Scalable Gaussian Processes with Grid-Structured Eigenfunctions (GP-GRIEF)

Trefor W. Evans
University of Toronto
trefor.evans@mail.utoronto.ca

Prasanth B. Nair
University of Toronto
pbn@utias.utoronto.ca

Abstract

We introduce a kernel approximation strategy that enables Gaussian process training and inference in $\mathcal{O}(dnp)$ time and $\mathcal{O}(dn)$ storage for a d-dimensional dataset of size n. Our GRIEF kernel consists of p eigenfunctions approximated on a dense Cartesian tensor product grid of inducing points. We show that by exploiting algebraic properties of Kronecker and Khatri-Rao tensor products, computational complexity of the training procedure can be *independent* of the number of inducing points, allowing us to use arbitrarily many to achieve a globally accurate kernel approximation. We benchmark our algorithms on real-world datasets with as many as two-million training points and up to 10^{32} inducing points.

Gaussian process (GP) modelling is a powerful Bayesian approach for classification and regression, however, it is restricted to modestly sized datasets since training and inference require $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ storage, where n is the number of training points [1]. This has motivated the development of approximate GP methods that use a set of m ($\ll n$) inducing points to reduce the computational cost and memory requirements to $\mathcal{O}(m^2 n)$ and $\mathcal{O}(mn)$, respectively [2–5]. However, such techniques perform poorly if too few inducing points are used, and any computational savings are lost on complex datasets that require m to be large. Wilson and Nickisch [6] exploited the structure of inducing points placed on a Cartesian product grid, allowing for $m > n$ while dramatically reducing computational demands over an exact GP. This inducing point structure enables significant performance gains in low-dimensions, however, time and storage complexities scale *exponentially* with the dataset dimensionality, rendering the technique intractable for general learning problems unless a dimensionality reduction procedure is applied. In the present work, a Cartesian product grid of inducing points is also considered, however, we show that these computational bottlenecks can be eliminated by identifying and exploiting further structure of the resulting matrices. The proposed approach leads to a highly scalable algorithm that performs exact training and inference with a non-degenerate GP in $\mathcal{O}(dnp)$ time and $\mathcal{O}(dn)$ memory, where d is the dimensionality of the dataset, and p is the number of eigenfunctions that we will describe next. We emphasize that our complexity is *independent* of m, which can be set arbitrarily high.

Our central idea is to approximate the exact kernel as a finite sum of eigenfunctions that can be found efficiently using a Nyström approximation [5]. This approach is similar to the "Nyström method" of Williams and Seeger [4], however, our final approximation is a valid probabilistic model. In other words, our model is sparse in the kernel eigenfunctions rather than the number of inducing points, which can greatly exceed the size of the training set due to the structure we introduce. This is attractive since it is well-known that eigenfunctions produce the most compact representation among orthogonal basis functions. Although our eigenfunctions are approximate, they convergence in the limit of large n [7]. Additionally, our ability to consider a huge value of m allows us to fill out the input space with inducing points, thereby enabling accurate global approximations of the eigenfunctions, even far from the training data. These basis functions also live in a reproducing kernel Hilbert space, unlike other sparse GPs whose bases have a pre-specified form (e.g. Lázaro-Gredilla

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

et al. [8]). In addition, we show how to recover a non-degenerate kernel by including a correction term that further improves our approximation without affecting computational complexity.

We begin by outlining our approximate kernel for a general distribution of inducing points. We then place the inducing points on a grid and discuss how our method remains computationally attractive even in high-dimensional problems with $m \gg n$. The discussion will focus on solving a linear system with the kernel covariance matrix which is the most expensive part of GP training and inference.

1 Approximate Kernel

Given a kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, we consider its approximation in terms of p eigenfunctions (obtained using the Nyström approximation) written in the following form [5]

$$\widetilde{k}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{p} \left(\frac{1}{\sqrt{\lambda_i}} \mathbf{K}_{\mathbf{x},\mathrm{U}} \mathbf{q}_i \right) \left(\frac{1}{\sqrt{\lambda_i}} \mathbf{K}_{\mathbf{z},\mathrm{U}} \mathbf{q}_i \right) = \mathbf{K}_{\mathbf{x},\mathrm{U}} \mathbf{Q} \mathbf{S}_p^T \mathbf{\Lambda}_p^{-1} \mathbf{S}_p \mathbf{Q}^T \mathbf{K}_{\mathrm{U},\mathbf{z}} \approx k(\mathbf{x}, \mathbf{z}), \qquad (1)$$

where $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$ are d-dimensional inputs; $\mathrm{U} = \{\mathbf{u}_i\}_{i=1}^m$ refers to the set of m inducing point locations; $\mathbf{K}_{\_,\_}$ refers to a matrix of exact kernel evaluations between the two sets in the subscript; $\mathbf{\Lambda}, \mathbf{Q} \in \mathbb{R}^{m \times m}$ are diagonal and unitary matrices containing the eigenvalues and eigenvectors of $\mathbf{K}_{\mathrm{U},\mathrm{U}}$, respectively; λ_i and $\mathbf{q}_i$ denote the ith largest eigenvalue and corresponding eigenvector of $\mathbf{K}_{\mathrm{U},\mathrm{U}}$, respectively; $\mathbf{S}_p \in \mathbb{R}^{p \times m}$ is a sparse selection matrix where $\mathbf{S}_p(i,:)$ contains one value set to unity in the column corresponding to the index of the ith largest value on the diagonal of $\mathbf{\Lambda}$; and we use the shorthand notation $\mathbf{\Lambda}_p = \mathbf{S}_p \mathbf{\Lambda} \mathbf{S}_p^T \in \mathbb{R}^{p \times p}$ to denote a diagonal matrix containing the p largest eigenvalues of $\mathbf{K}_{\mathrm{U},\mathrm{U}}$, sorted in descending order. Our kernel covariance matrix can then be written for a set of n inputs $\mathrm{X} = \{\mathbf{x}_i\}_{i=1}^n$ in our training set as

$$\widetilde{\mathbf{K}}_{\mathrm{X},\mathrm{X}} = \mathbf{K}_{\mathrm{X},\mathrm{U}} \mathbf{Q} \mathbf{S}_p^T \mathbf{\Lambda}_p^{-1} \mathbf{S}_p \mathbf{Q}^T \mathbf{K}_{\mathrm{U},\mathrm{X}}, \qquad (2)$$

which we observe is the same as the covariance matrix from the "Nyström method" of Williams and Seeger [4]. However, unlike the Nyström method, we replace the kernel and not just the covariance matrix to recover a valid probabilistic model (see discussion in [9, 10]).

2 Inducing point Structure

Peng and Qi [5] considered a variant of the approximate kernel in eq. (1), with an optimization scheme applied to find the best m inducing point locations, U, which we do not consider here. Instead, we would like to use *so many* inducing points that optimizing U is unnecessary, we will even consider $m \gg n$. This can be done efficiently by using a product correlation kernel for k and distributing inducing points on a Cartesian tensor product grid as considered by Wilson and Nickisch [6] in their "structured kernel interpolation" (SKI) approach. We do this by taking U to be distributed on a full grid with $\bar{m} = \sqrt[d]{m} \approx \mathcal{O}(10)$ points along each dimension. When this is done, the covariance matrix on the inducing point grid inherits a Kronecker product ($\otimes$) structure, $\mathbf{K}_{\mathrm{U},\mathrm{U}} = \bigotimes_{i=1}^d \mathbf{K}_{\mathrm{U},\mathrm{U}}^{(i)}$, enabling efficient Kronecker matrix algebra to be exploited. For instance, since $\mathbf{K}_{\mathrm{U},\mathrm{U}}^{(i)} \in \mathbb{R}^{\bar{m} \times \bar{m}}$ are small matrices, storage of $\mathbf{K}_{\mathrm{U},\mathrm{U}}$ only requires $\mathcal{O}(dm^{2/d}) = \mathcal{O}(d\bar{m}^2)$ memory. In addition, the eigen-decomposition of $\mathbf{K}_{\mathrm{U},\mathrm{U}}$ requires only $\mathcal{O}(dm^{3/d}) = \mathcal{O}(d\bar{m}^3)$ time and the eigenvector matrix $\mathbf{Q} = \bigotimes_{i=1}^d \mathbf{Q}^{(i)}$ also inherits a Kronecker product structure, enabling matrix-vector products with $\widetilde{\mathbf{K}}_{\mathrm{X},\mathrm{X}}$ in $\mathcal{O}(dm^{(d+1)/d}) = \mathcal{O}(d\bar{m}^{d+1})$ operations (see [11, 12] for details). In low-dimensions, exploiting this structure can be greatly advantageous, however, we can immediately see from the above complexities that the cost of matrix-vector products with $\widetilde{\mathbf{K}}_{\mathrm{X},\mathrm{X}}$ increases *exponentially* in d. The storage requirements will similarly increase exponentially in d since a vector of length $m = \bar{m}^d$ needs to be stored when a matrix-vector product is made with $\widetilde{\mathbf{K}}_{\mathrm{X},\mathrm{X}}$, and $\mathbf{K}_{\mathrm{X},\mathrm{U}}$ requires $\mathcal{O}(\bar{m}^d n)$ storage. This poor scaling poses a serious impediment to the successful application of this approach to high-dimensional datasets. Such restrictions apply similarly to SKI where its use is recommended only for very low-dimensional problems, $d \leqslant 5$ [13].

We now show how to massively decrease time and storage requirements from exponential to *linear* in d by identifying further matrix structure in our problem. From our covariance matrix $\widetilde{\mathbf{K}}_{\mathrm{X},\mathrm{X}}$ in eq. (2)

(and similarly for our kernel in eq. (1)), we discover that the exact cross-covariance matrices between test or train points and inducing points, e.g. $\mathbf{K}_{\mathrm{X,U}}$, admits a row-partitioned Khatri-Rao structure

$$\mathbf{K}_{\mathrm{X,U}} = \overset{d}{\underset{i=1}{\divideontimes}} \mathbf{K}_{\mathrm{X,U}}^{(i)} = \begin{pmatrix} \mathbf{K}_{\mathrm{X,U}}^{(1)}(1,:) \otimes \mathbf{K}_{\mathrm{X,U}}^{(2)}(1,:) \otimes \cdots \otimes \mathbf{K}_{\mathrm{X,U}}^{(d)}(1,:) \\ \mathbf{K}_{\mathrm{X,U}}^{(1)}(2,:) \otimes \mathbf{K}_{\mathrm{X,U}}^{(2)}(2,:) \otimes \cdots \otimes \mathbf{K}_{\mathrm{X,U}}^{(d)}(2,:) \\ \vdots \qquad\qquad \vdots \qquad\qquad \ddots \qquad\qquad \vdots \\ \mathbf{K}_{\mathrm{X,U}}^{(1)}(n,:) \otimes \mathbf{K}_{\mathrm{X,U}}^{(2)}(n,:) \otimes \cdots \otimes \mathbf{K}_{\mathrm{X,U}}^{(d)}(n,:) \end{pmatrix}, \tag{3}$$

where $*$ is the Khatri-Rao product (see [14] for details on this tensor product). Since $\mathbf{K}_{\mathrm{X,U}}^{(i)}$ are only of size $n \times \bar{m}$, the storage of $\mathbf{K}_{\mathrm{X,U}}$ has decreased to $\mathcal{O}(dn\bar{m}) \approx \mathcal{O}(dn)$. Further, by exploiting both Kronecker and Khatri-Rao matrix algebra, we can show that matrix-vector products can be made with $\widetilde{\mathbf{K}}_{\mathrm{X,X}}$ in $\mathcal{O}(dnp)$ time and only $\mathcal{O}(dn)$ storage is required for the entire GP training and inference process. To do this, we show how matrix-vector products can be made with $\widetilde{\mathbf{K}}$ within the promised complexity so that we can use a conjugate gradient method to solve a linear system with $\widetilde{\mathbf{K}} + \sigma^2\mathbf{I}$, where $\sigma^2 > 0$ is the variance of Gaussian noise in the training data[1].

To take fast matrix-vector products with $\widetilde{\mathbf{K}}_{\mathrm{X,X}}$, observe that $\mathbf{K}_{\mathrm{X,U}}\mathbf{Q} = *_{i=1}^{d} \mathbf{K}_{\mathrm{X,U}}^{(i)}\mathbf{Q}^{(i)}$ is a row-partitioned Khatri-Rao product matrix (using theorem 2 of [14]), and that $\mathbf{S}_p^T$ can be written as a column-partitioned Khatri-Rao product matrix. It can then be shown that a matrix-vector product with $(\mathbf{K}_{\mathrm{X,U}}\mathbf{Q})\mathbf{S}_p^T$ (a matrix product of row- and column-partitioned Khatri-Rao matrices) can be made in $\mathcal{O}(dnp)$ time and using no more than $\mathcal{O}(n)$ additional memory. This is done by expanding one (or several) rows of $\mathbf{K}_{\mathrm{X,U}}\mathbf{Q}\mathbf{S}_p^T$ at a time; further details can be found in the supplementary material. The same method can be used for fast matrix-vector multiplication with $\mathbf{S}_p\mathbf{Q}^T\mathbf{K}_{\mathrm{U,X}} = \mathbf{S}_p(\mathbf{K}_{\mathrm{X,U}}\mathbf{Q})^T$. Noting that $\mathbf{\Lambda}_p \in \mathbb{R}^{p\times p}$ is diagonal, it is evident that we do not exceed the promised complexities.

What results is a model with basis eigenfunctions that are structured on a grid of inducing points, and although m increases exponentially in d, the cost of GP training and inference is not affected. We call the resulting model GP-GRIEF (GP with GRId-structured Eigen-Functions).

It is possible to introduce a correction term to $\widetilde{k}$ (eq. (1)), giving

$$\widetilde{k}_{\mathrm{cor}}(\mathbf{x},\mathbf{z}) = \widetilde{k}(\mathbf{x},\mathbf{z}) + \delta_{\mathbf{x},\mathbf{z}}(k(\mathbf{x},\mathbf{z}) - \widetilde{k}(\mathbf{x},\mathbf{z})), \tag{4}$$

where $\delta_{\mathbf{x},\mathbf{z}} = 1$ if $\mathbf{x} = \mathbf{z}$, else 0. The kernel $\widetilde{k}_{\mathrm{cor}}$ then has an infinite number of basis functions (provided k does also) and our resulting GP will be non-degenerate. This correction term requires the diagonal of $\widetilde{\mathbf{K}}_{\mathrm{X,X}}$ and it can be shown that this can be computed in $\mathcal{O}(dnp)$ time and with no more than $\mathcal{O}(dn)$ storage by exploiting the matrix structure previously outlined (details in the supplement). Therefore, adding this correction does not affect the computational complexity of GP training.

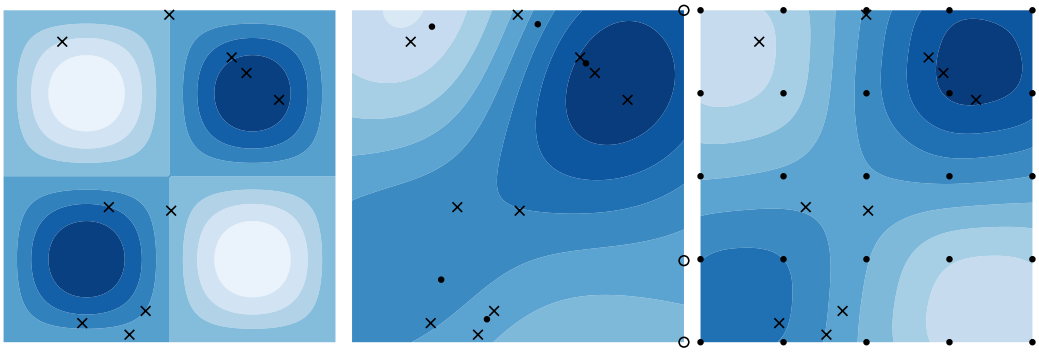
3 Experimental Studies

Figure 1 shows a comparison of the FITC method [3] versus the proposed GP-GRIEF method on a two-dimensional test problem, using a squared exponential kernel and $n = 10$ training points. FITC (using GPy [15]) with $m = 8$ achieves a root-mean squared error (RMSE) of 0.47 on a test set, whereas GP-GRIEF with only half the number of basis functions, $p = 4$, achieves an RMSE of 0.34, identical to that of an exact GP. GP-GRIEF has a dense grid of inducing points whose quantity does not affect training or inference complexity, as shown in section 2.

We next discuss some early results on large real-world regression datasets from the UCI repository. We report the mean and standard deviation of the RMSE from 10-fold cross validation[2] along with the mean training time per fold, which includes hyperparameter estimation through log marginal likelihood maximization using a machine with two E5-2680 v3 processors. For all runs, we use a squared-exponential kernel with automatic relevance determination (SE-ARD) and we vary $\bar{m}$ and p. Also presented are test errors reported by Yang et al. [16] for the same train-test splits with Fastfood

[1] The other computationally demanding task of GP training and inference is computation of $\log|\widetilde{\mathbf{K}}_{\mathrm{X,X}} + \sigma^2\mathbf{I}_n|$ which we can approximate in $\mathcal{O}(d\bar{m}^3)$ using the Nyström approximation from [6].

[2] 90% train, 10% test per fold. We use folds from `https://people.orie.cornell.edu/andrew/code/`.



(a) Test Data. (b) FITC, $m = 8$. (c) GP-GRIEF, $m = 25,\ p = 4$.

Figure 1: Comparison of FITC vs GP-GRIEF on the function $f(x, y) = \sin(x)\sin(y)$. The crosses denote the $n = 10$ training point positions whose responses are corrupted with $\mathcal{N}(0, 0.1)$ noise. Dots denote inducing point locations within bounds and circles show the direction of those outside bounds.

			GP-GRIEF				Yang et al. [16]
Dataset	n	d	p	$m{=}\bar{m}^d$	Time (mins)	RMSE	RMSE
Pumadyn	8192	32	100	10^{32}	1.6	0.21 ± 0.00	0.20 ± 0.00
			1000	10^{32}	9.3	0.20 ± 0.00	
Elevators	16599	18	100	5^{18}	0.7	0.097 ± 0.001	0.090 ± 0.001
			100	10^{18}	0.8	0.096 ± 0.001	
			1000	10^{18}	6	0.092 ± 0.002	
			5000	10^{18}	30.9	0.091 ± 0.001	
Protein	45730	9	100	10^{9}	0.9	0.63 ± 0.01	0.53 ± 0.01
			5000	10^{9}	34.2	0.58 ± 0.01	
Electric	2049280	11	100	10^{11}	65.6	0.068 ± 0.002	0.120 ± 0.120

Table 1: Mean and standard deviation of test error and average training time (including hyperparmater estimation) from 10-fold cross validation (90% train, 10% test per fold) on UCI regression datasets.

expansions to approximate the SE-ARD kernel. GP-GRIEF shows comparable test error to Yang et al. [16] on all datasets but performs considerably better on the Electric dataset with two-million training points. Further, only $p{=}100$ basis functions were needed for this high quality model on the Electric dataset and the entire training process took just about one hour.

Results for the Elevators dataset demonstrate the effective independence of training time on the number of inducing points: comparing the first and second rows of the Elevator dataset, we see that training time increases by only $0.14\times$ when m is increased by $262143\times$. This complexity independence allows enormous numbers of inducing points to be used; we use $m{=}10^{32}$ for the Pumadyn dataset demonstrating the efficiency of the matrix algebra employed since storing a double-precision vector of this length requires 800 billion Zettabytes, far more than all hard-disk space on earth, combined. Lastly, we demonstrate linear scaling with respect to p, in-fact the scaling is usually sub-linear.

4 Conclusion & Future Work

Our new technique, GP-GRIEF, has been outlined along with some promising initial results on real-world datasets where we demonstrated GP training and inference in $\mathcal{O}(dnp)$ time with $\mathcal{O}(dn)$ storage. We demonstrated that our complexities are independent of m, allowing us to use up to 10^{32} inducing points for an accurate global kernel approximation. In the future, we would like to consider some kernel parametrizations used by Peng and Qi [5] which may conflict with our goal of approximating the exact kernel k, however, may enable a better data-fit when $p \ll n$. Also, we will explore applications of our approximate eigenfunctions to accelerate other kernel methods.

Acknowledgements: Work funded by an NSERC Discovery Grant and the Canada Research Chairs program.

References

[1] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[2] A. J. Smola and P. Bartlett. "Sparse greedy Gaussian process regression". In: *Advances in Neural Information Processing Systems*. 2001, pp. 619–625.

[3] E. Snelson and Z. Ghahramani. "Sparse Gaussian processes using pseudo-inputs". In: *Advances in Neural Information Processing Systems* 18 (2006), p. 1257.

[4] C. K. I. Williams and M. Seeger. "Using the Nyström method to speed up kernel machines". In: *Advances in Neural Information Processing Systems*. 2001, pp. 682–688.

[5] H. Peng and Y. Qi. "EigenGP: Gaussian Process Models with Adaptive Eigenfunctions." In: *International Joint Conference on Artificial Intelligence*. 2015, pp. 3763–3769.

[6] A. G. Wilson and H. Nickisch. "Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP)". In: *Proceedings of The 32nd International Conference on Machine Learning*. 2015, pp. 1775–1784.

[7] C. T. H. Baker. *The numerical treatment of integral equations*. Oxford: Clarendon press, 1977.

[8] M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. Figueiras-Vidal. "Sparse spectrum Gaussian process regression". In: *Journal of Machine Learning Research* 11.6 (2010), pp. 1865–1881.

[9] J. Quiñonero-Candela and C. E. Rasmussen. "A unifying view of sparse approximate Gaussian process regression". In: *Journal of Machine Learning Research* 6.12 (2005), pp. 1939–1959.

[10] C. K. I. Williams, C. E. Rasmussen, A. Scwaighofer, and V. Tresp. *Observations on the Nyström method for Gaussian process prediction*. Tech. rep. University of Edinburgh and University College London, 2002.

[11] C. F. Van Loan. "The ubiquitous Kronecker product". In: *Journal of Computational and Applied Mathematics* 123.1 (2000), pp. 85–100.

[12] Y. Saatçi. "Scalable inference for structured Gaussian process models". PhD thesis. University of Cambridge, 2011.

[13] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. "Deep kernel learning". In: *Artificial Intelligence and Statistics*. 2016, pp. 370–378.

[14] S. Liu and G. Trenkler. "Hadamard, Khatri-Rao, Kronecker and other matrix products". In: *International Journal of Information and Systems Sciences* 4.1 (2008), pp. 160–177.

[15] GPy. *GPy: A Gaussian process framework in python*. `http://github.com/SheffieldML/GPy`. since 2012.

[16] Z. Yang, A. J. Smola, L. Song, and A. G. Wilson. "À la Carte – Learning Fast Kernels". In: *Artificial Intelligence and Statistics*. 2015, pp. 1098–1106.

[17] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. "A set of level 3 basic linear algebra subprograms". In: *ACM Transactions on Mathematical Software (TOMS)* 16.1 (1990), pp. 1–17.

Supplementary Material

Fast Matrix-Vector Product Algorithm

In this section we describe how fast matrix-vector products can be made with $\widetilde{\mathbf{K}}_{\mathrm{X,X}} + \sigma^2\mathbf{I}_n$ to enable GP training and inference in $\mathcal{O}(dnp)$ time and with $\mathcal{O}(dn)$ storage. In section 2, we discussed that this comes down to performing matrix-vector multiplication with a matrix that is a product of row- and column- Khatri-Rao matrices, such as $(\mathbf{K}_{\mathrm{X,U}}\mathbf{Q})\mathbf{S}_p^T$. Algorithm `mvKRrowcol` describes how this is done. The algorithm effectively computes the matrix row-by-row in the inner loop, however, it can be easily modified to compute multiple rows simultaneously by considering j to be a vector of indices whose length corresponds to the number of rows that are possible to fit in available memory. This increases the size of the matrix operations considerably, thereby offering opportunities for parallelization and the use of BLAS level-3 routines [17].

Algorithm `mvKRrowcol` Computes matrix-vector product $\mathbf{RCb}$ where $\mathbf{R} \in \mathbb{R}^{n\times m}$, $\mathbf{C} \in \mathbb{R}^{m\times p}$ are Khatri-Rao products of row- and column-partitioned matrices, respectively. Requires $\mathcal{O}(dnp)$ time if we assume that one of $\mathbf{R}$ or $\mathbf{C}$ are dense and the other is sparse with one non-zero per row. Also, the algorithm only requires $\mathcal{O}(n)$ additional memory. Note that $\circ$ denotes the element-wise, Hadamard product.

Input: $\mathbf{b} \in \mathbb{R}^p$, $\mathbf{R} = \overset{d}{\underset{i=1}{\circledast}} \mathbf{R}^{(i)} \in \mathbb{R}^{n\times m}$, $\mathbf{R}^{(i)} \in \mathbb{R}^{n\times \overline{m}}$, $\mathbf{C} = \overset{d}{\underset{i=1}{\circledast}} \mathbf{C}^{(i)} \in \mathbb{R}^{m\times p}$, $\mathbf{C}^{(i)} \in \mathbb{R}^{\overline{m}\times p}$
Output: $\mathbf{f} = \mathbf{RCb} \in \mathbb{R}^n$
for $j = 1$ **to** n **do**
  $\mathbf{t} = \mathbf{R}^{(1)}(j,:)\mathbf{C}^{(1)}$
  for $i = 2$ **to** d **do**
    $\mathbf{t} = \mathbf{t} \circ \mathbf{R}^{(i)}(j,:)\mathbf{C}^{(i)}$
  end for
  $\mathbf{f}(j) = \mathbf{tb}$
end for

Fast Computation of Kernel Correction

We discuss how the kernel correction term in eq. (4) can be efficiently implemented to enable GP training and inference within $\mathcal{O}(dnp)$ time and with $\mathcal{O}(dn)$ storage. The resulting covariance matrix from this corrected kernel (eq. (4)) simply adds a diagonal term to $\widetilde{\mathbf{K}}_{\mathrm{X,X}}$ and so, it does not affect the time for matrix-vector products, however, we need to show that this diagonal term can be computed within the promised complexities.

Computing the diagonal correction term simply requires the diagonal of $\widetilde{\mathbf{K}}_{\mathrm{X,X}}$ which we show how to efficiently compute in algorithm `rr_cov_diag` in $\mathcal{O}(dnp)$ time. We can see that this is a modification of algorithm `mvKRrowcol` where the inner loop multiplication only needs to consider the jth column of $\mathbf{C}$.

Since this diagonal correction is cheap to compute and leads to a non-degenerate GP, we find that it is always worthwhile to consider.

Algorithm `rr_cov_diag` Computes the diagonal of the covariance matrix formed by the kernel $\widetilde{k}$ (from eq. (1)); $\texttt{diag}(\widetilde{\mathbf{K}}_{\mathrm{X,X}}) = \texttt{diag}\big(\mathbf{K}_{\mathrm{X,U}}\mathbf{Q}\mathbf{S}_p^T\mathbf{\Lambda}_p^{-1}\mathbf{S}_p\mathbf{Q}^T\mathbf{K}_{\mathrm{U,X}}\big) = \texttt{diag}((\mathbf{RC})^T\mathbf{\Lambda}_p^{-1}(\mathbf{RC}))$, where we have denoted the row-partitioned Khatri-Rao product matrix $\mathbf{R} = \mathbf{S}_p \in \mathbb{R}^{p\times m}$, and the column-partitioned Khatri-Rao product matrix $\mathbf{C} = \mathbf{Q}^T\mathbf{K}_{\mathrm{U,X}} \in \mathbb{R}^{m\times n}$ for notational convenience. This computation requires $\mathcal{O}(dnp)$ time and only $\mathcal{O}(n)$ additional memory. Note that $\circ$ denotes the element-wise, Hadamard product.

Input: $\mathbf{R} = \overset{d}{\underset{i=1}{\circledast}}\, \mathbf{R}^{(i)} \in \mathbb{R}^{p\times m}$, $\mathbf{R}^{(i)} \in \mathbb{R}^{p\times \overline{m}}$,

$\mathbf{C} = \overset{d}{\underset{i=1}{\circledast}}\, \mathbf{C}^{(i)} \in \mathbb{R}^{m\times n}$, $\mathbf{C}^{(i)} \in \mathbb{R}^{\overline{m}\times n}$, $\mathbf{\Lambda}_p^{-1} \in \mathbb{R}^{p\times p}$, diagonal.

Output: $\mathbf{f} = \texttt{diag}((\mathbf{RC})^T\mathbf{\Lambda}_p^{-1}(\mathbf{RC})) \in \mathbb{R}^n$

$\mathbf{f} = \mathbf{0} \in \mathbb{R}^n$

for $j = 1$ **to** p **do**

  $\mathbf{t} = \mathbf{R}^{(1)}(j,:)\mathbf{C}^{(1)}$

  for $i = 2$ **to** d **do**

    $\mathbf{t} = \mathbf{t} \circ \mathbf{R}^{(i)}(j,:)\mathbf{C}^{(i)}$

  end for

  $\mathbf{f} = \mathbf{f} + \mathbf{\Lambda}_p^{-1}(j,j)(\mathbf{t} \circ \mathbf{t})$

end for