Gaussian Process Derivative at Uncertain Input for SE Kernel

Truong X. Nghiem
School of Informatics, Computing, and Cyber Systems
Northern Arizona University
truong.nghiem@nau.edu

Abstract

Given a Gaussian Process with a zero mean and a Squared Exponential (SE) kernel. We are interested in the exact mean and covariance of the predictive distribution of the latent function f and its gradient $\frac{\partial f}{\partial x}$, at an uncertain input $x \sim \mathcal{N}(\mu, \Sigma)$. This technical note develops the calculations of these quantities and documents an implementation of these calculations in a Matlab function called `gppred_exactmoments_se`.

Contents

1 Notations

1.1 SE covariance function

The Squared Exponential (SE) covariance function (ARD in the general case) is given by

$$\boldsymbol{k}_{\sigma_f,\Lambda}(x,z) = \sigma_f^2 \exp\left(-\frac{1}{2}(x-z)^T\Lambda^{-1}(x-z)\right).$$

Usually the matrix Λ is a diagonal matrix of the lengthscales, hence all computations involving Λ can be simplified significantly. The SE kernel has lengthscales λ_i, for $i = 1,\ldots,D$, and σ_f as hyperparameters. Therefore, $\Lambda = \text{diag}(\lambda_1^2,\ldots,\lambda_D^2)$. When the covariance function is written without subscripts, it implies Λ and σ_f of the GP model is used. Any omitted part of the subscript means the default value from the GP model is assumed.

The noise variance, $y = f(x) + \epsilon_n$ with $\epsilon_n \sim \mathcal{N}\left(0,\sigma_n^2\right)$, is σ_n^2.

1.2 Vectorization

If x is a vector then x_i is its i^{th} element. Given a collection X of vectors, we will write $x^{(i)}$ to denote the i^{th} vector in the collection. So $x_j^{(i)}$ is the j^{th} element of the i^{th} vector in the collection.

Given a scalar function $f(x,z)$, and X and Z are collections of vectors x and z (X and Z can be singletons). We will write $f(X,Z)$ to denote a matrix whose (i,j) element is $f(x^{(i)},z^{(j)})$.

The Hadamard product (element-wise product) of x and y is $x \odot y$. We extend this operator to make it a broadcasting operation, in which the dimensions of x and y do not need to match and the multiplication is broadcast along all dimensions. This is similar to the `.*` operators in Julia, and the function `bsxfun` in Matlab.

2 Problem Formulation

Given a latent function $f(x)$ with Gaussian Process prior with zero mean and SE covariance function $\boldsymbol{k}_{\sigma_f,\Lambda}(.,I)$t was shown that if the input x is deterministic, the gradient $\frac{\partial f}{\partial x}$ is a multi-variate Gaussian Process [1]. In fact, the vector $\left[f(x), \left(\frac{\partial f}{\partial x}\right)^T\right]^T$ has a joint Gaussian distribution.

In this technical note, we consider the case when x is non-deterministic and has a Gaussian distribution $x \sim \mathcal{N}(\mu,\Sigma)$. In other words, the input x is uncertain. Its uncertainty is propagated through the Gaussian Process f to the output $f(x)$, resulting in a non-Gaussian complex distribution of $f(x)$ and its gradient $\frac{\partial f}{\partial x}$. We will approximate this distribution by a multi-variate Gaussian distribution, by calculating the exact mean and covariance matrix of the non-Gaussian distribution and set the mean and variance of the approximate Gaussian distribution to these values. Therefore, the goal of this technical note is to calculate the first two moments of the joint distribution of $f(x)$ and $\frac{\partial f}{\partial x}$ at a Gaussian uncertain input $x \sim \mathcal{N}(\mu,\Sigma)$.

3 Mathematical results

Let's define the difference between the inputs in training data and a vector x as the columns of a matrix $\tilde{X}_x$:

$$\tilde{X}_x = X - x = [x^{(i)} - x].$$

3.1 Derivatives of the SE kernel

We derive several derivatives of the SE kernel.

Let's denote various derivatives of the covariance function as follows:

- $\boldsymbol{k}^{(1,0)}(x,x') = \nabla_x \boldsymbol{k}(x,x')$ is the gradient of $\boldsymbol{k}(x,x')$ with respect to the first argument x;
- $\boldsymbol{k}^{(0,1)}(x,x') = \mathrm{D}_{x'}\boldsymbol{k}(x,x')$ is the Jacobian of $\boldsymbol{k}(x,x')$ with respect to the second argument x';
- $\boldsymbol{k}^{(1,1)}(x,x') = \mathrm{D}_{x'}\left(\nabla_x \boldsymbol{k}(x,x')\right)$ is the $D \times D$ matrix such that $\boldsymbol{k}_{i,j}^{(1,1)}(x,x') = \frac{\partial^2}{\partial x_i \partial x'_j}\boldsymbol{k}(x,x')$.

We will derive these derivatives below.

The gradient $\boldsymbol{k}^{(1,0)}(x,x')$ is ((A.33) in [2])

$$\boldsymbol{k}^{(1,0)}(x,x') = -\Lambda^{-1}(x-x')\boldsymbol{k}(x,x').$$

Extending to the case when x' is the collection X, we have

$$\boldsymbol{k}^{(1,0)}(x,X) = \Lambda^{-1}\left(\tilde{X}_x \odot \boldsymbol{k}(x,X)\right) \in \mathbb{R}^{D\times N}.$$

The Jacobian $\boldsymbol{k}^{(0,1)}(x,x')$ is ((A.34) in [2])

$$\boldsymbol{k}^{(0,1)}(x,x') = -(\boldsymbol{k}^{(1,0)}(x,x'))^T = (x-x')^T\Lambda^{-1}\boldsymbol{k}(x,x')$$

where we note that Λ is symmetric (diagonal in fact). Extending to the case when x is the collection X and x' is x, we have

$$\boldsymbol{k}^{(0,1)}(X,x) = \left(\tilde{X}_x \odot \boldsymbol{k}(x,X)\right)^T \Lambda^{-1} = \boldsymbol{k}^{(1,0)}(x,X)^T \in \mathbb{R}^{N\times D}.$$

Finally ((A.37) in [2]),

$$\boldsymbol{k}^{(1,1)}(x,x') = \Lambda^{-1}\left(\mathbb{I} - (x-x')(x-x')^T\Lambda^{-1}\right)\boldsymbol{k}(x,x').$$

When $x = x'$ we will have

$$\boldsymbol{k}^{(1,1)}(x,x')\Big|_{x=x'} = \Lambda^{-1}\boldsymbol{k}(x,x) = \sigma_f^2\Lambda^{-1}.$$

3.2 Moments of SE kernel

Given an uncertain input $x \sim \mathcal{N}(\mu,\Sigma)$. Note that $\tilde{X}_\mu = X - \mu$.

Define the mean of the kernel ((A.30) in [2])

$$E_k(\mu, x^{(i)}, \Sigma, \Lambda) = \mathbb{E}_x\left[\boldsymbol{k}\left(x, x^{(i)}\right)\right] = \left|\Sigma\Lambda^{-1} + \mathbb{I}\right|^{-1/2} \boldsymbol{k}_{\Sigma+\Lambda}\left(\mu, x^{(i)}\right).$$

In particular for the entire data X

$$E_k = E_k(\mu, X, \Sigma, \Lambda) = \mathbb{E}_x\left[\boldsymbol{k}(x,X)\right] = \left|\Sigma\Lambda^{-1} + \mathbb{I}\right|^{-1/2} \boldsymbol{k}_{\Sigma+\Lambda}(\mu, X)$$

which is a row vector. Its transpose is the column vector

$$E_k(X, \mu, \Lambda, \Sigma) = \mathbb{E}_x\left[\boldsymbol{k}(X,x)\right] = \left|\Sigma\Lambda^{-1} + \mathbb{I}\right|^{-1/2} \boldsymbol{k}_{\Sigma+\Lambda}(X, \mu).$$

Define the mean of x times a kernel ((A.39) in [2])

$$E_{xk}(\mu, x^{(i)}, \Sigma, \Lambda) = \mathbb{E}_x\left[x\boldsymbol{k}\left(x, x^{(i)}\right)\right] = E_k(\mu, x^{(i)}, \Sigma, \Lambda)\Lambda(\Sigma+\Lambda)^{-1}(\Sigma\Lambda^{-1}x^{(i)} + \mu),$$

which is a column vector. For the entire data X we can define the matrix of the above column vectors for the $x^{(i)}$,

$$\begin{aligned} E_{xk} = E_{xk}(\mu, X, \Sigma, \Lambda) &= \left[\mathbb{E}_x\left[x\boldsymbol{k}\left(x, x^{(i)}\right)\right]\right]_i \\ &= \mathbb{E}_x\left[x \odot \boldsymbol{k}(x,X)\right] \\ &= \Lambda(\Sigma+\Lambda)^{-1}(\Sigma\Lambda^{-1}X + \mu)\,\mathrm{diag}\left(E_k(\mu, X, \Sigma, \Lambda)\right) \\ &= \Lambda(\Sigma+\Lambda)^{-1}(\Sigma\Lambda^{-1}X + \mu) \odot E_k(\mu, X, \Sigma, \Lambda). \end{aligned}$$

Define the mean of the product of the kernel and itself ((A.33) in [2])

$$\begin{aligned} E_{kk}(\mu, x^{(i)}, x^{(j)}, \Sigma, \Lambda) &= \mathbb{E}_x\left[\boldsymbol{k}\left(x, x^{(i)}\right)\boldsymbol{k}\left(x, x^{(j)}\right)\right] \\ &= \boldsymbol{k}_{\Lambda/2}\left(\frac{x^{(i)}}{2}, \frac{x^{(j)}}{2}\right) E_k\left(\mu, \frac{x^{(i)} + x^{(j)}}{2}, \Sigma, \frac{\Lambda}{2}\right) \end{aligned}$$

where we note that

$$\boldsymbol{k}_{\Lambda/2}\left(\frac{x^{(i)}}{2},\frac{x^{(j)}}{2}\right)=\sqrt{\sigma_f^2\boldsymbol{k}\left(x^{(i)},x^{(j)}\right)}$$

We also note that

$$\begin{aligned}E_k\left(\mu,\frac{x^{(i)}+x^{(j)}}{2},\Sigma,\frac{\Lambda}{2}\right)&=\left|2\Sigma\Lambda^{-1}+\mathbb{I}\right|^{-1/2}\boldsymbol{k}_{\Sigma+\Lambda/2}\left(\mu,\frac{x^{(i)}+x^{(j)}}{2}\right)\\&=\left|2\Sigma\Lambda^{-1}+\mathbb{I}\right|^{-1/2}\boldsymbol{k}_{4\Sigma+2\Lambda}\left(x^{(i)}-\mu,\mu-x^{(j)}\right)\end{aligned}$$

Therefore we can calculate E_{kk} for all X as

$$E_{kk}(\mu,X,X,\Sigma,\Lambda)=\left|2\Sigma\Lambda^{-1}+\mathbb{I}\right|^{-1/2}\boldsymbol{k}_{\Lambda/2}\left(\frac{X}{2},\frac{X}{2}\right)\odot\boldsymbol{k}_{4\Sigma+2\Lambda}\left(\tilde{X}_\mu,-\tilde{X}_\mu\right)$$

Define the mean of x times the product of the kernel and itself ((A.40) in [2])

$$\begin{aligned}E_{xkk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)&=\mathbb{E}_x\left[x\boldsymbol{k}\left(x,x^{(i)}\right)\boldsymbol{k}\left(x,x^{(j)}\right)\right]\\&=\boldsymbol{k}_{\Lambda/2}\left(\frac{x^{(i)}}{2},\frac{x^{(i)}}{2}\right)E_{xk}\left(\mu,\frac{x^{(i)}+x^{(j)}}{2},\Sigma,\frac{\Lambda}{2}\right)\\&=\boldsymbol{k}_{\Lambda/2}\left(\frac{x^{(i)}}{2},\frac{x^{(i)}}{2}\right)E_k\left(\mu,\frac{x^{(i)}+x^{(j)}}{2},\Sigma,\frac{\Lambda}{2}\right)\frac{\Lambda}{2}\left(\Sigma+\frac{\Lambda}{2}\right)^{-1}\left(\Sigma\Lambda^{-1}\left(x^{(i)}+x^{(j)}\right)+\mu\right)\\&=E_{kk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)\frac{\Lambda}{2}\left(\Sigma+\frac{\Lambda}{2}\right)^{-1}\left(\Sigma\Lambda^{-1}\left(x^{(i)}+x^{(j)}\right)+\mu\right)\\&=E_{kk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)(2\Sigma\Lambda^{-1}+\mathbb{I})^{-1}\left(\mu+\Sigma\Lambda^{-1}(x^{(i)}+x^{(j)})\right)\\&=E_{kk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)M\end{aligned}$$

where

$$M=(2\Sigma\Lambda^{-1}+\mathbb{I})^{-1}\left(\mu+\Sigma\Lambda^{-1}(x^{(i)}+x^{(j)})\right)$$

Define the mean of the product of $x^{(i)}$, $x^{(j)}$, and the kernels at these two inputs ((A.43) in [2])

$$\begin{aligned}E_{xxkk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)&=\mathbb{E}_x\left[xx^T\boldsymbol{k}\left(x,x^{(i)}\right)\boldsymbol{k}\left(x,x^{(j)}\right)\right]\\&=|\Lambda|^{1/2}\left|\frac{\Lambda}{2}+\Sigma\right|^{-1/2}\boldsymbol{k}_{\Lambda/2}\left(\frac{x^{(i)}}{2},\frac{x^{(i)}}{2}\right)\boldsymbol{k}_{\Sigma+\Lambda/2}\left(\mu,\frac{x^{(i)}+x^{(j)}}{2}\right)\left(S+MM^T\right)\\&=E_{kk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)\left(S+MM^T\right)\end{aligned}$$

where $S=(2\Lambda^{-1}+\Sigma^{-1})^{-1}=(2\Lambda^{-1}+\Sigma^{-1})^{-1}\Sigma^{-1}\Sigma=(2\Sigma\Lambda^{-1}+\mathbb{I})^{-1}\Sigma$.

Define the covariance ((A.37) in [2])

$$\begin{aligned}C_{kk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)&=\mathrm{Cov}_x\left(\boldsymbol{k}\left(x,x^{(i)}\right),\boldsymbol{k}\left(x,x^{(j)}\right)\right)\\&=E_{kk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)-E_k(\mu,x^{(i)},\Sigma,\Lambda)E_k(\mu,x^{(j)},\Sigma,\Lambda)\end{aligned}$$

and so for all X

$$C_{kk}(\mu,X,X,\Sigma,\Lambda)=E_{kk}(\mu,X,X,\Sigma,\Lambda)-E_k(\mu,X,\Sigma,\Lambda)^TE_k(\mu,X,\Sigma,\Lambda)$$

Define the covariance ((A.41) in [2])

$$\begin{aligned}C_{xkk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)&=\mathrm{Cov}_x\left(x\boldsymbol{k}\left(x,x^{(i)}\right),\boldsymbol{k}\left(x,x^{(j)}\right)\right)\\&=E_{xkk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)-E_{xk}(\mu,x^{(i)},\Sigma,\Lambda)E_k(\mu,x^{(j)},\Sigma,\Lambda)\end{aligned}$$

Define the covariance ((A.46) in [2])

$$\begin{aligned}C_{xkxk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)&=\mathrm{Cov}_x\left(x\boldsymbol{k}\left(x,x^{(i)}\right),x\boldsymbol{k}\left(x,x^{(j)}\right)\right)\\&=E_{xxkk}(\mu,x^{(i)},x^{(j)},\Sigma,\Lambda)-E_{xk}(\mu,x^{(i)},\Sigma,\Lambda)E_{xk}(\mu,x^{(j)},\Sigma,\Lambda)^T\end{aligned}$$

3.3 Mean and covariance of latent function

Given an uncertain input $x \sim \mathcal{N}(\mu, \Sigma)$. The calculations of the mean and variance of the latent function f are summarized below (see section 2.4.1 of [2] for details).

3.3.1 Mean

$$\mathbb{E}[f] = \overline{f} = \mu_f = q^T \alpha$$

where α is the constant weight vector used in calculating the predictive mean (at deterministic input), and

$$q = E_k = \mathbb{E}_x[\boldsymbol{k}(X, x)] = \left|\Sigma\Lambda^{-1} + \mathbb{I}\right|^{-1/2} \boldsymbol{k}_{\Sigma+\Lambda}(X, \mu)$$

is the common E_k calculated above.

3.3.2 Variance

$$\begin{aligned}
\mathrm{Var}(f) &= \mathbb{E}_x\left[\mathrm{Var}_f(f)\right] + \mathrm{Var}_x\left(\mathbb{E}_f\left[f\right]\right) \\
\mathbb{E}_x\left[\mathrm{Var}_f(f)\right] &= \sigma_f^2 - \mathrm{tr}\left((K + \sigma_n^2\mathbb{I})^{-1}\tilde{Q}\right) \\
\mathrm{Var}_x\left(\mathbb{E}_f\left[f\right]\right) &= \alpha^T\tilde{Q}\alpha - \overline{f}^2
\end{aligned}$$

where

$$\begin{aligned}
\tilde{Q}_{ij} &= \left|2\Sigma\Lambda^{-1} + \mathbb{I}\right|^{-1/2} \boldsymbol{k}\left(x^{(i)}, \mu\right) \boldsymbol{k}\left(x^{(j)}, \mu\right) Q_{ij} \\
Q_{ij} &= \exp\left((z_{ij} - \mu)^T P (z_{ij} - \mu)\right)
\end{aligned}$$

where $z_{ij} = (x^{(i)} + x^{(j)})/2$ and

$$P = \left(\Sigma + \frac{1}{2}\Lambda\right)^{-1} \Sigma\Lambda^{-1} = \Lambda^{-1} - \frac{1}{2}\left(\Sigma + \frac{1}{2}\Lambda\right)^{-1} = \left(\Lambda + \frac{1}{2}\Lambda\Sigma^{-1}\Lambda\right)^{-1}$$

We can write $\tilde{Q}$ succintly as

$$\tilde{Q} = \left|2\Sigma\Lambda^{-1} + \mathbb{I}\right|^{-1/2} \left(\boldsymbol{k}(X, \mu)\, \boldsymbol{k}(X, \mu)^T\right) \odot Q$$

3.4 Mean and covariance of derivative and latent function

Given an uncertain input $x \sim \mathcal{N}(\mu, \Sigma)$. This section develops the mean and covariance of the derivative (`GMEAN` and `GCOV`) and the covariance between the latent function and its derivative (`FGCOV`) at this uncertain input.

3.4.1 Mean of derivative

As derived in [2] (Appendix A.3), we can use the rule of iterated expectations to calculate the mean derivative as ((A.48) in [2]):

$$\begin{aligned}
\mathbb{E}[g] &= -\Lambda^{-1} \sum_{i=1}^{N} \alpha_i E_{xk}(\mu, x^{(i)}, \Sigma, \Lambda) - \alpha_i x^{(i)} E_k(\mu, x^{(i)}, \Sigma, \Lambda) \\
&= -\Lambda^{-1} \sum_{i=1}^{N} \alpha_i E_k(\mu, x^{(i)}, \Sigma, \Lambda) \left(\Lambda(\Lambda + \Sigma)^{-1}(\Sigma\Lambda^{-1}x^{(i)} + \mu) - x^{(i)}\right) \\
&= \sum_{i=1}^{N} \alpha_i E_k(\mu, x^{(i)}, \Sigma, \Lambda) \left(\Lambda^{-1}x^{(i)} - (\Lambda + \Sigma)^{-1}(\Sigma\Lambda^{-1}x^{(i)} + \mu)\right) \\
&= \sum_{i=1}^{N} \alpha_i E_k(\mu, x^{(i)}, \Sigma, \Lambda) \left(\left(\Lambda^{-1} - (\Lambda + \Sigma)^{-1}\Sigma\Lambda^{-1}\right) x^{(i)} - (\Lambda + \Sigma)^{-1}\mu\right).
\end{aligned}$$

We have

$$\Lambda^{-1} - (\Lambda + \Sigma)^{-1}\Sigma\Lambda^{-1} = \Lambda^{-1} - (\Lambda + \Sigma)^{-1}\left((\Lambda + \Sigma) - \Lambda\right)\Lambda^{-1} = (\Lambda + \Sigma)^{-1}.$$

Therefore,

$$\begin{aligned}\mathbb{E}[g] &= (\Lambda + \Sigma)^{-1}\sum_{i=1}^{N}\alpha_i E_k(\mu, x^{(i)}, \Sigma, \Lambda)(x^{(i)} - \mu) \\ &= (\Lambda + \Sigma)^{-1}\tilde{X}_\mu\left(\alpha \odot E_k(X, \mu, \Lambda, \Sigma)\right),\end{aligned}$$

or, by expanding E_k,

$$\mathbb{E}[g] = \left|\Sigma\Lambda^{-1} + \mathbb{I}\right|^{-1/2}(\Lambda + \Sigma)^{-1}\tilde{X}_\mu\left(\alpha \odot \boldsymbol{k}_{\Sigma+\Lambda}(X, \mu)\right).$$

3.4.2 Variance of derivative

Using the rule of total variance, we have ((A.49) in [2]),

$$\mathrm{Var}(g) = \mathbb{E}_x\left[\mathrm{Var}_f(g)\right] + \mathrm{Var}_x\left(\mathbb{E}_f\left[g\right]\right).$$

We have that

$$\mathbb{E}_f[g] = \mathbb{E}_f\left[\nabla_x f\right] = \nabla_x \mathbb{E}_f$$

because of linearity of differentiation.

It has been calculated in [2] that

$$\nabla_x \mathbb{E}_f = \Lambda^{-1}\tilde{X}_x(\boldsymbol{k}(X, x) \odot \alpha).$$

We also have that

$$\mathrm{Var}_f(g) = \boldsymbol{k}^{(1,1)}(x, x) - \boldsymbol{k}^{(1,0)}(x, X)\, K^{-1}\boldsymbol{k}^{(0,1)}(X, x).$$

Therefore,

$$\begin{aligned}\mathrm{Var}(g) &= \mathbb{E}_x\left[\boldsymbol{k}^{(1,1)}(x, x) - \boldsymbol{k}^{(1,0)}(x, X)\, K^{-1}\boldsymbol{k}^{(0,1)}(X, x)\right] + \mathrm{Var}_x\left(\Lambda^{-1}\tilde{X}_x(\boldsymbol{k}(X, x) \odot \alpha)\right) \\ &= \mathbb{G}^{(1)} - \mathbb{G}^{(2)} + \mathbb{G}^{(3)}.\end{aligned}$$

We will calculate each of the above terms separately.

1. First component

$$\mathbb{G}^{(1)} = \mathbb{E}_x\left[\boldsymbol{k}^{(1,1)}(x, x)\right] = \mathbb{E}_x\left[\sigma_f^2\Lambda^{-1}\right] = \sigma_f^2\Lambda^{-1}.$$

2. Second component

$$\mathbb{G}^{(2)} = \mathbb{E}_x\left[\boldsymbol{k}^{(1,0)}(x, X)\, K^{-1}\boldsymbol{k}^{(0,1)}(X, x)\right] = \mathbb{E}_x\left[\boldsymbol{k}^{(1,0)}(x, X)\, K^{-1}\left(\boldsymbol{k}^{(1,0)}(x, X)\right)^T\right]$$

This is a $D \times D$ matrix, whose (i, j) element is given by

$$\mathbb{G}^{(2)}_{i,j} = \mathbb{E}_x\left[\boldsymbol{k}_i^{(1,0)}(x, X)\, K^{-1}\left(\boldsymbol{k}_j^{(1,0)}(x, X)\right)^T\right]$$

where $\boldsymbol{k}_i^{(1,0)}(x, X)$ denotes the i^{th} row of $\boldsymbol{k}^{(1,0)}(x, X)$. This is essentially the expectation of an inner product of two random vectors. We know that $\mathbb{E}\left[X^T K Y\right] = \mathbb{E}[X]K\mathbb{E}[Y] + \mathrm{tr}\left(K\,\mathrm{Cov}(X, Y)\right)$. Therefore,

$$\mathbb{G}^{(2)}_{i,j} = \mathbb{E}_x\left[\boldsymbol{k}_i^{(1,0)}(x, X)\right] K^{-1}\mathbb{E}_x\left[\boldsymbol{k}_j^{(1,0)}(x, X)\right]^T + \mathrm{tr}\left(K^{-1}\,\mathrm{Cov}_x\left(\boldsymbol{k}_i^{(1,0)}(x, X), \boldsymbol{k}_j^{(1,0)}(x, X)\right)\right).$$

We have derived earlier that $\boldsymbol{k}^{(1,0)}(x, X) = \Lambda^{-1}\left(\tilde{X}_x \odot \boldsymbol{k}(x, X)\right)$. The expectation $\mathbb{E}_x\left[\boldsymbol{k}_i^{(1,0)}(x, X)\right]$ is the i^{th} row of the matrix

$$\mathbb{E}_x\left[\boldsymbol{k}^{(1,0)}(x, X)\right] = \Lambda^{-1}\mathbb{E}_x\left[\tilde{X}_x \odot \boldsymbol{k}(x, X)\right].$$

Since $\tilde{X}_x = X - x$, we have

$$\begin{aligned}\mathbb{E}_x\left[\boldsymbol{k}^{(1,0)}(x,X)\right] &= \Lambda^{-1}\left(X \odot \mathbb{E}_x\left[\boldsymbol{k}(x,X)\right] - \mathbb{E}_x\left[x \odot \boldsymbol{k}(x,X)\right]\right)\\ &= \Lambda^{-1}\left(X \odot E_k - E_{xk}\right)\end{aligned}$$

which can be calculated immediately from E_k and E_{xk}. We can also expand it by writing E_{xk} in terms of E_k:

$$\mathbb{E}_x\left[\boldsymbol{k}^{(1,0)}(x,X)\right] = \Lambda^{-1}\left(\left(X - \Lambda(\Sigma+\Lambda)^{-1}(\Sigma\Lambda^{-1}X + \mu)\right) \odot E_k\right).$$

We have that

$$\Lambda(\Sigma+\Lambda)^{-1}\Sigma\Lambda^{-1} = \Lambda(\Sigma+\Lambda)^{-1}(\Sigma+\Lambda-\Lambda)\Lambda^{-1} = \mathbb{I} - \Lambda(\Sigma+\Lambda)^{-1}.$$

Substituting this into the above equation gives us

$$\begin{aligned}\mathbb{E}_x\left[\boldsymbol{k}^{(1,0)}(x,X)\right] &= \Lambda^{-1}\left(\left(\Lambda(\Sigma+\Lambda)^{-1}(X-\mu)\right) \odot E_k\right)\\ &= (\Sigma+\Lambda)^{-1}\tilde{X}_\mu \odot E_k.\end{aligned}$$

This equation will be used in the code.

Let's consider the covariance

$$\mathrm{Cov}_x\left(\boldsymbol{k}_i^{(1,0)}(x,X), \boldsymbol{k}_j^{(1,0)}(x,X)\right)$$

which is an $N \times N$ matrix. First, note that

$$\boldsymbol{k}_i^{(1,0)}(x,X) = \Lambda_i^{-1}\left(\tilde{X}_x \odot \boldsymbol{k}(x,X)\right)$$

where $\tilde{\Lambda}_i^{-1}$ is the i^{th} row of $\tilde{\Lambda}^{-1}$, which is a row vector of all zeros except $1/\lambda_i^2$ in the i^{th} position. Define $\tilde{X}_{x,i}$ to be the i^{th} row of $\tilde{X}_x$. We then have

$$\mathrm{Cov}_x\left(\boldsymbol{k}_i^{(1,0)}(x,X), \boldsymbol{k}_j^{(1,0)}(x,X)\right) = \frac{1}{\lambda_i^2\lambda_j^2}\mathrm{Cov}_x\left(\tilde{X}_{x,i} \odot \boldsymbol{k}(x,X), \tilde{X}_{x,j} \odot \boldsymbol{k}(x,X)\right)$$

where the covariance matrix is an $N \times N$ symmetric matrix. Its (p,q) element is

$$\begin{aligned}&\mathrm{Cov}_x\left(\boldsymbol{k}\left(x,x^{(p)}\right)\left(x_i^{(p)} - x_i\right), \boldsymbol{k}\left(x,x^{(q)}\right)\left(x_j^{(q)} - x_j\right)\right)\\ &= x_i^{(p)}x_j^{(q)}\mathrm{Cov}_x\left(\boldsymbol{k}\left(x,x^{(p)}\right), \boldsymbol{k}\left(x,x^{(q)}\right)\right) - x_i^{(p)}\mathrm{Cov}_x\left(\boldsymbol{k}\left(x,x^{(p)}\right), x_j\boldsymbol{k}\left(x,x^{(q)}\right)\right) -\\ &\quad x_j^{(q)}\mathrm{Cov}_x\left(x_i\boldsymbol{k}\left(x,x^{(p)}\right), \boldsymbol{k}\left(x,x^{(q)}\right)\right) + \mathrm{Cov}_x\left(x_i\boldsymbol{k}\left(x,x^{(p)}\right), x_j\boldsymbol{k}\left(x,x^{(q)}\right)\right)\\ &= x_i^{(p)}x_j^{(q)}C_{kk}^{(p,q)} - x_i^{(p)}C_{xkk,j}^{(p,q)} - x_j^{(q)}C_{xkk,i}^{(p,q)} + C_{xkxk,(i,j)}^{(p,q)}\end{aligned}$$

From this equation, the trace in $\mathbb{G}_{i,j}^{(2)}$ can be computed as follows (note that K^{-1} is symmetric):

$$\begin{aligned}&\mathrm{tr}\left(K^{-1}\mathrm{Cov}_x\left(\boldsymbol{k}_i^{(1,0)}(x,X), \boldsymbol{k}_j^{(1,0)}(x,X)\right)\right)\\ &= \frac{1}{\lambda_i^2\lambda_j^2}\sum_{p=1}^{N}\sum_{q=1}^{N}K_{p,q}^{-1}\left(x_i^{(p)}x_j^{(q)}C_{kk}^{(p,q)} - x_i^{(p)}C_{xkk,j}^{(p,q)} - x_j^{(q)}C_{xkk,i}^{(p,q)} + C_{xkxk,(i,j)}^{(p,q)}\right)\end{aligned}$$

For an efficient implementation, see the implementation section below.

3. Third component

The (i,j) element of $\mathbb{G}^{(3)}$ is

$$\begin{aligned}\mathbb{G}_{i,j}^{(3)} &= \mathrm{Cov}_x\left(\frac{1}{\lambda_i^2}\tilde{X}_{x,i}(\boldsymbol{k}(X,x) \odot \alpha), \frac{1}{\lambda_j^2}\tilde{X}_{x,j}(\boldsymbol{k}(X,x) \odot \alpha)\right)\\ &= \frac{1}{\lambda_i^2\lambda_j^2}\mathrm{Cov}_x\left(\sum_{p=1}^{N}(x_i^{(p)} - x_i)\boldsymbol{k}\left(x^{(p)},x\right)\alpha_p, \sum_{q=1}^{N}(x_j^{(q)} - x_j)\boldsymbol{k}\left(x^{(q)},x\right)\alpha_q\right)\\ &= \frac{1}{\lambda_i^2\lambda_j^2}\sum_{p=1}^{N}\sum_{q=1}^{N}\alpha_p\alpha_q\left(x_i^{(p)}x_j^{(q)}C_{kk}^{(p,q)} - x_i^{(p)}C_{xkk,j}^{(p,q)} - x_j^{(q)}C_{xkk,i}^{(p,q)} + C_{xkxk,(i,j)}^{(p,q)}\right)\end{aligned}$$

Observe that this equation constains elements similar to those in the previous equation. This can be used to implement them more efficiently.

3.4.3 Covariance between latent function and derivative

The final equation is therefore:

$$\mathrm{Cov}(f,g)^T = \mathrm{Cov}(g,f) = \Lambda^{-1} \sum_{p=1}^{N} \sum_{q=1}^{N} \alpha_p \alpha_q \left(C_{kk}^{(p,q)} x^{(q)} - C_{xkk}^{(p,q)} \right) - K_{pq}^{-1} \left(E_{kk}^{(p,q)} x^{(q)} - E_{xkk}^{(p,q)} \right)$$

4 Matlab Implementation

4.1 Function Inputs and Outputs

The function's signature is:

```
function [POST, FMEAN, FCOV, GMEAN, FGCOV, GCOV] = ...
    gppred_exactmoments_se(GP, XMEAN, XCOV, POST)
```

4.1.1 Input arguments

`GP` the GP model object, of type `nextgp.GP`.

`XMEAN` a vector of length D of the mean of the uncertain input x

`XCOV` a matrix of size $D \times D$; the covariance matrix of the uncertain input x; if the covariance matrix is diagonal, it can also be a vector of length D of the diagonal.

`POST` internal structure to store reusable values; in the first call, this structure is created; in subsequent calls, this structure should be passed to the function to save computation time.

Note that $x \sim \mathcal{N}(\texttt{XMEAN}, \texttt{XCOV})$.

4.1.2 Outputs

`POST` the structure containing reusable values; see above; should always store this structure and pass to the next call.

`FMEAN` the mean of the latent function f, a scalar.

`FCOV` the self-covariance (variance) of the latent function f, a scalar.

`GMEAN` the mean of the derivative of f, vector of length D.

`FGCOV` the covariance between f and derivative of f, matrix of size $1 \times D$.

`GCOV` the self-covariance of the derivative of f, matrix of size $D \times D$.

The covariance matrix of $[f; g]$, where g is the derivative / gradient, is therefore:

$$\mathrm{Cov}\left(\begin{bmatrix} f \\ g \end{bmatrix} \right) = \begin{bmatrix} \texttt{FCOV} & \texttt{FGCOV} \\ \texttt{FGCOV}^T & \texttt{GCOV} \end{bmatrix}$$

4.2 Check arguments and basic setup

We check the arguments and set up some variables:

- `D` is the input dimension
- `N` is the number of training points
- `COVF` is the covariance function (`nextgp` covariance function object, $\boldsymbol{k}_{\sigma_f,\Lambda}(x, z)$ in the math)
- `calc_f_mean` if the mean of latent function f is to be calculated
- `calc_f_cov` if the self-covariance of latent function f is to be calculated
- `calc_g_mean` if the mean of derivative g is to be calculated
- `calc_g_cov` if the self-covariance of derivative g is to be calculated
- `calc_fg_cov` if the covariance between f and g is to be calculated
- `xcov_diag` if the input covariance is diagonal (`XCOV` is a vector rather than a matrix)
- `xtype` is the type of the input (0: numeric, 1: CasADi, 2: other symbolic type)
- `Lchol` is `GP.post.Lchol`

4.3 Common values and helper functions

We calculate some common values that will be used throughout the later calculations.

4.3.1 Kernel-related values

```
sf2 = COVF.m_sf2;                % \sigma_f^2 of the kernel
invLengthscales = COVF.m_ellinv(:); % inverse of lengthscales (not their squares)
Lambda = COVF.m_ell .^ 2;        % Lambda matrix but only the diagonal
invLambda = COVF.m_ell2inv;      % inversed Lambda matrix but only the diagonal
alpha = GP.post.alpha;           % alpha vector for posterior computation
```

4.3.2 Other values related to predictions

These are common values used by the calculations in the later sections. See Section 3.

We implement the above common values for the special case when `XCOV` is diagonal, specified as a vector. Some important variables:

- `Xdiff_mu` is $\tilde{X}_\mu$ as defined above.
- `kernel_SigmaLambda` is $(\Sigma + \Lambda)^{-1}$, used to compute the covariance function with $\Sigma + \Lambda$ instead of just Λ.
- `kernel_SigmaLambda_sqrt` is $(\Sigma + \Lambda)^{-1/2}$.
- `kernel_SigmaLambda_inv` is $\Sigma\Lambda^{-1}$.
- `E_k_det` is $\left|\Sigma\Lambda^{-1} + \mathbb{I}\right|^{-1/2}$.
- `E_k` is E_k (row vector) as defined in Moments of SE kernel.

These variables are only calculated in certain cases (related to the derivative):

- `kernel_2SigmaLambda_inv_I` is $(2\Sigma\Lambda^{-1} + \mathbb{I})$.
- `POST.Kinv` is K^{-1} but only calculated in certain cases (where it will be needed).
- `POST.K_half` is $\boldsymbol{k}_{\Lambda/2}(X, X)$.

- `POST.invLinvL` is $\text{diag}(\Lambda^{-1})\,\text{diag}(\Lambda^{-1})^T$, which is the matrix of $\frac{1}{\lambda_i^2 \lambda_j^2}$.
- `POST.aa` is $\alpha\alpha^T$
- `POST.aaKinv` is $\alpha\alpha^T - K^{-1}$
- `E_kk` is E_{kk} $(N \times N)$ as defined in Moments of SE kernel.
- `E_xk` is E_{xk} $(D \times N)$ as defined in Moments of SE kernel.
- `C_kk` is $C_{kk}(\mu, X, X, \Sigma, \Lambda)$ as defined in Moments of SE kernel.

```
% Kinv is calculated in certain cases, and only when it's not in POST
if Lchol && (xtype ~= 0 || calc_g_cov) && ~isfield(POST, 'Kinv')
    % if not numeric or if GCOV is calculated, we should compute Kinv
    % instead of calling \ or solve on symbolics
    POST.Kinv = GP.post.Kinv; % Kinv = solve_chol(GP.post.lhsA', eye(N,N));
end
if (calc_g_cov || calc_fg_cov) && ~isfield(POST, 'Kinv')
    % If GCOV or FGCOV is calculated, we will need Kinv
    if Lchol
        POST.Kinv = GP.post.Kinv;
    else
        if isnumeric(GP.post.L)
            POST.Kinv = -GP.post.L;
        else
            POST.Kinv = -GP.post.L(eye(N,N));
        end
    end
end

Xdiff_mu = trainXt - XMEAN;

% Calculations of other common values depend on the special or general cases
if xcov_diag
    % E_k is a row vector, as derived in the technote
    kernel_SigmaLambda = 1./(XCOV + Lambda);
    kernel_SigmaLambda_sqrt = sqrt(kernel_SigmaLambda);
    kernel_SigmaLambda_inv = XCOV .* invLambda;
    if xtype == 0
        % Numerical values
        E_k_det = 1 / sqrt(prod(kernel_SigmaLambda_inv + 1));
    else
        % Symbolic, prod() may not be defined, so we need to calculate
        % the cumulative product manually, or use det(diag(...))
        tmp = XCOV.*invLambda + 1;
        tmpprod = tmp(1);
        for kk = 2:numel(tmp)
            tmpprod = tmpprod * tmp(kk);
        end
        E_k_det = 1 / sqrt(tmpprod);
    end
    % Below, XMEAN is a vector, so no mean values need to be provided
    E_k = E_k_det*sf2* ...
          exp(-nextgp.sq_dist_casadi(...
              kernel_SigmaLambda_sqrt .* XMEAN, ...
              kernel_SigmaLambda_sqrt .* trainXt)/2);

    if calc_fg_cov
        % E_xk
```

```
        E_xk = (Lambda .* kernel_SigmaLambda .* ...
                (kernel_SigmaLambda_inv .* trainXt + XMEAN)) .* E_k;

        % E_kk
        kernel_2SigmaLambda_inv_I = 2*kernel_SigmaLambda_inv + 1;
        if ~isfield(POST, 'K_half')
            % covariance matrix of k_{Lambda/2}(X/2, X/2)
            POST.K_half = sqrt(sf2 * GP.post.K);
        end
        % covariance matrix to compute k() with 4*Sigma + 2*Lambda
        E_kk_cov = 1./sqrt(4*XCOV + 2*Lambda);
        E_kk_cov_Xdiff_mu = E_kk_cov .* Xdiff_mu;
        % Note that in the call below, the mean of input data is 0,
        % so we explicitly specify zeros()
        E_kk = (POST.K_half * (sf2 / sqrt(prod(kernel_2SigmaLambda_inv_I)))) .* ...
               exp(-nextgp.sq_dist_casadi(E_kk_cov_Xdiff_mu, -E_kk_cov_Xdiff_mu,...
               0, 0)/2);

        % C_kk
        C_kk = E_kk - E_k' * E_k;

        if ~isfield(POST, 'invLinvL')
            POST.invLinvL = invLambda * invLambda';
        end

        if ~isfield(POST, 'aa')
            POST.aa = alpha*alpha';
        end

        if ~isfield(POST, 'aaKinv')
            if Lchol
                POST.aaKinv = solve_chol(GP.post.lhsA', ...
                                         GP.training_data.y*alpha' - eye(N));
            else
                POST.aaKinv = POST.aa - POST.Kinv;
            end
        end
    end
else
    error('General case currently not supported.');
end
```

4.4 Implementation of mean and variance of latent function

The equations are derived in section 3.3.

- If Σ is diagonal, the above calculations can be specialized and we can avoid complex computations. In particular, $P = \text{diag}\left(\frac{1}{\Lambda_{ii}} - \frac{1}{2\Sigma_{ii}+\Lambda_{ii}}\right)$.
- Both Q and $\tilde{Q}$ are symmetric, so two nested loops can be used to calculate just the lower (or upper) triangular part of each matrix. Not for Matlab but for a language like C, Julia.
- $\text{tr}(AB)$ can be calculated faster in Matlab as `sum(sum(A.*B',2))`.
- If $(K + \sigma_n^2 \mathbb{I})$ (which will be written simply as K) is already factorized (Cholesky), perhaps by factorizing $\tilde{Q}$, the calculation can be done faster. In the code, if `Lchol` is true, `lhsA` is available such that `lhsA*lhsA' = K`, which mean `inv(lhsA)'*inv(lhsA) = inv(K)`. If $\tilde{Q} = \tilde{R}^T\tilde{R}$ (Cholesky decomposition) then $\text{tr}(K^{-1}\tilde{Q})$ is `sum(sum((lhsA\R_tilde').^2))`. Alternatively, we can use `solve_chol` function from GPML as: `trace(solve_chol(lhsA', Q_tilde))` (this calculates the full matrix but only uses its diagonal).

In my profiling, the first approach is faster but I am not sure if $\tilde{Q}$ is positive definite. In addition, the first approach won't work with symbolic inputs.

- When `Lchol` is false, `-post.L` is `inv(K)`.
- Q above can be calculated using the `sq_dist` function, by rewriting it as pair-wise distances between two collections of vectors.

$$Q_{ij} = \exp\left(\left((x^{(i)} - \mu) - (\mu - x^{(j)})\right)^T (P/4) \left((x^{(i)} - \mu) - (\mu - x^{(j)})\right)\right)$$

By factorizing $P = R^T R$, we can calculate $Q = \exp(\text{sq\_dist}(0.5R(X - \mu), 0.5R(\mu - X)))$.

4.4.1 The special but common case when Σ is diagonal

```
% Mean of latent function FMEAN
FMEAN = E_k * alpha; % q is E_k

% Variance of latent function FCOV
if calc_f_cov
    F_R_Xdiff_mu = (sqrt(invLambda - 1./(2*XCOV + Lambda)) / 2) .* Xdiff_mu;
    % Note that in the call below, the mean of input data is 0,
    % so we explicitly specify zeros()
    F_Q = exp(nextgp.sq_dist_casadi(F_R_Xdiff_mu, -F_R_Xdiff_mu, 0, 0));

    % Calculate k(X, mu): because the covariance function is SE-ARD,
    % we can directly calculate k() without calling cov()
    % kXmu = COVF.cov(trainX, XMEAN');
    if xtype == 0
        % Numeric XMEAN, do calculation directly
        kXmu = sf2 * exp(-sum((Xdiff_mu .* Xdiff_mu) .* invLambda, 1)'/2);
    else
        kXmu = sf2 * exp(-nextgp.sq_dist_casadi(invLengthscales .* trainXt,...
                                                invLengthscales .* XMEAN)/2);
    end

    if xtype == 0
        % Numeric
        F_Q = 1 / sqrt(prod(2*XCOV.*invLambda + 1)) * ...
              ((kXmu*kXmu') .* F_Q); % This is \tilde{Q}
    else
        % Symbolic, prod() may not be defined
        tmp = 2*XCOV.*invLambda + 1;
        tmpprod = tmp(1);
        for kk = 2:numel(tmp)
            tmpprod = tmpprod * tmp(kk);
        end
        F_Q = 1 / sqrt(tmpprod) * ((kXmu*kXmu') .* F_Q); % This is \tilde{Q}
    end

    FCOV = alpha' * F_Q * alpha - FMEAN^2 + sf2;

    switch xtype
      case 0  % numeric
        if isfield(POST, 'Kinv')
            % If Kinv is available, use it: trace(K*Q) = sum(sum(K'.*Q))
            % but note that Kinv is symmetric
            F_trace = sum(sum(POST.Kinv .* F_Q));
        elseif Lchol % L contains chol decomp => use Cholesky parameters (alpha,sW,L)
            F_trace = trace(solve_chol(GP.post.lhsA', F_Q));
```

```
        else                    % L is not triangular => use alternative parametrisation
            if isnumeric(GP.post.L)
                F_trace = -sum(sum(GP.post.L.*F_Q));
            else
                F_trace = -trace(GP.post.L(F_Q));
            end
        end

    case {1, 2}  % CasADi and other symbolic
        if Lchol    % L contains chol decomp => use Cholesky parameters (alpha,sW,L)
                    % CasADi doesn't override \ (mldivide) yet, but we
                    % can use solve() to compute V; or we can
                    % compute the inverse matrix and perform
                    % matrix multiplication, which is faster
                    % for CasADi's MX type.
                    %V = solve(casadi.DM(lhsA), Ks);
            F_trace = sum(sum(POST.Kinv .* F_Q));
        else % L is not triangular => use alternative parametrisation
            if isnumeric(GP.post.L)
                F_trace = -sum(sum(GP.post.L.*F_Q));
            else
                F_trace = -trace(GP.post.L(F_Q));
            end
        end
    end

    FCOV = FCOV - F_trace;
end
```

4.4.2 The general case

Not supported currently.

4.5 Implementation of mean and covariance of derivative and latent function

The equations are derived in section 3.4.

4.5.1 The special but common case when Σ is diagonal

The implementation of `GMEAN` is straightforward.

For the implementation of `GCOV`, that second part of $\mathbb{G}^{(2)}$ and $\mathbb{G}^{(3)}$ are complex. For each (i, j), we need to loop over p and q, which is both expensive and redundant. Instead, we will loop over p and q, noting that the resulting matrices are all symmetric, and for each pair (p, q), where $p \leq q$ due to symmetry, we calculate the matrices (for all $i, j \in \{1, \ldots, D\}$) in one shot and add them up. This can be seen from the two sums above: instead of calculating the sum over p and q for each (i, j), for each (p, q) we calculate the full $D \times D$ matrix for all (i, j) and add them up.

We will rewrite the above two sums as:

$$\begin{aligned}
&\left[\operatorname{tr}\left(K^{-1}\operatorname{Cov}_x\left(\boldsymbol{k}_i^{(1,0)}(x,X), \boldsymbol{k}_j^{(1,0)}(x,X)\right)\right)\right]_{i,j} \\
&= \left(\operatorname{diag}(\Lambda^{-1})\operatorname{diag}(\Lambda^{-1})^T\right) \odot \sum_{p=1}^{N}\sum_{q=1}^{N} K_{p,q}^{-1}\left(x^{(p)}\left(x^{(q)}\right)^T C_{kk}^{(p,q)} - x^{(p)}\left(C_{xkk}^{(p,q)}\right)^T - C_{xkk}^{(p,q)}\left(x^{(q)}\right)^T + C_{xkxk}^{(p,q)}\right)
\end{aligned}$$

and

$$\mathbb{G}^{(3)} = \left(\operatorname{diag}(\Lambda^{-1})\operatorname{diag}(\Lambda^{-1})^T\right) \odot \sum_{p=1}^{N}\sum_{q=1}^{N} \alpha_p\alpha_q\left(x^{(p)}\left(x^{(q)}\right)^T C_{kk}^{(p,q)} - x^{(p)}\left(C_{xkk}^{(p,q)}\right)^T - C_{xkk}^{(p,q)}\left(x^{(q)}\right)^T + C_{xkxk}^{(p,q)}\right)$$

Therefore, we have

$$\begin{aligned}\mathrm{Var}(g) &= \mathbb{G}^{(1)} - \mathbb{G}^{(2)} + \mathbb{G}^{(3)} \\ &= \sigma_f^2 \Lambda^{-1} - \mathbb{E}_x\left[\boldsymbol{k}^{(1,0)}(x, X)\right] K^{-1} \mathbb{E}_x\left[\boldsymbol{k}^{(1,0)}(x, X)\right]^T + \\ &\quad \left(\mathrm{diag}(\Lambda^{-1})\,\mathrm{diag}(\Lambda^{-1})^T\right) \odot \\ &\quad \sum_{p=1}^{N}\sum_{q=1}^{N} \left(\alpha_p \alpha_q - K_{p,q}^{-1}\right) \left(x^{(p)} \left(x^{(q)}\right)^T C_{kk}^{(p,q)} - x^{(p)} \left(C_{xkk}^{(p,q)}\right)^T - C_{xkk}^{(p,q)} \left(x^{(q)}\right)^T + C_{xkxk}^{(p,q)} \right)\end{aligned}$$

Let's consider the summand in the last term and denote

$$Z^{(p,q)} = \left(\alpha_p \alpha_q - K_{p,q}^{-1}\right) \left(x^{(p)} \left(x^{(q)}\right)^T C_{kk}^{(p,q)} - x^{(p)} \left(C_{xkk}^{(p,q)}\right)^T - C_{xkk}^{(p,q)} \left(x^{(q)}\right)^T + C_{xkxk}^{(p,q)} \right).$$

Note that in the above equation, we have $C_{kk}^{(p,q)} = C_{kk}^{(q,p)}$ and $C_{xkxk}^{(p,q)} = \left(C_{xkxk}^{(q,p)}\right)^T$. Therefore, we can loop over $p \le q$ and calculate $Z^{(p,q)} + Z^{(q,p)}$, for $p \ne q$, to save some computational operations. This is described in the following pseudocode.

- Initialize Z to zero matrix; Z will be the sum of all $Z^{(p,q)}$
- For $p \leftarrow 1 \ldots N$
  - For $q \leftarrow 1 \ldots N$
    - If $p = q$: $Z \leftarrow Z + Z^{(p,q)}$
    - Else: $Z \leftarrow Z + Z^{(p,q)} + Z^{(q,p)}$
- Calculate $\mathrm{Var}(g)$

Note also that $\left(\alpha_p \alpha_q - K_{p,q}^{-1}\right)$ is the (p, q) element of the following matrix

$$\alpha\alpha^T - K^{-1} = K^{-1} Y \alpha^T - K^{-1} = K^{-1}(Y\alpha^T - \mathbb{I})$$

For $\mathrm{Cov}(f, g)$, the equation is similar to the latter part of $\mathrm{Cov}(g)$ above and therefore can be implemented in the same loop of p and q (and reuse some values already calculated). Let's define a similar variable $Z_{fg}^{(p,q)}$ as

$$Z_{fg}^{(p,q)} = \alpha_p \alpha_q \left(C_{kk}^{(p,q)} x^{(q)} - C_{xkk}^{(p,q)} \right) - K_{pq}^{-1} \left(E_{kk}^{(p,q)} x^{(q)} - E_{xkk}^{(p,q)} \right)$$

which we can rewrite (for computation purpose) as

$$Z_{fg}^{(p,q)} = \alpha_p \alpha_q \left(E_{xk}^{(p)} E_k^{(q)} - E_k^{(p)} E_k^{(q)} x^{(q)} \right) + \left(\alpha_p \alpha_q - K_{pq}^{-1}\right) \left(E_{kk}^{(p,q)} x^{(q)} - E_{xkk}^{(p,q)} \right)$$

When $p \ne q$, we have

$$\begin{aligned} Z_{fg}^{(q,p)} &= \alpha_q \alpha_p \left(C_{kk}^{(q,p)} x^{(p)} - C_{xkk}^{(q,p)} \right) - K_{qp}^{-1} \left(E_{kk}^{(q,p)} x^{(p)} - E_{xkk}^{(q,p)} \right) \\ &= \alpha_p \alpha_q \left(C_{kk}^{(p,q)} x^{(p)} - C_{xkk}^{(q,p)} \right) - K_{pq}^{-1} \left(E_{kk}^{(p,q)} x^{(p)} - E_{xkk}^{(p,q)} \right) \end{aligned}$$

where we use the fact that K^{-1}, E_{kk}, and C_{kk} are symmetric and $E_{xkk}^{(p,q)} = E_{xkk}^{(q,p)}$. Hence, we can simplify the sum

$$Z_{fg}^{(p,q)} + Z_{fg}^{(q,p)} = \left(\alpha_p \alpha_q C_{kk}^{(p,q)} - K_{pq}^{-1} E_{kk}^{(p,q)} \right) \left(x^{(p)} + x^{(q)} \right) - \alpha_p \alpha_q \left(C_{xkk}^{(p,q)} + C_{xkk}^{(q,p)} \right) + 2 K_{pq}^{-1} E_{xkk}^{(p,q)}$$

The computation of $\mathrm{Cov}(f, g)$ will be fused inside the nested loops for calculating $\mathrm{Var}(g)$.

To make it easier to implement the most time-consuming part (the nested for loops) in C, that part of the code is implemented as a private Matlab function, after the main code block below. A C version of the loop was implemented in `+nextgp/private/gppredse_covloop_c*`. To build the MEX file, use `+nextgp/private/make_gpppredse_mex.m` and make sure that the generated MEX file is named `gppredse_covloop.mex*`.

```
% Mean of derivative GMEAN
GMEAN = kernel_SigmaLambda .* (Xdiff_mu * (alpha .* E_k'));

% Covariance of derivative GCOV
if calc_fg_cov
    % For cov(f,g), we will actually calculate cov(g,f) then transpose it

    if calc_g_cov
        % \EE[k^{(1,0)}(x,X)]
        Ek10 = kernel_SigmaLambda .* (Xdiff_mu .* E_k);  % D x N

        % The second term of GCOV
        if Lchol
            GCOV = Ek10*solve_chol(GP.post.lhsA', Ek10');
        else
            if isnumeric(GP.post.L)
                GCOV = Ek10*GP.post.L*Ek10';
            else
                GCOV = Ek10*GP.post.L(Ek10');
            end
        end

        % The first two terms of GCOV
        GCOV = diag(sf2 * invLambda) - GCOV;

        % Calculate the last term
        % Some common values that are not changed during iterations
        G_S = diag(1./(2*invLambda + 1./XCOV)); % S term

        [FGCOV, G_Z] = gppredse_covloop(...
            XMEAN, ...
            N, D, POST, trainXt, ...
            kernel_SigmaLambda_inv, kernel_2SigmaLambda_inv_I, invLambda, ...
            E_k, E_kk, E_xk, C_kk, ...
            G_S, true);

        GCOV = GCOV + POST.invLinvL .* G_Z;
    else
        [FGCOV] = gppredse_covloop(...
            XMEAN, ...
            N, D, POST, trainXt, ...
            kernel_SigmaLambda_inv, kernel_2SigmaLambda_inv_I, invLambda, ...
            E_k, E_kk, E_xk, C_kk, [], false); % G_S = [] is required for MEX to work
    end

end
```

The main nested loops:

```
function [FGCOV, G_Z] = ...
        gppredse_covloop(XMEAN, ...
                         N, D, POST, trainXt, ...
                         kernel_SigmaLambda_inv, kernel_2SigmaLambda_inv_I, invLambda, ...
                         E_k, E_kk, E_xk, C_kk, ...
                         G_S, ...
                         calc_g_cov)
    % Internal function to calculate the main nested loops in calculating the
    % covariance between latent function and derivative.
    % FGCOV does not need G_S; so only need to provide G_S if G_Z is computed
    %
```

```
% XMEAN: vector [D x 1]
% N, D: scalar
% POST: structure (see fields used in the code)
% trainXt: matrix [D x N]
% kernel_SigmaLambda_inv, kernel_2SigmaLambda_inv_I, invLambda: vectors [D x 1]
% E_k: vector [1 x N]
% E_kk: symmetric matrix [N x N]
% E_xk: matrix [D x N]
% C_kk: symmetric matrix [N x N]
% G_S: diagonal matrix [D x D]
% calc_g_cov: boolean; true if G_Z is returned, false if only FGCOV
%
% About calc_g_cov: While a Matlab function can detect the number of outputs
% (nargout), MEX function generated by Matlab Coder does not and silently
% ignore it, causing logical bugs in the code. Therefore, we explicitly
% specify it in the input arguments instead. If C code or MEX function is
% created manually, one can instead detect the number of outputs
% properly and does not need this.

% calc_g_cov = nargout > 1;

% In Matlab, these can be initialized to 0; but in MEX,
% they need to have correct sizes
G_Z = zeros(D, D);
FGCOV = zeros(D, 1);

% Because Matlab Coder is stupid, we must define G_Cxkxk here or else
% it will complain.
% If manually writing the code, don't need the following line at all
G_Cxkxk = zeros(D, D);

for p = 1:N
    for q = p:N
        xp = trainXt(:, p);
        xqT = trainXt(:, q)';

        % Calculate Mpq used in calculating Cxkk and Cxkxk (D x 1)
        sum_xpxq = xp + xqT';
        G_Mpq = (XMEAN + kernel_SigmaLambda_inv .* sum_xpxq) ./ ...
                kernel_2SigmaLambda_inv_I;

        % Calculate C_{xkk} (D x 1)
        G_Exkk = E_kk(p,q)*G_Mpq;
        G_ExkEk = E_xk(:,p)*E_k(q);
        G_Cxkk = G_Exkk -G_ExkEk;

        if calc_g_cov
            % Calculate C_{xkxk} (D x D)
            G_Cxkxk = E_kk(p,q)*(G_S + G_Mpq*G_Mpq') - E_xk(:,p)*E_xk(:,q)';
        end

        % Update G_Z
        if p == q
            % Update FGCOV
            FGCOV = FGCOV + ...
                    POST.aa(p,q)*(G_ExkEk - E_k(p)^2*xp) + ...
                    POST.aaKinv(p,q)*(E_kk(p,q)*xp - G_Exkk);

            if calc_g_cov
                % Update G_Z
                G_Z = G_Z + ...
```

```
                        (xp*xqT*C_kk(p,q) - xp*G_Cxkk' - G_Cxkk*xqT + G_Cxkxk)*...
                        POST.aaKinv(p,q);
                end
            else
                alpha_alpha = POST.aa(p,q);
                G_Cxkk_qp = G_Exkk - E_xk(:,q)*E_k(p);

                % Update FGCOV
                FGCOV = FGCOV + ...
                        (alpha_alpha * C_kk(p,q) - POST.Kinv(p,q)*E_kk(p,q))*sum_xpxq ...
                        - alpha_alpha *(G_Cxkk + G_Cxkk_qp) + (2*POST.Kinv(p,q))*G_Exkk;

                if calc_g_cov
                    xpxq = xp*xqT;
                    G_Z = G_Z + ...
                          ((xpxq + xpxq')*C_kk(p,q) - xp*G_Cxkk' - G_Cxkk*xqT ...
                           - xqT'*G_Cxkk_qp' - G_Cxkk_qp*xp' + ...
                           G_Cxkxk + G_Cxkxk') * POST.aaKinv(p,q);
                end
            end
        end
    end

    FGCOV = (invLambda .* FGCOV)';

    if calc_g_cov
        % For some reason (possibly due to accumulated calculation errors),
        % G_Z is not symmetric although it should be. It seems that somehow the
        % upper triangle part is more accurate than the lower part (compared to
        % Monte Carlo). So we will keep  the upper part only (by copying it to
        % the lower part).
        G_Z = triu(G_Z) + triu(G_Z,1)';
    end
end
```

4.5.2 The general case

Not supported currently.

References

[1] E. Solak, R. Murray-smith, W. E. Leithead, D. J. Leith, and Carl E. Rasmussen. Derivative Observations in Gaussian Process Models of Dynamic Systems. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1057–1064. MIT Press, 2003.

[2] Andrew McHutchon. *Nonlinear Modelling and Control Using Gaussian Processes*. Ph.D., University of Cambridge, 2014.