Sparse Inference for Bayesian Quadrature

Philipp Hummel

February 17, 2020

Abstract

Probabilistic numeric integration techniques, also known as Bayesian quadrature, are a class of algorithms for the numeric approximation of integrals that do not have an analytical solution. These methods rely heavily on Gaussian processes (GP) and therefore inherit the computational bottleneck of the covariance matrix inversion with cubic complexity in the number of evaluation points of the integrand. In this work, we make use of the Wendland kernel to reduce the computational effort by inducing a sparse covariance matrix. To make the Wendland kernel usable for Bayesian quadrature, we provide analytic integrals for it (against a uniform measure) that have been lacking in the literature so far. We perform a computational experiment on toy datasets and show that the induced sparse covariance matrix can speed-up the computations by a large factor for suitable data sets. Additionally, we investigate further tricks and their effect on computational efficiency.

1 Introduction

Integration is a central part of the mathematical toolbox and enables applications in many parts of science and engineering. Within machine learning, one particular use case for integration is Bayesian inference and prediction where it is needed for the normalization of the posterior distribution and defines the posterior predictive distribution. The arising integrals, however, cannot be solved analytically for many interesting cases. Therefore in practice, methods to approximate the integrals need to be used.

Historically, the first class of approximate integration methods has been (classic) numeric integration. In the most basic form, this amounts to approximating the integral of a function with a Riemann sum (think roughly: partitioning the support of the function into equally spaced intervals, computing the function value at interval points and summing the areas of the rectangles with height determined by the function values at the grid points). Only slightly more sophisticated is the trapezoidal rule. Here, the area is not a sum of rectangles, but a sum of trapezoids, that arise by linearly interpolating the function values at the interval points. Many more sophisticated classic integration techniques exist, but they all suffer very strongly from the curse of dimensionality. When the number of dimensions of the support of the function to be integrated (the integrand) grows (linearly), the number of evaluation points on an equally spaced grid grows exponentially. Since that means that the wall clock time to achieve a similar level of accuracy also grows exponentially, classic numeric integration techniques are rarely used for problems with more than five dimensions. Another popular family of approximate integration methods is Monte Carlo Integration and particularly Markov Chain Monte Carlo. These methods are stochastic (but not probabilistic in the sense of probabilistic numerics) and scale much better to high dimensional problems. They are currently among the most used methods for Bayesian inference.

With the field of probabilistic numerics, there has also come probabilistic numeric integration methods known as Bayesian quadrature. These methods, in their most basic form, work by emulating the integrand of concern with a Gaussian process (GP) that describes a probability distribution over functions given the evaluations of the integrand. Since the Gaussian distribution is closed under linear operations, integrating this Gaussian process will yield a normal distribution with an expected value and a variance that describe our knowledge about the value of the integral. The probabilistic framework allows obtaining new evaluation locations in a decision-theoretic manner e.g. to minimize the variance of the integral distribution [1]. Thereby, allowing a very high sample efficiency. Interestingly, with this probabilistic numeric interpretation for integration, some classic numeric integration methods can be recovered. For example, the trapezoidal rule can be interpreted as the maximum a posteriori estimate of Bayesian quadrature with a Wiener process prior [1]. This way of looking at the classical numeric method showcases the power of the probabilistic view. By realizing that the trapezoidal rule is the maximum a posteriori estimate obtained by starting with a Wiener process prior, we can understand the assumptions governing the trapezoidal rule in a new light. In particular, using the Wiener process prior means not assuming differentiability of the integrand. However, often we know that our integrand is differentiable and we can make stronger prior assumptions that might lead to better estimation results.

In practice, we desire methods that approximate integrals to a reasonable degree of accuracy quickly in wall clock time. When performing Bayesian quadrature prior assumptions about the integrand can lead to very high efficiency per sample (if the prior assumptions are correct). But working with the underlying Gaussian processes introduces a great computational overhead since the inverse of the $N \times N$ covariance matrix is important in several of the necessary GP equations and computing its inverse has computational costs of $O(N^3)$. In practice, the inverse is usually not computed directly, since indirect, but faster methods exist to compute the necessary quantities. Two of the most prominent of those indirect methods are the Cholesky decomposition [2] and conjugate gradients [3].

In this work, we consider an additional practical speed-up, that has the potential to reduce the associated computational costs even more. Matrices that are sufficiently sparse (i.e. that contain many zeros) can be inverted much faster. We can achieve such sparsity in the covariance matrix by using a covariance function (also called kernel) with finite support (i.e. that assigns 0 covariance (as opposed to just a small covariance) to points that are sufficiently far away from each other). In this work, we will use the Wendland kernel as it has exactly this property and can be integrated, which is an important requirement to be able to compute the expected value and variance of the integral of the GP.

2 Using the Wendland Kernel for Bayesian Quadrature

The most standard kernel used for Gaussian process regression is perhaps the squared exponential or Radial Basis Function (RBF) kernel. For two points x and $x' \in \mathbb{R}$ it assigns a covariance that is exponentially decaying with the Euclidean distance of the points $k(x, x') = \theta \exp(-\frac{(x-x')^2}{2\lambda})$, where θ acts as a scaling factor and λ acts as a lengthscale that controls how fast the function is assumed to change in the x-direction. We will instead consider a kernel that is built from Wendland functions. As we will see below this kernel assigns similar covariances but has advantageous computational properties.

The Wendland functions arise out of some principled computational speed requirements for a

radial basis function $\phi(r)$ in the field of applied mathematics. In particular, the first requirement is that ϕ must be compactly supported on the interval $[0, 1]$ (i.e. $\phi(r) = 0 \; \forall r \notin [0, 1]$). Thinking of r as some kind of distance between two locations (r is the radius in the sense of radial basis functions) this will make sure that locations that are further apart than 1 do not interact. This kind of sparsity is useful for computational efficiency in many applications and we will use it here to construct a sparse covariance matrix. Additionally, the function must be easily evaluable and is therefore chosen to be

$$\phi(r) = \begin{cases} \text{poly}(r), & \text{if } 0 \leq r \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

where $\text{poly}(r)$ is the polynomial of minimal degree, such that ϕ still meets some smoothness criterion. For a space with dimensionality 1, the Wendland functions with a smoothness constraint (e.g. belonging to C^0) are [4]:

Function	Smoothness
$\phi(r) = (1-r)_+$	C^0
$\phi(r) = (1-r)_+^3(3r+1)$	C^2
$\phi(r) = (1-r)_+^5(8r^2+5r+1)$	C^4

, where $(a)_+ := \max(0, a)$

These Wendland functions can easily be turned into a 1D kernel by using $r = |x - x'|$ for two input locations x and x'. See figure 1a) for a visualization of the three different smoothness constrained kernels. Additionally, we want to be able to control the support of the Wendland function by a lengthscale variable λ similar to that of the squared exponential kernel. We can achieve this by rescaling the distance $|x - x'|$ to $\frac{|x-x'|}{\lambda}$. Furthermore, a scaling factor θ can be added similarly to the RBF case yielding $k(x, x') = \theta\phi(\frac{|x-x'|}{\lambda})$.

Obtaining higher dimensional Wendland functions is possible but will yield more complicated expressions. For our purposes, we can regard all dimensions as independent and use the product of the Wendland functions over individual dimensions. For N_D-dimensional vectors $x = [x_1, \ldots, x_{N_D}]$ and $x' = [x'_1, \ldots, x'_{N_D}]$ this means using

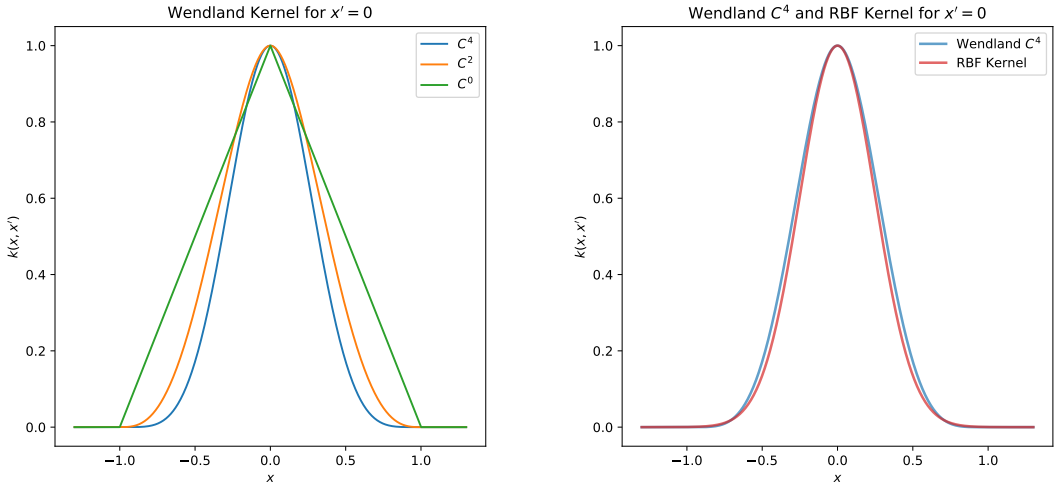
$$k(x, x') = \theta \prod_{d=1}^{N_D} \phi(\frac{|x_d - x'_d|}{\lambda_d})$$

where we also introduced an individual lengthscale λ_d for each dimension[1]. Figure 1b) compares the Wendland kernel with the squared exponential kernel. They are almost indistinguishable but the Wendland kernel has much nicer computational properties.

Analytic Integrals for the Wendland Kernel

Having obtained a kernel with good computational properties does not yet mean that we can use it for Bayesian quadrature. For it to be useful we need to be able to get an analytic expression for the distribution of the integral value. According to [5] (and backed by [6]), the Wendland kernel indeed has a closed-form solution for the integral when integrated against a uniform measure and thus qualifies for Bayesian quadrature; yet to the best of our knowledge an analytic expression for

[1] An implementation of the Wendland kernel can be found in python script 1. Additionally, the Wendland kernel has been integrated into GPyTorch and can be found at this pull request https://github.com/cornellius-gp/gpytorch/pull/957



(a) (b)

Figure 1: Visualization of the Wendland kernel with x' fixed at 0. **a)** 1D Wendland kernel with three different smoothness constraints. **b)** Comparison of the C^4 Wendland kernel and the Squared Exponential kernel. Visually the two kernels are almost identical.

the integral equations is lacking in the literature. Therefore one of the contributions of this work is to provide these results.

To compute the distribution of the integral value resulting from using a Wendland kernel, we first review the general case of vanilla Bayesian quadrature. We know that the distribution of the integral value F is Gaussian since the integrand f is modelled by a Gaussian process and integration is a linear operation (and Gaussian distributions are closed under linear operations). The expected value E and variance V of this distribution are (from [7] (notation heavily adapted)):

$$\begin{aligned} E_{f|D}[F] &= \int m_D(x)dx \\ V_{f|D}[F] &= \iint C_D(x, x')dxdx' \end{aligned} \tag{1}$$

where $m_D(x)$ and $C_D(x, x')$ are the posterior mean and the posterior covariance of the underlying GP on the integrand and $x \in \mathbb{R}^{N_D}$. The mean and the covariance are standardly defined as:

$$\begin{aligned} m_D(x) &= k(x, X)K^{-1}y \\ C_D(x, x') &= k(x, x') - k(x, X)K^{-1}k(X, x') \end{aligned} \tag{2}$$

where $X = \{x_1, \ldots, x_N\} \in \mathbb{R}^{N \times N_D}$ are the locations of the N samples obtained and $y = \{f(x_1), \ldots, f(x_N)\} \in \mathbb{R}^N$ their corresponding function values and X and y are summarized as D (Data). $K \in \mathbb{R}^{N \times N}$ denotes the covariance matrix (also called kernel gram matrix).

To find the desired mean and variance, we need to plug equations 2 into 1, use the Wendland kernel, and solve the integrals.

Let's start with the expected value:

$$\begin{aligned} E_{f|D}[F] &= \int m_D(x)dx \\ &= \int k(x,X)K^{-1}ydx \\ &= \int k(x,X)wdx \qquad \text{defining } \underset{N\times 1}{w} = \underset{N\times N}{K}^{-1}\underset{N\times 1}{y} \\ &= \int \sum_{i=1}^{N} w_i k(x,x_i)dx \\ &= \sum_{i=1}^{N} w_i \int k(x,x_i)dx \end{aligned} \tag{3}$$

So we find that we will need to solve $\int k(x,x_i)dx$.

Looking at the variance:

$$\begin{aligned} V(F) &= \iint k(x,x') - k(x,X)K^{-1}k(X,x')dxdx' \\ &= \iint k(x,x')dxdx' - \iint k(x,X)K^{-1}k(X,x')dxdx' \\ &= \iint k(x,x')dxdx' - \sum_{ij} \iint k(X_i,x')k(x,X_j)K_{ji}^{-1}dxdx' \\ &= \iint k(x,x')dxdx' - \sum_{ij} K_{ji}^{-1} \int k(X_i,x')dx' \int k(x,X_j)dx \end{aligned} \tag{4}$$

we find that we need to solve the three different integrals $\iint k(x,x')dxdx'$, $\int k(X_i,x')dx'$ and $\int k(x,X_j)dx$, where the last two are identical due to the symmetry of the kernel.

Let's start with the latter integral, introducing a user-defined, rectangular lower and upper bound $l = [l_1,\ldots,l_{N_D}]; u = [u_1,\ldots,u_{N_D}]$ and replacing X_j with an arbitrary but fixed point $p \in [l,u]$ (for slightly easier readability):

$$\begin{aligned} \int_l^u k(x,p)dx &= \int_l^u \theta \prod_{d=1}^{N_D} \phi(\frac{|x_d - p_d|}{\lambda_d})dx \\ &= \theta \prod_{d=1}^{N_D} \int_{l_d}^{u_d} \phi(\frac{|x_d - p_d|}{\lambda_d})dx_d \end{aligned} \tag{5}$$

Now to solve the one dimensional integral $\int_{l_d}^{u_d} \phi(\frac{|x_d-p_d|}{\lambda_d})dx_d$ (We drop all dimensional indices now to avoid notational clutter). First remember that $\phi(r) = (1-r)_+^k \text{poly}(r)$ with $k \in 1,3,5$. To be able to drop the $()_+$ we can simply shift the integral boundaries to only include the region around p, where $(1 - \frac{|x-p|}{\lambda}) \geq 0$. Defining: $a = \max(l, p-\lambda)$ and $b = \min(u, p+\lambda)$, this region becomes $[a,b]$. Additionally, we need to deal with the absolute value. The sign induced by the absolute value will change when $x = p$ and we can split the integral into two integrals for the regions where the sign will be constant.

$$\int_a^b \phi(\frac{|x-p|}{\lambda})dx = \int_a^p \phi(\frac{p-x}{\lambda})dx + \int_p^b \phi(\frac{x-p}{\lambda})dx \tag{6}$$

Finally, we have arrived at integrals over polynomials that can easily be solved. The solutions can be found in Appendix B.1. The polynomials have been integrated with the symbolic math

package SymPy [8] and the solutions numerically verified. The symbolic integration is documented in the notebooks 2, 3, 4. They provide the integrals as copy-pasteable python and latex code. The numerical verification of the integration result (for C^4) is documented in notebook 6.

Next, we need to solve $\int_l^u \int_l^u k(x,x')dxdx'$:

$$\begin{aligned}\int_l^u \int_l^u k(x,x')dxdx' &= \int_l^u \int_l^u \theta \prod_{d=1}^{N_D} \phi(\frac{|x_d - x'_d|}{\lambda_d})dxdx' \\ &= \theta \int_{l_{N_D}}^{u_{N_D}} \dots \int_{l_1}^{u_1} \int_{l_{N_D}}^{u_{N_D}} \dots \int_{l_1}^{u_1} \prod_{d=1}^{N_D} \phi(\frac{|x_d - x'_d|}{\lambda_d})dx_1 \dots dx_{N_D} dx'_1 \dots dx'_{N_D} \\ &= \theta \int_{l_{N_D}}^{u_{N_D}} \dots \int_{l_1}^{u_1} \prod_{d=1}^{N_D} \int_{l_d}^{u_d} \phi(\frac{|x_d - x'_d|}{\lambda_d})dx_d dx'_1 \dots dx'_{N_D} \\ &= \theta \prod_{d=1}^{N_D} \int_{l_d}^{u_d} \int_{l_d}^{u_d} \phi(\frac{|x_d - x'_d|}{\lambda_d})dx_d dx'_d\end{aligned} \tag{7}$$

Now we find that we need to solve the one dimensional double integral $\int_{l_d}^{u_d} \int_{l_d}^{u_d} \phi(\frac{|x_d - x'_d|}{\lambda_d})dx_d dx'_d$ (again dropping the dimensional indices for increased readability). Here it becomes harder to drop the $()_+$ function since for every location of x' in $[l,u]$ x will also range from l to u (whereas above we had a fixed point p). To tackle the problem we differentiate three cases.

Let's first assume that $l + \lambda < u - \lambda$ (i.e. $u - l > 2\lambda$) (Case 1). Next we split the outer integral into three summands:

$$\begin{aligned}\int_l^u \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' &= \int_l^{l+\lambda} \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' \\ &+ \int_{l+\lambda}^{u-\lambda} \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' \\ &+ \int_{u-\lambda}^u \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx'\end{aligned} \tag{8}$$

Dealing with the first summand first, we find that we need to change the inner integral's boundaries to be $\max(l, x' - \lambda) = l$ and $\min(u, x' + \lambda) = x' + \lambda$ to be able to drop the $()_+$ function (Remember: $l \le x' \le l + \lambda$ (first summand) and $l + \lambda < u - \lambda$ (Case 1)). Additionally, we need to deal with the absolute value. The sign induced by the absolute value will change when $x = x'$ and therefore we differentiate between regions where $x < x'$ and $x \ge x'$ and finally get:

$$\int_l^{l+\lambda} \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' = \int_l^{l+\lambda} \left(\int_l^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^{x'+\lambda} \phi(\frac{x-x'}{\lambda})dx \right) dx' \tag{9}$$

Dealing with the second summand, we can use a similar procedure. We find that we need to change the inner integral's boundaries to be $\max(l, x' - \lambda) = x' - \lambda$ and $\min(u, x' + \lambda) = x' + \lambda$ to deal with the $()_+$ function. Again we also need to deal with the absolute value and split the inner integral into two summands:

$$\int_{l+\lambda}^{u-\lambda} \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' = \int_{l+\lambda}^{u-\lambda} \left(\int_{x'-\lambda}^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^{x'+\lambda} \phi(\frac{x-x'}{\lambda})dx \right) dx' \tag{10}$$

Dealing with the third and last summand, we can use the same procedure. We find that the inner integral's boundaries need to be $\max(l, x' - \lambda) = x' - \lambda$ and $\min(u, x' + \lambda) = u$ and that we need to split it into two summands because of the absolute value:

$$\int_{u-\lambda}^u \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' = \int_{u-\lambda}^u \left(\int_{x'-\lambda}^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^u \phi(\frac{x-x'}{\lambda})dx \right) dx' \tag{11}$$

Finally putting it all together: Case 1: $l + \lambda < u - \lambda$; (i.e. $u - l > 2\lambda$):

$$\begin{aligned}\int_l^u \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' = &\int_l^{l+\lambda} \left(\int_l^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^{x'+\lambda} \phi(\frac{x-x'}{\lambda})dx) \right) dx' \\ &+ \int_{l+\lambda}^{u-\lambda} \left(\int_{x'-\lambda}^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^{x'+\lambda} \phi(\frac{x-x'}{\lambda})dx \right) dx' \\ &+ \int_{u-\lambda}^{u} \left(\int_{x'-\lambda}^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^{u} \phi(\frac{x-x'}{\lambda})dx \right) dx'\end{aligned} \tag{12}$$

For the other two cases, Case 2: $\lambda \leq u - l \leq 2\lambda$ and Case 3: $u - l < \lambda$ we can follow a similar procedure to arrive at the solutions:
Case 2: $\lambda \leq u - l \leq 2\lambda$:

$$\begin{aligned}\int_l^u \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' = &\int_l^{u-\lambda} \left(\int_l^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^{x'+\lambda} \phi(\frac{x-x'}{\lambda})dx \right) dx' \\ &+ \int_{u-\lambda}^{l+\lambda} \left(\int_l^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^{u} \phi(\frac{x-x'}{\lambda})dx \right) dx' \\ &+ \int_{l+\lambda}^{u} \left(\int_{x'-\lambda}^{x'} \phi(\frac{x'-x}{\lambda}) + \int_{x'}^{u} \phi(\frac{x-x'}{\lambda})dx \right) dx'\end{aligned} \tag{13}$$

And finally Case 3: $u - l < \lambda$:

$$\int_l^u \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' = \int_l^u \left(\int_l^{x'} \phi(\frac{x'-x}{\lambda})dx + \int_{x'}^{u} \phi(\frac{x-x'}{\lambda})dx \right) dx' \tag{14}$$

Again we arrived at integrals over polynomials. The full solutions to these integrals can be found in Appendix B.2. Copy-pasteable latex and python source code is available in the notebooks 2, 3, 4. The numerical verification of the integration result (for C^4) is documented in notebook 7.

Analytic Integrals for WSABI

Additional to vanilla Bayesian quadrature there is a more advanced Bayesian quadrature algorithm, called WSABI [9], that we worked on. WSABI (Warped Sequential Active Bayesian Integration) assumes that the integrand is nonnegative (as appears e.g. in Bayesian inference) and puts a Gaussian process on the square root of the integrand. The resulting distribution on the actual integrand then follows a Non-Central χ^2 process and thereby becomes intractable. The authors remedy this by approximating the χ^2 process with another GP. Two possible approximation schemes are provided and we focus here on the linearization approximation scheme. One of the advantages of WSABI is improved active sampling which leads to faster convergence in wall clock time. The derivation and solution for the expected value of the integral distribution can be found in the markdown document 9 and corresponding PDF document 10. The integration of the polynomials (for C^0) was performed with SymPy and can be found in notebook 5 where the solutions are available as copy-pastable latex and python code (it can be adapted to the C^2 and C^4 cases). The result has been numerically verified in notebook 8. We could not find a solution for the variance of the integral distribution (since this would have led to too many case distinctions) (but some initial notes can be found in the markdown document 11 and corresponding PDF document 12).

3 Sparse Covariance Matrix Speed-ups

Having found analytic results for the expected value and variance of the normal distribution of the integral, we can now turn to issues of computational efficiency. Since the Wendland kernel induces a sparse covariance matrix, methods that use sparse representations of matrices could potentially bring a speed-up. It is important at this point to realize that there is a conceptual difference between a sparse matrix and a matrix that is represented in a data format especially suitable for sparse matrices (for brevity: sparse data format). A sparse matrix is generally one that contains many zeros and we can quantify its density as the number of zeros contained divided by the total number of elements. The sparsity of the matrix is then $1 -$ density. A sparse data format is one where only the nonzero elements are explicitly represented. An example of this would be to store the matrix as three lists. The first containing the row indices of the m nonzero elements, the second containing the corresponding column indices and the third containing the corresponding values. Additionally, the shape of the matrix needs to be stored (as it cannot be inferred from the three lists). The described data format is called COOrdinate (COO) format. It is useful for constructing sparse matrices and as an illustration but it does not allow for fast matrix-vector operations. After an initial benchmark, we decided to use SciPy's [10] Compressed Sparse Column (CSC) format for the rest of the experiments.

As mentioned in the beginning, in practice the inverse of the covariance matrix is rarely computed explicitly since there are faster, indirect methods. The two most notable methods are the Cholesky decomposition and conjugate gradients (CG). For both of these algorithms, versions that can deal with sparse data formats exist but we will focus on conjugate gradients in the following[2]. We use conjugate gradients to find a solution for the Woodbury vector $w = K^{-1}y$ by rearranging the equation to become $Kw = y$.

In theory, the computational complexity of conjugate gradients is in $O(m\sqrt{\kappa})$ where κ is the condition number of the matrix K and m is the computational complexity of multiplying a vector with the matrix. For dense matrices m is N^2 but for a sparse matrix m is the number of nonzero entries in the matrix. Depending on the sparsity of the covariance matrix, the speedups could, therefore, be quite large. In practice, however, the speed gains depend on the details of the underlying implementation.

To understand the efficiency effects of using the Wendland kernel we ran a computational experiment. It would be useful to compare the timings of CG for different covariance matrix sparsities and also for different numbers of data points N. To make this possible we sampled data points from a Wendland GP and varied the lengthscale of the first dimension λ_1. Shorter lengthscales induce a higher sparsity in the covariance matrix since the support of the kernel is smaller. To get a better feeling for what the sampled data looks like we visualize three different data sets sampled from a 2-D Wendland GP for three different lengthscales λ_1 in Figure 2.

For the actual experiment, we used a 5-D Wendland GP with lengthscales $\lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = 8$ and varied λ_1 to induce varying densities in the covariance matrix. The x-axis was labeled with the resulting density, rather than the values for λ_1 (Remember that the kernel involves a dimension-wise product. Therefore one very short length scale is sufficient to induce a high sparsity). In Figure 3 we plot the timings for 10 independent runs to compute the Woodbury vector

[2]Sparse Cholesky decomposition is not readily available in python. It depends on an external library, SuiteSparse (http://faculty.cse.tamu.edu/davis/suitesparse.html), whose installation procedure comes with some annoyances. In contrast, sparse conjugate gradients is readily available in SciPy and is also the default method used by GPyTorch which is the GP package we chose. Therefore, conjugate gradients was the natural choice.

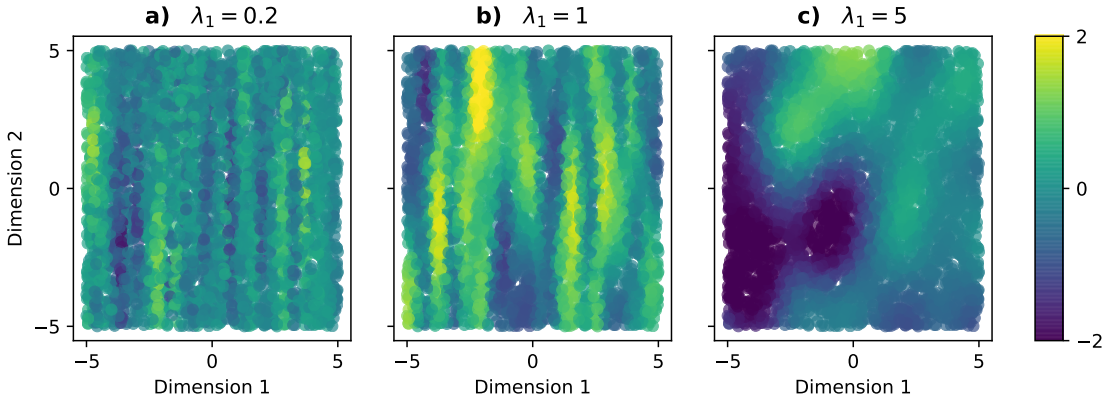


Figure 2: Visualization of $N = 4000$ samples from a 2-D Wendland GP prior with $\lambda_2 = 8$ and varying λ_1. **a)** Extremely short lengthscale for the first dimension ($\lambda_1 = 0.2$). Induced covariance matrix density: $\sim 4\%$. **b)** Intermediate lengthscale for the first dimension ($\lambda_1 = 1$). Induced covariance matrix density: $\sim 18\%$. **c)** Long lengthscale for the first dimension ($\lambda_1 = 5$). Induced covariance matrix density: $\sim 72\%$.

with CG (mean and standard error) for a dense representation of the Wendland GP covariance matrix, a sparse CSC representation of the same matrix and an RBF GP covariance matrix for the same data as a comparison. The covariance hyperparameters λ_i and θ were taken as the known underlying parameters. Four different versions with the number of data points N varying from $[500, 1000, 2000, 4000]$ were executed. For very sparse covariance matrices the sparse representation version of CG runs faster than the one with a dense representation. These speedups can be very large. In the most extreme case we considered ($N = 4000; \lambda_1 = 0.01$ and a covariance matrix density of $\sim 0.2\%$) the sparse algorithm runs ~ 19 times faster than the dense one. For the case of $N = 4000; \lambda_1 = 0.2$ and a density of $\sim 4\%$ roughly corresponding to Figure 2a) (except for the number of dimensions) the speed-up is still a factor of 4. The density at which the dense algorithm becomes faster than the sparse one varies in $[\sim 1\%, \sim 4\%, \sim 13\%, \sim 13\%]$ for N in $[500, 1000, 2000, 4000]$. This shows that sparse representation algorithms can be quite useful for very sparse problems.

Another interesting effect is visible though. For almost all densities considered so far, dense CG runs considerably faster for the Wendland GP than for the RBF GP fitting the same data. This is an unexpected effect that would suggest that convergence can be achieved more easily on the CG problem induced by the Wendland GP. However, comparing the condition numbers shows a larger κ for the Wendland covariance matrix in all cases considered. The condition numbers are visualized in Figure 4 for the $N = 1000$ case but were highly similar for the three other cases (except overall increasing κ for increasing N). This is a surprising finding since the theory predicts that CG should converge faster for lower κ. Additionally, when considering even larger λ_1 (that induce denser Wendland covariance matrices) the run-times for the RBF case actually go down again and fall below the runtimes for the Wendland GP. This behavior is visualized in Figure 5 for the $N = 1000$ case but was highly similar for the other numbers of data points. It appears as if there was some additional factor that slowed down the computation for the RBF case for the intermediate values of λ_1, although the condition numbers are smaller for the RBF than for the Wendland matrices. On a related note, we found that due to the finite representation capacity of floating-point numbers the covariance matrix of the RBF GP also contains many zeros (numbers

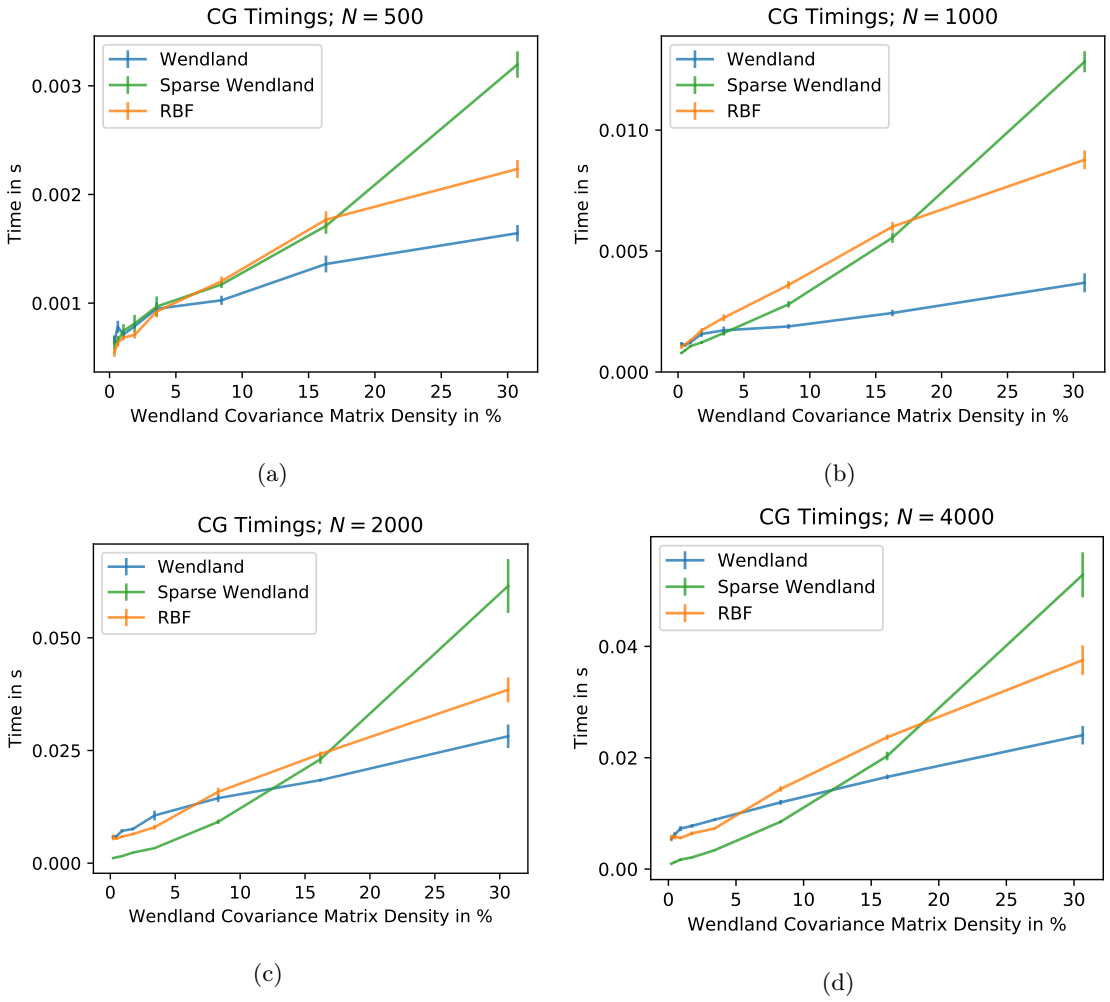


Figure 3: Visualization of the timings of 10 independent runs of CG (mean and standard error) for computing the Woodbury vector for data sets with differing λ_1 which induce different sparsities in the Wendland covariance matrix which are plotted on the x-axis. The data sets in the four sub figures have different sizes $N \in [500, 1000, 2000, 4000]$. Plotted are the timings for sparse CG with the Wendland covariance matrix, dense CG with the same matrix, and dense CG with the RBF covariance matrix as comparison.

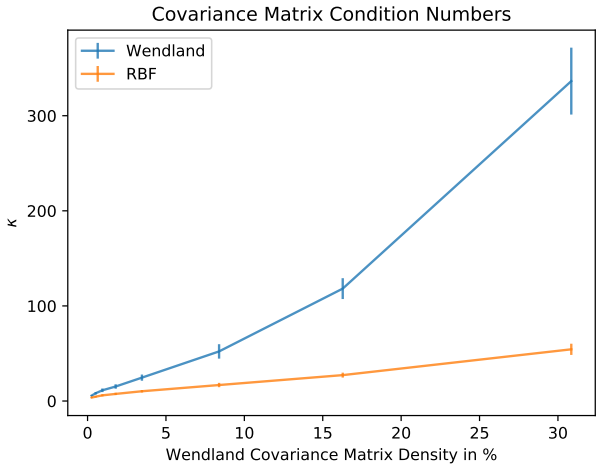


Figure 4: Visualization of the corresponding condition numbers (mean and standard error) to the $N = 1000$ condition from Figure 3. The condition numbers of the Wendland GP's covariance matrix are considerably higher than that of the RBF covariance matrix.

that are too small to be represented just appear as 0). As a rule of thumb the covariance matrix of the RBF GP was about three times denser than that of the Wendland GP (but of course saturated at 100%). When cutting off all numbers below 10^{-10} in the RBF covariance matrix the densities become comparable. Interestingly, doing so speeds up the computation for the RBF case in the range of intermediate λ_1, such that the slow-down in the RBF case in this regime is removed and the RBF case is almost always superior to the dense Wendland case (visualized in Figure 6). This behavior matches much better with what we would expect given the condition numbers.

On an additional note, we found that a simple rescaling trick had a very similar effect. First, we added a small constant to the diagonal of the inverse of the covariance matrix $w = (K+\sigma I)^{-1}y$ and then rearranged the problem to be $w = (\frac{1}{\sigma}K + I)^{-1}\frac{1}{\sigma}y$ (we chose $\sigma = 10^{-6}$). The effect of this rescaling on the timings was very similar to that of cutting the small values in the covariance matrix.

The described slow-down for the RBF case for intermediate λ_1 (and intermediate induced densities) and the remedy of cutting off small values or rescaling the problem is a phenomenon that we have not fully understood. However, we currently hypothesize that floating-point multiplications are slower the larger the range of numbers involved[3]. We came to this idea as another member of the group has made a similar observation previously. It remains an unconfirmed hypothesis though since we could not explore the precise reason in more detail[4].

In summary, we believe that sparse CG in conjunction with the Wendland kernel can lead to large speed gains when used on a problem that induces a very sparse covariance matrix. This is usually given when the integrand changes especially quickly in at least one of the coordinate directions (since the induced λ will then be small). However, we do not expect that using the Wendland kernel alone without a sparse representation algorithm will give consistent speed-ups over using the standard RBF kernel (at least when using either the cut-off or rescaling trick for the RBF case that we explained above).

[3]Or the more extreme the values themselves are

[4]However, we believe this might have something to do with the implementation details of BLAS.

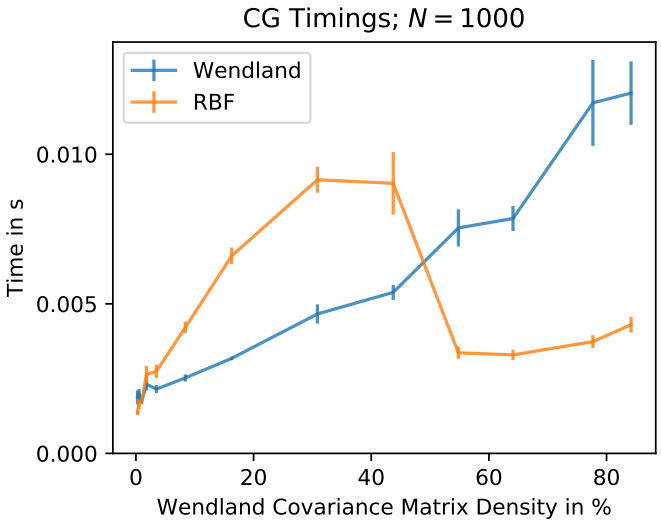


Figure 5: Visualization of the timings of 10 independent runs of CG (mean and standard error) for computing the Woodbury vector for data sets with differing λ_1, which induce different sparsities in the Wendland covariance matrix which are plotted on the x-axis. Compared to Figure 3 this figure also displays larger values for λ_1 and thereby induces higher densities in the Wendland covariance matrix. The timings of the sparse CG for the Wendland case have been omitted. They follow a similar rising trend as in Figure 3 and thereby mask (since the y-axis has to span a large range) the interesting differences between the dense Wendland and RBF timings.

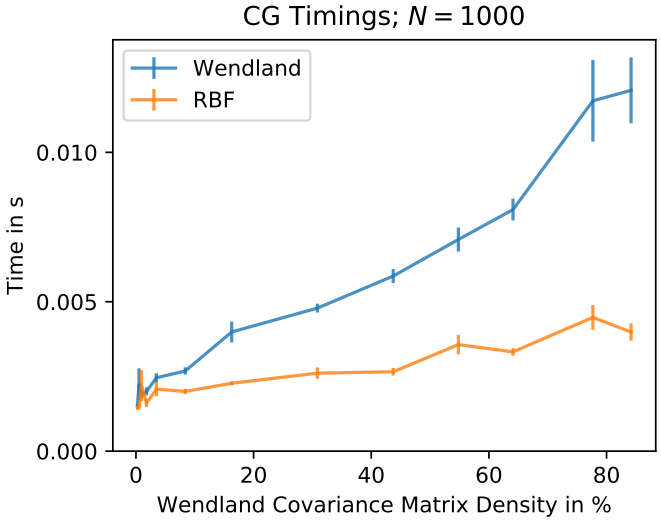


Figure 6: Visualization of the same setting as in Figure 5 with one change. All values below 10^{-10} in the RBF covariance matrix have been cut, thereby inducing a comparable sparsity in both matrices. Apparently this leads to removing the slow-down for the RBF case for intermediate values of λ_1 (i.e. intermediate densities).

4 Conclusion

In this work, we have made use of the Wendland kernel to induce a sparse covariance matrix for Bayesian quadrature. First, we provided the analytic integrals for the expected value and variance of the distribution of the integral for vanilla Bayesian quadrature (and the expected value for WSABI). These had been lacking in the literature but are essential for performing Bayesian quadrature with the Wendland kernel. In the second part, we showed with a computational experiment that the wall clock speed of some of the bottleneck computations of Bayesian quadrature can be greatly reduced when using the Wendland kernel in conjunction with a sparse representation version of the conjugate gradients algorithm. We also investigated another effect that made it appear as if dense CG would run faster with the Wendland induced covariance matrix as compared to the covariance matrix induced by an RBF kernel. We believe now that the initially observed speed advantage of dense CG on the Wendland covariance matrix was more of a slow down of CG with the RBF covariance matrix happening due to a computational artifact. While we do not fully understand this artifact, it was shown to be removed when cutting small values out of the RBF covariance matrix or using the rescaling trick defined above. Therefore, our current recommendation is to use the Wendland kernel in conjunction with sparse CG for data sets that induce a very sparse covariance matrix and use the RBF kernel in conjunction with cutting small values or the rescaling trick for other data sets.

It is important to note, that our experiments were performed on toy data and we believe that further work should be done to address whether the Wendland kernel in conjunction with sparse CG can speed-up the computations on suitable real-world datasets in a similar fashion to the speed-up observed on our toy data.

References

[1] Philipp Hennig and Michael A Osborne. *Probabilistic Numerics*. Unpublished manuscript. 2019.

[2] William H Press et al. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[3] Magnus R Hestenes, Eduard Stiefel, et al. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49.6 (1952), pp. 409–436.

[4] Holger Wendland. *Scattered data approximation*. Vol. 17. Cambridge university press, 2004.

[5] François-Xavier Briol et al. "Probabilistic integration: A role in statistical computation?" In: *Statistical Science* 34.1 (2019), pp. 1–22.

[6] Chris Oates and Mark Girolami. "Control Functionals for Quasi-Monte Carlo Integration". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Ed. by Arthur Gretton and Christian C. Robert. Vol. 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, May 2016, pp. 56–65. URL: http://proceedings.mlr.press/v51/oates16.html.

[7] Carl Edward Rasmussen and Zoubin Ghahramani. "Bayesian monte carlo". In: *Advances in neural information processing systems* (2003), pp. 505–512.

[8] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[9] Tom Gunter et al. "Sampling for inference in probabilistic models with fast Bayesian quadrature". In: *Advances in neural information processing systems*. 2014, pp. 2789–2797.

[10] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* (2020). DOI: https://doi.org/10.1038/s41592-019-0686-2.

A List of Additional Resources

Additional to the report we provide useful resources. This includes Python code, jupyter notebooks, and markdown and corresponding pdf files for mathematical derivations

1. wendland_kernel.py
2. wendland_sympy_integration_vanilla_bq_c0.ipynb
3. wendland_sympy_integration_vanilla_bq_c2.ipynb
4. wendland_sympy_integration_vanilla_bq_c4.ipynb
5. wendland_sympy_integration_WSABI.ipynb
6. numeric_integral_test_vanilla_bq_mean.ipynb
7. numeric_integral_test_vanilla_bq_covariance.ipynb
8. numeric_integral_test_WSABI_mean.ipynb
9. WSABI mean wendland integration.md
10. WSABI mean wendland integration.pdf
11. WSABI covariance wendland integration try.md
12. WSABI covariance wendland integration try.pdf

B Full Equations

B.1 Vanilla Bayesian Quadrature; Kernel Integral

In the following, the solutions to the kernel integral $\int_a^b \phi(\frac{|x-p|}{\lambda})dx = \int_a^p \phi(\frac{p-x}{\lambda})dx + \int_p^b \phi(\frac{x-p}{\lambda})dx$, needed for the expected value of the normal distribution on the integral value, are presented for the Wendland kernel with the three smoothness constraints C^0, C^2 and C^4.

Smoothness class C^0:

$$\begin{aligned}\int_a^b \phi(\frac{|x-p|}{\lambda})dx &= \int_a^p \phi(\frac{p-x}{\lambda})dx + \int_p^b \phi(\frac{x-p}{\lambda})dx \\ &= \int_a^p (1-\frac{p-x}{\lambda})dx + \int_p^b (1-\frac{x-p}{\lambda})dx \\ &= \frac{-\frac{a^2}{2} - a\lambda + ap - \frac{b^2}{2} + b\lambda + bp - p^2}{\lambda}\end{aligned} \tag{15}$$

Smoothness class C^2:

$$\begin{aligned}\int_a^b \phi(\frac{|x-p|}{\lambda})dx &= \int_a^p \phi(\frac{p-x}{\lambda})dx + \int_p^b \phi(\frac{x-p}{\lambda})dx \\ &= \int_a^p (1-\frac{p-x}{\lambda})^3(\frac{3(p-x)}{\lambda}+1)dx + \int_p^b (1-\frac{x-p}{\lambda})^3(\frac{3(x-p)}{\lambda}+1)dx \\ &= \frac{1}{\lambda^4}(\frac{3a^5}{5} + 2a^4\lambda - 3a^4p + 2a^3\lambda^2 - 8a^3\lambda p + 6a^3p^2 - 6a^2\lambda^2 p + 12a^2\lambda p^2 \\ &\quad - 6a^2p^3 - a\lambda^4 + 6a\lambda^2p^2 - 8a\lambda p^3 + 3ap^4 - \frac{3b^5}{5} + 2b^4\lambda + 3b^4p - 2b^3\lambda^2 \\ &\quad - 8b^3\lambda p - 6b^3p^2 + 6b^2\lambda^2 p + 12b^2\lambda p^2 + 6b^2p^3 + b\lambda^4 - 6b\lambda^2p^2 - 8b\lambda p^3 \\ &\quad - 3bp^4 + 4\lambda p^4)\end{aligned} \tag{16}$$

Smoothness class C^4:

$$\begin{aligned}\int_a^b \phi(\frac{|x-p|}{\lambda})dx &= \int_a^p \phi(\frac{p-x}{\lambda})dx + \int_p^b \phi(\frac{x-p}{\lambda})dx \\ &= \int_a^p (1-\frac{p-x}{\lambda})^5(8(\frac{p-x}{\lambda})^2 + 5\frac{p-x}{\lambda}+1)dx \\ &\quad + \int_p^b (1-\frac{x-p}{\lambda})^5(8(\frac{x-p}{\lambda})^2 + 5\frac{x-p}{\lambda}+1)dx \\ &= -\frac{1}{3\lambda^7}(3a^8 + 15a^7\lambda - 24a^7p + 28a^6\lambda^2 - 105a^6\lambda p + 84a^6p^2 + 21a^5\lambda^3 \\ &\quad - 168a^5\lambda^2p + 315a^5\lambda p^2 - 168a^5p^3 - 105a^4\lambda^3 p + 420a^4\lambda^2p^2 - 525a^4\lambda p^3 \\ &\quad + 210a^4p^4 - 7a^3\lambda^5 + 210a^3\lambda^3p^2 - 560a^3\lambda^2p^3 + 525a^3\lambda p^4 - 168a^3p^5 \\ &\quad + 21a^2\lambda^5 p - 210a^2\lambda^3p^3 + 420a^2\lambda^2p^4 - 315a^2\lambda p^5 + 84a^2p^6 + 3a\lambda^7 \\ &\quad - 21a\lambda^5p^2 + 105a\lambda^3p^4 - 168a\lambda^2p^5 + 105a\lambda p^6 - 24ap^7 + 3b^8 - 15b^7\lambda \\ &\quad - 24b^7p + 28b^6\lambda^2 + 105b^6\lambda p + 84b^6p^2 - 21b^5\lambda^3 - 168b^5\lambda^2p - 315b^5\lambda p^2 \\ &\quad - 168b^5p^3 + 105b^4\lambda^3p + 420b^4\lambda^2p^2 + 525b^4\lambda p^3 + 210b^4p^4 + 7b^3\lambda^5 \\ &\quad - 210b^3\lambda^3p^2 - 560b^3\lambda^2p^3 - 525b^3\lambda p^4 - 168b^3p^5 - 21b^2\lambda^5p + 210b^2\lambda^3p^3 \\ &\quad + 420b^2\lambda^2p^4 + 315b^2\lambda p^5 + 84b^2p^6 - 3b\lambda^7 + 21b\lambda^5p^2 - 105b\lambda^3p^4 \\ &\quad - 168b\lambda^2p^5 - 105b\lambda p^6 - 24bp^7 + 56\lambda^2p^6 + 6p^8)\end{aligned} \tag{17}$$

B.2 Vanilla Bayesian Quadrature; Kernel Double Integral

In the following to solutions to the kernel double integral $\int_l^u \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx'$ are presented for the smoothness classes C^0, C^2 and C^4. In the main text, we differentiated between three cases for all smoothness classes:

Case 1: $l + \lambda < u - \lambda$ (i.e. $u - l > 2\lambda$)
Case 2: $\lambda \leq u - l \leq 2\lambda$
Case 3: $u - l < \lambda$
The solutions for Case 1 and Case 2 are equivalent for all three smoothness classes, suggesting that we would not have needed to differentiate between them. Therefore, we introduce here a condensed case distinction:
Case 1*: $u - l \geq \lambda$
Case 2*: $u - l < \lambda$

Smoothness class C^0:

Case 1*:

$$\int_l^u \int_l^u \phi(\frac{|x - x'|}{\lambda}) dx dx' = \frac{\lambda\,(-3l - \lambda + 3u)}{3} \tag{18}$$

Case 2*:

$$\int_l^u \int_l^u \phi(\frac{|x - x'|}{\lambda}) dx dx' = \frac{\frac{l^3}{3} + l^2\lambda - l^2 u - 2l\lambda u + l u^2 + \lambda u^2 - \frac{u^3}{3}}{\lambda} \tag{19}$$

Smoothness class C^2:

Case 1*:

$$\int_l^u \int_l^u \phi(\frac{|x - x'|}{\lambda}) dx dx' = \frac{\lambda\,(-4l - \lambda + 4u)}{5} \tag{20}$$

Case 2*:

$$\begin{aligned}\int_l^u \int_l^u \phi(\frac{|x - x'|}{\lambda}) dx dx' = &-\frac{1}{5\lambda^4}(l^6 + 4l^5\lambda - 6l^5 u + 5l^4\lambda^2 - 20l^4\lambda u + 15l^4 u^2 - 20l^3\lambda^2 u \\ &+ 40l^3\lambda u^2 - 20l^3 u^3 - 5l^2\lambda^4 + 30l^2\lambda^2 u^2 - 40l^2\lambda u^3 + 15l^2 u^4 \\ &+ 10l\lambda^4 u - 20l\lambda^2 u^3 + 20l\lambda u^4 - 6l u^5 - 5\lambda^4 u^2 + 5\lambda^2 u^4 - 4\lambda u^5 \\ &+ u^6)\end{aligned} \tag{21}$$

Smoothness class C^4:

Case 1*:

$$\int_l^u \int_l^u \phi(\frac{|x - x'|}{\lambda}) dx dx' = \frac{\lambda\,(-24l - 5\lambda + 24u)}{36} \tag{22}$$

Case 2*:

$$\begin{aligned}\int_l^u \int_l^u \phi(\frac{|x-x'|}{\lambda})dxdx' =& \frac{1}{36\lambda^7}(8l^9 + 45l^8\lambda - 72l^8u + 96l^7\lambda^2 - 360l^7\lambda u + 288l^7u^2 + 84l^6\lambda^3 \\ & - 672l^6\lambda^2u + 1260l^6\lambda u^2 - 672l^6u^3 - 504l^5\lambda^3u + 2016l^5\lambda^2u^2 \\ & - 2520l^5\lambda u^3 + 1008l^5u^4 - 42l^4\lambda^5 + 1260l^4\lambda^3u^2 - 3360l^4\lambda^2u^3 \\ & + 3150l^4\lambda u^4 - 1008l^4u^5 + 168l^3\lambda^5u - 1680l^3\lambda^3u^3 + 3360l^3\lambda^2u^4 \\ & - 2520l^3\lambda u^5 + 672l^3u^6 + 36l^2\lambda^7 - 252l^2\lambda^5u^2 + 1260l^2\lambda^3u^4 \\ & - 2016l^2\lambda^2u^5 + 1260l^2\lambda u^6 - 288l^2u^7 - 72l\lambda^7u + 168l\lambda^5u^3 \\ & - 504l\lambda^3u^5 + 672l\lambda^2u^6 - 360l\lambda u^7 + 72lu^8 + 36\lambda^7u^2 - 42\lambda^5u^4 \\ & + 84\lambda^3u^6 - 96\lambda^2u^7 + 45\lambda u^8 - 8u^9)\end{aligned} \tag{23}$$