Scalable Gaussian Process Regression for Massive Astronomy Data Sets

Steven Stetzler

Introduction

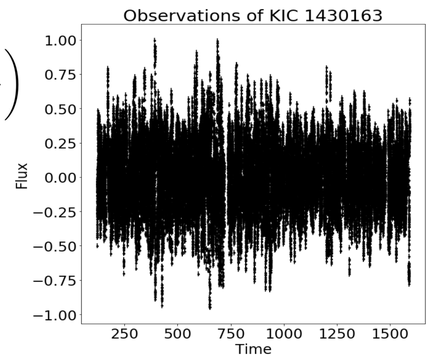
Gaussian Processes (GPs) have proven to be powerful, flexible, and interpretable machine learning models with applications in a broad range of astronomy topics, including modelling of how the brightness of stars change over time, exoplanet transits, and multi-band supernovae observations. At the core of each of these applications is the need to perform parameter inference through the GP model. In the language of computer science, this takes the form of hyperparameter optimization or kernel learning. In this work we explore efficient methods for performing hyperparameter optimization of a GP model over large datasets.

Data and Model

Our dataset (right) is a single light curve of the star KIC 1430163, which contains 51,505 measurements of the star's brightness over time. Datasets of this size are already intractable for a large number of exact GP solvers. We seek to model the variability of this star using a periodic kernel:

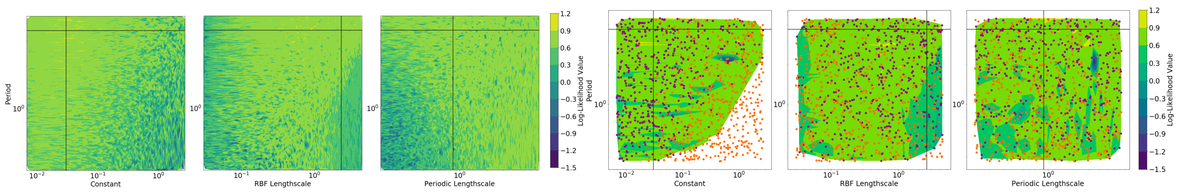
$$k_{periodic}(\mathbf{x}_1, \mathbf{x}_2) = A \exp\left(-\frac{1}{2}||\mathbf{x}_1 - \mathbf{x}_2||_2^2/l_R^2 + 2\frac{\sin^2(\pi||\mathbf{x}_1 - \mathbf{x}_2||/P)}{l_P^2}\right)$$

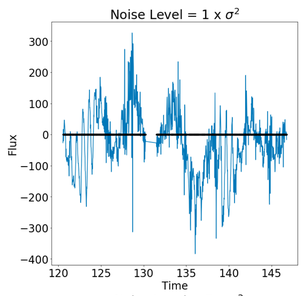
Modelling of the star's rotational period P is the same as finding the hyperparameter P of this kernel that best fits the data.

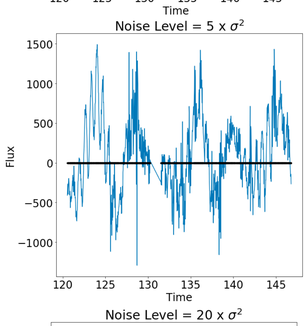


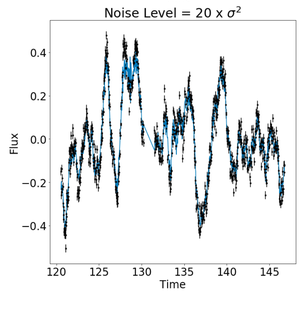
Evaluating Conjugate Gradient Methods

In astronomy applications, we are interested in hyperparameter optimization, and thus need the approximate method to accurately reproduce the loss surface of our problem. We explore how the use of conjugate gradient methods changes the loss surface. We've uncovered that conjugate gradient methods are not always great at doing this, as shown below. Below on the left is the loss surface plotted in two dimensions at a time when using an exact solver with the best fitting parameters overplotted. On the right is the loss surface using the same kernel with a solver based on conjugate gradients. **Many of the points tested did not produce converging gradients**!



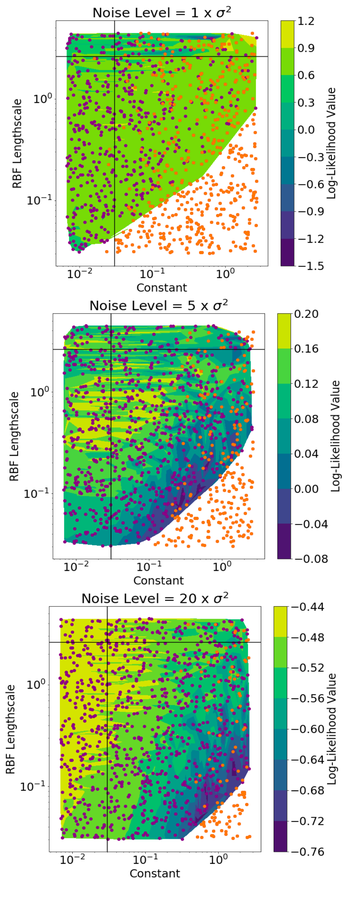






Noise May Be The Issue

We attempted to isolate the diverging conjugate problem by taking a hint from the preconditioner matrix **P** to vary the amount of noise in our model. To the left we show plots of the predictive mean compared with the actual data. To the right we show the estimated loss surface using a GP with a conjugate gradient solver. Each panel shows a different level of noise, at the observed noise level (top), five times that level (middle), and 20 times that level (bottom). **Only when we increase the noise substantially do we obtain a predictive mean and loss surface consistent with an exact solve**.



Background

A Gaussian Process is a probabilistic model centered around the multivariate normal distribution who's covariance matrix is specified with a kernel evaluated at the data. This kernel can be thought of as a similarity metric between points, with kernels assigning nearby points high similarity scores. Hyperparameters in the kernel specify what "nearby" means. The covariance matrix **K** is defined as:

$$\mathbf{K} = \begin{pmatrix} k(x_1, x_1) & \dots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \dots & k(x_n, x_n) \end{pmatrix}$$

Predictions using a GP require **inverting this matrix** and computing the log-likelihood requires an **inverse** and a **log-determinant** of this matrix.

Exact GPs on large data are intractable: $\mathbf{K}^{-1}, \log|\mathbf{K}| \rightarrow \mathcal{O}(n^3)$!

Inverses and log-determinants, can be drastically sped up using **linear conjugate gradients** to compute inverses and stochastic trace techniques to compute determinants. Use of a preconditioner matrix **P** speeds up these methods. GPyTorch is a python package that uses these methods to speed up exact inferences with the preconditioner $\mathbf{P} = (\mathbf{P}_k + \sigma^2\mathbf{I})$

Further speed up is obtained through **approximations** of the kernel. The **Structure Kernel Interpolation** (SKI) method uses **inducing points** that are interpolated from the data and are forced to lie on a grid, giving an approximation of the kernel: $\mathbf{K}_n \approx \mathbf{W}\mathbf{K}_m\mathbf{W}^T$

Covariance matrix over original data

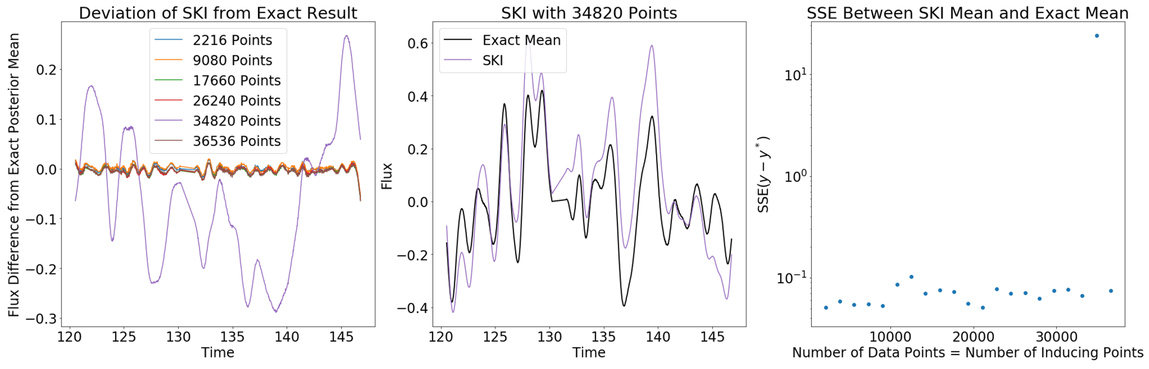
Covariance matrix over inducing points

Interpolation matrix

SKI speeds things up: $\mathbf{K}^{-1}, \log|\mathbf{K}| \sim \mathcal{O}(n + m\log m)$!!

Scaling to Large Data

These experiments show that SKI and other methods based on conjugate gradients may not be suitable for hyperparameter optimization, as arbitrary points of the parameter space may have misleading or diverging evaluations of the loss function and its gradient. However we can still perform tests to see if hyperparameters learned using an exact solver can be transferred to an approximate method. Below, we show the deviation of the predictive mean on the first 1000 data points of our dataset from the exact GP solver for an SKI model trained with increasingly larger sets of the data. **Deviation from the exact solution remain tolerable** over a broad range of data sizes.



We also performed experiments to verify the scaling of SKI methods on large data. We find that the runtime for computing the log-likelihood scales above linear (bottom right), consistent with the log-linear guarantee for SKI computations of the inverse and determinant of **K**. We find production of the posterior fit to the training data is more consistent with an n^2 scaling relation (bottom left), which is the guarantee for conjugate gradient methods.