Extending and Applying the Gaussian Process Autoregressive Regression Model

Justin Bunker

Supervisor:
Dr. Richard E. Turner

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Magdalene College August 2019

I would like to dedicate this thesis to my sister and both of my parents whose love and support I will always be thankful for.

Declaration

I, Justin Bunker of Magdalene College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. The word count of this thesis does not exceed 10500 words. No existing software, apart from several key standard libraries, was used for the work required in this thesis. A list of these libraries is provided within the experiments section.

Justin Bunker
August 2019

Acknowledgements

I would personally like to thank my supervisor, Dr. Turner. During our numerous meetings, I was able to learn a lot about how to approach problems with a researcher's mindset and this insight has greatly enhanced my experience. I am also very grateful for all of the advice he has provided me.

Also, I would like to thank Wessel Bruinsma for all of the great discussions and constructive feedback used towards the writing of this report. Despite being very busy, Wessel always found the time to provide valuable advice and ideas on different matters related to this project. It was always a pleasure to interact with him.

Abstract

The goal of this thesis is to investigate different properties and applications of the Gaussian Process Autoregressive Regression (GPAR) model [9]. Through the use of multiple Gaussian processes, GPAR can efficiently model multi-output systems by using information about the relationship between the various outputs. The goal of this thesis is twofold. First, we examine and describe GPAR behaviour when dealing with random hyperparameter initialisation, compositional kernel search, and optimal conditional ordering. Second, we discuss several approaches for using GPAR to enhance performance in solving multiple integrals for related integrands using Bayesian quadrature [12] and also for hyperparameter tuning with the freeze-thaw approach [11].

Contents

List of Figures

List of Tables

1 Introduction

When modelling multiple time-series outputs, it is often the case that the different outputs are related to each other in a causal manner. For instance, we may be interested in modelling the following financial securities: an index representing a basket of equities of different sectors, a different index representing a basket of technology equities, and finally a technology stock. Knowledge of the values of the equities index can indicate what the values could be for the technology index. The same logic follows for using the values of the technology index to help predict values of the technology stock. Exploiting such relationships is paramount to the success of GPAR. It is also important for one to be mindful of situations where the number of outputs, modelled by GPAR, increases in size. More specifically, we are interested in how larger GPAR models can be trained efficiently through the use of random restarts. We also wish to study how sensitive the GPAR predictive performance is to the selection of the kernel, a major component of GPAR. If it turns out that only a select few kernels are appropriate for a given task, then it is critical to come up with strategies to find good ones efficiently. Finally, for similar reasons as given for kernel selection, we also want to investigate the importance of another major element of GPAR: output ordering.

Bayesian quadrature is an approach, rooted in probabilistic numerics, that treats integration as an inference problem. The success of this approach relies on how well the integrand is being modelled. Given a set of related integrands, it would be sensible to explore whether GPAR can exploit the relationship between the integrands to provide better models which would then be incorporated into the Bayesian quadrature framework. We investigate a few possible approaches that use GPAR for Bayesian quadrature. Unfortunately, Bayesian quadrature requires the computation of a certain number of integrals, which are usually expensive. One possible idea, which was originated by Wessel Bruinsma, is to flip the problem around by using a clever trick involving derivatives. This idea will be explored in greater detail.

Automated approaches to finding the right set of hyperparameters for machine learning models are usually expensive since a model must be trained for each of the proposed hyperparameter configurations. Each of these trained models is usually assessed using a heuristic, such as a loss function, which attempts the quantify the performance of the model. This loss function is usually plotted while the model is being trained. Hyperparameter tuning is a good use case for Bayesian optimisation, which is an approach that is typically used to optimise functions which are expensive or difficult to evaluate. The particular Bayesian optimisation approach that is explored in more detail within this report attempts to model the loss for models, each of which corresponds to a different hyperparameter configuration. Since these hyperparameters are being used within the same model, it may be sensible to suggest that their respective losses could be related to one another. Therefore, GPAR can be used to improve the process of modelling these loss functions, and hopefully, this could translate to finding appropriate hyperparameters much faster. We investigate to what extent GPAR is able to model these loss curves.

2 GPAR Model

2.1 Overview

Multi-output systems are ubiquitous and span many different domains. Examples include forecasting battery cell capacities [2], traffic data imputation [10], blood glucose trajectory predictions [4], and early sepsis detection [5]. Having efficient and accurate modelling approaches for these systems would provide immense practical benefits. Models for such systems must exploit inter-output relationships to characterise the behaviour of every single output better.

The idea behind GPAR is simple. First, a suitable ordering of the various outputs is chosen. Second, for each output, we fit a Gaussian process that receives as inputs the outputs of the functions that are before it in the order that was previously defined. This strategy hinges on the fact that information about one function can increase the predictive capabilities for other functions.

This section presents a small overview of GPs and then builds on top of this to give a complete description of the GPAR model. Particular attention is brought to the training methodology and how specific design choices could be taken into account. Finally, several considerations about dealing with noisy observations, sharing hyperparameters, and random initialisations are then discussed in more detail.

2.2 Gaussian Processes

A Gaussian Process (GP) is a particular stochastic process. A stochastic process $\{Y(\mathbf{x}), \mathbf{x} \in \mathbf{X}\}$ may be defined as a collection of random variables $Y(\mathbf{x})$ that are each associated to an index $\mathbf{x}$. For a stochastic process to be Gaussian, all finite subsets of this collection of random variables must follow a joint Gaussian distribution.

Just like the multivariate Gaussian distribution is a generalisation of the univariate Gaussian distribution into multiple dimensions, one can regard the GP as a generalisation of the multivariate Gaussian distribution into the function space. In other words, a GP can be seen as a multivariate Gaussian distribution with an infinite number of dimensions. Namely, it allows for the specification of a probability distribution over functions such that any finite values for each of those functions follow a Gaussian distribution.

Any GP can be fully characterised by specifying a mean function $m(\mathbf{x})$ and a kernel $K(\mathbf{x}, \mathbf{x}')$. The term "kernel" is often used interchangeably with the term "covariance function" since $K(x, x') = \text{cov}(f(x), f(x'))$ where x and x' are indices that are respectively associated to $f(x)$ and $f(x')$. The mean function and kernel pair encapsulate prior beliefs about the behaviour of any given function which we seek to model. The mean function can describe the average value of the modelled function as we move away from observations while the kernel can specify how smooth the function is and how the function values relate to one another. To demonstrate the significance of these functions, let us suppose that we are interested in modelling a function for which we have seen no values yet. In Figure 1, we plot the mean and a confidence interval that is two standard deviations away from the mean for three different combinations of mean functions and kernels.

After acquiring a few observations and plotting the posterior mean and variances in Figure 2, it becomes clear that the use of different priors results in different behaviours for the mean and confidence intervals between the observations. For example, in the left-most plot of Figure 2, an assumption was used to establish that the underlying function is periodic. As a consequence of this, we are left with a poor fit that serves as an example to show that proper care must be taken when choosing an appropriate prior.

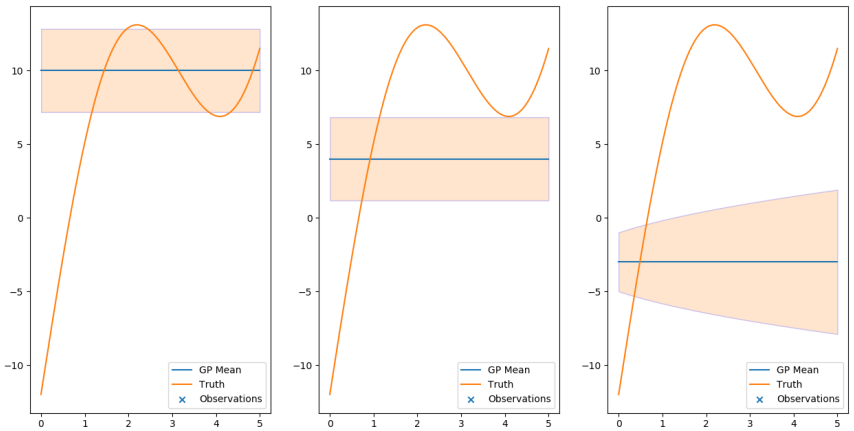


Figure 1: Examples of GP prior distributions

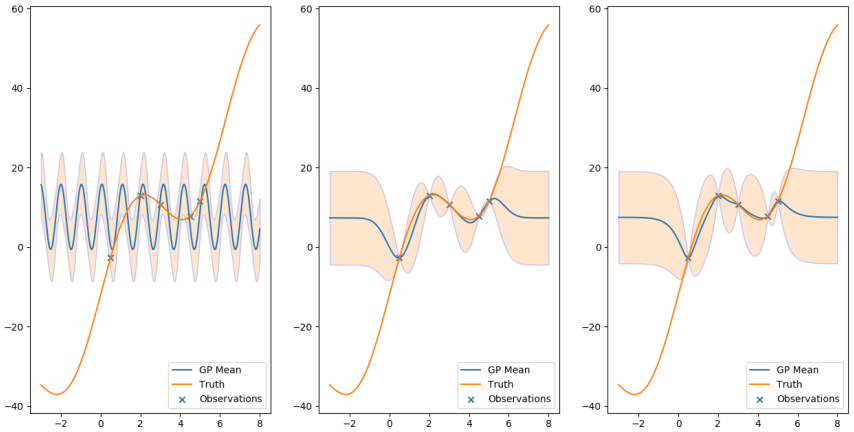


Figure 2: Examples of GP posterior distributions

Note that it is common to avoid specifying a mean function as we may let the data determine what the posterior mean is. Providing a mean function adds more hyperparameters to the GP which, after optimising the log-likelihood, can give information about how the function behaves as we move away from the observations. Going forward, we assume a constant mean function of 0 unless otherwise specified.

The kernel defines a measure of similarity between two values based on the respective indices of these values. In other words, for a set of test points for which we want predictions, the kernel is used to construct a covariance matrix for the respective function values at those points. For a kernel k to be valid it must follow two conditions:

1. The kernel k must be symmetric (i.e. $k(x, y) = k(y, x)$).
2. All gram matrices $\mathbf{K}$ constructed from k must be positive semidefinite.

In practice, $\mathbf{K}$ gets inverted, which means we must occasionally add noise to the diagonal of this matrix to make it positive definite. One case where this is required is when two indices from the training set are very close to each other. This can cause the rows of the covariance matrix to no longer be considered linearly independent which in turn results in a singular covariance matrix. For stability and performance reasons, we invert matrices using a Cholesky decomposition which requires the matrix to be positive definite.

In more technical terms, given a GP prior $\mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$, the following Equations are used to specify the mean function and kernel of the posterior distribution $\mathcal{GP}(m_N(\mathbf{x}), k_N(\mathbf{x}, \mathbf{x}'))$ given we have seen N observations (see appendix for derivations).

$$m_N(\mathbf{x}) = k(\mathbf{x}, \mathbf{X})k(\mathbf{X}, \mathbf{X})^{-1} f(\mathbf{X}) \tag{1}$$

$$k_N(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{X})k(\mathbf{X}, \mathbf{X})^{-1} k(\mathbf{X}, \mathbf{x}') \tag{2}$$

Given matrices $\mathbf{A}$ and $\mathbf{B}$ of respective sizes $N \times D$ and $M \times D$, we define $k(\mathbf{A}, \mathbf{B})$ to be a matrix $\mathbf{K}$ such that $K_{ij} = k(\mathbf{A}_{\text{row } i}, \mathbf{B}_{\text{row } j})$. We can interpret $\mathbf{A}$ and $\mathbf{B}$ as each holding a certain number of D-dimensional points as rows. Typically, $\mathbf{x}$ would be the point we are interested in predicting at and $\mathbf{X}$ would contain the indices for which we have observations. Therefore, using this logic and assuming we have seen N observations: $k(\mathbf{x}, \mathbf{X})$, $k(\mathbf{X}, \mathbf{X})$, $k(\mathbf{X}, \mathbf{x})$ would respectively be matrices of sizes $1 \times N$, $N \times N$, and $N \times 1$. This notation will be used throughout this document.

The training process of the GP (i.e. finding hyperparameter values) is typically done through an optimisation process that seeks to maximise the log-likelihood of the hyperparameters given the observations seen so far. Also, given that a GP has N observations and we want to compute the posterior distribution over a series of test points, there is a worst-case time complexity of $O(N^3)$ and a worst-case space complexity of $O(N^2)$. The former complexity is due to the matrix inversion in Equations 1 and 2 while the latter is due to the storing of the $\frac{n(n+1)}{2}$ entries in the same matrix.

Several specific kernels are used ubiquitously throughout this work presented in this document. Details of these kernels are shown in Table 1. We can get an idea of what kinds of functions each of these kernels are suitable for by looking at prior samples, as shown in Figure 3. It is also worth mentioning that the SE kernel is infinitely differentiable (see chapter 2 in [8]) as this property becomes relevant in the discussion about Bayesian quadrature.

Name	$k(x, x')$	Hyperparameters
Squared Exponential (SE)[1]	$\sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$	σ^2, ℓ
Rational Quadratic	$\sigma^2 \left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$	σ^2, ℓ, α
Linear	$\sigma^2(x-c)(x'-c)$	σ^2, c

Table 1: Common kernels with associated hyperparameters such as the variance σ^2 and lengthscale l

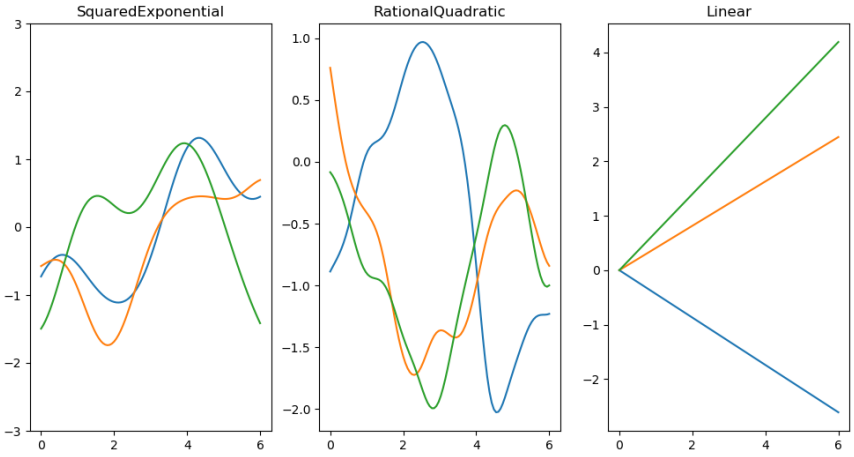


Figure 3: GP prior samples

2.3 Complete GPAR description

As previously mentioned, we want to use GPAR to model a system consisting of multiple outputs which are represented in the form of time-series. The underlying process that is generating the data for a particular output may possess a causal relationship with one or many of the other outputs. For instance, we might have the following system of equations:

$$\begin{aligned} y_1(x) &= f_1(x) \\ y_2(x) &= f_2\left(x, y_1(x)\right) \\ &\dots \\ y_n(x) &= f_n\left(x, y_1(x), \dots, y_{n-1}(x)\right) \end{aligned}$$

In this case, for a given index value, each function $y_k(x)$ may rely in some way on the values of the functions that precede it (i.e. y_{k-i} for $i = 1, \dots, k+1$). Of course, we could model each individual output with an independent GP (IGP). However, by doing so, we would be omitting the use of important information that could help bolster the predictive performance of our model as the IGPs would enforce an erroneous independence assumption

[1] Also known as Radial Basis Function (RBF) or Exponentiated Quadratic (EQ)

across each of the functions.

To explain how GPAR captures this information, we may use the following simple idea. Given a collection of random variables $\{Y_k|k = 1, \ldots, N\}$, the joint distribution of this collection could be factored into univariate distributions by making use of the product rule:

$$p(Y_1, Y_2, \ldots, Y_N) = p(Y_1) \prod_{i=2}^{N} p(Y_i|Y_1, \ldots, Y_{i-1})$$

GPAR applies this same idea but towards stochastic processes:

$$p(Y_1(x), Y_2(x), \ldots, Y_N(x)) = p(Y_1(x)) \prod_{i=2}^{N} p(Y_i(x)|Y_1(x), \ldots, Y_{i-1}(x))$$

Hence, every conditional term can now be modelled with a single GP. The GP that is responsible for modelling $Y_k(x)$ would use an indexing variable $X \in \mathbb{R}^k$ where one dimension would correspond to the native index of the system (e.g. time), while the remaining dimensions respectively relate to each of the $k-1$ outputs from the preceding functions. Since conditional outputs are now modelled separately by single GPs, a significant advantage of this approach is that we can make full use of the complete GP machinery. Note that it is not the case that $p(Y_1(x), Y_2(x), \ldots, Y_N(x))$ is jointly Gaussian under the GPAR model.

2.4 Training Methodology

The methodology to train GPAR can be seen in Algorithm 1. Note that in a world with unlimited computational power, we would seek to try all possible order combinations. However, this would not be practical as there are $n!$ ways to order n outputs. Thus, a greedy approach is used instead, which trains $n(n-1)/2$ GPs, resulting in a $O(n^2)$ time complexity. We discuss this in more detail within the experiment section of this report.

Algorithm 1: GPAR training process

```
N ← Output Count;
Remaining Y ← 1, ..., N;
Current X ← X;
Models ← ∅;
for i = 1, ..., N do
    Model, Output ID ← MaxLikelihoodWithBestY(Current X, Remaining Y);
    Current X ← Add(Current X, Remaining Y[Output ID]);
    Remaining Y ← Remove(Remaining Y, Output ID);
    Models ← Add(Models, Model);
end for
return Models;
```

2.5 Kernel Selection

Kernel selection could drastically affect how well GPAR performs. Of course, it is not always clear which kernels should be used to model inter-output relationships. Furthermore, it may

	Output Dep.	Kernel
1	Linear	$k_{\mathrm{EQ}}(x, x') + k_{\mathrm{Linear}}(y(x), y(x'))$
2	Linear + x Dep.	$k_{\mathrm{EQ}}(x, x') + k_{\mathrm{RQ}}(x, x')k_{\mathrm{Linear}}(y(x), y(x'))$
3	Non-linear	$k_{\mathrm{EQ}}(x, x') + k_{\mathrm{RQ}}(y(x), y(x'))$
4	Non-linear + x Dep.	$k_{\mathrm{EQ}}(x, x') + k_{\mathrm{RQ}}((x, y(x)), (x', y(x')))$

Table 2: Sensible choices for the GPAR kernel

be impractical to manually construct the kernel when the number of outputs is sufficiently large: different subsets of outputs might have different relationships amongst each other.

One possible solution is to use compositional kernel search [3]. This technique uses the fact that kernels are closed under addition and multiplication. This fact allows the use of a greedy approach which successively adds and multiplies kernels that were initially predetermined. With unlimited computational power, it would be feasible to use compositional kernel search while finding the appropriate ordering. However, with limited computational resources, it would be too expensive to compute a new GPAR ordering for every proposed kernel in the search procedure. Table 2 can be used as a reasonable starting point to avoid having to perform such an expensive search procedure [9]. One approach could be to start with a kernel from Table 2 to establish the ordering, then use compositional kernel search to find a better kernel. Then, with the new kernel, recompute a potentially better order and continue repeating this whole process iteratively.

2.6 Considerations

Recall that the goal of a model such as GPAR is to separate noise from the true data-generating functions. In cases where the signal-to-noise ratio is low, this task is complicated by the fact that noisy values of the preceding functions are fed forward into the models for subsequent functions. Having noise carried forward in this way might harm the predictive performance of the model. To mitigate this problem, one possible solution could be only to feed the predictive means forward and exclude the actual noisy observations.

Another important consideration is the number of hyperparameters that are being used throughout the entire GPAR model. Since each output is modelled with a single GP, the number of hyperparameters scales linearly with the number of outputs. A large number of outputs combined with an overabundance of hyperparameters may severely slow down the training time. It may be interesting to explore possible ways of sharing hyperparameters in order to prevent this.

Finally, a common practice to fit GP hyperparameters is by maximising the likelihood. However, this surface is very multi-modal, which makes the result of the optimisation process dependent on initialisation. Although it is possible to come up with very reasonable initialisations by inspecting the data, this is not always possible (e.g. in situations where there is too much data to inspect). For this reason, it is recommended to use multiple different initializations by sampling hyperpriors and then taking the post-optimisation configuration that yields the highest likelihood.

3 Bayesian Quadrature

3.1 Overview

Numerical quadrature, also known as numerical integration, is the process of finding the numerical value of an integral k that is usually difficult, if not impossible, to solve analytically [7]:

$$k = \int_{\mathcal{X}} f(x)\mathrm{d}x$$

Typically, for numerical quadrature, the approximation of the integral is the result of a rule which is defined as a function of integrand evaluations $f(x_1)$, $f(x_2)$, ... , $f(x_N)$. This process can be framed as an inference problem: we are interested in inferring the value of an unknown quantity k as a result of seeing a finite number of observations $f(x_n)$ where $n = 1, \ldots, N$.

To turn this into an inference problem, we first begin by specifying a prior distribution over the integrand. Using a GP for this prior is a good idea because of the property that GPs are closed under affine transformation and integration is a linear operator.

After combining our prior distribution over the integrand with some observations, we get a GP posterior that can then be pushed through the integral operator and which will result in a posterior Gaussian distribution over the integral value:

$$\mu_N = \left\{ \int_{\mathcal{X}} k(\cdot, \mathbf{X})\mathrm{d}\mathbf{x} \right\} k(\mathbf{X}, \mathbf{X})^{-1} f(\mathbf{X}) \tag{3}$$

$$\sigma_N^2 = \left\{ \int_{\mathcal{X}} \int_{\mathcal{X}} k(\cdot, \cdot)\mathrm{d}\mathbf{x}\mathrm{d}\mathbf{x}' \right\} - \left\{ \int_{\mathcal{X}} k(\cdot, \mathbf{X})\mathrm{d}\mathbf{x} \right\} k(\mathbf{X}, \mathbf{X})^{-1} \left\{ \int_{\mathcal{X}} k(\mathbf{X}, \cdot)\mathrm{d}\mathbf{x} \right\} \tag{4}$$

Proof. To demonstrate this result we can use the following simple proof [12]. First, we define our GP as a function $g : \mathcal{X} \times \Omega \to \mathbb{R}$ where $\mathcal{X}$ and Ω are respectively defined to be the sets of indices and samples. Therefore, $g(x, \omega)$ is the evaluation of a sample $\omega \in \Omega$ at a particular index $x \in \mathcal{X}$. Since GPs are closed under affine transformations and the integral is a linear operator, we can compute both the mean and the variance of the integral and this will give us the mean and the variance of the GP over the integral. Also, we remind the reader that provided we have a bounded integral $\int_{(X \times Y)} |f(x, y)|\mathrm{d}(x, y) < \infty$, Fubini's theorem allows

us to swap the order of integration such that $\int_X \left(\int_Y f(x,y)\mathrm{d}y\right) \mathrm{d}x = \int_Y \left(\int_X f(x,y)\mathrm{d}x\right) \mathrm{d}y$.

$$\begin{aligned}
\mathbb{E}_N\left[\int_{\mathcal{X}} g(x,\omega)\mathrm{d}x\right] &= \int_\Omega \int_{\mathcal{X}} g(x,\omega)p(\omega)\mathrm{d}x\mathrm{d}\omega \\
&= \int_{\mathcal{X}} \int_\Omega g(x,\omega)p(\omega)\mathrm{d}\omega\mathrm{d}x \\
&= \int_{\mathcal{X}} m_N(x)\mathrm{d}x \\
\mathbb{V}_N\left[\int_{\mathcal{X}} g(x,\omega)\mathrm{d}x\right] &= \mathbb{E}_N\left[\left(\int_{\mathcal{X}} g(x,\omega)\mathrm{d}x - \int_{\mathcal{X}} m_N(x)\mathrm{d}x\right)^2\right] \\
&= \mathbb{E}_N\left[\int_{\mathcal{X}}\int_{\mathcal{X}} \left(g(x,\omega) - m_N(x)\right)\left(g(x',\omega) - m_N(x')\right)\mathrm{d}x\mathrm{d}x'\right] \\
&= \int_\Omega \int_{\mathcal{X}}\int_{\mathcal{X}} \left(g(x,\omega) - m_N(x)\right)\left(g(x',\omega) - m_N(x')\right)p(\omega)\mathrm{d}x\mathrm{d}x'\mathrm{d}\omega \\
&= \int_{\mathcal{X}} \int_{\mathcal{X}}\int_\Omega \left(g(x,\omega) - m_N(x)\right)\left(g(x',\omega) - m_N(x')\right)p(\omega)\mathrm{d}\omega\mathrm{d}x\mathrm{d}x' \\
&= \int_{\mathcal{X}} \int_{\mathcal{X}} k_N(x,x')\mathrm{d}x\mathrm{d}x'
\end{aligned}$$

Finally, by considering $g(x,\omega)$ to be a posterior distribution over the integrand, we can combine this result with Equations 1 and 2 to respectively arrive at Equations 3 and 4.

□

3.2 Application of GPAR

Let us consider the case of modelling N integrals $y_1, y_2, \ldots, k_N$ where the respective integrands $f_1(x), f_2(x), \ldots, f_N(x)$ may have causal relationships among one another. GPAR could be used to model each of the integrands more accurately as opposed to simply using IGPs. For the first output in the GPAR stack, which we can assume to be f_1, we can use the standard BQ approach. However, modelling the second output f_2 requires a bit more thought because the kernel now expects each of its arguments to be of two dimensions. For each move up the GPAR output stack, the dimensionality of the kernel arguments increase by one, and we need to integrate that kernel when dealing the intermediate integrals (i.e. not the original integration problem, but the ones in the posterior distribution). How do we deal with the dimensionality increase?

A possible solution would be as follows. Let us assume we have an oracle g_1 that tells us the exact value f_1 would have for a given input x in our example for no extra cost. As we are only interested in integrating over our input domain and we know that the kernel makes use of the added information of the previous outputs, we could use our oracle to tell us the corresponding output for every specific argument as we're integrating:

$$\left(\int_{\mathcal{X}} k((\mathbf{x}, f_2(\mathbf{x})), \mathbf{X}^{\mathrm{obs}})\mathrm{d}\mathbf{x}\right)_i = \int_{\mathcal{X}} k((\mathbf{u}, g_2(\mathbf{u})), (\mathbf{x_i}, f_2(\mathbf{x_i})))\mathrm{d}\mathbf{u}$$

Unfortunately, we do not have access to such an oracle, and we want to avoid directly evaluating the integrand, assuming it is expensive to do so. Hence, we have to resort to using GPAR means to provide these corresponding values.

3.2.1 Naive Numerical Methods Approach

To compute the intermediate integrals as described above, we can resort to using standard numerical method approaches. One drawback to this way of proceeding is that the intermediate integrand requires at least one GP prediction (and possibly many more), which could significantly increase the computation time if a large number of integrand evaluations are done.

3.2.2 Naive Monte Carlo Approach

The approach mentioned above may have problems when the domain of integration increases and the numerical methods responsible for approximating the integrals of the key terms in the process require more integrand evaluations. Rather than using deterministic quadrature methods, we could instead use Monte Carlo sampling:

$$\underbrace{\int_a^b \int_a^b \dots \int_a^b}_{\text{M}} f(x_1, x_2, \dots, x_M) \mathrm{d}x_1 \mathrm{d}x_2 \dots \mathrm{d}x_M \simeq (b-a)^M \sum_{i=1}^{N} f(x_i)$$

A fixed number of N of samples can be taken with the hope that it is reasonable enough to provide a sensible final result at the cost of perhaps less precision. The challenge here would be to balance the trade-off between computational cost and results properly. Unfortunately, this trade-off would depend on the nature of the integrand.

3.3 Derivative Approach

An approach to doing Bayesian quadrature without having to deal with integration in the intermediate steps would be very attractive. Such an approach may be possible by differentiating the kernel that we use for the prior that is placed over the integrand.

The first form of the Fundamental Theorem of Calculus (FTC) (see chapter 5 in [6]) states that given we have a differentiable function F on the interval (a, b) and the function $f(x) = \frac{\mathrm{dF(x)}}{\mathrm{d}x}$ is Riemann integrable on [a, b] then we have the following identity:

$$\int_a^b f = F(a) - F(b)$$

Adapting the integral term from Equation 3 in order to integrate over the interval $[a, b]$ on the real number line, we have:

$$\int_a^b k(x, \mathbf{X}) \mathrm{d}x$$

Assuming that the kernel $k(x, x')$ is differentiable, let $k_\Delta(x, x') = \frac{\mathrm{d}k(x,x')}{\mathrm{d}x}$. Combining this with the integral above yields:

$$\int_a^b k_\Delta(x, \mathbf{X}) \mathrm{d}x = k(b, \mathbf{X}) - k(a, \mathbf{X})$$

Typically, k would be an EQ kernel because it has the property that it is infinitely differentiable. Also, since the EQ kernel is stationary and $k(-\infty, c) = 0$ for $c \in \mathbb{R}$ we may claim

the following:

$$\int_{-\infty}^{c} k_{\Delta}(x, \mathbf{X}) \mathrm{d}x = k(c, \mathbf{X})$$

The same idea could be brought forward to the double integral term in Equation 4 with $k_{\Delta\Delta'}(x, x') = \frac{\mathrm{d}^2 k(x,x')}{\mathrm{d}x\mathrm{d}x'}$. Using the same logic as shown above we can arrive at the following identity:

$$\int_{-\infty}^{c}\int_{-\infty}^{d} k_{\Delta\Delta'}(x, x') \mathrm{d}x\mathrm{d}x' = k(c, d)$$

Using this identity, we can come up with an equivalent expression for Equation 4 as shown below. Note that we shortened the notation for the sake of clarity.

$$\begin{aligned}\int_{a}^{b}\int_{a}^{b} k_{\Delta\Delta'}(x, x') \mathrm{d}x\mathrm{d}x' &= \left\{\int_{-\infty}^{b}\int_{-\infty}^{b} k_{\Delta\Delta'}\right\} - 2\left\{\int_{-\infty}^{a}\int_{-\infty}^{b} k_{\Delta\Delta'}\right\} + \left\{\int_{-\infty}^{a}\int_{-\infty}^{a} k_{\Delta\Delta'}\right\} \\ &= k(b, b) - 2k(a, b) + k(a, a)\end{aligned}$$

To summarise what was demonstrated so far: we are effectively modelling the observations of the integrand using a kernel $k_{\Delta\Delta'}(x, x')$ that was obtained by differentiating each argument of a known kernel such as the SE. The advantage of proceeding in this way is that the integral terms, which are usually computational bottlenecks, can be evaluated by using the closed-form of the known kernel. To get the differentiated kernel, we may rely on automatic differentiation or other numerical differentiation techniques that are less costly than numerical integration.

One final difficulty must be addressed: the results that are shown above are only valid is the quadrature is performed on an integrand that takes a one-dimensional index as an input. One of the difficulties of Bayesian quadrature is that it has trouble dealing with integrands which have highly multi-dimensional domains. Fortunately, the results shown above could easily be generalised into higher dimensions.

Given we have a function $g : \mathbb{R}^D \to \mathbb{R}$ where $D \in \mathbb{N}^+$, we want to demonstrate a way to evaluate $\int_{a_D}^{b_D}\int_{a_{D-1}}^{b_{D-1}} \ldots \int_{a_1}^{b_1} g(x_1, x_2, \ldots, x_D)\mathrm{d}x_1\mathrm{d}x_2 \ldots \mathrm{d}x_D$ as a set of finite evaluations of a function $f : \mathbb{R}^D \to \mathbb{R}$ where $f(x_1, x_2, \ldots, x_d) = \frac{\partial^D}{\partial x_1 \partial x_2 \ldots \partial x_D} g(x_1, x_2, \ldots, x_d)$.

For the simplest case $D = 1$, as already mentioned, we may use the first form of the FTC:

$$\int_{a_1}^{b_1} g(x_1)\mathrm{d}x_1 = f(b_1) - f(a_1)$$

For $D = 2$:

$$\begin{aligned}\int_{a_2}^{b_2}\int_{a_1}^{b_1} g(x_1, x_2)\mathrm{d}x_1\mathrm{d}x_2 &= \int_{a_2}^{b_2} \left(g(b_1, x_2) - g(a_1, x_2)\right) \mathrm{d}x_2 \\ &= \left(g(b_1, b_2) - g(a_1, b_2)\right) - \left(g(b_1, a_2) - g(a_1, a_2)\right)\end{aligned}$$

Note that in the results presented earlier we used the fact that kernels are symmetric as well as $a_1 = a_2$ and $b_1 = b_2$. These assumptions would allow $g(a_1, b_2) = g(b_1, a_2)$.

For $D = 3$:

$$\begin{aligned}\int_{a_3}^{b_3}\int_{a_2}^{b_2}\int_{a_1}^{b_1} g(x_1,x_2,x_3)\mathrm{d}x_1\mathrm{d}x_2\mathrm{d}x_3 &= \int_{a_3}^{b_3}\left(g(b_1,b_2,x_3)-g(a_1,b_2,x_3)\right)\mathrm{d}x_3 \\ &\quad - \int_{a_3}^{b_3}\left(g(b_1,a_2,x_3)-g(a_1,a_2,x_3)\right)\mathrm{d}x_3 \\ &= g(b_1,b_2,b_3) - g(a_1,b_2,b_3) \\ &\quad - g(b_1,a_2,b_3) + g(a_1,a_2,b_3) \\ &\quad - g(b_1,b_2,a_3) + g(a_1,b_2,a_3) \\ &\quad + g(b_1,a_2,a_3) - g(a_1,a_2,a_3)\end{aligned}$$

This recursive relationship continues for as many dimensions as we desire. One drawback is that for any value of D, the time complexity in the number of terms to evaluate is $O(2^D)$. When working with kernels in the current context, D would be equal to twice the dimensionality of the input space because we are treating each dimension for both arguments as a separate argument for the kernel (e.g. for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$, $k(\mathbf{x}, \mathbf{y}) = k(x_1, x_2, y_1, y_2)$, and $D = 4$). By exploiting the kernel symmetry, we can lower the number of evaluations, but we still cannot escape the exponential time complexity. This increase in dimensionality limits the number of multiple related integrals that can be jointly modelled. Further improvements can be obtained if the kernel is separable.

Finally, we must also verify that the result of differentiating our kernel D times is also a valid kernel since we will be using it to model the integrand. For any given integrand, we will require $k_\Delta = \frac{\partial^{D/2}}{\partial x_1 \partial x_2 \ldots \partial x_{D/2}} k$ and $k_{\Delta\Delta'} = \frac{\partial^{D/2}}{\partial x_{D/2+1} \partial x_{D/2+2} \ldots \partial x_D} k_\Delta = \frac{\partial^D}{\partial x_1 \partial x_2 \ldots \partial x_D} k$, yet only the latter must be considered a valid kernel because it is being used to model the observations whereas the former is merely used as an intermediate mathematical step within the Bayesian quadrature framework.

4 Freeze-Thaw Bayesian Optimisation

4.1 Overview

Many machine learning models require the setting of what is known as hyperparameters. These are a special type of parameters that are set before the training process begins. These hyperparameters help specify the characteristics of the machine learning model. They may characterise the structure of the model (e.g. the architecture of a neural network) which affects model capacity. They may also establish behaviour within the training process (e.g. stopping condition for an optimiser). Several notable examples of this can be seen below:

1. Learning rate in an optimiser such as gradient descent.
2. Kernel selection for models such as SVMs and GPs.
3. Number of layers and units per layer within in a neural network.
4. Type of activation functions.

These hyperparameters are usually set by combining practitioner experience with trial and error. Unfortunately, this approach can be both time-consuming and very inefficient if the practitioner does not have a reasonable amount of experience with the particular modelling approach at hand. Another approach is to search the hyperparameter space using random or grid searches. Even for a small number of hyperparameters, this could be very time consuming because, for each new hyperparameter configuration, one must train a model with that particular configuration to see whether the initial settings were proper. These search methods do not work really well because they are not efficient and still require a large number of tries to be successful.

To evaluate a particular hyperparameter configuration, the practitioner would have to run the model on a cross-validation data set and then look at a loss metric such as mean squared error or one minus the percent accuracy. During the training process, the loss metric would be plotted every time the model has iterated through all of the training data (i.e. at every epoch). The best hyperparameter configuration would be the one with the lowest cross-validation loss at its asymptote. The plot of the cross-validation loss over the number of epochs usually looks like an exponential decay function which means that even after only a few epochs, it can sometimes be easy for a human being to infer what the asymptotic loss will eventually look like, by inspecting this plot.

Freeze-thaw Bayesian optimisation is a principled approach that can be used to find a set of hyperparameters in a relatively efficient manner [11]. It leans on the fact that a partial loss curve may sometimes be enough to infer how good a hyperparameter configuration is. Meanwhile, Bayesian optimisation is a technique that is used to minimise a function that may be very difficult or expensive to evaluate, where a model, such as a GP, is used as a surrogate for this function. Observations are made by evaluating the function at specific points in the domain. A different function, known as the acquisition function, uses the posterior distribution of the surrogate model and returns a measure of how beneficial it would be to evaluate the original function at a specific point in the domain. At every iteration of the Bayesian optimisation procedure, the acquisition function is maximised to get the next point that the function of interest should be evaluated at. The goal is to find the minimum

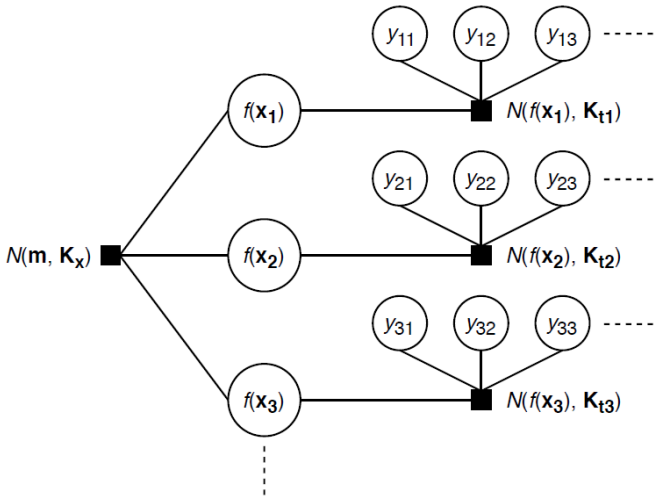


Figure 4: Freeze-thaw graphical model

of a function with the least amount of function evaluations. Proper care must be taken for selecting the appropriate acquisition function for the task at hand.

A naive way to use Bayesian optimisation for finding the best set of hyperparameters is to use a single GP to model the cross-validation loss for a specific time value. For example, assume we are interested in finding an optimal set of 3 hyperparameters α, β, and γ then the domain of this GP would be of four dimensions (i.e. $x_1 = [\alpha_1, \beta_1, \gamma_1, t_1]^T$ where t_1 represents the number of epochs). Assuming we have N unique hyperparameter configurations and a maximum of T epochs in which we evaluate these configurations, then this leaves us NT observations. This results in the particular worst-case time and space complexities of $O(N^3T^3)$ and $O(N^2T^2)$ which are deemed expensive.

Through a reasonable conditional independence assumption, freeze-thaw lowers the complexity while still providing excellent results. We can describe this assumption by explaining the generative procedure behind freeze-thaw. First, we begin with a global GP, which uses a Matérn-5/2 kernel, over the domain of all possible hyperparameter configurations. Given we have a basket of hyperparameter configurations, we can sample this global GP to get what the model believes is an appropriate asymptotic loss for each configuration. The loss function for each of the configurations is in turn modelled by a separate GP. We may sample these GPs to get a loss function for each of the configurations in our basket. A graphical representation of this model can be found in Figure 4. The nodes containing $f(\mathbf{x_k})$, represent the asymptotic loss for loss curve k, and the nodes containing y_{kv} represents the loss of curve k at epoch v.

In order to perform freeze-thaw optimisation, we first train an initial set of models with different hyperparameter configurations. The freeze thaw model hyperparameters are then set by maximising the log marginal likelihood, denoted as $P(\{\mathbf{y}_n\}_{n=1}^N | \{\mathbf{x}_n\}_{n=1}^N)$, over all of the observations of the loss curves that were seen so far. We define $\mathbf{y}_n$ to be a vector containing the points for the nth loss curve and $\mathbf{x_n}$ is its corresponding hyperparameter

configuration. A closed form of this likelihood could be found by marginalising out the latent variable $\mathbf{f}$:

$$P\left(\{\mathbf{y}_n\}_{n=1}^N \mid \{\mathbf{x}_n\}_{n=1}^N\right) = \int \left[\prod_{n=1}^N \mathcal{N}\left(\mathbf{y}_n; f_n \mathbf{1}_n, \mathbf{K}_{tn}\right)\right] \mathcal{N}\left(\mathbf{f}; \mathbf{m}, \mathbf{K}_{\mathbf{x}}\right) d\mathbf{f} \tag{5}$$

We use $\mathbf{1}_n$ to denote a vector of ones with the same length as the number of observations for the nth loss curve.

Second, Bayesian optimisation is used to construct a basket containing both old and new models, each with a unique hyperparameter configuration. Third, we again use Bayesian optimisation to select one of the models from the constructed basket to train. The selected model can be completely new, or it can be a model that had already been trained (i.e. has observations for its loss). This is the reason behind why this approach is called freeze-thaw: the training of one model can be stopped (i.e. "frozen") and we may continue the training of an existing model (i.e. "thawed" model) or start training a completely new model.

4.2 Model Selection Methodology

Two types of selections are required in freeze-thaw. The first type is the selection of B_{old} old models (i.e. models for which loss observations were already made) and B_{new} new models for hyperparameter configurations that were not considered yet. The second type of selection deals with choosing the model to train within the basket of $B_{\text{old}} + B_{\text{new}}$ models.

4.3 Custom Exponential Kernel

We are interested in using GPs to model the cross-validation loss for a specific model. Since this loss will generally follow an exponential decay, the authors behind freeze-thaw have suggested to use a kernel which is an infinite mixture of exponentially decaying basis functions:

$$k(x, x') = \int_0^\infty e^{-\lambda x} e^{-\lambda x'} p(\lambda) \mathrm{d}\lambda$$

If we put a Gamma(α, β) prior on λ we may arrive at the following closed form expression (see appendix for the derivation):

$$k(x, x') = \frac{\beta^\alpha}{(x + x' + \beta)^\alpha}$$

This shows the desirable property where $\lim_{x \to \infty} k(x, x) = 0$, which allows us to model the asymptotic loss by using the constant mean function. To give an idea of the type of functions that are suitable candidates for being modelled with this type of kernel, we present GP samples associated with this kernel in Figure 5.

4.4 Acquisition Functions

Freeze-thaw makes use of two types of acquisition functions: expected improvement (EI) and entropy search (ES). Expected improvement is used to construct the basket of models and entropy search is used to select which model is trained from within that basket. The following is an explanation for how each of these acquisition functions would be used to

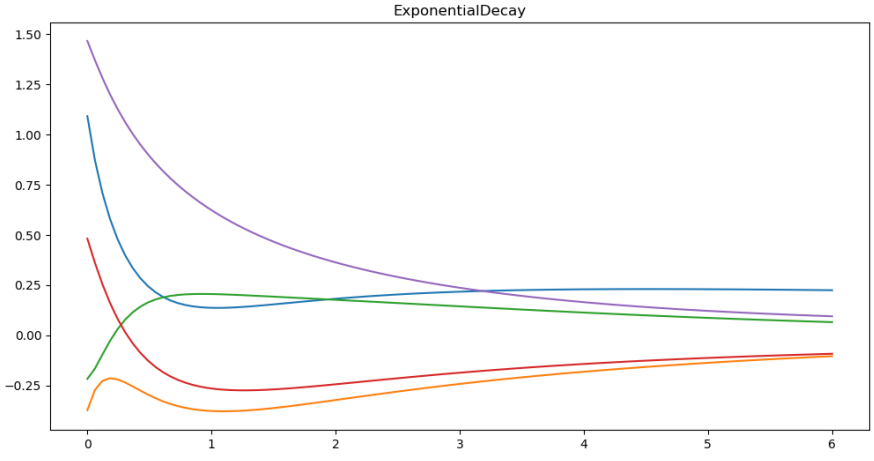


Figure 5: Samples from a GP that uses the custom exponential kernel

find the minimum of a function f. We also define $f_{\min} = \min_{x \in \mathcal{X}_{\text{obs}}} f(x)$ where $\mathcal{X}_{\text{obs}}$ is the set of configurations that were seen so far. We will also use a GP as our model for f: $\mathcal{GP}\left(m(x), k(x, x')\right)$.

For EI we define the following utility function:

$$u(x) = \max\left\{0, f_{\min} - f(x)\right\}$$

For a particular x, $f(x)$ is a random variable. The EI acquisition function is defined as the expected value of this utility function:

$$\begin{aligned}\alpha_{EI}(x) &= \mathbb{E}[u(x)] \\ &= \left(f_{\min}(x) - m(x)\right)\Phi(f_{\min}|m(x), k(x,x)) + k(x,x)\phi(f_{\min}, m(x), k(x,x))\end{aligned}$$

We respectively denote both $\phi(x|\mu, \sigma^2)$ and $\Phi(x|\mu, \sigma^2)$ as the PDF and CDF for a univariate Gaussian distribution with mean μ and variance σ^2. Therefore, we end up with a closed-form solution that is easy to evaluate. The derivation of this result can be found in the appendix.

For ES we are interested in finding the next observation that lowers the entropy of the probability of the minimum value. The entropy of a probability distribution P over the continuous random variable X is defined as:

$$H[P] = -\int_{\infty}^{\infty} P(x)\log P(x)\mathrm{d}x$$

If X is instead defined as a discrete random variable that can take on a finite set of values $\omega_1, \omega_2, \ldots, \omega_n$, we would have the following definition:

$$H[P] = -\sum_{i=1}^{n} P(\omega_i)\log P(\omega_i)$$

Entropy can be thought of as a measure of how uncertain the outcome of a stochastic event is. For example, an event that is entirely deterministic would have zero entropy while an event tied to a uniform distribution would have high entropy. Another way of interpreting entropy is the average amount of information that is required to send the result of an output that follows a given probability distribution. If an event is uniformly random, then each outcome would be encoded using a message of the same length. However, if the distribution is non-uniform, then an encoding scheme could encode more frequent outcomes with small messages at the expense of less frequent outcomes which would have longer messages. By proceeding in this way, the average message length would be smaller (i.e. meaning smaller entropy).

Given that $x* = \text{argmax}_{x \in X} f(x)$, the utility function for ES can be defined as:

$$u(x) = H\left[P(x^*|\mathcal{D})\right] - H\left[P(x^*|\mathcal{D}, x, f(x))\right]$$

A point x that results in a high value of $u(x)$ would imply that knowing the value of the function evaluated at that point, $f(x)$, would lower the entropy of the probability distribution over the point $x*$. Due to the usually complex nature of this probability distribution, it is common to use Monte Carlo (MC) sampling to get an approximate representation.

When picking a model from a basket, EI should be avoided since it favours selecting a new configuration because an untrained model typically has a higher expectation to improve than the ones that have already been trained. Therefore, ES is used instead: we want to lower the entropy of the probability over the configuration with the lowest asymptotic loss.

For each of the configurations in a basket of configurations, we would use MC to compute the probability distribution $P_{\min}(\mathbf{x}^*)$ (i.e. approximation for $P_{\min}(\mathbf{x}^*|D)$) that a specific configuration $\mathbf{x}^*$ has the minimum asymptotic loss. The closed form of Equation 6 is used as a stepping stone for this approximation. Refer to Algorithm 2 for more details.

$$P\left(f^*|\mathbf{y}, \{\mathbf{x}_n\}_{n=1}^N, \mathbf{x}^*\right) = \int P\left(f^*|\mathbf{f}, \mathbf{x}^*\right) P\left(\mathbf{f}|\mathbf{y}, \{\mathbf{x}_n\}_{n=1}^N\right) d\mathbf{f} \tag{6}$$

To compute the acquisition function, we will also need the probability distribution over the minimum given a specific observation y^*. We will denote this distribution as $P_{\min}^{y^*}(\mathbf{x}^*)$ (i.e. approximation for $P_{\min}(\mathbf{x}^*|\mathbf{D}, \mathbf{f}(\mathbf{x}) = \mathbf{y}^*)$). Again using marginalisation, Equations 7 and 8 will be used to sample observations that respectively depend on if the configuration at hand has already seen training or if it is completely new. Refer to Algorithm 3 for more details.

$$P\left(y_n^*|\left\{\mathbf{x}_n\right\}_{n=1}^N, \mathbf{y}\right) = \int P\left(y_n^*|\mathbf{y}_n, \mathbf{f}_n\right) P\left(\mathbf{f}_n|\mathbf{y}, \{\mathbf{x}_n\}_{n=1}^N\right) d\mathbf{f}_n \tag{7}$$

$$P\left(y^*|\left\{\mathbf{x}_n\right\}_{n=1}^N, \mathbf{y}, \mathbf{x}^*\right) = \int P\left(y^*|f^*\right) P\left(f^*|\mathbf{y}, \{\mathbf{x}_n\}_{n=1}^N, \mathbf{x}^*\right) df^* \tag{8}$$

Finally, putting the pieces together, we can compute an approximation of the ES acquisition function as seen in Algorithm 4.

Algorithm 2: Computing $P_{\min}$

Given a Basket $(x_1, x_2, \ldots, x_N)$;
$P \leftarrow (0, 0, \ldots, 0)$;
for $i = 1, 2, \ldots, n_{samples}$ **do**
  Samples $\leftarrow (0, 0, \ldots, 0)$;
  for $k = 1, 2, \ldots, N$ **do**
    $x_k^* \leftarrow$ Basket(k);
    $f_k^* \leftarrow$ Sample from (6) using x_k^*;
    Samples$(k) \leftarrow f_k^*$;
  end for
  $i_{\min} \leftarrow \text{argmin}_i$ Samples(i);
  $P(i_{\min}) \leftarrow \frac{1}{n_{\text{samples}}}$;
end for
return P;

Algorithm 3: Computing $P_{\min}^{y}$

Given a Basket $(x_1, x_2, \ldots, x_N)$ and a specific configuration x_k;
if x_k *is old* **then**
  $y_k^* \leftarrow$ Sample from (7) using x_k;
else
  $y_k^* \leftarrow$ Sample from (8) using x_k;
end if
Backup $\leftarrow$ current freeze-thaw parameters;
Update freeze-thaw parameters with (5) by incorporating y_k^* as an observation;
$P_{\min}^{y_k^*} \leftarrow$ Result of running Algorithm (2) with Basket and new parameters;
current freeze-thaw parameters $\leftarrow$ Backup;
return $P_{min}^{y_k^*}$;

Algorithm 4: Performing ES to find the next model to train

Given a Basket $(x_1, x_2, \ldots, x_N)$;
$P_{\min} \leftarrow$ Result from Algorithm 2;
$\alpha \leftarrow (0, 0, \ldots, 0)$;
for $k = 1, 2, \ldots, N$ **do**
  $x_k^* \leftarrow$ Basket(k);
  for $i = 1, 2, \ldots, n_{samples}$ **do**
    $P_{\min}^{y_{ki}^*} \leftarrow$ Result from Algorithm 3 with x_k^*;
    $\alpha(i) \leftarrow \frac{H[P_{\min}] - H[P_{\min}^{y_{ki}^*}]}{n_{\text{samples}}}$
  end for
end for
$k_{\max} \leftarrow \text{argmax}_k \alpha(k)$;
return k_{max};

4.5 Application of GPAR

The seemingly obvious way to use GPAR is to improve the modelling of the loss functions. It might be sensible to claim that losses from a given model with different hyperparameter configurations could possess causal relationships between one another. Therefore, for a given set of losses, GPAR could be used to find more accurate asymptotic losses. However, because each loss function would be conditioned on the loss functions that come before it on the GPAR stack, this approach would violate the conditional independence assumption that is made between all the GPs. Therefore, it is not possible to directly incorporate GPAR into freeze-thaw.

Despite this setback, we can inspire ourselves from the mechanics of freeze-thaw to create a new optimisation procedure which still uses GPAR to model the loss functions. First, EI could still be used to create the basket of hyperparameter configurations, and ES would be used to determine which model to train. Once the model is trained for a predetermined amount of time (e.g. one epoch), GPAR can run over all of the loss functions to update the constant mean functions that are used to model each loss. The values that the aforementioned mean functions take on could be used as observations for a global GP that is then trained separately. The procedure repeats itself in this way until a stopping condition is met (e.g. fixed number of training epochs).

5 Experiments

5.1 Overview

All of the relevant experiments that were done in the context of this project are presented in this section. For the sake of cohesion, after an experiment is presented, a short discussion of the results and implications immediately follows.

All of the software that was created to run the experiments were written from scratch with the help of the following libraries:

- TensorFlow: Graph-based mathematical library
- GPy: GP library
- GPflow: GP library with TensorFlow backbone
- Matplotlib: Plotting library
- NumPy: Numerical computing library (matrix computations)
- Pandas: Data manipulation and analysis library
- SciPy: Scientific computing library (numerical methods)

The experiments were all run locally on a laptop with the following specifications:

- Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
- NVIDIA GeForce GTX 1050 Ti
- 16.0 GB Memory

Also, for many of the experiments, the error metric of choice is the standardised mean squared error (SMSE), which is simply the mean squared error normalised by the variance of the truth:

$$\text{SMSE} = \frac{1}{N \times \text{Var}(\{y_{\text{truth}}^{(i)}\}_{i=1}^{N})} \sum_{i=1}^{N} (y_{\text{pred}}^{(i)} - y_{\text{truth}}^{(i)})^2 \tag{9}$$

One consideration that is worth mentioning is the risk of introducing bias in favour of GPAR when comparing against IGPs. In the implementation of GPAR, each possible ordering of the functions undergoes a certain number of random restarts before optimising the hyper-parameters. For example, assuming that each GP performs 10 random restarts and that we are interested in modelling 3 different functions, then each function could see more than 10 initialisations when training as part of GPAR. Consider the GPAR example shown in Table 3: y_1, y_2 and y_3 respectively see 10, 20, and 30 initialisations in the case where every GP uses 10 random initialisations. However, in the IGP case, each function is strictly restricted to only seeing 10 initialisations per function. Furthermore, we want to limit the amount of stochasticity that the initialisation can bring. Under a very stochastic regime, a certain set of results and conclusions could be acquired for one run, and a completely different set could be found for a different run. Hence, to limit this behaviour, we use a relatively high number of initialisations (e.g. 50+). Past a certain threshold, the number of initialisations was found to provide a negligible impact on performance.

Round	Candidates	Highest Likelihood
1	$P(y_1)$, $P(y_2)$, $P(y_3)$	$P(y_1)$
2	$P(y_2\|y_1)$, $P(y_3\|y_1)$	$P(y_2\|y_1)$
3	$P(y_3\|y_1, y_2)$	$P(y_3\|y_1, y_2)$

Table 3: Example of GPAR iterations for 3 hypothetical outputs

5.2 GPAR Model

5.2.1 Synthetic Functions

Since GPAR will be used extensively, it is important to run some sanity checks to establish correctness. We first begin by specifying 3 functions as shown below. For future reference, these functions will be referred to as the *base* scheme.

$$y_1(x) = \frac{-\sin(10\pi(x+1))}{2x+1} - x^4 \tag{10}$$

$$y_2(x) = \cos(y_1(x))^2 + \sin(3x) \tag{11}$$

$$y_3(x) = y_1(x)^2 y_2(x) + 3x \tag{12}$$

For 35 uniformly-spaced index values, the observations for each of these outputs are generated by evaluating the corresponding equation and then adding some Gaussian noise $\mathcal{N}(0, 0.0025)$. The selected kernel is the one from Table 2 corresponding to non-linear output dependencies with a dependency on the index variable (i.e. Kernel 4). Note that we will keep using this kernel unless otherwise specified. The visualisation of the predictions, as well as the SMSE for GPAR and IGP, can be seen in Figure 6.

The plots in Figure 6 are ordered in the way that they are modelled by GPAR: y_1, y_2, y_3. The fact that the first function that is modelled, y_1, does not benefit by being modelled by GPAR is expected. The models for the remaining functions do, however, benefit from the additional information used by GPAR.

Noise structure recovery could be used as an additional sanity check to validate GPAR. The noise parameters $\epsilon_1 \sim \mathcal{N}(0, 0.01)$, $\epsilon_2 \sim \mathcal{N}(0, 0.0016)$, and the functions shown below will be used.

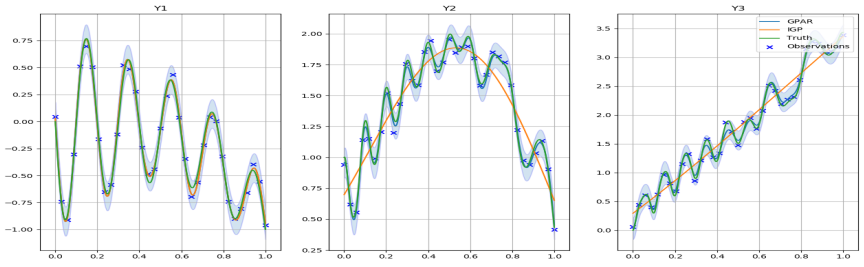
$$y_1(x) = \frac{-\sin(10\pi(x+1))}{2x+1} - x^4 + \epsilon_1 \tag{13}$$

$$y_2(x) = \sin(2\pi x)^2 \epsilon_1 + \cos(2\pi x)^2 \epsilon_2 \tag{14}$$

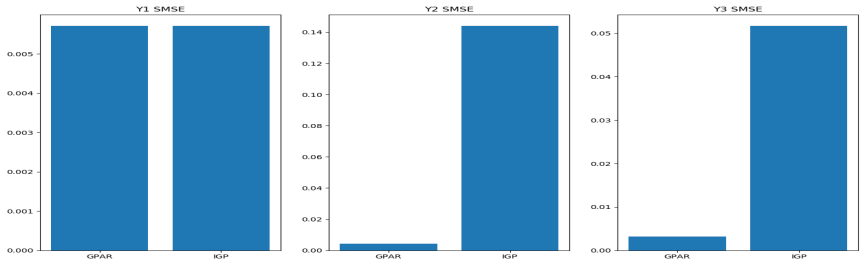
$$y_3(x) = \sin(2\pi\epsilon_1) + \epsilon_2 \tag{15}$$

$$y_4(x) = \sin(2\pi x)\epsilon_1 + \epsilon_2 \tag{16}$$

We wish to plot samples of (13) vs those of (14) at a specific x value. To create the truth, we directly sample both functions 100000 times and use a scatter plot with a 2D histogram, which helps better visualise the density of the points. To create an equivalent plot using IGPs, we sample from each corresponding GP, rather than directly from the functions. Finally, for GPAR, we sample the first GP, which could be either for (13) or for (14) depending on the order found by the training procedure. Then, the noisy sample from the first GP is fed into the second GP to produce the corresponding sample for the remaining output.



(a) Modelling of Synthetic Functions



(b) SMSE for Synthetic Functions Model

Figure 6: GPAR vs IGP performance comparison

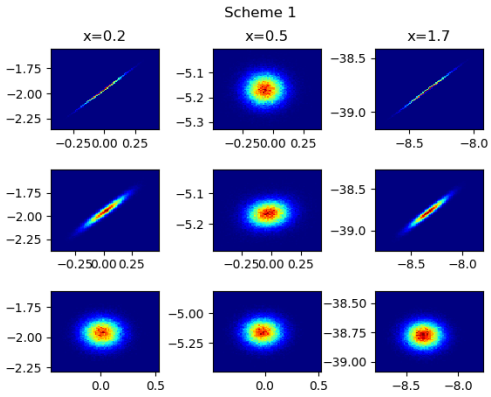
This entire procedure is also done for (13) vs (15) and (13) vs (16). The results can be seen in Figure 7. The samples from GPAR resemble the truth more than the ones from IGPs.

For the IGP, each plot is simply the product of two independent Gaussian distributions. On the other hand, for GPAR, by modelling the conditional distribution, we can capture the relationship between the two functions which are linked through shared noise. Visually, this means that the two-dimensional Gaussian distribution can stretch diagonally rather than only horizontally or vertically.

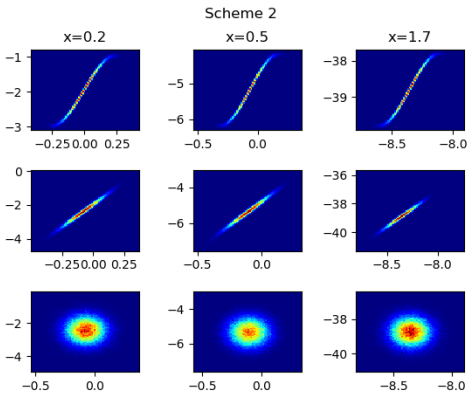
5.2.2 Error Sensitivity to Random Initialisation

Improperly selecting initial hyperparameters could negatively impact GPAR performance. For simple cases, it is possible to specify a reasonable starting point manually. For more sophisticated cases, it might be a better idea to perform random initialisations. We seek to demonstrate that systems which are more involved (i.e. more outputs with a broader variety of inter-output relationships) require more random initialisations as opposed to simpler systems.

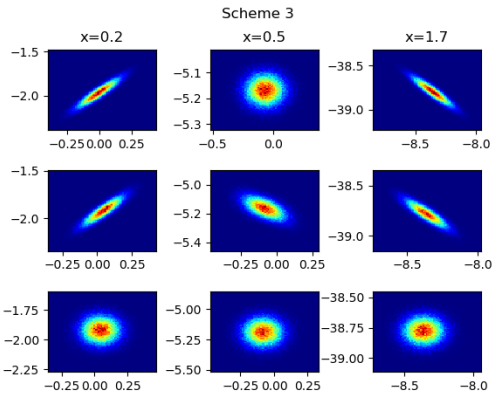
In order to do this, we will use GPAR to model 3 different schemes of low, medium, and high complexities, which are described in more detail below. For each of these schemes, 20 GPAR runs are performed where each run corresponds to training GPAR once with a set number of initialisations and then summing up the total SMSE across all functions. The sums are then averaged over the total number of runs (i.e. 20) the result of which is a point



(a) Noise Structure (13) vs (14)



(b) Noise Structure (13) vs (15)



(c) Noise Structure (13) vs (16)

Figure 7: Truth (row 1) vs GPAR (row 2) vs IGP (row 3) samples for modelling functions that share noise

on our plot for the specified number of restarts.

To come up with a set of random initialisations, each of the hyperparameters is sampled from a standard Gaussian distribution, and afterwards, if required, a constraint is applied (e.g. positive value constraint). For each of these schemes, we use an EQ kernel, 50 uniformly-spaced observations, and compute the total SMSE over 1000 test points. The EQ kernel is used because it results in a faster runtime, which allows us to use a higher number of restarts and sample sizes.

Low complexity scheme:

$$y_1(x) = 5\sin(x) + 5 \tag{17}$$
$$y_2(x) = 7y_1(x)^{0.5x} + 12 \tag{18}$$

Medium complexity scheme:

$$y_1(x) = 5x^2 + 100\sin(x) + 5 \tag{19}$$
$$y_2(x) = 0.005y_1(x) + \frac{x}{y_1(x)} + 200 \tag{20}$$
$$y_3(x) = x^2 + y_1(x) + 100\cos(y_2(x)) \tag{21}$$

High complexity scheme:

$$y_1(x) = 5(\cos(x)^2) + x\sin(\cos(x))^2 + 5 \tag{22}$$
$$y_2(x) = 5\sin(0.0001y_1(x)^3) + xy_1(x)\cos(\sin(x))^2 + 100 \tag{23}$$
$$y_3(x) = 0.0001(y_2(\cos(x)y_1(x))^2) + x\sin(\sin(x))^3 - 200 \tag{24}$$
$$y_4(x) = \frac{y_1(x)^2)}{y_2(x)} + \frac{y_2(x)}{y_3(x)} + \frac{y_3(x)^2}{y_1(x)} \tag{25}$$
$$y_5(x) = \sqrt{(}y_1(x)y_4(x)) + y_2(x) + y_3(x) + 10\cos(x) \tag{26}$$

In Figure 8, we notice the trend that more random initialisations are needed for more complicated schemes. To further take advantage of random initialisation when the complexity scales, one can achieve even better performance by using a prior distribution over the hyperparameters (i.e. a hyperprior). The Gamma distribution is a popular choice because its support is the set of all positive real numbers $\mathbb{R}^+$ which corresponds to valid values for many hyperparameters.

5.2.3 Kernel Selection for Characterising Relationships

Choosing the right kernel is of critical importance when using GPs. What if we mischaracterise the relationships between various outputs (i.e. swapping a linear dependence assumption vs a non-linear one)? We wish to investigate if there is a basis for exploring kernel search methods to improve performance.

In order to do this, a new simple variant of compositional kernel search is used to find a good kernel efficiently. First, we define a base set containing the following four kernels families: EQ, RatQuad, Linear, and Matern 5/2. In this starting set, for each kernel family, there is

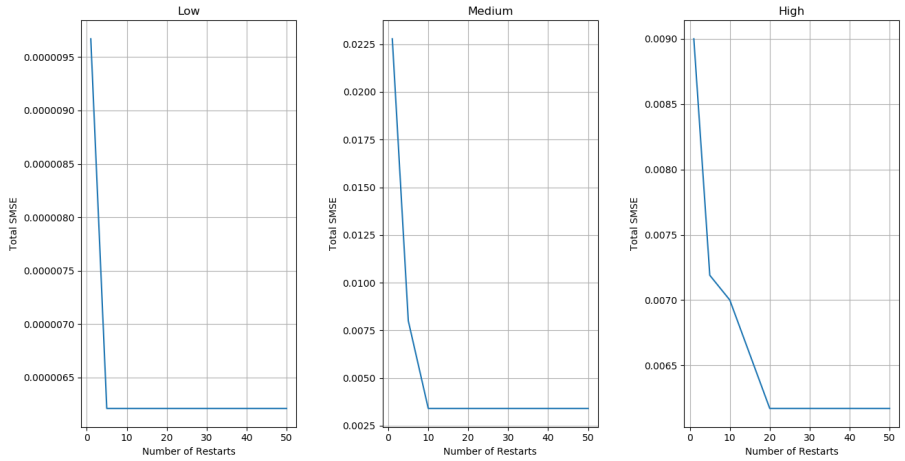


Figure 8: Error plotted over the number of restarts for different levels of complexity

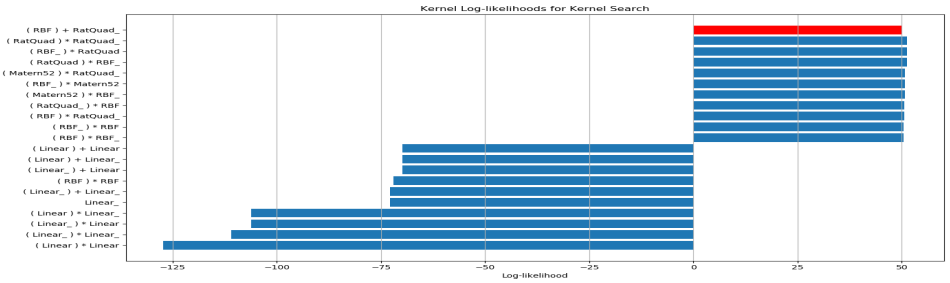


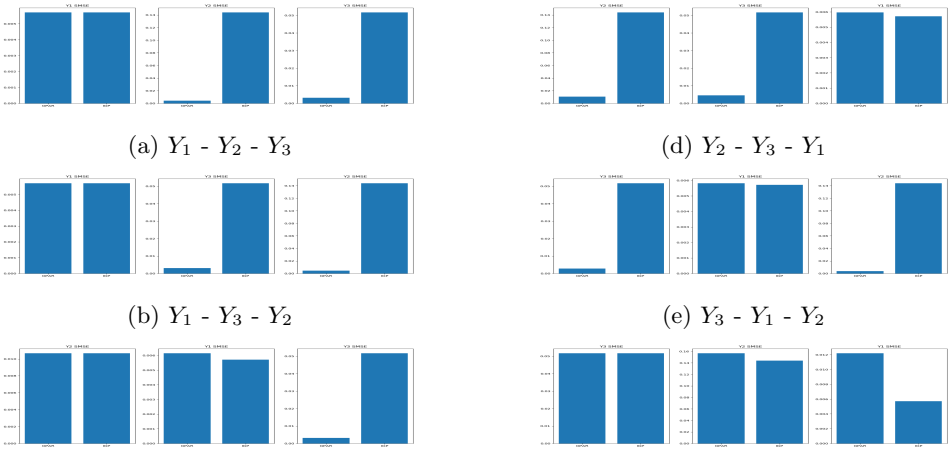
Figure 9: Top 10 and bottom 10 log-likelihood for different kernel configurations

the option of only being applied to the index (i.e. k(x, x')) or of being applied to both the index and the preceding outputs (i.e. k([x, y], [x, y'])). Thus, the starting set will contain eight kernels: four only applied to the index, and four applied to all dimensions of the input.

In the first round of the search, we start with the eight kernels. The next search round combines the kernels of the previous round through addition and multiplication with each of the eight base kernels. This continues until the search depth is reached.

For a search depth of 2, we end up trying $8 + 8 \times (2 \times 8) = 136$ kernels. The results for the top ten and bottom ten kernels according to likelihood are shown in Figure 9. Kernels that are applied to both the index *and* preceding outputs are denoted with an underscore. Also, in this figure, Kernel 4 from Table 2 is shown in red as a reference.

Looking at the results from Figure 9, one may notice that the top 10 kernels are very similar to each other when measured by their log-likelihood. This suggests that it may not be nec-



(a) Y_1 - Y_2 - Y_3

(d) Y_2 - Y_3 - Y_1

(b) Y_1 - Y_3 - Y_2

(e) Y_3 - Y_1 - Y_2

(c) Y_2 - Y_1 - Y_3

(f) Y_3 - Y_2 - Y_1

Figure 10: Base scheme ordering comparison (Noise: $\mathcal{N}(0, 0.0025)$)

essary to use significant resources to find an absolute best kernel. There is also a tendency to favour EQ, RatQuad and Matern 5/2 kernels over the Linear kernel. However, this is expected as the search ran on observations which come from an underlying source that is not very linear in nature.

5.2.4 Importance of Optimal Ordering

Another property that is worth investigating is how much the predictive performance of GPAR relies on having the correct output ordering. Is it worth spending significant resources on getting the absolute best order? Furthermore, if there is a very strong dependency, then it would be interesting to see under which conditions it becomes critical to be more mindful of ordering.

Using the base scheme, we present results for all of the 6 possible orders in Figure 10. These results indicate that out of the 6 orders, more than half are of similar performance (i.e. orders *a*, *b*, *d*, and *e*) are quite similar in performance. It may not be necessary to spend the computational resources to train and compute likelihoods for every input to establish order.

5.3 Bayesian Quadrature

5.3.1 Evaluation of Proposed GPAR Solution

Through a simple example, we start by examining the feasibility of the proposed naive GPAR solution. We choose 3 Gaussian distributions to use as integrands: $\mathcal{N}(1,2)$, $\mathcal{N}(-3,1)$, and $\mathcal{N}(0,4)$. An EQ kernel over all input dimensions is used in every GP. We also perform 1000 random restarts for training each GP. Furthermore, a small amount of noise is added to the covariance matrix to ensure that it is positive definite. Lastly, we uniformly select observations within the interval of integration.

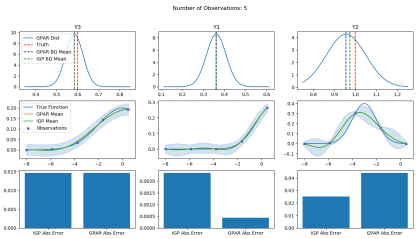
$$y_1 = \int_{-8}^{0.5} \mathcal{N}(x|1,2)\mathrm{d}x$$
$$y_2 = \int_{-8}^{0.5} \mathcal{N}(x|-3,1)\mathrm{d}x$$
$$y_3 = \int_{-8}^{0.5} \mathcal{N}(x|0,4)\mathrm{d}x$$

In Figure 11, we show the results of this experiment. Each column corresponds to a different integrand. The two plots on the left demonstrate how well IGP and GPAR can model the integrals. The first row of these plots shows the Gaussian distribution over the integral from the perspective of the naive GPAR approach. Vertical lines display the means of both approaches as well as the ground truth, which is acquired through standard numerical methods. The second row shows how well the integrands are being modelled. The confidence intervals that are shown come from GPAR. Finally, the third row compares the absolute value of the errors for each approach. Lastly, the two plots on the right side compare the SMSE of GPAR and IGPs over the integrand.

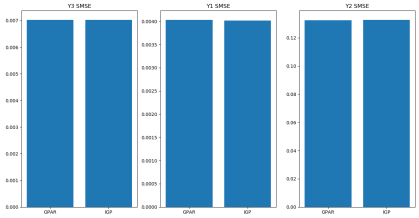
Looking at plots (a) and (c), it appears that GPAR is doing slightly better at modelling y_1. This can be explained by the fact that information from y_3 (i.e. the first function that is being modelled) has some usefulness for modelling y_1 possibly because y_1 and y_3 are very similar. However, it performs noticeably less well on y_2. It could be speculated that, when modelling y_3, the preceding model outputs within GPAR are ruining the integration process used to compute the posterior. It is natural to ask if this behaviour persists by increasing the number of observations.

Unfortunately, plots (b) and (d) demonstrate that doing so reduces the performance of GPAR relative to its counterpart even more. Numerical results can be found in Tables 4 and 5 (note that μ_i and σ_i respectively denote the mean and standard deviation obtained through the Bayesian quadrature procedure for y_i). Looking at these results, we can make two observations. First, for the baseline, all of the means get closer to the truth while the standard deviation gets smaller. This is the expected behaviour. On the other hand, for the naive GPAR approach, as the number of observations gets higher, the means for y_1 and y_2 seem to get worse and the confidence increases. This is not a good sign. Finally, as a sanity check, we confirm that the statistics that correspond to the first function modelled by GPAR, y_3, are the same as for IGPs.

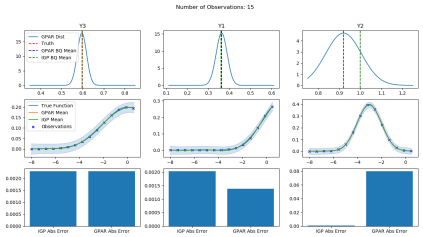
Unfortunately, the blame cannot be passed on to the modelling process of the integrand. We point the readers attention to plot (d) of Figure 11. According to these results, GPAR



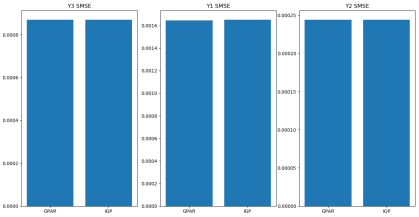
(a) Integral modelling comparison (5 Obs.)



(c) SMSE for integrand models (5 Obs.)



(b) Integral modelling comparison (15 Obs.)



(d) SMSE for integrand models (15 Obs.)

Figure 11: Baseline vs Naive GPAR and Gaussian integrand modelling comparison for y_1, y_2, and y_3

Obs	μ_3	σ_3	μ_1	σ_1	μ_2	σ_2
5	0.584031	0.041260	0.359485	0.044020	0.974550	0.061393
15	0.596376	0.022263	0.359807	0.022309	0.999099	0.022474
30	0.597741	0.015587	0.360689	0.015600	0.999513	0.015644
Truth	0.598674	-	0.361836	-	0.999767	-

Table 4: Baseline statistics comparison

Obs	μ_3	σ_3	μ_1	σ_1	μ_2	σ_2
5	0.584031	0.041260	0.361407	0.046628	0.955750	0.093960
15	0.596376	0.022263	0.363216	0.026232	0.920098	0.085130
30	0.597741	0.015587	0.364319	0.020785	0.915754	0.082817
Truth	0.598674	-	0.361836	-	0.999767	-

Table 5: GPAR statistics comparison

Integrands	Max $\int$ Evals.	Max $\iint$ Evals.
$\{y_n\}_{n=1}^{3}$	63	819
$\{y_n\}_{n=4}^{6}$	231	40047

Table 6: Intermediate integrand evaluations (15 Obs.)

is on par with IGPs, yet fails to achieve, at the very least, the same performance.

5.3.2 Complex Integrands with Explicit Causality

We now attempt a more difficult set of integrands to model. This example has been designed so that GPAR has a modelling advantage. The goal of this is twofold:

1. To see if GPAR can use a significant modelling advantage to offset the shortcomings discovered in the previous experiment.

2. To motivate the use of MC sampling due to the increased integrand complexity.

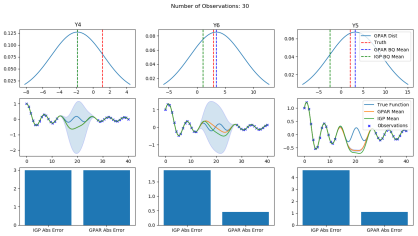
Using the Bessel function of the first kind $J_\alpha(x)$, we construct three different integrands (see below). We will use forty observations: twenty in the interval [0, 12] and the remaining twenty in [28, 40].

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}$$

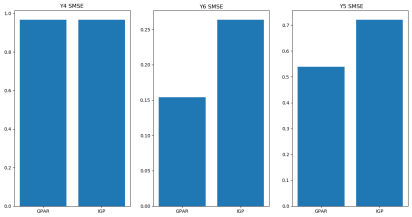
$$\begin{aligned} y_4 &= \int_0^{40} J_0(x)\mathrm{d}x \\ y_5 &= \int_0^{40} (J_0(x) + J_1(x))\mathrm{d}x \\ y_6 &= \int_0^{40} (J_0(x) + J_1(x) + J_2(x))\mathrm{d}x \end{aligned}$$

This is an example where, in the naive GPAR case, the intermediate integration steps will require more integrand evaluations. In order to show this, we could print the number of evaluations that the numerical integration subroutine (i.e. *nquad* from the SciPy library) uses. For each set of integrals (i.e. Gaussian and Bessel), Table 6 shows the maximum number of integrand evaluations for the single and double integral computations for the two experimental setups. As a reminder, we refer the reader to Equations 3 and 4 to point out that the single integrals are performed for each observation, and the double integrals are performed once. Because GP predictions are performed within the integrand, Table 6 demonstrates that these integrand evaluations could be a problem because the number of evaluations can become very high.

The results of predicting the integrals and modelling the integrands can be found in Figure 12. Naive GPAR has a lower absolute error for both y_5 and y_6. These results seem to support the argument that if the modelling advantage is significant enough, then GPAR could outperform the baseline.

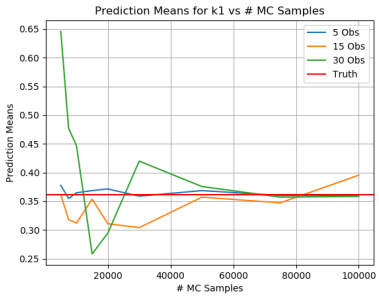


(a) Integral modelling comparison (30 Obs.)

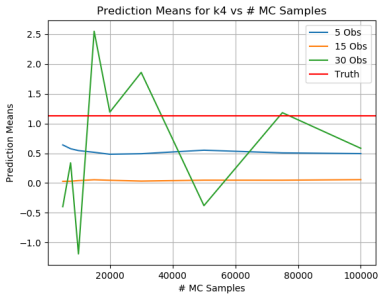


(b) SMSE for integrand models (30 Obs.)

Figure 12: Baseline vs Naive GPAR and Bessel integrand modelling comparison for y_4, y_5, and y_6



(a) y_1 Means vs # samples



(b) y_4 Means vs # samples

5.3.3 Varying MC Sample Size

We can explore the effects of imprecise computation for the intermediate integrals in Bayesian Quadrature. In Figures 13a and 13b, we respectively plot the means over both y_1 and y_4 over the number of MC samples when performing regular Bayesian quadrature. We plot these means for different numbers of observations in order to identify any relevant trends. In Figure 13a, it is unsurprising to see that the means seem to converge to the true value as the number of samples is increasing. Also, a lower number of observations seems to translate to having less variance, particularly for lower numbers of samples. This is also expected as Equations 3 and 4 show that for each observation there is one single integral computation. Hence, a low number of samples will provide weak estimates, and the aggregation of all of these estimates will contribute to the overall error. In other words, having more observations increases the significance of the poor approximations caused by an insufficient number of samples. Unfortunately, as shown in Figure 13b, limiting the number of observations is not possible because this causes a second problem: if there are not enough observations, which can be the case when the domain of integration gets large, then even with a reasonable number of samples, it will be difficult to converge to the true value. Also, to achieve comparable results to ones from the previous section, it seems like MC sampling requires many more evaluations of the intermediate integrands without providing any clear benefits.

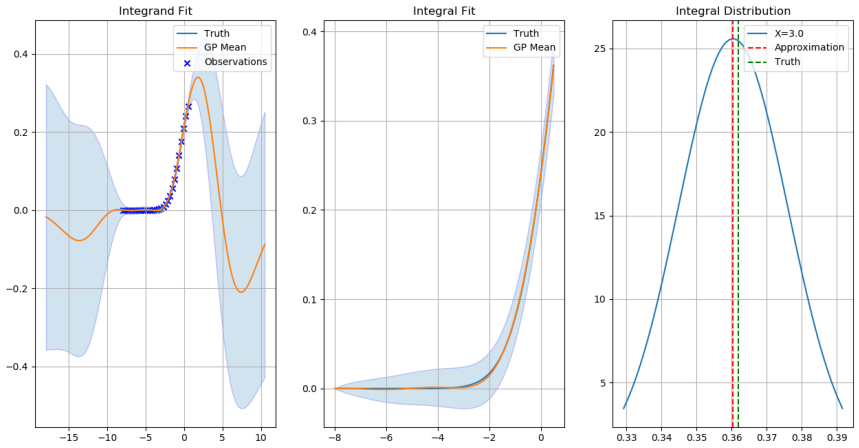


Figure 14: Derivative trick to model y_1 (30 Obs.)

5.3.4 One Dimensional Derivative Trick

A possible alternative to using MC to deal with the problems related to integrating, is to model the observations using a differentiated kernel. Figures 14 and 15 shows the results of modelling y_1 and y_4 using 30 observations.

The leftmost plot in Figure 14 shows how the integrand is being modelled by the GP, which uses a differentiated EQ kernel. As a reminder, y_1 is the integral from -8 to 0.5 of a Gaussian distribution with mean 1 and variance 2. The true value is also displayed for that range. As we move away from the region of integration, the predictive mean becomes more unwieldy. In the middle plot, we show the predictive mean for $g(x) = \int_{-8}^{x} \mathcal{N}(x'|1, 2)dx'$ that we compare with a Gaussian CDF which could be seen as a very close approximation for the true value. Finally, in the rightmost plot, we show the prediction of the integral at $x = 0.5$ (i.e. prediction for $g(0.5)$). Overall, this shows excellent potential for this approach. We also show similar good results for y_4 in Figure 15.

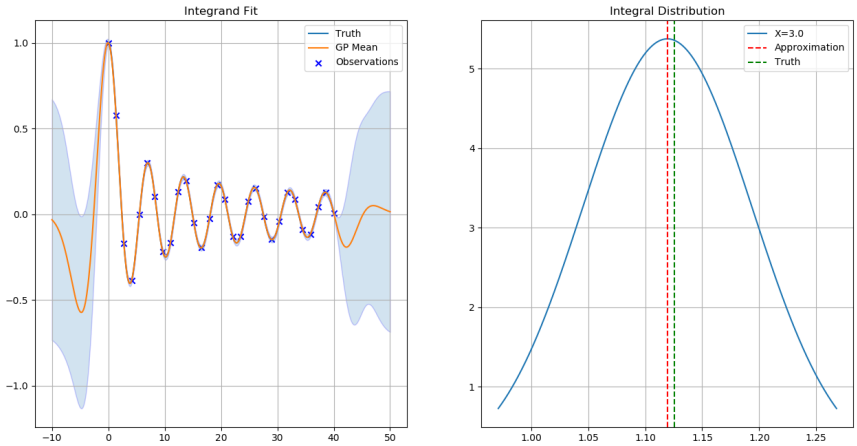


Figure 15: Derivative trick to model y_4 (30 Obs.)

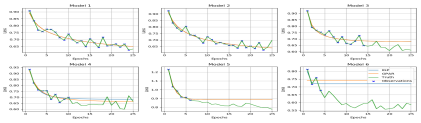
5.4 Freeze-Thaw Bayesian Optimisation

5.4.1 Loss curve Modelling

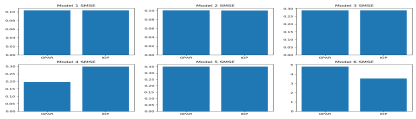
We want to investigate whether GPAR can model loss functions efficiently. First, we create a set of 6 neural networks, each with a varying number of units and layers. Then, we generate a series of loss functions for a variety of different epoch lengths. The GPs used in the modelling process are made up of a constant mean function as well as a kernel that is composed of the sums of the custom exponential kernel, where the domain is the number of epochs, and an EQ kernel, that takes in previous outputs. The latter component of the kernel is only relevant for GPAR as IGPs will only make use of the former component.

In Figure 16, we present three different runs of the same loss functions where different runs have different numbers of observations for each loss function. We notice that GPAR performs well when the number of observations for a particular curve is low because it can rely more on preceding outputs.

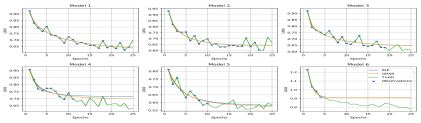
The training process also includes finding an appropriate value for the constant mean function. This may attenuate the advantages that GPAR has relative to IGPs. Indeed, Figure 17 shows the same configuration that was presented above but with the difference that both GPAR and IGP have a mean function that is always set to 0. By removing the mean function from any consideration, the overall effectiveness of GPAR appears to increase.



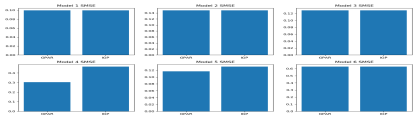
(a) Run 1 Predictions



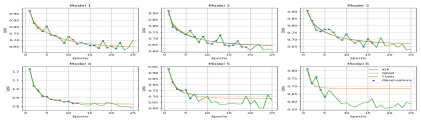
(d) Run 1 SMSE



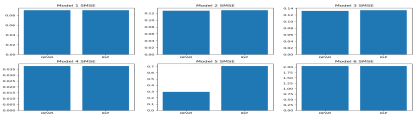
(b) Run 2 Predictions



(e) Run 2 SMSE

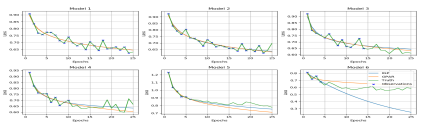


(c) Run 3 Predictions

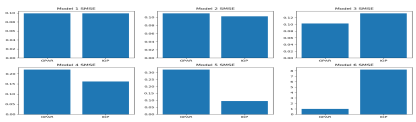


(f) Run 3 SMSE

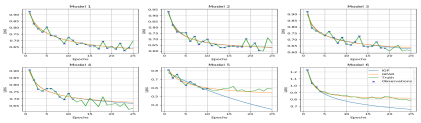
Figure 16: GPAR vs IGP models for loss functions (regular)



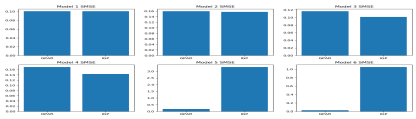
(a) Run 1 Predictions



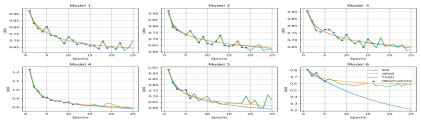
(d) Run 1 SMSE



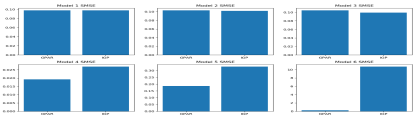
(b) Run 2 Predictions



(e) Run 2 SMSE



(c) Run 3 Predictions



(f) Run 3 SMSE

Figure 17: GPAR vs IGP models for loss functions (0 mean function)

6 Conclusion and Future Work

6.1 GPAR

As shown in multiple experiments, GPAR provides meaningful predictive performance benefits. Several important conclusions could be made from the experimental results.

First, we note that the number of random initialisations had to scale with the complexity of the system of functions. Within the GPAR framework, each function is modelled with a GP that has its hyperparameters. Increasing the number of functions that are being modelled requires more GPs, which in turn results in a higher hyperparameter count. As future work, it would be useful to mathematically quantify the rate at which the recommended number of restarts increases. Furthermore, this also serves as a motivation for finding ways to limit the number of hyperparameters in the model (e.g. parameter sharing).

Second, while running a simplified kernel search, it may not always be necessary to spend the resources to find an absolute best kernel. Although there may still be terrible kernel choices that can negatively impact performance in a significant way, a large number of kernels can have similar results. Further work could involve constructing a more efficient kernel search procedure that is less computationally intensive but still provides admirable results.

Thirdly, some function orders do provide terrible results. However, just like with the kernel search results, we can relax the requirement to find the absolute best order in favour of more computational efficiency. Finding cheaper ways of establishing order that does not involve computing a GP over every output could be an area for further investigation.

6.2 Bayesian Quadrature

For Bayesian quadrature, we are interested in seeing how the proposed naive GPAR implementation would perform. Unfortunately, for integrals where the integrand is relatively simple (i.e. very smooth), then it is better to stick to IGPs. However, when GPAR can significantly outperform IGPs in modelling the integrand, then using the naive GPAR implementation seems to be the better option.

Unfortunately, as the integrands get more sophisticated, the naive GPAR implementation gets more costly due to the increase in the number of function evaluations required to solve the intermediate integrals. This motivates an alternative approach. Using MC sampling to compute the intermediate integrals initially seemed like an attractive solution. The hope was that capping the number of samples would not result in a significant decrease in modelling performance. Sadly, MC was shown not to be a valid option. A larger domain of integration requires more observations, which in turn requires more MC samples because performance is more sensitive to the number of MC samples when the number of observations is high. In the end, this behaviour results in needing an unreasonable number of samples to model the integral under consideration correctly. Therefore, other approaches to solving this problem should be considered.

One promising way to avoid dealing directly with the intermediate integrals is to use a differentiated kernel to model integrand observations. This method was shown to function

well on a few examples. However, further work is still required to incorporate this approach within the GPAR framework.

6.3 Freeze-thaw Optimisation

The presented experiments show that GPAR can offer an advantage in modelling loss functions of a common model with different hyperparameters configurations. Modelling these curves is a key component in the freeze-thaw approach. Concerning future work, one possible improvement would be to find more efficient ways to use a mean function while still allowing GPAR to maintain the same level of performance difference with IGPs. Also, more generally, it would be interesting to quantify to what extent better models of the loss curves benefits the overall performance of freeze-thaw.

7 Appendix

7.1 GP Posterior Equations

Assuming a training set of N observations is provided, and we are interested in predicting the values of a function at some test indices. For the training set, we define $\mathbf{y}$ and $\mathbf{X}$ to respectively correspond to the function values and their associated indices. Each entry in the vector $\mathbf{y}$ is a function value and each row in matrix $\mathbf{X}$ is an index. Following the same logic, we use $\mathbf{y}_*$ and $\mathbf{X}_*$ for the set of function observations and indices that we are interested in getting predictions for.

With a kernel, we can specify the join distribution over the values for both the training and the desired sets of indices:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} K(\mathbf{X},\mathbf{X}) & K(\mathbf{X},\mathbf{X}_*) \\ K(\mathbf{X}_*,\mathbf{X}) & K(\mathbf{X}_*,\mathbf{X}_*) \end{bmatrix} \right)$$

Using the relevant Equations for the conditional distribution of a multivariate Gaussian (see chapter 2 in [1]), we condition over the training set to obtain the following:

$$\begin{aligned} \mu_{\mathbf{y}_*|\mathbf{y},\mathbf{X},\mathbf{X}_*} &= \mathbf{0} + K(\mathbf{X}_*,\mathbf{X})K(\mathbf{X},\mathbf{X})^{-1}(\mathbf{y}-\mathbf{0}) \\ &= K(\mathbf{X}_*,\mathbf{X})K(\mathbf{X},\mathbf{X})^{-1}\mathbf{y} \\ \mathbf{\Sigma}_{\mathbf{y}_*|\mathbf{y},\mathbf{X},\mathbf{X}_*} &= K(\mathbf{X}_*,\mathbf{X}_*) - K(\mathbf{X}_*,\mathbf{X})K(\mathbf{X},\mathbf{X})^{-1}K(\mathbf{X},\mathbf{X}_*) \end{aligned}$$

7.2 Custom Exponential Kernel

We show a derivation for the closed form of the custom kernel, in freeze-thaw optimisation, when using a Gamma distribution $p(x) = \frac{1}{\Gamma(\alpha)}\beta^\alpha x^{\alpha-1}e^{-\beta x}$ over λ. Note that $\Gamma(\alpha)$ is the Gamma function (i.e. $\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-x}\mathrm{d}x$).

$$\begin{aligned} k(x,x') &= \int_0^\infty e^{-\lambda x}e^{-\lambda x'}p(\lambda)\mathrm{d}\lambda \\ &= \int_0^\infty e^{-\lambda x}e^{-\lambda x'}\frac{1}{\Gamma(\alpha)}\beta^\alpha\lambda^{\alpha-1}e^{-\beta\lambda}\mathrm{d}\lambda \\ &= \int_0^\infty \frac{1}{\Gamma(\alpha)}\beta^\alpha\lambda^{\alpha-1}e^{-\lambda(x+x'+\beta)}\mathrm{d}\lambda \\ &= \frac{\beta^\alpha}{(x+x'+\beta)^\alpha}\int_0^\infty \frac{1}{\Gamma(\alpha)}(x+x'+\beta)^\alpha\lambda^{\alpha-1}e^{-\lambda(x+x'+\beta)}\mathrm{d}\lambda \\ &= \frac{\beta^\alpha}{(x+x'+\beta)^\alpha} \end{aligned}$$

7.3 Expected Improvement Acquisition Function

Below is a derivation of the closed-form of the expected improvement acquisition function. We respectively denote $\phi(x, \mu, \sigma^2)$ and $\phi(x, \mu, \sigma^2)$ as the probability density function and cumulative density function for a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$.

$$
\begin{aligned}
\alpha_{EI}(x) &= \mathbb{E}[u(x)] \\
&= \int_{-\infty}^{\infty} u(x) p(f) \mathrm{d}f \\
&= \int_{-\infty}^{\infty} u(x) \mathcal{N}(f | m(x), k(x,x)) \mathrm{d}f \\
&= \int_{-\infty}^{f_{\min}} (f_{\min} - f) \mathcal{N}(f | m(x), k(x,x)) \mathrm{d}f \\
&= f_{\min}(x) \Phi(f_{\min} | m(x), k(x,x)) - \int_{-\infty}^{f_{\min}} f \mathcal{N}(f | m(x), k(x,x)) \mathrm{d}f \\
&= f_{\min}(x) \Phi(f_{\min} | m(x), k(x,x)) + \int_{-f_{\min}}^{\infty} \frac{f}{\sqrt{2\pi k(x,x)}} e^{\frac{-(f-m(x))^2}{2k(x,x)}} \mathrm{d}f \\
&= f_{\min}(x) \Phi(f_{\min} | m(x), k(x,x)) + \frac{1}{2} \int_{-f_{\min}}^{\infty} \frac{2(f - m(x))}{\sqrt{2\pi k(x,x)}} e^{\frac{-(f-m(x))^2}{2k(x,x)}} \mathrm{d}f \\
&\quad + \int_{-f_{\min}}^{\infty} \frac{m(x)}{\sqrt{2\pi k(x,x)}} e^{\frac{-(f-m(x))^2}{2k(x,x)}} \mathrm{d}f \\
&= f_{\min}(x) \Phi(f_{\min} | m(x), k(x,x)) + \frac{1}{2} \int_{(f_{\min} - m(x))^2}^{\infty} \frac{1}{\sqrt{2\pi k(x,x)}} e^{\frac{-u}{2k(x,x)}} \mathrm{d}u \\
&\quad + \int_{-f_{\min}}^{\infty} \frac{m(x)}{\sqrt{2\pi k(x,x)}} e^{\frac{-(f-m(x))^2}{2k(x,x)}} \mathrm{d}f \\
&= f_{\min}(x) \Phi(f_{\min} | m(x), k(x,x)) + \frac{k(x,x)}{\sqrt{2\pi k(x,x)}} e^{\frac{-u}{2k(x,x)}} \Bigg|_{u=(f_{\min} - m(x))^2}^{\infty} \\
&\quad + \int_{-f_{\min}}^{\infty} \frac{m(x)}{\sqrt{2\pi k(x,x)}} e^{-\frac{(f-m(x))^2}{2k(x,x)}} \mathrm{d}f \\
&= f_{\min}(x) \Phi(f_{\min} | m(x), k(x,x)) + k(x,x) \phi(f_{\min}, m(x), k(x,x)) \\
&\quad + \int_{-f_{\min}}^{\infty} \frac{m(x)}{\sqrt{2\pi k(x,x)}} e^{-\frac{(f-m(x))^2}{2k(x,x)}} \mathrm{d}f \\
&= f_{\min}(x) \Phi(f_{\min} | m(x), k(x,x)) + k(x,x) \phi(f_{\min}, m(x), k(x,x)) \\
&\quad - m(x) \int_{-\infty}^{f_{\min}} \frac{1}{\sqrt{2\pi k(x,x)}} e^{-\frac{(f-m(x))^2}{2k(x,x)}} \mathrm{d}f \\
&= f_{\min}(x) \Phi(f_{\min} | m(x), k(x,x)) + k(x,x) \phi(f_{\min}, m(x), k(x,x)) \\
&\quad - m(x) \Phi(f_{\min} | m(x), k(x,x)) \\
&= (f_{\min}(x) - m(x)) \, \Phi(f_{\min} | m(x), k(x,x)) + k(x,x) \phi(f_{\min}, m(x), k(x,x))
\end{aligned}
$$

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] Abdallah A. Chehade and Ala A. Hussein. Latent function decomposition for forecasting li-ion battery cells capacity: A multi-output convolved gaussian process approach, 2019.

[3] David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search, 2013.

[4] Ian Fox, Lynn Ang, Mamta Jaiswal, Rodica Pop-Busui, and Jenna Wiens. Deep multi-output forecasting: Learning to accurately predict blood glucose trajectories. 2018.

[5] Joseph Futoma, Sanjay Hariharan, Mark Sendak, Nathan Brajer, Meredith Clement, Armando Bedoya, Cara O'Brien, and Katherine Heller. An improved multi-output gaussian process rnn with real-time validation for early sepsis detection, 2017.

[6] J. Lebl. *Basic Analysis: Introduction to Real Analysis*. Createspace Independent Pub, 2014.

[7] A. O'Hagan. Bayes–hermite quadrature. *Journal of Statistical Planning and Inference*, 29(3):245–260, nov 1991.

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

[9] James Requeima, Will Tebbutt, Wessel Bruinsma, and Richard E. Turner. The gaussian process autoregressive regression model (gpar), 2018.

[10] Filipe Rodrigues, Kristian Henrickson, and Francisco C. Pereira. Multi-output gaussian processes for crowdsourced traffic data imputation. 2018.

[11] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. 06 2014.

[12] Xiaoyue Xi, Francois-Xavier Briol, and Mark Girolami. Bayesian quadrature for multiple related integrals. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5373–5382, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.