Scalable Gaussian Process Regression for Massive Astronomy Data Sets

Steven Stetzler
Department of Astronomy
University of Washington
Seattle, WA 98195
stevengs@uw.edu

Abstract

I present background work and my own initial results in investigating applying scalable Gaussian Process (GP) regression to astronomy data sets. I focus on comparing fast approximate GP solvers with exact solvers in the context of hyperparameter learning. We find that approximate solvers that utilize conjugate gradients for fast matrix inverses are not suitable for hyperparameter learning. We perform initial work trying to integrate kernel learning utilizing exact solvers with the scalability of approximate methods.

1 Introduction

Gaussian Process (GP) regression has shown great promise in astronomical applications e.g. for modelling of stellar variability (how the brightness of stars change over time) [1], exoplanet transits (the dip in brightness of a far away star due to a planet passing in front of it) [2], and multi-band supernovae observations (how the brightness of a supernovae rises and decays on short timescales in multiple wavelengths) [3]. GPs have also proven to be powerful, flexible, and interpretable function approximators in many machine learning settings where physical parameter inference is not as important as learning an arbitrary target distribution. The main challenge of using a GP is its scalability as exact inference with a GP requires inverting and computing the determinant of a $n \times n$ matrix. These operations require $\mathcal{O}(n^3)$ computations, which is intractable for a single machine when $n > 10^{3-4}$. Future astronomical surveys expect to produce massive data sets with $\sim 10^9$ individual objects, that will each produce a data stream of 10^{3-5} observations with $d \geq 1$. Since we expect to be measuring many more individual objects than there are observations for each object ($10^9 >> 10^{3-5}$), then the speedup possible through the use of distributed computing will be more significant if we distribute computations over the objects as a group instead of over the GP inferences of each object. Thus, we cannot expect to take advantage of distributed computing methods to speed up GP inference, and the only path forward is to improve the scalability of the GP inference itself. In this report, I review prior work to improve the scalability of GP inference and apply those methods to astronomical data sets in the context of kernel learning. In §2 I provide the mathematical background necessary to understand Gaussian Process regression. In §3, I outline the prior applicable work on improving GP scalability, in §4 I outline our data and models, and in §5 I provide preliminary results in applying these methods to astronomical data sets and outline my further work.

2 Mathematical Background

Gaussian Process regression is one approach to modeling data with the form

$$Y = f(X) + \epsilon$$

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

where X is an input, Y is an output and ϵ is Gaussian random noise, which in the case of astronomical data sets takes the form of observational noise σ^2. A Gaussian Process can be thought of as a probability distribution over function values $f(X)$ which is updated based on data observed. In fact, the GP specifies that for any collection of data points $\mathbf{x} = \{x_i\}_{1 \leq i \leq n}$, the collection of function values $\mathbf{f} = \{f(x_i)\}_{1 \leq i \leq n}$ will be jointly Gaussian given some mean function μ and covariance matrix $\mathbf{K} = [k(x_i, x_j)]_{1 \leq i,j \leq n}$ specified by a *kernel function* over *pairs* of points $k(x_i, x_j)$:

$$\mathbf{f}(\mathbf{x}) \sim \mathcal{N}\left(\mu(\mathbf{x}), \mathbf{K}\right)$$

$$\begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu(x_1) \\ \vdots \\ \mu(x_2) \end{pmatrix}, \begin{pmatrix} k(x_1, x_1) & \dots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \dots & k(x_n, x_n) \end{pmatrix}\right)$$

Frequently, the mean function is taken to be zero: $\mu(X) = 0$. (We can always demean the data, and frequently subtract obvious trends e.g. linear ones before performing regression.). A common choice for the kernel function is the RBF kernel (or squared exponential kernel): $k(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right)$, although we will discuss the use of other kernels that can be used to model physical processes that astronomers are interested in.

The above equation represents a *prior* over functions, and does not yet incorporate training data. Given a location of a test point x_i, the above gives us a way to construct an $f(x_i)$ as a sample from a multivariate normal distribution. Rasmussen & Williams (2006) [4] derive the expressions for the *posterior* distribution that represents the distribution from which function values can be drawn once the GP has been conditioned on observed training data points $\mathbf{X}$ with corresponding targets $\mathbf{y}$ and noise measurements σ_n. For a new set of test points $\mathbf{X}_*$:

$$\begin{aligned} \mathbf{f}_* | \mathbf{X}, \mathbf{y}, \mathbf{X}_* &\sim \mathcal{N}\left(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)\right) \\ \bar{\mathbf{f}}_* &= \mathbf{K}(\mathbf{X}_*, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}_n]^{-1}\mathbf{y} \\ \text{cov}(\bar{\mathbf{f}}_*) &= \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}_n]^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}_*). \end{aligned}$$

Here, $\mathbf{I}_n$ represents the $n \times n$ identity matrix. Whereas before we may have specified that our mean function be $\mu(X) = 0$, we see now that once we have seen the training data $\mathbf{X}$, the new mean function for the multivariate normal from which predictions are drawn is $\bar{\mathbf{f}}_*$ and the covariance matrix of the multivariate normal has changed as well. This process of updating the mean function and covariance matrix is how the GP learns from the training data $\mathbf{X}, \mathbf{y}$ to predict (a distribution of) values $\mathbf{y}_*$ for new test data $\mathbf{X}_*$.

The above formalism gives one a way to make test predictions given observed data. However, using this formalism requires a choice of kernel function over pairs of points $k(x_i, x_j)$, which frequently depends on one or more *hyperparameters*. For the RBF kernel, the "lengthscale" σ^2 is a hyperparameter that should be chosen to best fit the training data (possibly best fitting a validation set as well). In astronomy, there are physical models that produce kernels, and the parameters of physical systems (for example, the rotation period of a star) are represented as hyperparameters of the GP model. To perform hyperparameter optimization, and thus to perform parameter inference in astronomy applications, one usually aims to minimize the log marginal likelihood of the observations. Again from [4], we find this expression is

$$\begin{aligned} \log \mathcal{L}(\theta) &= \log p(\mathbf{y}|\mathbf{X}, \theta) \\ &= -\frac{1}{2}\left(\mathbf{y} - \mu(\mathbf{X})\right)^T \left(\mathbf{K}_\theta + \sigma_n^2 \mathbf{I}_n\right)^{-1} \left(\mathbf{y} - \mu(\mathbf{X})\right) - \log |\mathbf{K}_\theta + \sigma_n^2 \mathbf{I}_n| - \frac{n}{2} \log 2\pi. \end{aligned}$$

where θ are our set of hyperparameters, and the subscript $\mathbf{K}_\theta$ signifies that these hyperparameters are specifying the covariance. The negative log marginal likelihood $-\log \mathcal{L}$ represents a *loss function* that we can pass to any optimization routine to minimize over hyperparameters θ and thus find the best fitting parameters of our model.

In performing both prediction (computing $\bar{\mathbf{f}}_*$) and loss minimization (computing $\log \mathcal{L}$), we must perform a matrix inversion, $(\mathbf{K} - \sigma_n^2 \mathbf{I}_n)^{-1}$, and compute a determinant, $|\mathbf{K} - \sigma_n^2 \mathbf{I}_n|$, of an $n \times n$ matrix. These operations typically require $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ storage, which is where the computational bottleneck of using GPs comes into play. This bottleneck is especially relevant when performing hyperparameter optimization, as the determinant and inversion must be performed once per evaluation of the log marginal likelihood, which usually happens once per iteration of an optimization routine.

3 Previous Work

A lot of work has been put into overcoming the computational bottleneck of matrix inversion and determinant computation. In this section, I summarize previous work on this topic that is relevant to the work I will be performing.

3.1 Structured Kernel Interpolation: KISS-GP

Wilson et al. (2015) [5] introduced a new framework for performing approximate GP inference in linear time, which they call Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). The authors build upon previous methods of using *inducing* points (or interpolating points) to approximate the kernel $\mathbf{K}$. For a kernel learned on n data points $\mathbf{X}$, the kernel $\mathbf{K_{X,X}}$ can be approximated using $m \leq n$ inducing points $\mathbf{U}$:

$$\mathbf{K_{X,X}} \approx \mathbf{K_{X,U}} \mathbf{K_{U,U}^{-1}} \mathbf{K_{U,X}}$$

where $\mathbf{K_{X,U}}$, $\mathbf{K_{U,U}}$ $\mathbf{K_{U,X}}$ are $n \times m$, $m \times m$, $m \times n$ respectively. This speeds up computations of the inverse of $\mathbf{K_{X,X}}$ to $\mathcal{O}(m^2 n + m^3)$ computations. These methods work great as long as the inducing points are chosen well to approximate the covariance matrix accurately, and computing these inverses will be tractable as long as the number of inducing points remains small: $m << n$. The authors build upon this and provide substantial speed up by noticing that the matrix $\mathbf{K_{X,U}}$ can be approximated by interpolating points from $\mathbf{K_{U,U}}$. For example, if the inducing points are assumed to lie on a grid, then the kernel evaluation between data point x_i and inducing point u_j can be written as

$$k(x_i, u_j) \approx w_i k(u_a, u_j) + (1 - w_i) k(u_b, u_j)$$

where u_a and u_b are the two inducing points that most closely bound x_i from above and below: $u_a \leq x_i \leq u_b$ (i.e. x_i lies between u_a and u_b), and w_i is an interpolation weight. Here we assume a linear interpolation, but more complex interpolations can be used. Thus the matrix $\mathbf{K_{X,U}}$ can be written as

$$\mathbf{K_{X,U}} \approx \mathbf{W} \mathbf{K_{U,U}}$$

where $\mathbf{W}$ is an extremely sparse matrix of interpolation weights. Here we assumed a linear interpolation, and thus for each data point x_i there exist only two interpolation weights and thus two non-zero entries per row of $\mathbf{W}$. We can use more complex interpolation schemes at the cost of sparsity of $\mathbf{W}$. In a complementary manner, $\mathbf{K_{U,X}} = \mathbf{K_{U,U}} \mathbf{W}^T$, giving us the approximation for $\mathbf{K_{X,X}}$ of:

$$\begin{aligned} \mathbf{K_{X,X}} &\approx \mathbf{K_{X,U}} \mathbf{K_{U,U}^{-1}} \mathbf{K_{U,X}} \\ &\approx \mathbf{W} \mathbf{K_{U,U}} \mathbf{K_{X,U}^{-1}} \mathbf{K_{U,U}} \mathbf{W}^T \\ &= \mathbf{W} \mathbf{K_{U,U}} \mathbf{W}^T \end{aligned}$$

Inverses of $\mathbf{K_{X,X}}$ can be computed efficiently through the use of linear conjugate gradients and matrix matrix-vector products, which come at a cost of $\mathcal{O}(n + m^2)$ computations. Computation of these inverses converges quickly, or at least with many fewer iterations than n.

Further speed up comes from a specific choice of the inducing points $\mathbf{U}$. If the inducing points are constrained to lie on a grid, then we can take advantage of Toeplitz structure (where the diagonal of a matrix is constant) of $\mathbf{K_{U,U}}$ to compute matrix-vector products in $\mathcal{O}(n + m \log m)$ computations and $\mathcal{O}(n + m)$ storage. In $d > 1$ dimensions, Kronecker structure of $\mathbf{K_{U,U}}$ can be used to compute matrix-vector products in $\mathcal{O}(dm^{1+1/d})$ computations with $\mathcal{O}(n + dm^{2/d})$ storage. Kronecker structure can be formed when the kernel factorizes over the dimensions of the data: $k(x_i, x_j) = \prod_{p=1}^{d} k(x_i^p, x_j^p)$, which allows one to write the covariance matrix as a Kronecker product over dimensions:

$$\mathbf{K} = \mathbf{K}_1 \otimes ... \otimes \mathbf{K}_d$$

These speedups are linear in n and almost linear in m. This allows us to perform almost exact inference by setting $m \sim n$. This speed up comes from the grid structure of the inducing points, and for data that already lie on a regularly spaced grid (e.g. image data), no inducing points are needed to obtain this speedup. However, for almost all astronomy applications data will never lie on a uniform grid naturally, and this inducing point method is required to obtain these speedups.

The `GPyTorch`[1] Python package implements regression with the KISS-GP framework, and acts as a general environment within which to perform GP inference. I use this package to perform inference within the KISS-GP framework.

3.2 GPyTorch: Efficient Gaussian Processes

`GPyTorch`[6] is a Python package that accelerates Gaussian Process regression and classification by utilizing parallel hardware and linear conjugate gradients. Linear conjugate gradients[2] attempt to solve systems of equations of the form

$$\mathbf{Ax} = \mathbf{b}$$

by using the fact that the deviation from the optimal solution, $\mathbf{Ax} - \mathbf{b}$, is also the gradient of the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{x}^T\mathbf{b}.$$

This allows for the optimal solution $\mathbf{x}^*$ to be found in an iterative manner by minimizing $f(\mathbf{x})$. One important quantity computed in this iterative method is the residual $\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k$ where k is the iteration number. If this vector does not converge to $\mathbf{0}$ after many iterations, that means that the conjugate gradient method is not converging.

`GPyTorch` uses *preconditioning* of the matrix $\mathbf{A}$ to help the iterative method converge. Preconditioning methods find solutions to the equation

$$\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b}$$

which for some choices of the preconditioner, $\mathbf{P}$, drastically speeds up this iterative method. `GPyTorch` specifically uses the Pivoted Cholesky Decomposition of the covariance matrix with an added noise term as the preconditioner:

$$\mathbf{P} = (\mathbf{P}_k + \sigma^2\mathbf{I}) \tag{1}$$

where $\mathbf{P}_k$ is the rank k pivoted Cholesky decomposition of the covariance matrix $\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \mathbf{P}_k$ and σ^2 is the modelled noise term of our observations. In our results, we evaluate the ability of linear conjugate gradient methods to converge when σ^2 is both fixed (not a model parameter) and small.

4 Data and Model

The dataset we are concerned with in this work is the stellar flux of the star KIC 1430163[3], a star with noticeable variability in its brightness due to stellar rotation. In total, this star has $n = 51505$ observations for it, shown in Fig. 1. Astronomers are interested in inferring the period of the star's rotation, which can be inferred directly from the variability in the data. In order to model this variability, we use the semi-periodic kernel

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

which is a scaled combination of an RBF kernel and a periodic kernel. Here, A is a scaling constant, referred to as "Constant" throughout, l_R is the lengthscale associated with the RBF kernel ("RBF Lengthscale" throughout), P is the period ("Period"), and l_P is the lengthscale of the periodicity ("Periodic Lengthscale" throughout).

We are interested in performing a hyperparameter optimization to find the value of P that best fits the data, usually marginalizing over choices for other parameters. (See e.g. [7] for a similar approach using this same data.) However, with 51505 data points, exact GP solvers will struggle to fit the data in a reasonable amount of time. In the following section, we explore methods used to speed up this process and expose flaws in the process of using these approximate methods for hyperparameter learning.

[1] https://github.com/cornellius-gp/gpytorch

[2] See https://en.wikipedia.org/wiki/Conjugate_gradient_method

[3] Extracted from the Kepler data set: https://archive.stsci.edu/kepler/

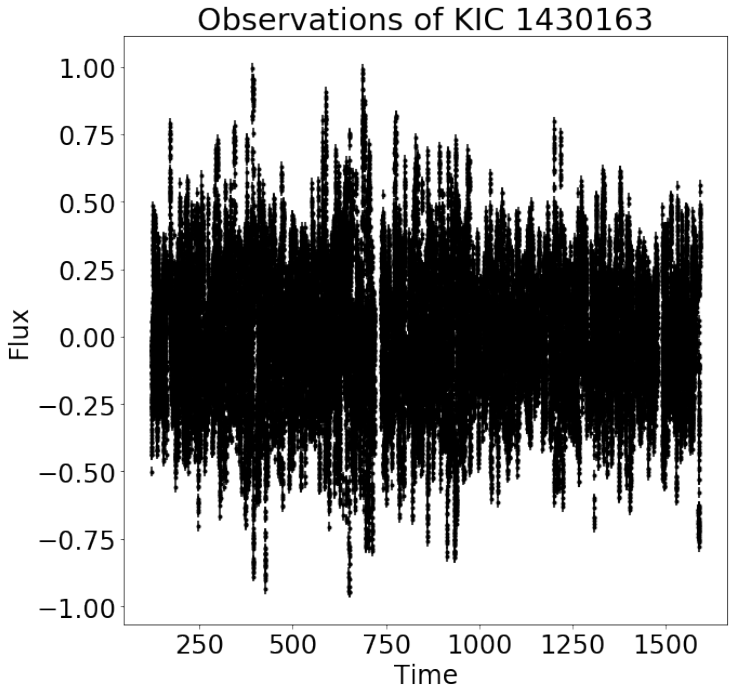

Figure 1: Observation of the star KIC 1430163

5 Results

Expanding the work from [5] (§3.1) to astronomy related data sets has one large hurdle: this method provides an *approximation* of the posterior distribution. In scientific applications, it is frequently the case that computational resources will be sacrificed as long as an exact result can be obtained. However, this trend may not continue as astronomy data sets become larger. It is then necessary to show that approximate methods are sufficient for parameter estimation on a data set.

My initial goal was the perform a thorough investigation of how the Structured Kernel Interpolation (SKI) method performs in the parameter inference regime. Specifically, I was hoping to perform a fully Bayesian estimation of the marginal posterior distribution of periods that best fit the data by using a Markov chain Monte Carlo (MCMC) routine. This would give me both an estimate of the inferred period and a measure of the uncertainty of that period. I would then tweak the SKI method to make it more or less approximate and see how the inferred period changes. This would provide a definitive answer as to whether we can trust the SKI approximate to produce near exact results. However, I quickly ran into issues that made this compelling path impossible.

First, I noticed that the SKI method as implemented in GPyTorch did not provide a converging fit to the data in a consistent manner when varying the hyperparameters of the model. What this means is that parameter inference through kernel learning would be essentially impossible, as we could not get an accurate measure of the log-likelihood for some values of the hyperparameters. Next, I backed away from using the SKI method to using an exact kernel in GPyTorch. I found the exact same behavior of the fit not converging for some values of the hyperparameters. I then pivoted my project to thoroughly understanding these inconsistencies instead of focusing on producing a final estimate of the ability of SKI to provide accurate parameter inferences. In the following sections, I outline this procedure. Unless otherwise stated, the semi-periodic kernel (Eqn. 2) was used as our base model. For exact inferences, we only fit the model to the first 1000 points of the time series, as fitting many more would prove infeasible for exact GP solvers.

5.1 Kernel Learning with Exact Kernel and Exact Solver

Since we are interested in learning the hyperparameters of our model, I first set out to produce a plot of the loss surface of our model under the data we are considering (the first 1000 points of the time series). Shown in Fig. 2 are contour plots produced by randomly and uniformly sampling 10^4 points over fixed bounds in our four dimensional parameter space and evaluating the log-likelihood of a model fit to our data using the semi-periodic kernel. `GPyTorch` produces inconsistent behavior (due to bugs in the code) when turning off settings that accelerate the GP fit. Because of this, I performed experiments that utilize an exact solver (no conjugate gradients) using the Gaussian Process Regression implementation in the `sklearn` package. This map of our parameter space serves as a baseline for evaluating the inconsistencies that occur when fitting using conjugate gradients and the SKI method. We see from this map that the loss function is relatively well behaved. We take the parameters with maximum likelihood from these tests as our "ground truth" for the best fitting parameters which we use extensively later

	Constant	RBF Lengthscale	Periodic Lengthscale	Period	Log-Likelihood
Value	0.03	2.34	0.84	3.54	0.91

Table 1: The best fitting parameters from a hyperparameter search using an exact GP solver on the first 1000 data points of the time series.

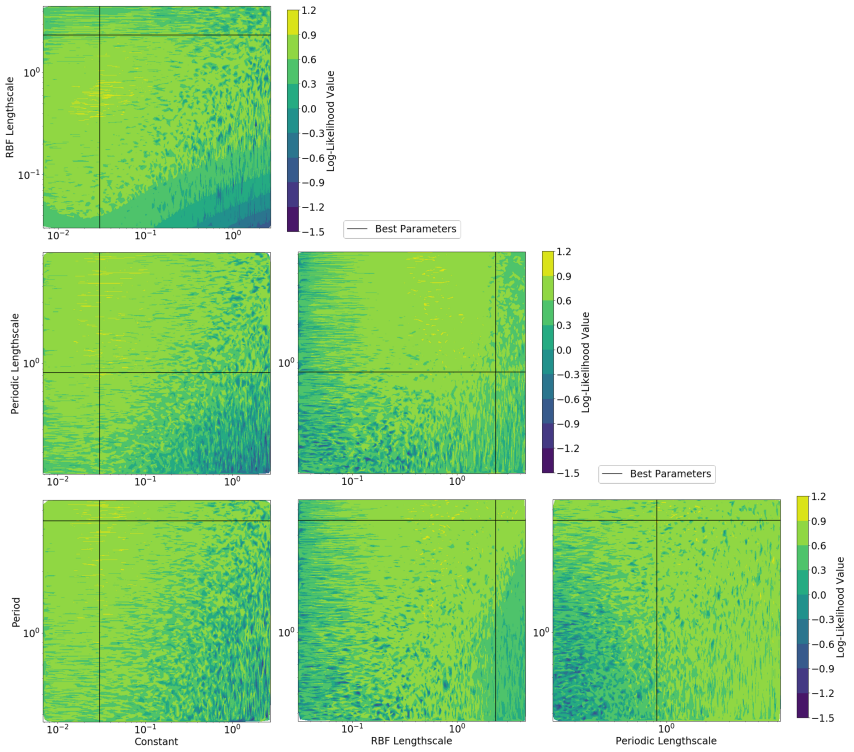

Figure 2: The loss surface of the semi-periodic kernel plotted using pairwise combinations of the hyperparameters. Color indicates value of the log-likelihood (higher is better) and black lines indicate the point where the maximum likelihood occurred.

5.2 Kernel Learning with Exact Kernel: Linear Conjugate Gradients

Linear conjugate gradients provide a way to find an approximate solution to a linear system. An iterative process can be used to find the approximate solution using just a few iterations, which usually makes these methods much faster than computing exact matrix inverses. GPyTorch uses these gradients extensively to speed-up computation, even for exact kernels. Here we evaluate at which values of the hyperparameters these gradients converge. Figure 3 shows the loss surface of

our model under the data along with the locations of tested points in the parameter space. At each point, I tested for convergence of the conjugate gradients by evaluating the residual norm produced by the iterative conjugate gradient method. If conjugate gradients always converged and produced solutions close to exact values, we would expect that Fig. 3 and Fig. 2 would be identical. That is, the loss surfaces would look the same as the inverses and log determinants of the kernel matrices would be identical in both cases. However, we notice that for almost all of the hyperparameters tested, the linear conjugate gradients did not converge. And even in regions where they did converge, the loss surface does not resemble the one produced through an exact solver. This is very worrisome as this means that for arbitrary combinations of the hyperparameters, even in regions close to the parameters that supposedly maximize the likelihood of the model, these conjugate gradients cannot converge and their results should not be trusted.

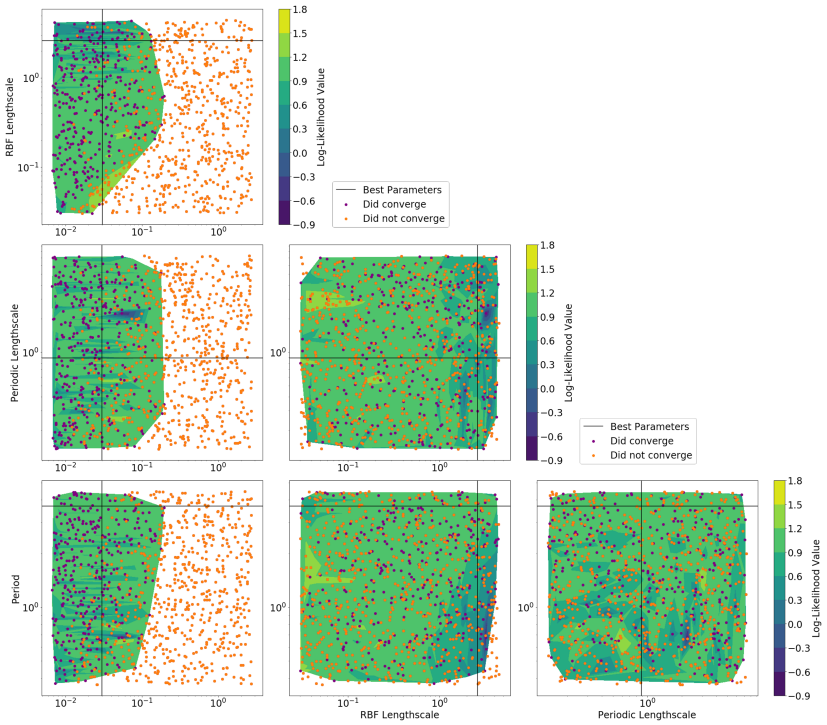

Figure 3: The log-likelihood evaluated over 10^3 randomly and uniformly chosen points in our parameter space using an exact kernel with acceleration from the use of conjugate gradients. Color of the contour indicates the value of the log-likelihood. Purple points indicate where the GP solver was able to converge using conjugate gradients, and orange points indicate where the solver did not converge. We observe that the majority of chosen points did not allow for the model to converge on a fit.

5.3 Fixed Noise as a Potential Source of Diverging Gradients

By inspecting Eqn. 1, we can see that the preconditioner used by GPyTorch to fit the Gaussian Process quickly includes a noise term. In contrast to many other settings, we are using a fixed noise model as our data have inherent observational noise that has already been characterized and estimated. Thus, no further modeling of the noise is required. For our dataset, this fixed noise is quite low, ~ 0.018, and this led to a hunch that the preconditioner being used might not be a good enough preconditioner when the noise is small. To test this, we varied the amount of noise in the data from the base level of $\sigma^2 \sim 0.018$ to $10 \times \sigma^2$ and $20 \times \sigma^2$. Shown in Fig.4 is the posterior mean that the `GPyTorch` fit converged to at these three noise levels. In these three cases, we use an arbitrary set of hyperparameters for our model, *not* the maximum likelihood parameters. We see that for the base noise level and the $10 \times \sigma^2$ noise level, the fit has obviously not converged. Only when we increase the noise level to $20 \times \sigma^2$ does the model actually converge to a reasonable answer under these hyperparameters, however the fit is not as good as one might expect since the modeled noise is

high and so larger deviations from the mean are tolerated. Some may point out that this behavior is due to the set of hyperparameters chosen being inconsistent with the noise level i.e. *no posterior exists* that can fit the data. To investigate this, we tested the exact solver from `sklearn` with the same choice of hyperparameters and fixed noise levels and found that a reasonable posterior mean could be produced in each case. This means that it is likely that the small noise level is interfering with the preconditioning of the conjugate gradients.

Finally, for the $20 \times \sigma^2$ noise level, we again perform a hyperparameter search to evaluate the loss function, shown in Fig. 5. We see that by increasing the noise level, we've increased the chance that the conjugate gradient method converges for an arbitrary choice of hyperparameters. However, there are still points where convergence is impossible, proving kernel learning to be intractable on this data set with linear conjugate gradient methods.

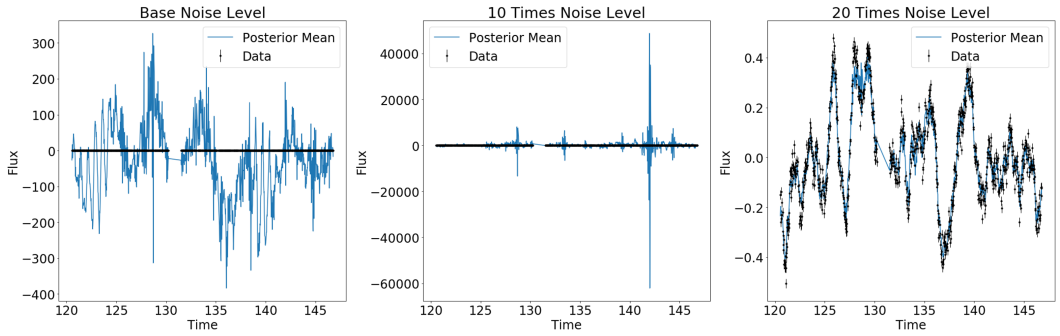

Figure 4: The posterior mean produced by a fit to the data using the linear conjugate gradient method at a noise level $\sigma^2 \sim 0.018$ (left), $10 \times \sigma^2$ (middle), and $20 \times \sigma^2$ (right). We can see that the only reasonable posterior mean is produced for noise much higher than actually exists in our data.

5.4 Extending Exact Solutions Over Large Data

While the previous experiments showed that inferring the best fitting hyperparameters is simply intractable on our dataset, thus ruling out the use of the SKI method to infer the best fitting parameters, we can still try to evaluate the accuracy of the SKI method when expanded over larger data. We performed an experiment where we used the best fitting hyperparameters from the exact GP solver over the first 1000 data points in the time series and extrapolated out the solve over larger portions of the dataset using the SKI method. We then compare the posterior mean produced from each of these fits on the first 1000 data points with the posterior mean produced by the exact solver, as shown in Fig. 6. Figure 6 shows that as we expand the dataset fed to the SKI method past the first 1000 points, the posterior mean of the SKI model remains relatively constant. Over the first 1000 data points, the sum squared error between the exact solver's predicted mean and the SKI method's predicted mean stays below the 10^{-1} level. These results are promising. They tell us that even if we can't perform kernel learning with SKI, and even if we can't scale our exact method to large data, there is still hope that we can merge these two methods by performing kernel learning on a subset of the data with an exact solver and using the SKI method to scale that solution to a larger data set and make future predictions.

We also explored how the accuracy of the SKI method fares as its "approximation knob", the number of inducing points used to approximate the covariance matrix, is tuned to make the approximation better or worse. Figure 7 shows the difference between the SKI method's predictive posterior mean and the mean produced by an exact solver. Both models use the best fitting hyperparameters and are fit over the first 1000 data points alone. We can see that the mean prediction only starts to deviate when we use fewer than $m = 0.1 \times n$ inducing points where $n = 1000$ is the number of data points. This means that the SKI method is a robust approximator over a large range of number of inducing point and that having as many as or more inducing points than data points is not necessary in order to produce near-exact results.

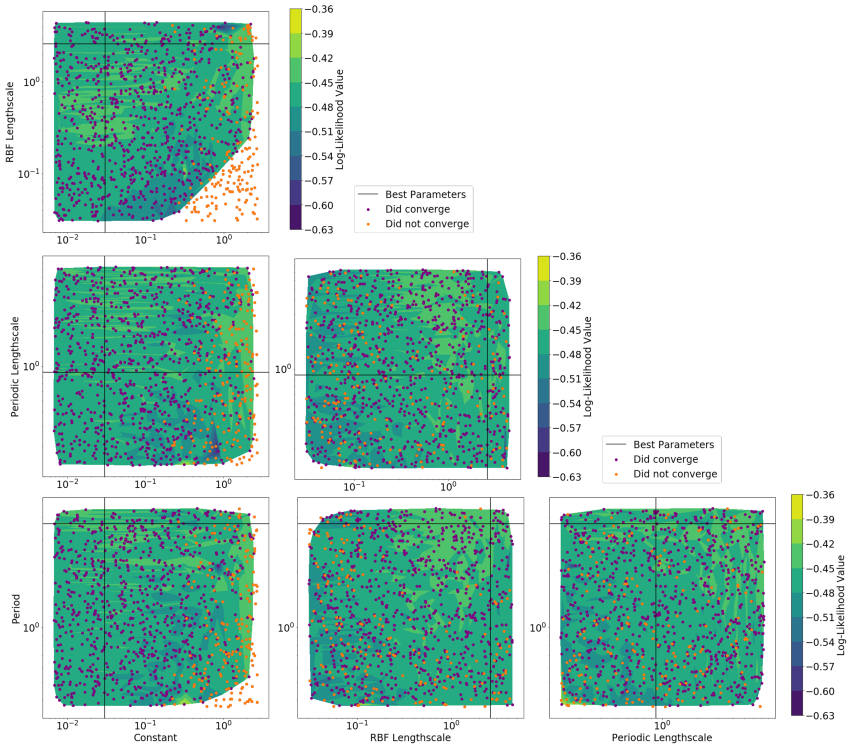

Figure 5: The log-likelihood evaluated over 10^3 randomly and uniformly chosen points in our parameter space using a fixed noise level of $20 \times \sigma^2$. Color of the contour indicates the value of the log-likelihood. Purple points indicate where the GP solver was able to converge using conjugate gradients, and orange points indicate where the solver did not converge.

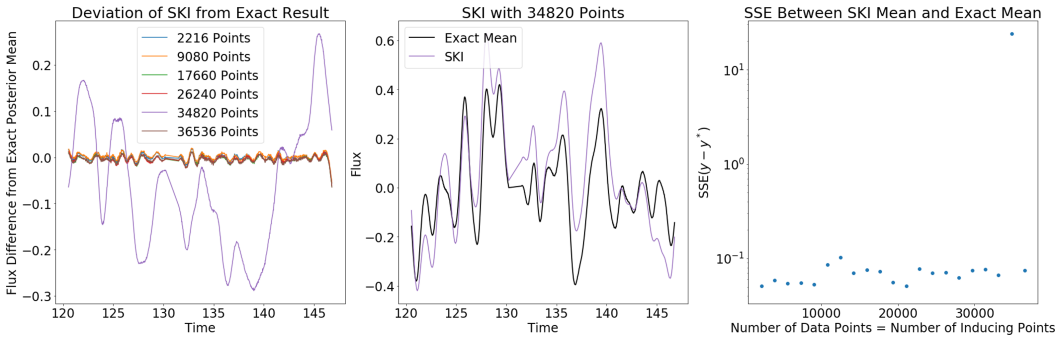

Figure 6: The left plot shows the deviation from the exact solver's predictive posterior mean over the first 1000 data points of the SKI method when trained on many more data points. The SKI method produces a predictive mean close to that of the exact solver, except in one case (middle) which is still not understood. The right plot shows the sum squared error between the two means as a function of the number of data points used to fit the SKI method.

5.5 Runtime Analysis of SKI

Finally, we perform a run time analysis of the SKI method to confirm its promise of a fast solve. Figure 8 shows the wall clock runtime of both the fit of the GP model to the data (required for prediction), and the evaluation of the log likelihood (required for kernel learning). All experiments were performed using a single computer with $2\times$ AMD EPYC 7401 CPUs (48 cores) and 1 TB of RAM. `GPyTorch` was allowed to use as many resources as required and did not saturate the machine. We see that the posterior fitting time scales as expected at $\mathcal{O}(n + m^2) = \mathcal{O}(n^2)$ for $m = n$, where

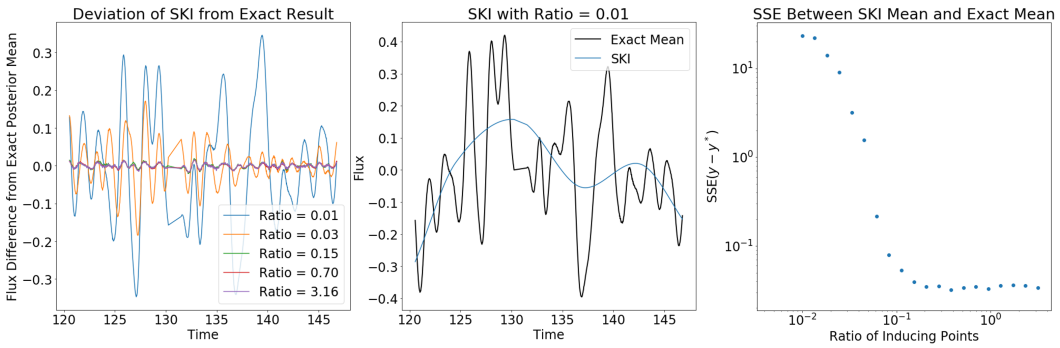

Figure 7: The left plot shows the deviation from the exact solver's predictive posterior mean over the first 1000 data points of the SKI method when the number of inducing points is changed. The middle plot shows the behavior of the predictive mean produced by SKI when the number of inducing points is much smaller than the number of data points. The right plot shows the sum squared error between the means, and indicates that acceptable SSE is obtained for $m \geq 0.1n$.

in these experiments we use as many inducing points as data points. This is the recovered scaling from matrix-vector products when using linear conjugate gradients. Evaluations of the log-likelihood recover a linear scaling $\mathcal{O}(n)$, which is a tractable result for kernel learning, as promised by the SKI method.

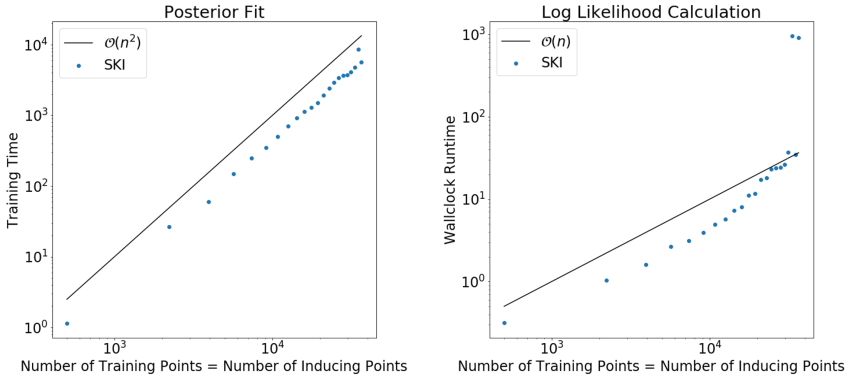

Figure 8: The wall clock runtime of producing a fit to the training data (left) and evaluation of the log-likelihood (right). Both of these results recover the expected scaling. The deviation in the run time for the experiments with two largest number of training points is not understood.

6 Conclusions

We set out to compare the performance of exact GP solvers and the Structured Kernel Interpolation method for providing an approximate solves in the context of kernel learning, where we are interested in finding hyperparamters that are the best fit to our training data. We found that limiting the size of fixed noise in our data makes kernel learning intractable on our dataset with SKI methods, as the preconditioning of the linear conjugate gradients used for approximate solves is not robust to small noise terms. In essence, for an arbitrary choice of hyperparameters, the SKI method is not guaranteed to produce a converging fit whereas the exact solver is. We attempted to merge kernel learning with an exact solver and approximate posterior inference with the SKI method in order to redeem the SKI method in the kernel learning context. We found that our method produced tolerable errors, indicating that this may be a promising method to merge exact and approximate solvers. Finally, we performed tests to examine the runtime scaling of SKI methods and recovered the promised $\mathcal{O}(n^2)$ complexity for GP fitting and $\mathcal{O}(n)$ complexity for evaluation of the log-likelihood.

References

[1] Ruth Angus, Timothy Morton, Suzanne Aigrain, Daniel Foreman-Mackey, and Vinesh Rajpaul. Inferring probabilistic stellar rotation periods using Gaussian processes. *Monthly Notices of the Royal Astronomical Society*, 474(2):2094–2108, 09 2017.

[2] N. P. Gibson, S. Aigrain, S. Roberts, T. M. Evans, M. Osborne, and F. Pont. A Gaussian process framework for modelling instrumental systematics: application to transmission spectroscopy. *Monthly Notices of the Royal Astronomical Society*, 419(3):2683–2694, 01 2012.

[3] A. G. Kim, R. C. Thomas, G. Aldering, P. Antilogus, C. Aragon, S. Bailey, C. Baltay, S. Bongard, C. Buton, A. Canto, F. Cellier-Holzem, M. Childress, N. Chotard, Y. Copin, H. K. Fakhouri, E. Gangler, J. Guy, M. Kerschhaggl, M. Kowalski, J. Nordin, P. Nugent, K. Paech, R. Pain, E. Pecontal, R. Pereira, S. Perlmutter, D. Rabinowitz, M. Rigault, K. Runge, C. Saunders, R. Scalzo, G. Smadja, C. Tao, B. A. Weaver, and C. Wu. STANDARDIZING TYPE ia SUPERNOVA ABSOLUTE MAGNITUDES USING GAUSSIAN PROCESS DATA REGRESSION. *The Astrophysical Journal*, 766(2):84, mar 2013.

[4] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[5] Andrew Gordon Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). *International Conference on Machine Learning (ICML)*.

[6] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. *arXiv e-prints*, September 2018.

[7] Daniel Foreman-Mackey, Eric Agol, Sivaram Ambikasaran, and Ruth Angus. Fast and scalable gaussian process modeling with applications to astronomical time series. *The Astronomical Journal*, 154(6):220, nov 2017.