Fast, Adaptive Methods for Gaussian Process Regression

Adam Lee, Dr. Peter Green, Prof. Vassil Alexandrov, Robert Allison

EPSRC Centre for Doctoral Training in Distributed Algorithms, University of Liverpool, Liverpool, UK

Gaussian Process Regression

Gaussian Process regression is an extension of Bayesian linear regression to the space of functions, allowing for non-parametric model estimation representing non-linear relationships between variables.

$$f(\mathbf{x}) \sim \mathcal{GP}(\ 0, \kappa\left(\mathbf{x}, \mathbf{x}'\right)\ ) \Rightarrow \mathbf{f} \sim \mathcal{N}\left(0, K_{nn}\right), \tag{1}$$

where f is our underlying function & $\mathbf{f}$ are noise-free realisations of this function. We then assume our observed data are our $\mathbf{f}$ with added noise such that $\mathbf{y} \sim \mathcal{N}(0, K_{nn} + \sigma^2 I)$. Training this model involves optimizing the model negative log-likelihood and its gradient with respect to a vector of model hyperparameters $\boldsymbol{\theta}$

$$\mathcal{L} = \log p(\mathbf{y}|X, \boldsymbol{\theta}) \propto -\mathbf{y}^{\mathrm{T}} \hat{K}_{XX}^{-1} \mathbf{y} - \log|\hat{K}_{XX}|, \tag{2}$$

$$\frac{d\mathcal{L}}{d\boldsymbol{\theta}_i} = \mathbf{y}^{\mathrm{T}} \hat{K}_{XX}^{-1} \frac{d\hat{K}_{XX}}{d\boldsymbol{\theta}_i} \hat{K}_{XX}^{-1} \mathbf{y} + \mathrm{Tr}\left(\hat{K}_{XX}^{-1} \frac{d\hat{K}_{XX}}{d\boldsymbol{\theta}_i}\right). \tag{3}$$

which is traditionally executed using the Cholesky factorisation of $\hat{K}_{nn} = K_{nn} + \sigma^2 I$. Alternatively, iterative system of linear algebraic equation (SLAE) solvers are used, namely the Conjugate Gradients algorithm.

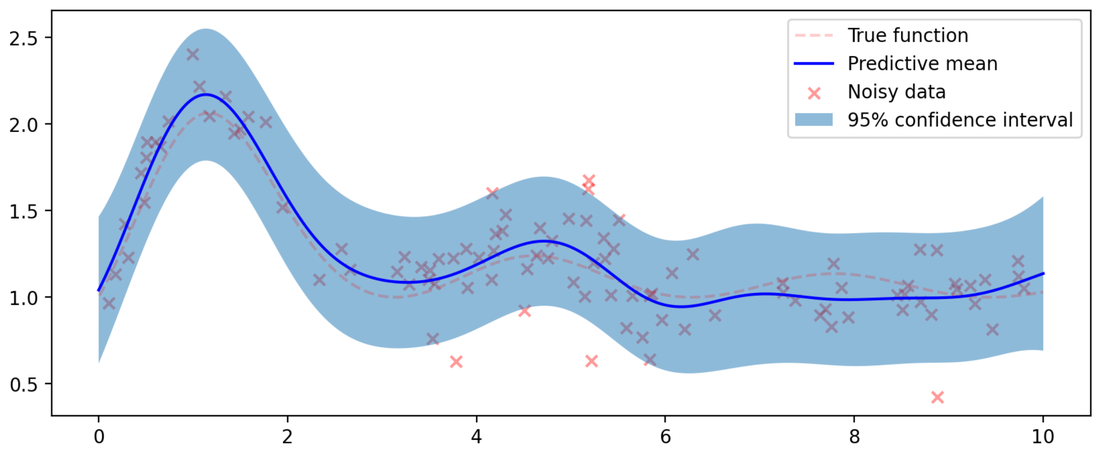

Figure 1: A simple one-dimensional Gaussian Process posterior on a non-linear function.

Conjugate Gradients and Iterative GPs

The Conjugate Gradients (CG) algorithm is an iterative routine for computing the solution to a positive-definite SLAE requiring n lots of matrix-vector products between the system itself and an iteratively updated solution vector $\mathbf{x}_i$ for the system $\hat{K} \in \mathbb{R}^{n \times n}$. Slight modifications to the standard algorithm result in the ability to compute both the system's solution and the log-determinant $\log|\hat{K}_{nn}|$. In practice, we truncate the work done by terminating CG after $m < n$ iterations, resulting in estimations for the required GP training quantities computed in more desirable time. The use of system preconditioners improves the rate of convergence for CG.

Algorithms

Algorithm 1: Pseudocode for Gaussian Process Training

1: Define objective function

$$\mathcal{L}_0 = -\mathbf{y}^{\mathrm{T}} \hat{K}_{XX|\boldsymbol{\theta}_0}^{-1} \mathbf{y} - \log|\hat{K}_{XX|\boldsymbol{\theta}_0}|$$

2: Define optimisation algorithm (SGD, Adam, L-BFGS-B) and gradients
3: $i = 0$
4: **while** $\mathcal{L}_i < -\epsilon$ **do**
5: $\quad i \leftarrow i + 1$
6: $\quad \boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} + \Delta_i$, using optimiser with PCG
7: $\quad$ Calculate $\mathcal{L}_i$
8: **end while**
9: $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_i$ is our MLE of hyper-paramters

Algorithm 2: Pseudocode for computing training quantities

1: Draw i.i.d $\mathbf{z}_i \sim \mathcal{N}(0, I_n)$ for $i = 1, \ldots, t$
2: Use a modified **PCG** to compute

$$[\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_t] = \hat{K}_{XX}^{-1}[\mathbf{y}, \mathbf{z}_1, \ldots, \mathbf{z}_i],$$

and partial tridiagonalisations

$$\tilde{T}_1, \ldots, \tilde{T}_t.$$

3: $\mathrm{Tr}\left(\hat{K}_{XX}^{-1} \frac{d\hat{K}_{XX}}{d\boldsymbol{\theta}_i}\right) \approx \frac{1}{t} \sum_{j=1}^{t} \left(\mathbf{z}_j^{\mathrm{T}} \hat{K}_{XX}^{-1}\right) \left(\frac{d\hat{K}_{XX}}{d\boldsymbol{\theta}_i} \mathbf{z}_j\right)$
4: $\det|\hat{K}_{XX}| \approx \frac{1}{t} \sum_{j=1}^{t} \mathbf{e}_1^{\mathrm{T}} \tilde{T}_j \mathbf{e}_1$

Algorithm 3: Pseudocode for modified, batch Preconditioned Conjugate Gradients

Input: func: mmp_A( ), $n \times t$ system rhs B, func: $\mathrm{P}^{-1}(\ )$
Output: $A^{-1}B$, $\tilde{T}_1, \ldots, \tilde{T}_t$
$U_0, R_0, Z_0, D_0 \leftarrow \mathbf{0},\ mmp\_A(U_0) - B,\ P^{-1}(R_0),\ Z_0$
$\tilde{T}_1, \ldots, \tilde{T}_t \leftarrow \mathbf{0}, \ldots, \mathbf{0}$
j = 0
for $j \leftarrow 0$ **to** t **do**
$\quad$ j $\leftarrow$j + 1
$\quad$ Perform standard conjugate gradients for search directions D_{j-1}
$\quad$ Convergence check with R_j
$\quad$ For all i, compute $\left[\tilde{T}_i\right]_{j-1:j,j-1:j}$ using gradients in 6 :
end for

Preconditioning

The need for preconditioning arises from the convergence analysis of Conjugate Gradients which suggests the number of iterations j for a CG solve to converge scales like the system's condition number

$$j = \mathcal{O}(\varkappa), \tag{4}$$

$$\varkappa = \frac{\max_s(\lambda_s)}{\min_t(\lambda_t)}. \tag{5}$$

Preconditioning a system $A \in \mathbb{R}^{n \times n}$ with preconditioner $P \approx A^{-1}$ in the context of CG involves simply multiplying our preconditioner with various column vectors. A common choice for a preconditioner for Gaussian Process systems is the incomplete/pivoted Cholesky decomposition, which computes the first k columns of the full Cholesky decomposition of K. An inverse matrix-vector product is then performed using the Woodbury inversion lemma. This project's current work involves the exploration of a family of potential preconditioning strategies developed from Gaussian Process approximation methods, such as the Nystrom method and structured kernel interpolation (SKI) method.

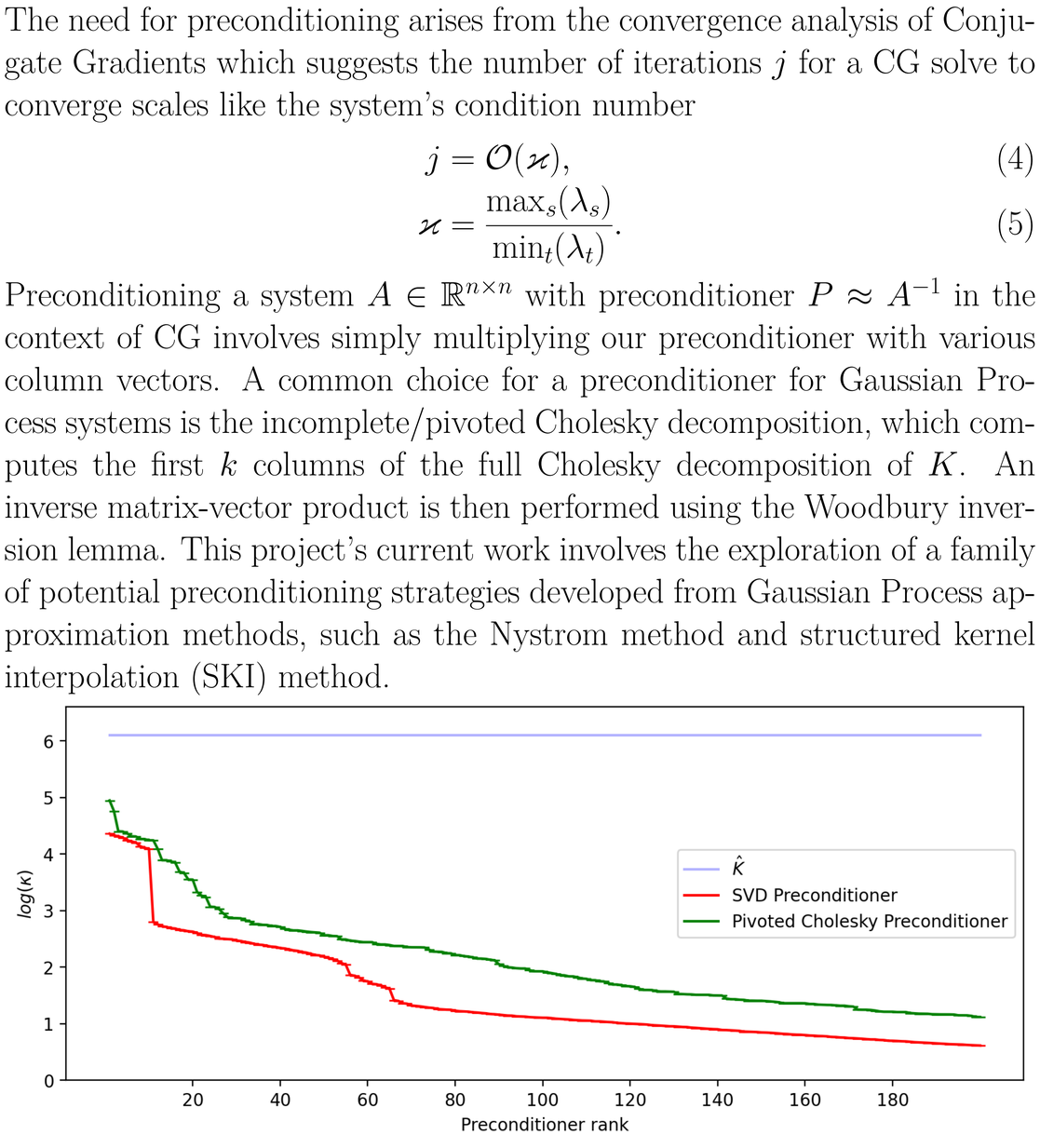

Figure 2: A comparison of the decay in condition number of a system before and after preconditioning

Engineering and Physical Sciences Research Council
UNIVERSITY OF LIVERPOOL
GCHQ