1,1);
  var1 ~ lognormal(0,1);
  var2 ~ lognormal(0,1);
  sigma1 ~ lognormal(0,1);
  sigma2 ~ lognormal(0,1);
  pi ~ uniform(0,1);
  lp1 <- 0;
  for(i in 1:n2)
   lp1 <- lp1 + multi_normal_log(y[i],zero1, Sigma1);
  lp2 <- 0;
  for(j in 1:n1)
   lp2 <- lp2 + multi_normal_log(yt[j],zero2, Sigma2);
  increment_log_prob(log_mix(pi,lp1,lp2));
}
```

A.8 Trace plots and summary statistics

For the multitask model, we ran 4 chains for 600 iterations with 200 iterations of warm-up. The effective sample size was above 1000 for each parameter, with the Gelman-Rubin potential scale reduction statistic $\hat{R} \leq 1.01$. Trace plots are shown in Figure A2.

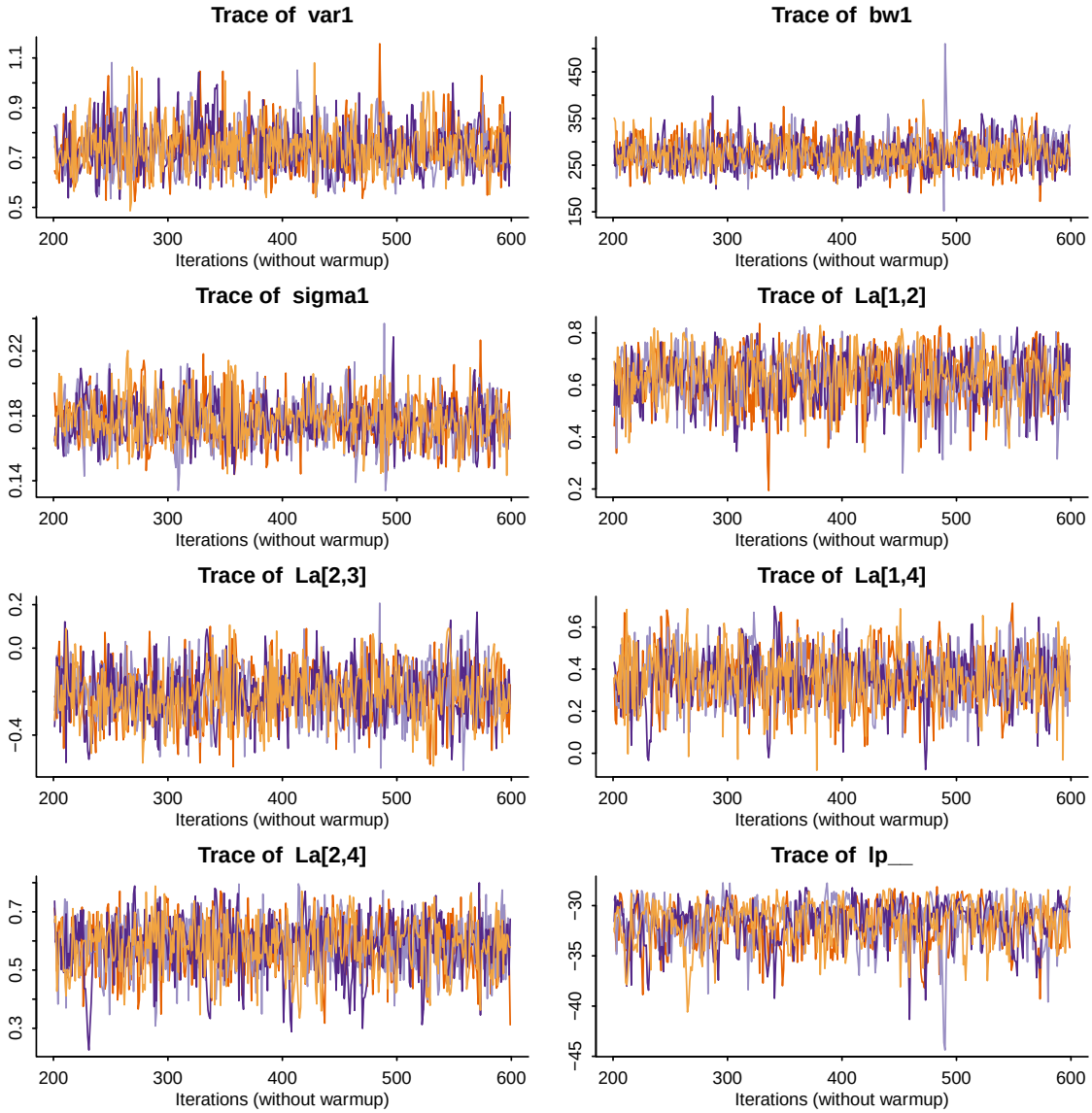

Figure A2: Traces show good convergence for the 4 chains, each of which was run for 600 iterations after 200 iterations of warm-up.