Probabilistic Numerical Linear Algebra for Machine Learning

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von

Jonathan Wenger

aus Nürnberg

Tübingen
2023

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation: 25. Juli 2023

Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Philipp Hennig
2. Berichterstatter:	Prof. Dr. Robert C. Williamson

Abstract

Machine learning models are becoming increasingly essential in domains where critical decisions must be made under uncertainty, such as in public policy, medicine or robotics. For a model to be useful for decision-making, it must convey a degree of certainty in its predictions. Bayesian models are well-suited to such settings due to their principled uncertainty quantification, given a set of assumptions about the problem and data-generating process. While in theory, inference in a Bayesian model is fully specified, in practice, numerical approximations have a significant impact on the resulting posterior. Therefore, model-based decisions are not just determined by the data but also by the numerical method. This begs the question of how we can account for the adverse impact of numerical approximations on inference.

Arguably, the most common numerical task in scientific computing is the solution of linear systems, which arise in probabilistic inference, graph theory, differential equations and optimization. In machine learning, these systems are typically large-scale, subject to noise and arise from generative processes. These unique characteristics call for specialized solvers. In this thesis, we propose a class of probabilistic linear solvers, which infer the solution to a linear system and can be interpreted as learning algorithms themselves. Importantly, they can leverage problem structure and propagate their error to the prediction of the underlying probabilistic model. Next, we apply such solvers to accelerate Gaussian process inference. While Gaussian processes are a principled and flexible model class, for large datasets inference is computationally prohibitive both in time and memory due to the required computations with the kernel matrix. We show that by approximating the posterior with a probabilistic linear solver, we can invest an arbitrarily small amount of computation and still obtain a provably coherent prediction that quantifies uncertainty exactly. Finally, we demonstrate that Gaussian process hyperparameter optimization can similarly be accelerated by leveraging structural prior knowledge in the model via preconditioning of iterative methods. Combined with modern parallel hardware, this enables training Gaussian process models on datasets with hundreds of thousands of data points.

In summary, we demonstrate that interpreting numerical methods in linear algebra as probabilistic learning algorithms unlocks significant performance improvements for Gaussian process models. Crucially, we show how to account for the impact of numerical approximations on model predictions via uncertainty quantification. This enables an explicit trade-off between computational resources and confidence in a prediction. The techniques developed in this thesis have advanced the understanding of probabilistic linear solvers [1], they have shifted the goalposts of what can be expected from Gaussian process approximations [2] and they have defined the way large-scale Gaussian process hyperparameter optimization is performed in GPyTorch [3, 4], arguably the most popular library for Gaussian processes in Python.

Zusammenfassung

Maschinelle Lernmodelle werden immer wichtiger in Bereichen, in denen kritische Entscheidungen unter Unsicherheit getroffen werden müssen, wie z. B. in Politik, Medizin oder Robotik. Damit ein Modell nützlich ist, muss es genau sein und die Unsicherheit in seine Vorhersagen quantifizieren, sodass das Treffen einer Entscheidung gegen das Erheben weiterer Daten abgewogen werden kann. Bayes'sche Modelle eignen sich aufgrund ihrer präzisen Unsicherheitsquantifizierung besonders für derartige Situationen. Theoretisch ist die Inferenz in einem Bayes'schen Modell zwar vollständig spezifiziert, in der Praxis allerdings haben numerische Approximationen einen erheblichen Einfluss. Daher werden modellbasierte Entscheidungen nicht nur von den Daten beeinflusst, sondern problematischerweise auch von numerischen Methoden. Dies wirft die Frage auf, wie man den Einfluss numerischer Approximationen berücksichtigen kann.

Die wohl häufigste numerische Problemstellung im wissenschaftlichen Rechnen ist die Lösung linearer Systeme, die in der Wahrscheinlichkeitsrechnung, der Graphentheorie, bei Differentialgleichungen und in der Optimierung auftreten. Im maschinellen Lernen sind diese Systeme in der Regel hochdimensional, verrauscht und entstehen durch generative Prozesse. Diese spezifischen Eigenschaften erfordern spezialisierte Lösungsmethoden. Im Rahmen dieser Arbeit beschreiben wir eine Klasse probabilistischer linearer Löser, die die Lösung eines linearen Systems schätzen und selbst als Lernalgorithmen interpretiert werden können. Solche Löser nutzen die Problemstruktur explizit und können ihren intrinsischen Approximationsfehler an ein übergeordnetes probabilistisches Modell propagieren. Im Anschluss wenden wir solche Löser an um die Inferenz mit Gaußschen Prozessen zu beschleunigen. Während Gaußprozesse eine gut fundierte und flexible Modellklasse sind, ist Inferenz für große Datensätze sowohl zeitlich als auch speichertechnisch nicht realisierbar. Wir demonstrieren, dass durch die Approximation des Posteriors mit einem probabilistischen linearen Löser ein beliebig geringer Rechenaufwand dennoch eine beweisbar kohärente Vorhersage mit exakter Unsicherheitsquantifikation zulässt. Abschließend zeigen wir, dass die Optimierung der Hyperparameter von Gaußprozessen in ähnlicher Weise beschleunigt werden kann, indem Struktur im Modell durch Präkonditionierung von iterativen Methoden automatisch genutzt wird. In Kombination mit moderner paralleler Hardware ermöglicht dies das Training von Gaußprozessen auf Datensätzen mit Hunderttausenden von Datenpunkten.

Zusammenfassend kann die Inferenz und das Optimieren von Hyperparametern von Gaußschen Prozessen signifikant beschleunigt werden, indem man numerische Methoden in der linearen Algebra als probabilistische Lernalgorithmen interpretiert. Insbesondere zeigen wir, dass die Auswirkungen numerischer Approximation auf Modellvorhersagen durch Unsicherheit quantifiziert werden können. Dies ermöglicht eine explizite Abwägung zwischen Rechenressourcen und Unsicherheit bei Vorhersagen. Die in dieser Arbeit entwickelten Methoden haben das

Verständnis von probabilistischen linearen Löser erweitert [1], sie haben gezeigt, was von Approximationen von Gaußprozessen erwartet werden kann [2], und sie haben die Art und Weise definiert, wie Hyperparameteroptimierung in GPyTorch [3, 4], der größten Softwarebibliothek für Gaußprozesse in Python, durchgeführt wird.

Acknowledgements

This thesis would not have been possible without the direct and indirect support of many people.

First and foremost, I would like to thank Philipp Hennig for his scientific guidance, personal and professional mentoring and unwavering support. You've been an inspiration to me as a scientist and I hope I can one day pass on what you have taught me.

I would also like to thank John Cunningham for his belief in me early on, the opportunity to come to Columbia University and of course for giving Geoff, Taiga and me a place to sleep. I am beyond excited for the scientific challenges to come.

I am thankful for the inspiring work with my collaborators, who have taught me more than any book or paper ever will. Special thanks go out to Geoff Pleiss, Marvin Pförtner, Jacob Gardner and the ProbNum development team.

I would also like to thank the entire Methods of Machine Learning group who've made the past four years so enjoyable. In particular, for the comradery, which was invaluable when times were tough. Special thanks to Felix and Agustinus for always being available for a run and the entire MvL6 cycling community, as well as to Franziska Weiler who's always been supportive in any situation.

I would like to thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for their support, especially Leila Masri and Sara Sorce who were always the first to help and answer questions.

I am grateful to my parents, Lydia and Johann Wenger, for giving me early access to educational resources and instilling curiosity in me about the world. You've always supported me in whatever quest I have pursued. I am also thankful for the support of my sister Katharina. You've been there for me from the very start and have always inspired me. I would also like to thank Bob Kamen and Trudi Veldman for fostering a love for science in me and providing guidance and the necessary freedom in a formative period of my life.

Finally, I would like to thank Hanna Dettki for all the deep conversations, shared moments and your emotional support throughout. Thank you for bringing color to my life. I cannot wait for what the future will hold.

Thank you!

Jonathan Wenger
Tübingen, April 28, 2023

Contents

Notation

Basic Mathematical Objects

a	A scalar
$\boldsymbol{a}$	A (column) vector
$\boldsymbol{A}$	A matrix
$\boldsymbol{A}_{i,j}$	The (i, j)th entry of the matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$ (or $\boldsymbol{A}_{:,j}$)	The ith row (or jth column) of the matrix $\boldsymbol{A}$
f	A function

Topology

S^{o}	Interior of a subset S of a topological space
$\overline{S}$	Closure of a subset S of a topological space
∂S	Boundary of a subset S of a topological space

Linear Algebra

$\mathrm{im}(\boldsymbol{A})$	Image of a linear map ($=$ column space or range)
$\ker(\boldsymbol{A})$	Kernel of a linear map ($=$ null space)
$\mathrm{rank}(\boldsymbol{A})$	Rank of a linear map
$\mathrm{tr}(\boldsymbol{A})$	Trace of a (square) matrix
$\det(\boldsymbol{A})$	Determinant of a (square) matrix

Structured and special matrices.

$\boldsymbol{I}$	Identity matrix
$\mathrm{diag}(\cdot)$	Diagonal matrix

Analysis

$\mathbb{N}$	Natural numbers
$\mathbb{Z}$	Integers
$\mathbb{R}$	Real numbers
$\mathbb{C}$	Complex numbers
$\mathbb{R}^d$	Euclidean space of dimension d
$\mathbb{R}^{\mathbb{X}}$	Space of functions from $\mathbb{X}$ to $\mathbb{R}$

Vector and matrix norms, metrics and inner products.

$\lvert \cdot \rvert$	(Elementwise) absolute value
$\lVert \cdot \rVert_{\mathbb{V}}$	Norm of a vector in a normed space $\mathbb{V}$
$\lVert \boldsymbol{x} \rVert_p$	p-Norm, where $p \geq 1$
$\lVert \boldsymbol{A} \rVert_F$	Frobenius norm
$\lVert \boldsymbol{A} \rVert_*$	Nuclear norm
$\langle \cdot\,, \cdot \rangle_{\mathbb{H}}$	Inner product in a (pre-)Hilbert space $\mathbb{H}$

Probability Theory

a	A random scalar
$\mathbf{a}$	A random vector
$\mathbf{A}$	A random matrix
$\mathrm{P}(\cdot)$	Probability of an event
$\mathrm{P}(\cdot \mid \cdot)$	Conditional probability of an event
$\mathbb{E}(\cdot)$	Expectation of a random variable
$\mathrm{Cov}(\cdot, \cdot)$	Covariance between two random variables
$\mathrm{Var}(\cdot)$	Variance of a random variable

Distributions and densities.

$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution
$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian probability density function
$\mathcal{GP}(\mu, k)$	Gaussian process

Statistical distances.

$d_{\mathrm{KL}}(p \parallel q)$	Kullback-Leibler divergence

Numerical Analysis

i	Iteration index of an iterative algorithm
$\boldsymbol{P}$	A preconditioner of a matrix

Algorithms

$\mathcal{O}(\cdot)$	Landau "Big O" notation
$o(\cdot)$	Landau "small O" notation

Machine Learning

$\mathbb{X}$	Input space
d	Dimension of input space
$\mathbb{F}$	Output space of a model
$\mathbb{Y}$	Output / target space
d'	Dimension of output / target space
n	Number of training data
$\boldsymbol{X}$	Training data inputs
$\boldsymbol{y}$	Training data targets
$\boldsymbol{X}_\diamond$	Test data inputs
$\boldsymbol{w}$	Parameters of a model
$\boldsymbol{\theta}$	Hyperparameters of a model
ℓ	Loss function
$\bar{\ell}$	Regularized loss function

Acronyms & Abbreviations

(P)CG	(Preconditioned) conjugate gradient method
CGGP	Conjugate-gradient-based Gaussian processes
DTC	Deterministic training conditional
ELBO	Evidence lower bound
EVD	Eigenvalue decomposition
GP	Gaussian process
GPU	Graphics processing unit
i.i.d.	independent and identically distributed
KL	Kullback-Leibler divergence
L-BFGS	Limited memory BFGS, a type of quasi-Newton method
NLL	Negative log-likelihood

ODE	Ordinary differential equation
PDE	Partial differential equation
PLS	Probabilistic linear solver
QFF	Quadrature Fourier features
RBF	Radial basis function
RFF	Random Fourier features
RKHS	Reproducing kernel Hilbert space
RMSE	Root mean square error
SLQ	Stochastic Lanczos quadrature
s.p.d.	symmetric positive definite
STE	Stochastic trace estimate
SoD	Subset of data
SoR	Subset of regressors
SVD	Singular value decomposition
SVGP	Scalable variational Gaussian processes

Chapter

1

Introduction

On New Year's Day in 1801, Italian astronomer Guiseppe Piazzi discovered what was then thought to be an unknown planet between the orbits of Mars and Jupiter. Piazzi observed the location of the celestial object, later named Ceres, multiple times over the coming weeks until Ceres's position had shifted too close to the sun's glare to be observable. When Piazzi published his observations, the news about a potential discovery of an unknown planet sparked instant excitement in the astronomical community. The leading European astronomers of the time tried repeatedly to confirm Piazzi's findings by observing Ceres after its reemergence out of the sun's

obfuscation. However, searching the sky for an object like Ceres was a momentous challenge in 1801 without knowing precisely where to look and no accurate models for the movements of celestial bodies were known. At least until the problem of determining Ceres's orbit caught the attention of a young mathematician by the name of Carl Friedrich Gauss. Only 24 years of age at the time, Gauss accurately predicted the position of Ceres from only a handful of Piazzi's measurements, leading to its rediscovery by astronomers by the end of 1801. Gauss achieved this magnificent feat by combining the assumption of an elliptical orbit with an elegant way to incorporate approximate data, now known as the *method of least squares*, and lots of tedious calculations by hand [5, 6]. The rediscovery of Ceres made Gauss almost instantaneously famous throughout the scientific community in Europe, in part due to the popular appeal of astronomy at the time [7]. The method of least squares as employed by Gauss[1] is not only still in use in astrophysics today but in all of science. In fact, a large part of this thesis can be viewed as a modern treatment of the method of least squares in an age of large-scale data and computation.

1.1 Learning to Predict from Data

In his quest to predict the position of Ceres, Gauss combined assumptions about the movement of celestial bodies with the approximate measurements of Ceres's position by Piazzi. This combination of *prior knowledge* with noise-corrupted *measurements* allowed him to make highly accurate predictions. Such a strategy for making predictions is not only fundamental to scientific discovery but a key part of our everyday life. All of us constantly rely on our prior experience in combination with observations to predict the world around us. How long will it take to get from A to B? What will the weather be like this afternoon? Which direction is a ball going to bounce off the ground? One of the hallmarks of intelligent behavior is to take actions to observe the environment, learn to predict based on observed patterns and make decisions accordingly [9, 10]. Replicating such behavior in machines is the overarching goal of the field of machine learning.

1.1.1 Machine Learning

Machine learning describes the concept of a machine improving its performance on a complex task by leveraging data [11]. This is fundamentally different from the classic programming paradigm, in which a computer is given an explicit set of instructions to complete a task. Instead, a machine learning algorithm is programmed to learn from examples [12, 13].

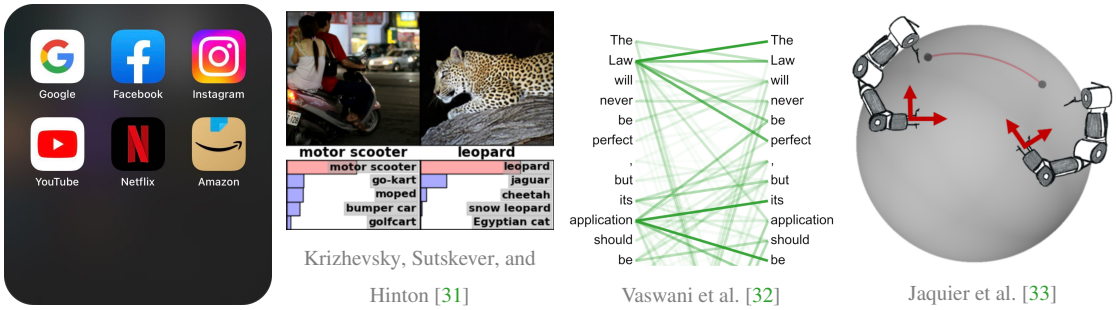
Advantages of Learning Algorithms While humans can be formidable at prediction, as the story of Gauss and Ceres illustrates, learning algorithms have some key advantages. We are fundamentally limited in how much information we can absorb, process and share with others in a given amount of time. To learn a new language we cannot read hundreds of thousands of pages of translated text in a few days to pick up a language. And if we meet a native speaker, they can try to explain the grammar of the language to us, but not share their language proficiency instantly.

[1] The discovery of the method of least squares is often credited to Adrien-Marie Legendre, who published before Gauss, although Gauss likely had used the method years before [8].

In contrast, one can train a machine learning model to translate text to reasonable accuracy in a few hours and the trained model can be duplicated instantaneously. This is why we aim to equip machines with the ability to learn from experience. Their ability to efficiently learn to make predictions and automated decisions at scale from a large number of observations, while being almost seamlessly replicable.

Applications Machine learning is used in a wide range of domains from commercial to medical applications, robotics and engineering, as well as the natural sciences (see Figure 1.1). Arguably, the most commercially successful application of machine learning is content discovery and personalized advertisement. Some of the largest companies in the 21st century have built their businesses on providing individualized content based on the predicted interests of their users, be it search results [14], social media or advertisement [15]. Recently, generative models have made rapid progress in their ability to produce photo-realistic images [16], natural language text [17] and machine code [18]. They are trained on vast amounts of data typically scraped from the web and generate content based on user-specified input prompts. In many engineering applications, as well as in medicine, predicting the content of images can aid decision-making at scale. For example, to rapidly diagnose faulty parts during quality control [19] or in a hospital to diagnose potentially malignant tissue from imaging data [20]. In the last decade, machine learning has also increasingly been applied in the physical sciences to accelerate scientific discovery [21]. Among other domains, it has been used for climate modeling [22], astronomy [23, 24], fundamental physics [25–27], neuroscience [28, 29] and protein folding [30]. The physical sciences are a particularly promising domain for machine learning since many experiments nowadays produce large amounts of unstructured data, e.g. in fundamental physics or genetics, and one can often rely on strong inductive biases about physical phenomena from established mechanistic models.

Learning Paradigms Depending on what kind of data is available and whether a machine learning model can interact with its environment different learning frameworks exist, loosely inspired by how humans learn. The most common learning paradigm is *supervised learning*, where the goal is to learn an unknown functional relationship from a set of example input-output pairs. Classic examples of this task are predicting the CO_2 content in the atmosphere over time or classifying images based on what they depict. Sometimes the labels for the inputs are not explicitly given but can be determined from the data itself without human annotation. This paradigm is known as *self-supervised learning* and is often applied to data that has some inherent time structure. For example, a language model can evaluate its performance by predicting the next word in a sentence, given all the previous words. If we do not have labeled examples available only an unstructured dataset, we can use *unsupervised learning* to discover clusters in the dataset or detect anomalies when observing new data. Finally, if the learning agent can interact with its environment by taking actions and obtaining observations, it is solving a *reinforcement learning* problem to maximize its internal notion of reward. This is often used for robotics and gameplay. If labeled data is expensive to obtain, one can employ *active learning*, meaning the model explicitly queries for specific labeled examples to learn efficiently in a supervised manner. Finally, we might want to transfer what we have learned on one dataset to another to learn more efficiently on a new problem where we only have little data available. This is known as *transfer learning*.

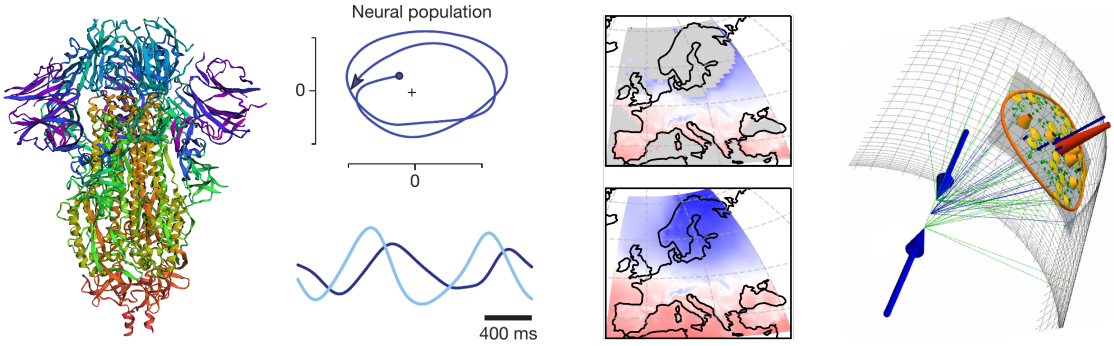


(a) Content discovery

(b) Computer Vision — Krizhevsky, Sutskever, and Hinton [31]

(c) Language modeling — Vaswani et al. [32]

(d) Robotics — Jaquier et al. [33]



(e) Structural biology — Walls et al. [34]

(f) Neuroscience — Churchland et al. [28]

(g) Climate modeling — Axen et al. [35]

(h) Particle physics — Komiske et al. [26]

Figure 1.1: *Examples of machine learning applications.*

1.1.2 A Mathematical Theory of Learning

In this thesis, we focus on the supervised learning setting, which we will now formalize. Assume there exists an unknown functional relation between inputs $\boldsymbol{x} \in \mathbb{X}$ and targets $y \in \mathbb{Y}$. The problem of learning this relationship from data is known as *regression* if $\mathbb{Y} \subset \mathbb{R}^{d'}$ and as *classification* if $\mathbb{Y} = \{1, \dots, C\}$.

In order to learn the relationship between inputs and targets, we define a model $f_{\boldsymbol{w}} : \mathbb{X} \to \mathbb{F}$ that depends on parameters $\boldsymbol{w} \in \mathbb{W}$, where either $\mathbb{F} = \mathbb{R}^{d'}$ or $\mathbb{F} = \mathbb{R}^{C}$. We want to identify parameters $\boldsymbol{w}$ such that our model has good predictive performance as measured by a loss function $\ell : \mathbb{Y} \times \mathbb{F} \to \mathbb{R}$. In other words, we want to minimize the *risk*

$$\mathbb{E}_{p_{\text{data}}(\boldsymbol{x},y)}(\ell(y, f_{\boldsymbol{w}}(\boldsymbol{x}))) \,, \tag{1.1}$$

defined as the expected loss of the model under the *data distribution* $p_{\text{data}}(\boldsymbol{x}, y)$, with respect to the parameters $\boldsymbol{w}$.

Example 1.1 (Linear regression with square loss)
Linear regression assumes a model of the form $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\intercal \boldsymbol{w}$, where $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^p$ is a feature map. The loss function is given by the square loss, defined as

$$\ell(y_j, f_{\boldsymbol{w}}(\boldsymbol{x}_j)) = (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2.$$

Example 1.2 (Logistic regression with binary cross-entropy loss)
Binary classification via logistic regression defines a model of the form $f_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma(\boldsymbol{\phi}(\boldsymbol{x})^\intercal \boldsymbol{w})$, where $\sigma : \mathbb{R} \to \mathbb{R}$ is the logistic function. The loss function is given by the binary cross-entropy, defined as

$$\ell(y_j, f_{\boldsymbol{w}}(\boldsymbol{x}_j)) = -y_j \log(f_{\boldsymbol{w}}(\boldsymbol{x}_j)) - (1 - y_j) \log(1 - f_{\boldsymbol{w}}(\boldsymbol{x}_j)).$$

Unfortunately, in practice the true distribution of the data as defined by the underlying data-generating process is unknown and we cannot minimize the risk (1.1) directly. However, in the supervised setting we have access to a training dataset $(\boldsymbol{X}, \boldsymbol{y})$ of n input and output pairs $(\boldsymbol{x}_j, y_j) \in \mathbb{X} \times \mathbb{Y}$ assumed to be independent and identically distributed draws from the latent data distribution $p_{\text{data}}(\boldsymbol{x}, y)$. These might come from a series of physical experiments or passive observations. Therefore, instead, we aim to find parameters $\boldsymbol{w}$ which minimize the *(regularized) empirical risk* $\bar{\ell}$ as a readily available proxy of the risk (1.1), i.e.

$$\boldsymbol{w}_* = \arg\min_{\boldsymbol{w}} \bar{\ell}(\boldsymbol{w}) \qquad \text{where} \qquad \bar{\ell}(\boldsymbol{w}) = \sum_{j=1}^{n} \ell(y_j, f_{\boldsymbol{w}}(\boldsymbol{x}_j)) + R(\boldsymbol{w}). \tag{1.2}$$

The sum over the training in- and output pairs encourages the model to predict well on the training data as measured by the per-sample-loss $\ell(y_j, f_{\boldsymbol{w}}(\boldsymbol{x}_j))$. However, it is not enough for a model to predict well on the training data. We often have other desired properties of a predictive model which we want to explicitly encourage or discourage. For example, to avoid models which only fit the training data, but do not perform well on unseen test data, we can introduce a regularizer $R(\boldsymbol{w})$ which penalizes "extreme" parameter values preventing the model from overfitting.

1.1.3 Desired Properties of a Predictive Model

For a predictive model to be useful in practice, it is not sufficient to achieve good performance on the training data. Ideally, we would like to use a model which makes accurate predictions on arbitrary new data, can quantify the certainty in its predictions, is resource-efficient and adheres to ethical and societal constraints.

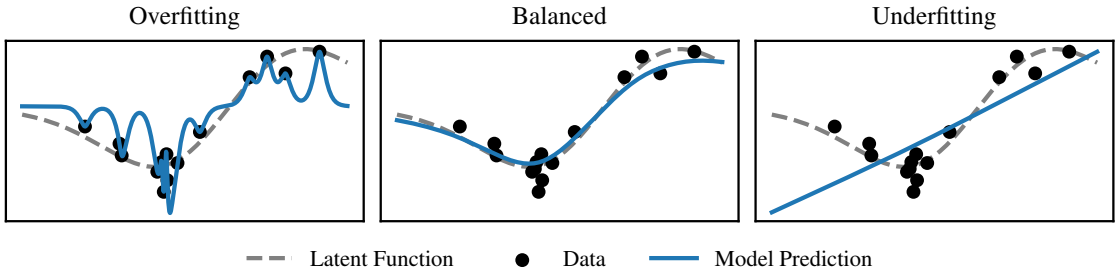


Figure 1.2: *Bias and variance trade-off.* Overly flexible models overfit to the training data resulting in larger variance across different training datasets from the same data distribution. In contrast, inflexible models have strong inductive bias and do not adapt to the training data very much. An ideal model balances inductive bias and model fit, such that it generalizes well.

Predictive Performance and Generalization First and foremost when using a machine learning model we are interested in how accurate its predictions are. In practice, we optimize performance on the training dataset we have at our disposal. However, this is not necessarily indicative of predictive performance on unseen data, known as *generalization*. Instead, there is a trade-off between the data-adaptivity of a model and its ability to generalize, as Figure 1.2 illustrates. An overly flexible model can perfectly fit the training data without learning about the true latent function. In contrast, a rigid model with strong inductive bias might introduce unavoidable error but generalize better away from the training data. In practice finding the right balance between bias and variance is an important criterion for generalization performance.

Uncertainty Quantification: Knowing When We Don't Know Whenever we make a prediction it is important to know how much trust we should place in that prediction. This is crucial if we rely on that prediction to make a downstream decision. Therefore a model must accurately quantify the uncertainty in its prediction (see Figure 1.3). For example, say we have a model which predicts the number of infections during a pandemic. If there is large uncertainty in our prediction, we should more strictly enforce contact restrictions to avoid a potential population-wide outbreak. Uncertainty also serves as an important diagnostic for whether and how we can improve predictions relative to the amount and quality of data we have and the resources we have invested in our model. If we only become marginally more certain by collecting more data on a certain scenario, such data is no longer useful to improve our predictions.

Data and Computational Efficiency In every practical setting, we are resource constrained. We can only collect a certain amount of data and we only have a certain amount of computational resources available. A desirable predictive model uses very little data, can handle low-quality data gracefully and only requires few computational resources to train and make a prediction. Whether this is possible often depends on the inductive bias of the model. Meaning, prior to seeing any data, how much information about the relationship we are trying to learn is encoded in the model as is illustrated in Figure 1.4. In practice, the inductive biases we choose are unfortunately often more informed by computational considerations than by modeling choices reflecting properties of the latent function.

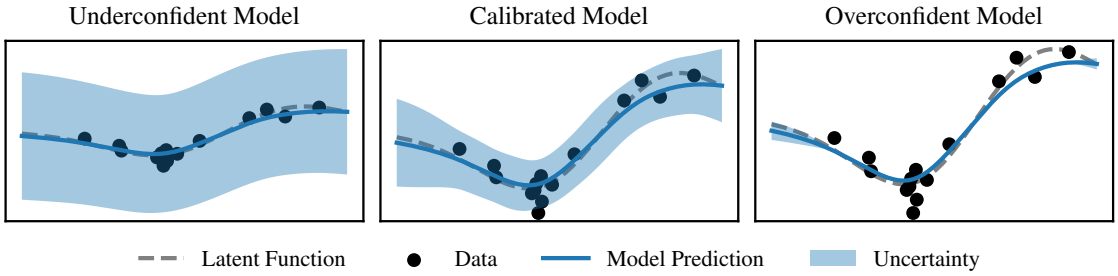


Figure 1.3: *Uncertainty quantification.* Models trained on the same dataset with identical prediction, but qualitatively different uncertainty quantification. A model can be underconfident in its predictions, in other words overly cautious, or overconfident, suggesting it is more accurate than it is. An ideal model is calibrated and reflects uncertainty coming from all sources, the model choice, the data and the amount of computation performed.

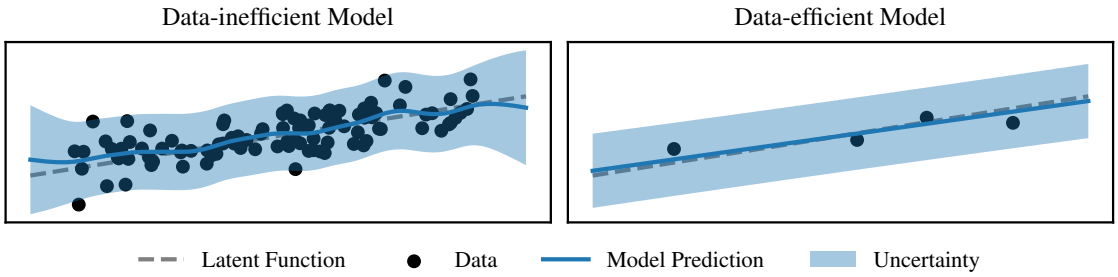


Figure 1.4: *Data efficiency.* Two models with different inductive biases applied to the same regression problem. A flexible model can learn more complicated latent functions but in the case of a simple latent function needs considerably more data. In contrast, a model that explicitly encodes the fact that the latent function is linear can generalize well even when trained on just a few data points.

Societal and Behavioral Aspects Machine learning models always exist in a societal context and therefore their use and behavior must be subject to societal and ethical norms. As a problematic example, take the case where a model might be able to improve its predictions according to a given loss function at the cost of discriminating against a subpopulation [36]. Further, a model might output discriminatory or dehumanizing content, which reflects social biases [37]. To guard against unintended consequences, predictive models must be carefully evaluated in their behavior and their application (e.g. [38, 39]). Additionally, how humans interact with a model is also an important consideration. If a machine learning model is hard to use, its users distrust the model output or experience it as unreliable, then it will not be adopted. In contrast, if a model is easy to misuse, then its widespread adoption may result in uncontrollable harm.

1.2 Bayesian Machine Learning

In this thesis we take a probabilistic perspective on learning, meaning we describe our assumptions about the problem and how the data was generated using probability theory. In contrast to the empirical risk minimization framework, there is no single best model $f_{\boldsymbol{w}_*}$ but a range of hypotheses $f_{\boldsymbol{w}}$ that are more or less likely to model the latent function well. Using a probabilistic approach we do not only obtain a point estimate $f_{\boldsymbol{w}}(\boldsymbol{x}_\diamond)$ as a prediction, but a distribution over possible values representing the certainty of the model. This is a useful way to approach questions of how can we effectively collect new data, how much information can we transfer to a new problem, how much should we trust the predictions of our model and how useful are they for a downstream task. Unfortunately, it can also result in challenging problems of having to approximate the formally correct way to perform inference and make predictions. Resolving this difficulty for regression will be one of the main themes of this work.

1.2.1 Bayes' Theorem

From a probabilistic perspective the parameters $\boldsymbol{w}$ of a model $f_{\boldsymbol{w}}$ are modeled as a random variable with an associated *prior* distribution $p(\boldsymbol{w})$, which captures our belief about which parameters are likely to generalize well, before seeing any data. The prior thus describes the set of possible hypotheses and their probability. We then define a *likelihood* $p(\boldsymbol{y} \mid \boldsymbol{w})$, which describes how we assume the training data was generated given the parameters. As a function of the parameters and for fixed data it measures the compatibility of the observed data with a given hypothesis for the parameters. We then update our belief when observing data according to *Bayes' theorem*:

$$\underbrace{p(\boldsymbol{w} \mid \boldsymbol{y})}_{\text{posterior}} = \frac{\overbrace{p(\boldsymbol{y} \mid \boldsymbol{w})}^{\text{likelihood}}\overbrace{p(\boldsymbol{w})}^{\text{prior}}}{\underbrace{p(\boldsymbol{y})}_{\text{evidence}}} = \frac{p(\boldsymbol{y} \mid \boldsymbol{w})p(\boldsymbol{w})}{\int p(\boldsymbol{y} \mid \boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w}} \tag{1.3}$$

This results in a *posterior* distribution, which captures our updated belief about the parameters after observing the data. For an illustration of Bayes' theorem see Figure 1.5. The *evidence* $p(\boldsymbol{y}) = \int p(\boldsymbol{y} \mid \boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w}$ formally serves as a normalization constant, but can be interpreted as how likely the data was generated from the assumed model when marginalizing out the parameters. This interpretation is particularly useful for (Bayesian) model selection.

Prediction Given a posterior over the model parameters, we can predict at a new test input $\boldsymbol{x}_\diamond$ by averaging over all possible hypotheses for the parameters weighted according to the posterior $p(\boldsymbol{w} \mid \boldsymbol{y})$. This results in the *predictive distribution*

$$p(y_\diamond \mid \boldsymbol{y}) = \int p(y_\diamond \mid \boldsymbol{w})p(\boldsymbol{w} \mid \boldsymbol{y})d\boldsymbol{w} \tag{1.4}$$

characterizing our belief about a new data point $y_\diamond$ given the training data $\boldsymbol{y} = \begin{pmatrix} y_1 & \dots & y_n \end{pmatrix}$.

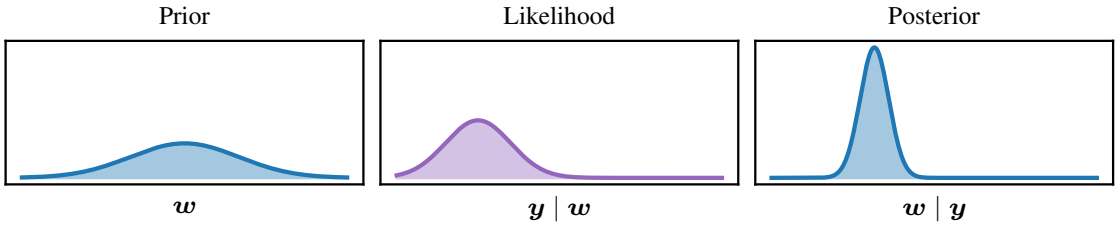


Figure 1.5: *Bayes' theorem.* A Gaussian prior, reflecting the belief over the parameters of the model, is conditioned on data, described by a Gaussian likelihood, leading to a Gaussian posterior over the parameters. Conceptually, the prior defines our assumptions about the parameters, while the likelihood models the data-generating process assuming the parameters were known. The posterior then describes our updated belief about the parameters after observing data.

One can make different arguments why Bayes' theorem is the "right" way to update a belief when faced with new information, either following an axiomatic approach [40] or a decision-theoretic one [41]. However, from a purely applied perspective being able to quantify uncertainty in a prediction at all, strictly adds new functionality to a model. For example, whether to make a decision based on the available information or whether to collect more data. Bayes' theorem serves as a recipe for how to quantify uncertainty given a set of assumptions.

1.2.2 Model Selection

So far we've treated the model f selected from a set of possible models $\mathbb{M}$ as given. However, if we make the dependence on a given model class explicit in Bayes' theorem, we can maximize the evidence

$$p(\boldsymbol{y}) = p(\boldsymbol{y} \mid f) = \int p(\boldsymbol{y} \mid \boldsymbol{w}, f) p(\boldsymbol{w} \mid f) d\boldsymbol{w} \tag{1.5}$$

over the set of models $f \in \mathbb{M}$ in question to perform *model selection* using the training data. The evidence (1.5) is also often referred to as the *marginal likelihood* since we can interpret it as a likelihood function over the set of possible models where we marginalize out the parameters $\boldsymbol{w}$. A popular alternative to selecting a single model via evidence maximization is to use an ensemble of models which averages the predictions of individual models weighed according to

$$p(f \mid \boldsymbol{y}) \propto p(\boldsymbol{y} \mid f) p(f),$$

where $p(f)$ describes our prior belief over the set of possible models.

1.2.3 Connection to Empirical Risk Minimization

So far it is not clear how the probabilistic viewpoint connects to the empirical risk minimization framework outlined earlier in Section 1.2.3 and whether potentially a Bayesian machine learning model trades off (empirical) risk for the ability to quantify uncertainty. As it turns out, the two perspectives are intrinsically linked.

As above, assume a probabilistic model for i.i.d. data $\boldsymbol{y} = \begin{pmatrix} y_1 & \dots & y_n \end{pmatrix}$ and parameters $\boldsymbol{w}$ given by the joint distribution

$$p(\boldsymbol{y}, \boldsymbol{w}) = \prod_{j=1}^{n} p(y_j \mid \boldsymbol{w}) p(\boldsymbol{w}).$$

If in the empirical risk minimization problem (1.2), we choose the loss function proportional to the log-likelihood, such that

$$\ell(y_j, f_{\boldsymbol{w}}(\boldsymbol{x}_j)) \propto -\log p(y_j \mid \boldsymbol{w}) \tag{1.6}$$

and the regularizer proportional to the log-prior such that $R(\boldsymbol{w}) \propto -\log p(\boldsymbol{w})$, then the solution of the empirical risk minimization problem is the *maximum a-posteriori* (MAP) estimate for the parameters of the model, since

$$\begin{aligned}
\boldsymbol{w}_* &= \arg\min_{\boldsymbol{w}} \bar{\ell}(\boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \sum_{j=1}^{n} \ell(y_j, f_{\boldsymbol{w}}(\boldsymbol{x}_j)) + R(\boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} -\sum_{j=1}^{n} \log p(y_j \mid \boldsymbol{w}) - \log p(\boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \log \left(\prod_{j=1}^{n} p(y_j \mid \boldsymbol{w}) p(\boldsymbol{w}) \right) \\
&= \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{y}, \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} p(\boldsymbol{w} \mid \boldsymbol{y}).
\end{aligned}$$

This means that the mode of the posterior over the parameters is a solution to the empirical risk minimization problem derived from the probabilistic model. From the point of view of empirical risk minimization, if we can derive a probabilistic model from a given loss function and regularizer, this readily lets us interpret the implicit assumptions made about how the data was generated and which parameters we implicitly favor.

Example 1.3 (Bayesian linear regression)
Consider a linear regression model as in Example 1.1. If we assume a Gaussian prior $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \lambda^2 \boldsymbol{I})$ over the parameters and a Gaussian likelihood $y_j \mid \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\phi}(\boldsymbol{x}_j)^\intercal \boldsymbol{w}, \sigma^2)$, then we obtain an equivalent empirical risk minimization problem of the form:

$$\begin{aligned}
\boldsymbol{w}_* &= \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w} \mid \boldsymbol{y}) \\
&= \arg\min_{\boldsymbol{w}} -\sum_{j=1}^{n} \log \mathcal{N}(y_j; \boldsymbol{\phi}(\boldsymbol{x}_j)^\intercal \boldsymbol{w}, \sigma^2) - \log \mathcal{N}(\boldsymbol{w}; \boldsymbol{0}, \lambda^2 \boldsymbol{I})
\end{aligned}$$

$$= \arg\min_{\boldsymbol{w}} \frac{1}{2\sigma^2} \sum_{j=1}^{n} (y_j - \boldsymbol{\phi}(\boldsymbol{x}_j)^\intercal \boldsymbol{w})^2 + \frac{1}{2\lambda^2} \|\boldsymbol{w}\|_2^2$$

$$= \arg\min_{\boldsymbol{w}} \sum_{j=1}^{n} (y_j - \boldsymbol{\phi}(\boldsymbol{x}_j)^\intercal \boldsymbol{w})^2 + \frac{\sigma^2}{\lambda^2} \|\boldsymbol{w}\|_2^2$$

This shows that for Bayesian linear regression, a Gaussian likelihood and prior corresponds to the square loss and L_2 regularization.

One can extend the connection between empirical risk minimization and the MAP estimate from the posterior mode to the entire posterior, by considering a generalized form of the empirical risk minimization problem known as the variational formulation as given by Zellner [42]. Instead of optimizing over parameters $\boldsymbol{w}$, one optimizes over candidate distributions $q(\boldsymbol{w})$, such that

$$q_*(\boldsymbol{w}) = \arg\min_{q(\boldsymbol{w})} \mathbb{E}_q \left(\sum_{j=1}^{n} \ell(y_j, f_{\boldsymbol{w}}(\boldsymbol{x}_j)) \right) + d_{\mathrm{KL}}(q(\boldsymbol{w}) \parallel p(\boldsymbol{w})). \tag{1.7}$$

Restricting the family of distributions q to Dirac delta distributions recovers the standard formulation of the empirical risk minimization problem (1.2) with the regularizer $R(\boldsymbol{w}) \propto -\log p(\boldsymbol{w})$. Choosing the loss proportional to the log-likelihood as in (1.6), the minimizing distribution $q_*(\boldsymbol{w})$ in (1.7) is the posterior distribution as given by Bayes' theorem [42, 43], since in that case

$$q_*(\boldsymbol{w}) = \arg\min_{q(\boldsymbol{w})} d_{\mathrm{KL}}(q(\boldsymbol{w}) \parallel p(\boldsymbol{w} \mid \boldsymbol{y}))$$

and the KL divergence is minimized when its arguments are identical.

1.3 Gaussian Processes

So far we've assumed a regression model $f_{\boldsymbol{w}}$, which is explicitly parametrized through a finite set of parameters $\boldsymbol{w}$. We do not have to restrict ourselves to parametrized functions, instead, we can directly learn in a specified function space using Gaussian processes. Intuitively, one can interpret a Gaussian process as a distribution over the functions in the hypothesis space.

1.3.1 From Parameter Space to Function Space

Consider a Bayesian linear model $f_{\boldsymbol{w}} : \mathbb{R}^d \to \mathbb{R}$, where

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\intercal \boldsymbol{w}$$

with a feature map $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^p$ and a Gaussian prior over the weights $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{S})$. The component functions $\phi_j(\cdot)$ of the feature map define basis functions, which form a linear

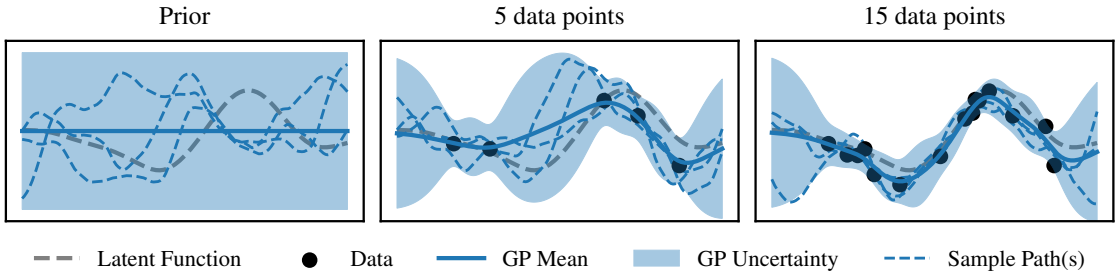


Figure 1.6: *Gaussian process conditioned on data.* A Gaussian process prior can be intuitively interpreted as defining a distribution over functions in a hypothesis class defined by the kernel. The sample paths of a Gaussian process are representative of the latent function one is trying to learn.[2] As a Gaussian process is conditioned on data, the posterior mean more closely resembles the latent function, assuming it lies in the hypothesis space, and its uncertainty contracts.

combination with weights given by the parameters w_j. From this perspective, the prior over the weights directly induces a prior over the functions in the hypothesis space, since by the properties of Gaussians, it holds for every $\boldsymbol{x} \in \mathbb{X}$ that

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\intercal \boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\phi}(\boldsymbol{x})^\intercal \boldsymbol{S} \boldsymbol{\phi}(\boldsymbol{x})).$$

This model is a special case of a Gaussian process, known as a parametric Gaussian process, since we explicitly parametrize the hypothesis space $\operatorname{span}(\phi_j(\cdot))$ in which we search for the latent function. However, we do not have to restrict ourselves to finite-dimensional function spaces. Instead, we can specify a covariance function or kernel, allowing us to use "infinitely many features" implicitly.

1.3.2 Gaussian Process Inference

A *Gaussian process* (GP)

$$\mathrm{f} \sim \mathcal{GP}(\mu, k)$$

is a stochastic process with *mean function* $\mu : \mathbb{R}^d \to \mathbb{R}$ and *covariance function* or *kernel* $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that any collection of function values

$$\mathbf{f} = (\mathrm{f}(\boldsymbol{x}_1), \ldots, \mathrm{f}(\boldsymbol{x}_n))^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$$

is jointly Gaussian with mean vector $\boldsymbol{\mu}_j = \mu(\boldsymbol{x}_j)$ and covariance matrix $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The mean function $\mu(\cdot)$ represents the best a priori guess of the latent function to be modeled, while the kernel $k(\cdot, \cdot)$ specifies how correlated function values are at different input points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, as well as, how confident we are in our belief about the latent function at a given input point.

[2] Technically, the sample paths of a Gaussian process are elements of a larger space than the reproducing kernel Hilbert space, in which the latent function is sought. See Kanagawa et al. [44] for a detailed discussion.

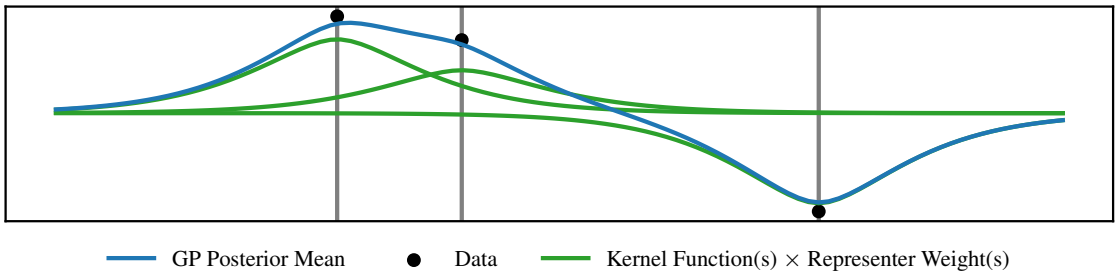


Figure 1.7: *Posterior mean of a Gaussian process as a linear combination.* The posterior mean of a Gaussian process can be viewed as a sum over kernel functions $k(\cdot, \boldsymbol{x}_j)$ placed at the training data points $\boldsymbol{x}_j$ scaled by the representer weights $(\boldsymbol{v}_*)_j$.

Inference Assume we are given a training data set $(\boldsymbol{X}, \boldsymbol{y})$ and the observations $\boldsymbol{y} = f(\boldsymbol{X}) + \boldsymbol{\varepsilon}$ are corrupted by i.i.d. Gaussian noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ such that

$$\boldsymbol{y} \mid f \sim \mathcal{N}(f(\boldsymbol{X}), \sigma^2 \boldsymbol{I}).$$

Then the posterior is again a Gaussian process $\mathcal{GP}(\mu_*, k_*)$ with mean and covariance functions

$$\begin{aligned} \mu_*(\cdot) &= \mu(\cdot) + k(\cdot, \boldsymbol{X}) \hat{\boldsymbol{K}}^{-1} (\boldsymbol{y} - \boldsymbol{\mu}) \\ k_*(\cdot, \cdot) &= k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \hat{\boldsymbol{K}}^{-1} k(\boldsymbol{X}, \cdot) \end{aligned}$$

where $\hat{\boldsymbol{K}} = \boldsymbol{K} + \sigma^2 \boldsymbol{I}$.

Representer Weights The posterior mean of a Gaussian process can be interpreted as a linear combination of kernel functions placed at training data points $\boldsymbol{x}_j$ weighted by the so-called *representer weights*

$$\boldsymbol{v}_* = \hat{\boldsymbol{K}}^{-1} (\boldsymbol{y} - \boldsymbol{\mu}),$$

since $\mu_*(\cdot) = \mu(\cdot) + \sum_{j=1}^{n} k(\cdot, \boldsymbol{x}_j)(\boldsymbol{v}_*)_j$. This is illustrated in Figure 1.7.

1.3.3 Modeling with Kernels

Kernels provide a flexible and expressive language to encode properties of the latent function and in turn inductive biases into a Gaussian process model. This is important since encoding (approximately) known characteristics of the latent function allows us to learn more efficiently and generalize better away from the training data. Encoding prior knowledge via a function space prior can be simpler and more direct than placing a prior over the parameters of an explicitly parametrized model. Especially, when the model is a non-linear function of the parameters, as is the case for example in Bayesian deep learning.

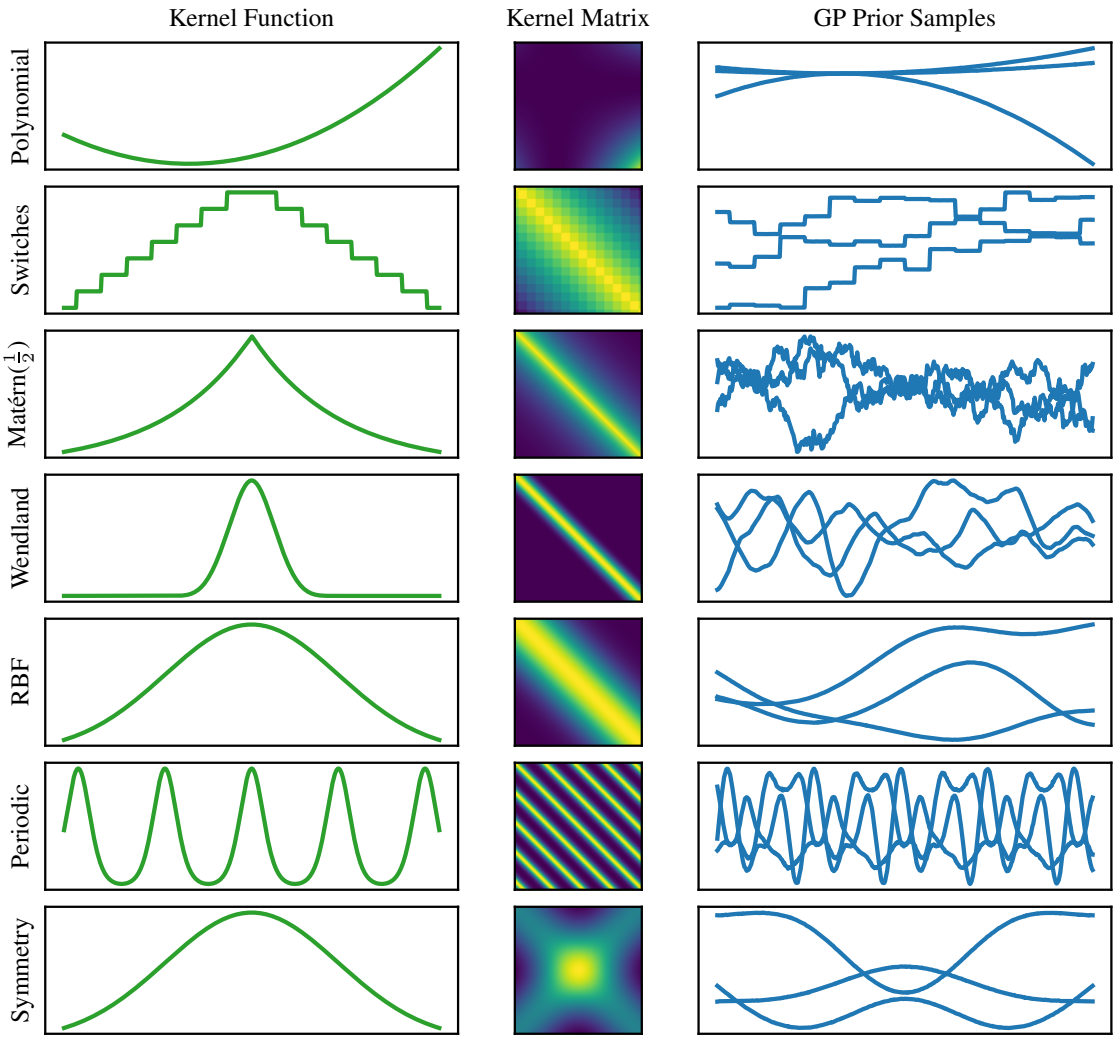


Figure 1.8: *Modeling with kernels.* Kernel functions $k(\cdot, \mathbf{0})$, kernel matrices $k(\boldsymbol{X}, \boldsymbol{X})$ and corresponding samples drawn from a Gaussian process prior $\mathcal{GP}(0, k)$. Choosing a kernel allows one to flexibly encode prior knowledge about the latent function into a Gaussian process model, for example, known properties like smoothness or symmetry.

Function Properties via Kernel Choice To demonstrate how choosing the kernel of a Gaussian process defines properties of functions in the hypothesis class, we give examples of some common properties, which can be encoded via a kernel below. These are also illustrated in Figure 1.8.

- *Parametric:* If the latent function is a linear combination of known feature functions $\phi_j : \mathbb{R}^d \to \mathbb{R}$, we can define a kernel $k(\cdot, \cdot) = \langle \boldsymbol{\phi}(\cdot), \boldsymbol{\phi}(\cdot) \rangle$. The "Switches" function [45] is an example of this where the features are step functions with different step locations.
- *Smoothness:* Often the differentiability of the latent function is known, for example when solving PDEs [46]. Kernels like the Matérn(ν) [47] or compactly supported Wendland-type kernel [48] give control over the smoothness of the functions in the hypothesis space.
- *Symmetry:* Explicitly enforcing a known symmetry can often be a very useful inductive bias to generalize away from the data and to obtain data-efficient models. For example, a known reflectional symmetry can cut the number of training data required in half.

Kernel Algebra Kernels do not only explicitly model known properties of the latent function, but they can also be easily combined into new kernels via algebraic rules. This is useful if the latent function is known to be a composition of subprocesses, for example, a linear combination of periodic functions at different scales. We give a basic list of algebraic rules here:

- *Sum:* The sum of two kernels $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel.
- *Product:* The product of two kernels $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel.
- *Direct Sum:* The direct sum of two kernels $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}_1, \boldsymbol{x}_1') + k_2(\boldsymbol{x}_2, \boldsymbol{x}_2')$ is a kernel.
- *Tensor Product:* The tensor product of two kernels $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}_1, \boldsymbol{x}_1')k_2(\boldsymbol{x}_2, \boldsymbol{x}_2')$ is a kernel.

For a more detailed overview see Section 4.2.4 of Rasmussen and Williams [47].

1.3.4 Hyperparameter Optimization

We can perform Bayesian model selection as in Section 1.2.2 for Gaussian processes by optimizing the *marginal likelihood* with respect to the kernel hyperparameters. In other words, we choose the hyperparameters $\boldsymbol{\theta}$ as follows:

$$\begin{aligned}
\boldsymbol{\theta}_* &= \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{y} \mid \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \frac{1}{2} \Bigg(\underbrace{(\boldsymbol{y} - \boldsymbol{\mu})^\intercal \hat{\boldsymbol{K}}^{-1} (\boldsymbol{y} - \boldsymbol{\mu})}_{\text{quadratic loss}} + \underbrace{\log\det(\hat{\boldsymbol{K}})}_{\text{model complexity}} + n \log(2\pi) \Bigg)
\end{aligned}$$

The two model-dependent terms in the log marginal likelihood can be readily interpreted as a *quadratic loss* term, penalizing if the model does not fit the training data well, as well as a *model complexity* term, which increases in magnitude for kernels that define more flexible or

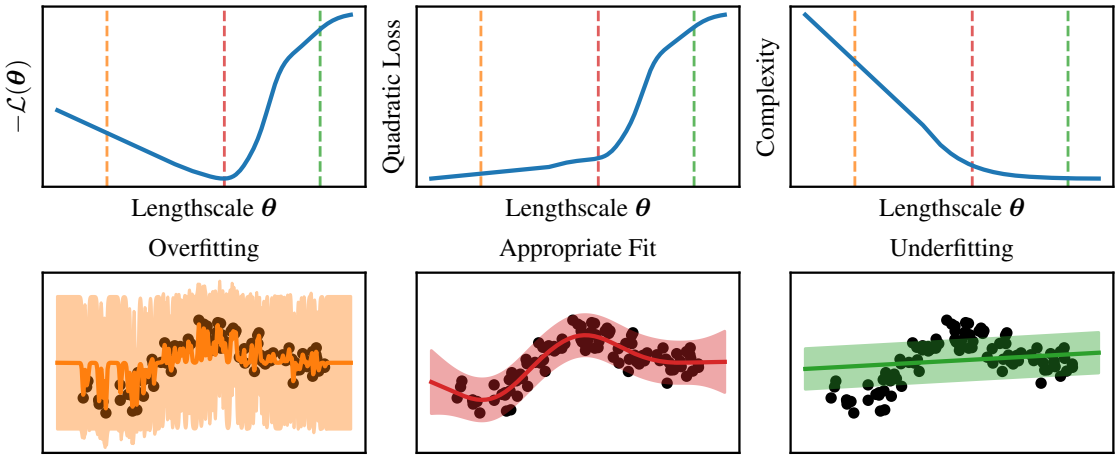


Figure 1.9: *Gaussian process hyperparameter optimization.* Top: Negative log-marginal likelihood $-\mathcal{L}(\boldsymbol{\theta})$ of a Gaussian process with a Matérn($\frac{5}{2}$) kernel as a function of the lengthscale, as well as the quadratic loss and model complexity terms. Bottom: Gaussian process posteriors for three different choices of lengthscale corresponding to a complex model overfitting the data (—), the optimal model minimizing the loss function (—), and a simple model with large bias (—).

complex models. The latter is a regularization term and a form of Occam's razor, promoting simpler hypotheses for the latent function over more complex ones. For an illustration of this decomposition see Figure 1.9. Choosing hyperparameters for Gaussian processes in this way is a very basic form of representation learning, where the representation is implicitly defined by the kernel and changes with the values of the hyperparameters.

1.3.5 Advantages

Gaussian processes have several advantages that make them attractive as a model class. In particular, they satisfy many of the desired properties of predictive models outlined in Section 1.1.3.

Expressive Modeling Gaussian processes can directly encode prior knowledge about the latent function via the kernel framework and therefore provide a powerful modeling language to enforce desirable inductive biases. This is particularly attractive for scientific machine learning, where we have well-established mechanistic models for physical phenomena. In that case, often properties of the latent function such as its smoothness or its symmetries are known to be true and therefore must be satisfied by our model, for example when learning the solution to a PDE in a data-driven manner. Finally, this modeling language is compositional via the rules of kernel algebra, so if the latent function can be decomposed, this can directly be reflected in the model.

Tractable Inference Under the assumption of Gaussian observation noise, the posterior of a Gaussian process can be computed in closed form. It reduces to basic, well-studied operations of linear algebra, specifically the solution of a linear system. Even if observations are made via a bounded linear operator, inference is still tractable [46, Thm. 1]. This stands in contrast to other

models, such as neural networks, where inference requires solving a challenging optimization problem.

Uncertainty Quantification Gaussian processes inherently provide not only a point estimate but also structured uncertainty for a prediction. Therefore one can reason about how much a given prediction should be relied upon or whether additional data needs to be collected. Importantly, uncertainty can be used to inform decisions for downstream applications. For example, consider a Gaussian process model for the temperature in a simulated electronic component. If the temperature inside the component is likely to exceed a critical temperature threshold based on the uncertainty of the model, we need to weigh redesigning the component before going into production against the risk of its failure.

1.3.6 Limitations

While Gaussian processes have been widely used in scientific applications such as geospatial statistics and astrophysics, their use in machine learning is less broad than one might expect from their theoretical advantages as a model class. This is mainly due to the following reasons.

Representation Learning The success of deep learning in domains such as computer vision and language modeling has shown the importance of representation learning for challenging machine learning applications. It is difficult to achieve similar performance in computer vision to a deep learning model using Gaussian processes. One of the main reasons is that when specifying a prior for Gaussian processes, the representation of the data (up to kernel hyperparameters) is fixed. This can be very beneficial if the corresponding inductive bias matches the latent function to be learned. But for data in pixel space, it is difficult to define a kernel that captures the key features of the data a priori. This is one of the reasons deep learning originally became popular because it removed the need to design hand-crafted features. In a way, many priors induced by the kernel are too restrictive to allow for good generalization. Recently, methods have emerged which can tackle this challenge either by using convolutional kernels directly [49] or by learning a hierarchical representation via a neural network and then defining a neural kernel [50].

Computational Cost Possibly the biggest challenge for the wider adoption of Gaussian processes is their computational cost. Gaussian processes scale poorly with the number of training data points both in time and memory. While inference is mathematically tractable and well-understood, the required linear solves to compute the posterior have cubic time complexity $\mathcal{O}(n^3)$ in the number of data points n and storing the kernel matrix requires $\mathcal{O}(n^2)$ memory. This quickly becomes prohibitive even on modern hardware as Figure 1.10 illustrates. For example, storing a dense kernel matrix in memory for $n = 100{,}000$ training data points requires roughly 80GB of RAM. For hyperparameter optimization, we not only have to perform a kernel matrix solve and compute the log-determinant but do so *repeatedly*. Even if there is sufficient low-latency memory available to store the kernel matrix, for just a few thousand data points naive hyperparameter optimization can already take hours. For modern machine learning applications with datasets that easily exceed $n \geq 100{,}000$ training examples, this is not feasible.

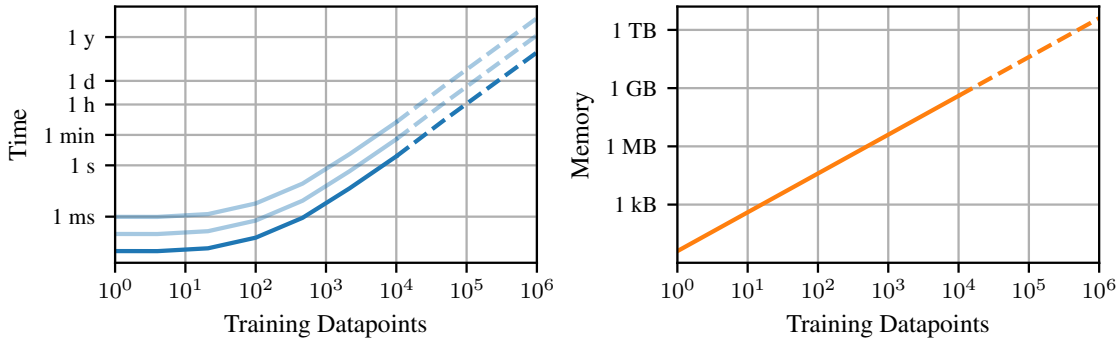


Figure 1.10: *Computational cost of Gaussian processes.* Gaussian process hyperparameter optimization requires repeated solves with, and determinants of, the kernel matrix. This is often implemented via a Cholesky decomposition of the (damped) kernel matrix, which scales cubically in time and quadratically in memory with the size of the training dataset.

1.4 Limited Resources and the Role of Computation

The chasm between the model we would like to use and the computation we actually perform is arguably one of the most ignored but influential parts of the learning framework. Computational resources are always limited in practice which necessitates the use of approximations and in turn, significantly impacts the accuracy and reliability of a model. Often the available resources even dictate the model choice itself, rather than the assumptions about how the data was generated.

1.4.1 Inference in Theory and Practice

This is an all too familiar problem for any machine learning practitioner. Even if in theory the mathematical problem to be solved for inference has an analytic solution, in practice this solution cannot be found with limited computational resources. This introduces a host of implementation choices, all of which impact the resulting predictions of the approximated model. For example, consider a large-scale non-convex empirical risk minimization problem. Even though in theory a unique solution exists, approximating it in practice requires choosing an optimizer, its learning rate schedule, the batch size and how many steps to take. As another example take generic Bayesian inference. Computing the posterior requires evaluating a high-dimensional integral. This is a notoriously hard problem, which is why one typically resorts to a Monte-Carlo approximation. However, the chosen sampling method, the burn-in period and the number of samples all impact the predictions made with the underlying model in unexpected ways.

To illustrate this impact on reliability, consider Figure 1.11 depicting a toy Gaussian process inference problem and three commonly used approximation methods. Each choice of approximation method introduces its own biases into the prediction and strongly affects uncertainty quantification. For example, both a Nyström approximation and SVGP are more certain about their predictions in some regions of the input space where the model has seen no data compared to other regions where it has.

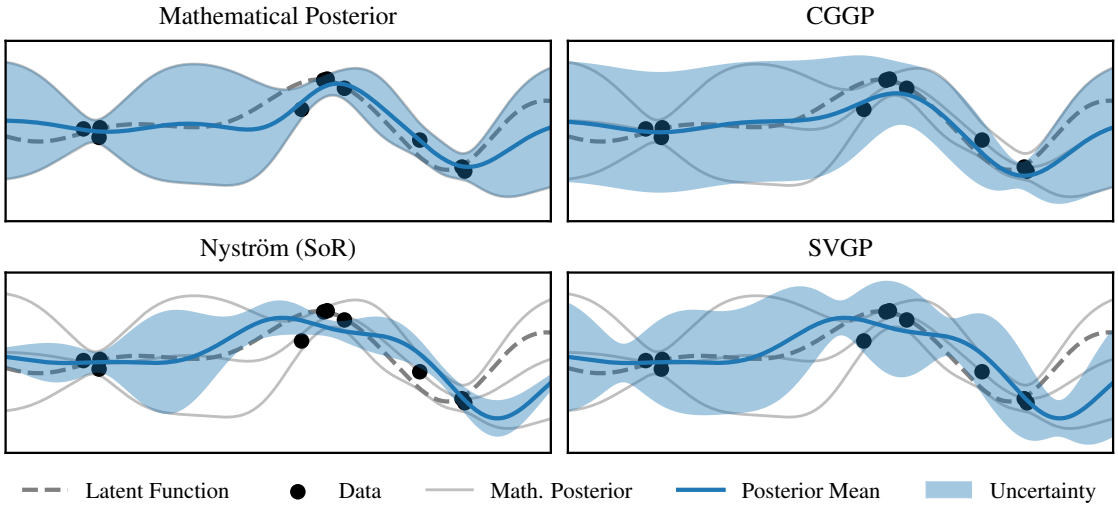


Figure 1.11: *Impact of approximation on the reliability of a model.* Mathematical, i.e. exact, Gaussian process posterior and different approximations (CGGP, Nyström and SVGP) on a toy dataset. While the shown approximations differ in their computational cost, they all introduce different pathologies, in particular regarding uncertainty quantification. This illustrates the impact limited computational resources have on the reliability of a model.

The unaccounted-for impact of numerical approximations creates a *disconnect between theory and practice* and crucially affects the reliability of a model. Even in the ideal case, where mathematical guarantees on the error of a numerical method are known, the prediction of the model is still *not* informed by this approximation error. This raises the important question: How do we ensure that, given finite computational resources, we still obtain reliable predictions from a model? This problem is a key motivation for the field of probabilistic numerics.

1.4.2 Probabilistic Numerics: Computation as Machine Learning

Probabilistic numerics [51–54] interprets problems in numerical analysis as probabilistic inference tasks. As a framework it is based on two core insights [54]. First, *the true solution to a numerical problem is fundamentally uncertain*, either due to limited computational resources, or inherent stochasticity. And, second, in analogy to machine learning agents that learn from data and interact with their environment, *numerical algorithms are learning agents* that take actions, collect observations of the numerical problem they are solving and make predictions.

As an example problem consider the solution of a linear system of the form

$$\boldsymbol{A}\boldsymbol{x}_* = \boldsymbol{b}.$$

The quantity of interest when solving this numerical problem is the solution $\boldsymbol{x}_*$ or when solving multiple problems the inverse $\boldsymbol{A}^{-1}$. For any problem outside of a toy problem, given the matrix $\boldsymbol{A}$ and the right-hand-side $\boldsymbol{b}$, we are fundamentally uncertain about the solution $\boldsymbol{x}_*$ before performing

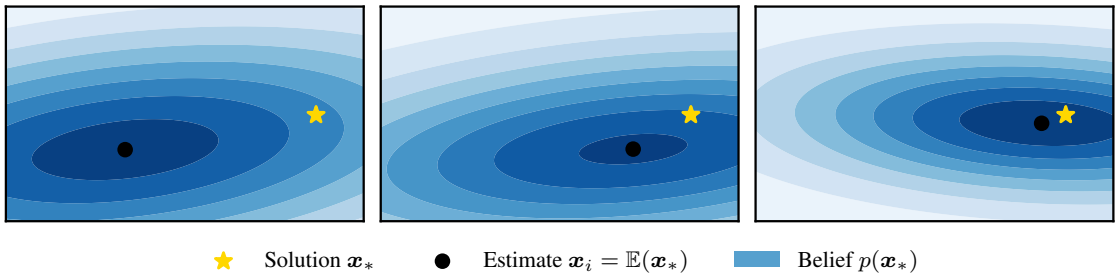


Figure 1.12: *Probabilistic belief about the solution of a linear system.* Probabilistic numerical methods quantify the uncertainty about the solution of a numerical problem via a probability measure. The plot shows the estimated solution of a linear system with $n = 20$ projected down to a random two-dimensional subspace. As the solver collects more matrix-vector product observations, the belief contracts onto the true solution of the linear system.

any computation. Numerical methods for the solution of a linear system then perform operations on $\boldsymbol{A}$ and $\boldsymbol{b}$ to improve their estimate of $\boldsymbol{x}_*$. Even in exact arithmetic, prior to convergence the true solution $\boldsymbol{x}_*$ is only known approximately.

To reason about this uncertainty inherent in computation, probabilistic numerical methods use the tools of probability theory to quantify their belief about the quantity of interest. Instead of a point estimate for the true solution of a numerical problem, such as the solution of a linear system, they return a probability measure. Quantifying the approximation error of a numerical method in this way promises multiple benefits, such as a more structured description of error than a worst-case bound, the seamless integration of stochastic computations and the propagation of approximation error to a downstream task. For example when performing approximate inference in a probabilistic machine learning model, if the approximation error of the numerical method is quantified probabilistically, it can ideally be propagated to the posterior.

The operations a numerical method performs to approximate the solution of a numerical problem are interpreted in probabilistic numerics as the actions of a learning agent, that updates its belief. When solving a linear system, the actions are vectors $\boldsymbol{s}_i$ that are multiplied with the matrix $\boldsymbol{A}$ or the current residual $\boldsymbol{r}_{i-1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{i-1} = \boldsymbol{A}(\boldsymbol{x}_* - \boldsymbol{x}_{i-1})$. The solver then observes the result of this operation and updates its belief about the solution $\boldsymbol{x}_*$ accordingly. Such a belief update for the solution of a linear system is illustrated in Figure 1.12. Describing a numerical method as a learning agent allows the design of policies, which define the sequence of actions an algorithm takes, that automatically adapt to the problem at hand and the current belief about the quantity of interest. For example, based on a decision-theoretic framework, which guides the exploration-exploitation tradeoff of the algorithm. It also enables the design of problem-specific stopping criteria that are based on the belief of the agent about its quantity of interest and the goal it is trying to achieve. This way computation can be adaptively saved compared to the case where the stopping criteria are chosen generically for a large problem class as is typically the case for classic numerical algorithms.

1.5 Thesis Contributions

This thesis makes significant contributions to the field of probabilistic numerical linear algebra both in theory and in practice. In particular, it develops the theoretical understanding of probabilistic linear solvers from first principles (Chapter 2); it shifts the goalposts of what can be expected from Gaussian process approximations by showing that it is possible to account for the impact of numerical approximation on prediction (Chapter 3); and it demonstrates that large-scale Gaussian process hyperparameter optimization can be significantly accelerated by exploiting structural prior knowledge via preconditioning (Chapter 4). From a practical perspective, this thesis has led to the first comprehensive software package for probabilistic numerical methods, ProbNum [55], and it has defined the default large-scale training routine in GPyTorch [3].

Chapter 2: Probabilistic Linear Solvers for Machine Learning Arguably, one of the most fundamental numerical operations in machine learning and scientific computation at large is the solution of linear systems. The linear systems that arise in machine learning pose specific challenges due to their scale, characteristic structure and often stochastic nature. To meet these challenges, in Chapter 2 we propose a unifying class of probabilistic linear solvers, which jointly infer the matrix, its inverse and the solution from matrix-vector product observations. Such solvers interpret the numerical task of solving a linear system as probabilistic inference. This allows them to leverage problem structure and propagate their approximation error upstream to a probabilistic model. This class of solvers emerges from a fundamental set of desiderata that constrains the space of possible algorithms and recovers the method of conjugate gradients as a special case. We demonstrate how to incorporate prior spectral information to calibrate uncertainty and experimentally showcase the potential of such solvers for machine learning. Our solver is implemented in a publicly available software package, called ProbNum [55].

ProbNum  `https://probnum.readthedocs.io`

Disclaimer 1.1
Chapter 2 is based on the peer-reviewed conference publication

[1] J. Wenger and P. Hennig. "Probabilistic Linear Solvers for Machine Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020

with the following co-author contributions:

	Ideas	Analysis	Code	Experiments	Writing
J. Wenger	50 %	90 %	100 %	100 %	90 %
P. Hennig	50 %	10 %	0 %	0 %	10 %

Chapter 3: Posterior and Computational Uncertainty in Gaussian Processes Gaussian processes are a principled and flexible probabilistic model class. However, they scale prohibitively with the size of the training data. In response, many approximation methods have been developed,

which inevitably introduce approximation error. This additional source of uncertainty, due to limited computation, is entirely ignored when using the approximate posterior. Therefore in practice, GP models are often as much about the approximation method as they are about the data. In Chapter 3, we develop a new class of methods that consistently estimates the combined uncertainty arising from *both* the finite number of data observed *and* the finite amount of computation expended. The most common GP approximations map to an instance in this class, such as methods based on the Cholesky factorization, conjugate gradients, and inducing points. For any method in this class, we prove (i) convergence of its posterior mean in the associated RKHS, (ii) decomposability of its combined posterior covariance into mathematical and computational covariances, and (iii) that the combined variance is a tight worst-case bound for the squared error between the method's posterior mean and the latent function. Finally, we empirically demonstrate the consequences of ignoring computational uncertainty and show how implicitly modeling it improves generalization performance on benchmark datasets. Our framework is implemented in a publicly available software package called IterGP [2].

https://itergp.readthedocs.io

Disclaimer 1.2
Chapter 3 is based on the peer-reviewed conference publication

[2] J. Wenger et al. "Posterior and Computational Uncertainty in Gaussian Processes". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022

with the following co-author contributions:

	Ideas	Analysis	Code	Experiments	Writing
J. Wenger	50 %	100 %	80 %	60 %	70 %
G. Pleiss	10 %	0 %	0 %	40 %	15 %
M. Pförtner	0 %	0 %	20 %	0 %	0 %
P. Hennig	20 %	0 %	0 %	0 %	0 %
J. P. Cunningham	20 %	0 %	0 %	0 %	15 %

Chapter 4: Preconditioning for Scalable GP Hyperparameter Optimization Gaussian process hyperparameter optimization requires linear solves with, and log-determinants of, large kernel matrices. Iterative numerical techniques are becoming popular to scale to larger datasets, relying on the conjugate gradient method (CG) for the linear solves and stochastic trace estimation for the log-determinant. In Chapter 4 we introduce new algorithmic and theoretical insights for preconditioning these computations. While preconditioning is well understood in the context of CG, we demonstrate that it can also accelerate convergence and reduce the variance of the estimates for the log-determinant and its derivative. We prove general probabilistic error bounds for the preconditioned computation of the log-determinant, log-marginal likelihood and its derivatives. Additionally, we derive specific rates for a range of kernel-preconditioner combinations, showing that up to exponential convergence can be achieved. Our theoretical results enable

provably efficient optimization of kernel hyperparameters, which we validate empirically on large-scale benchmark problems. There our approach accelerates training by up to an order of magnitude. Our methodology is implemented in GPyTorch [3] and defines its default large-scale training routine for Gaussian process models.

GPyTorch https://docs.gpytorch.ai

Disclaimer 1.3
Chapter 4 is based on the peer-reviewed conference publication

[4] J. Wenger et al. "Preconditioning for Scalable Gaussian Process Hyperparameter Optimization". In: *International Conference on Machine Learning (ICML)*. 2022

with the following co-author contributions:

	Ideas	Analysis	Code	Experiments	Writing
J. Wenger	50 %	80 %	20 %	20 %	65 %
G. Pleiss	20 %	10 %	50 %	30 %	10 %
J. P. Cunningham	5 %	0 %	0 %	0 %	5 %
P. Hennig	5 %	0 %	0 %	0 %	5 %
J. Gardner	20 %	10 %	30 %	50 %	15 %

Summary This thesis presents techniques in probabilistic numerical linear algebra to accelerate and scale Gaussian process models, while crucially accounting for the impact of approximation error. This is made possible by interpreting numerical methods in linear algebra as probabilistic learning algorithms and informing them about the structure of the underlying model. Our work enables the use of Gaussian process models in large-scale settings without compromising their ability to quantify uncertainty – a fundamental prerequisite for optimal decision-making. A complete list of publications (co-)authored during the doctoral qualification period is given below.

	Authors	Title	Venue	Year
[56]	Wenger, Kjellström, and Triebel	Non-Parametric Calibration for Classification	AISTATS	2020
[1]	Wenger and Hennig	Probabilistic Linear Solvers for Machine Learning	NeurIPS	2020
[55]	Wenger, Krämer, Pförtner et al.	ProbNum: Probabilistic Numerics in Python	arXiv	2021
[4]	Wenger, Pleiss, Hennig, Cunningham, and Gardner	Preconditioning for Scalable Gaussian Process Hyperparameter Optimization	ICML	2022
[2]	Wenger, Pleiss, Pförtner, Hennig, and Cunningham	Posterior and Computational Uncertainty in Gaussian Processes	NeurIPS	2022
[46]	Pförtner, Steinwart, Hennig, and Wenger	Physics-Informed Gaussian Process Regression Generalizes Linear PDE Solvers	arXiv	2022

Chapter

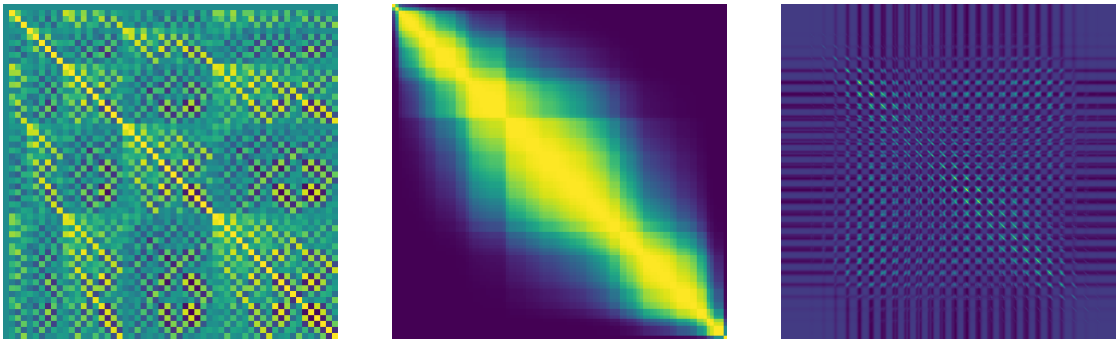
2

Probabilistic Linear Solvers for Machine Learning

Linear systems are the bedrock of virtually all numerical computation. Machine learning poses specific challenges for the solution of such systems due to their scale, characteristic structure, stochasticity and the central role of uncertainty quantification in machine learning. Unifying earlier work we propose a class of probabilistic linear solvers that jointly infer the matrix, its inverse and the solution from matrix-vector product observations. This class emerges from a fundamental set of desiderata that constrains the space of possible algorithms and recovers the method of conjugate gradients under certain conditions. We demonstrate how to incorporate prior spectral information in order to calibrate uncertainty and experimentally showcase the potential of such solvers for machine learning.

2.1 Introduction

Arguably one of the most fundamental problems in machine learning, statistics and scientific computation at large is the solution of linear systems of the form $\boldsymbol{A}\boldsymbol{x}_* = \boldsymbol{b}$, where $\boldsymbol{A} \in \mathbb{R}^{n\times n}_{\text{sym}}$



(a) Gram matrix $\boldsymbol{X}^\intercal \boldsymbol{X}$ (b) Kernel matrix $k(\boldsymbol{X}, \boldsymbol{X})$ (c) Hessian matrix

Figure 2.1: *Examples of structured matrices in machine learning.*

is a symmetric positive definite matrix [57–59]. Such matrices usually arise in the context of second-order or quadratic optimization problems and as Gram matrices. Some of the numerous application areas in machine learning and related fields are least-squares regression [60], kernel methods [61], Kalman filtering [62], Gaussian (process) inference [47], spectral graph theory [63], (linear) differential equations [64] and (stochastic) second-order optimization [65].

Linear systems in machine learning are typically large-scale, have characteristic structure arising from generative processes, and are subject to noise (see Figure 2.1). These distinctive features call for linear solvers that can explicitly make use of such structural information. While classic solvers are highly optimized for general problems, they lack key functionality for machine learning. In particular, they do not consider generative prior information about the matrix.

An important example are kernel Gram matrices, which exhibit specific sparsity structure and spectral properties, depending on the kernel choice and the generative process of the data. Exploiting such prior information is a prime application for probabilistic linear solvers, which aim to quantify computational uncertainty arising from limited computational resources. Another key challenge, which we will not yet address here, is posed by noisy matrix evaluations arising from data subsampling. Ultimately, linear algebra for machine learning should integrate all sources of uncertainty in a computational pipeline – aleatoric, epistemic and computational – into one coherent probabilistic framework.

Contribution This paper sets forth desiderata for probabilistic linear solvers which establish first principles for such methods. From these, we derive an algorithm incorporating prior information on the matrix $\boldsymbol{A}$ or its inverse $\boldsymbol{A}^{-1}$, which jointly estimates both via repeated application of $\boldsymbol{A}$. This results in posterior beliefs over the two operators and the solution which quantify computational uncertainty. Our approach unifies and extends earlier formulations and constitutes a new way of interpreting linear solvers. Further, we propose a prior covariance class that recovers the method of conjugate gradients as its posterior mean and uses prior spectral information for uncertainty calibration, one of the primary shortcomings of probabilistic linear solvers. We conclude by presenting simplified examples of promising applications of such solvers within machine learning.

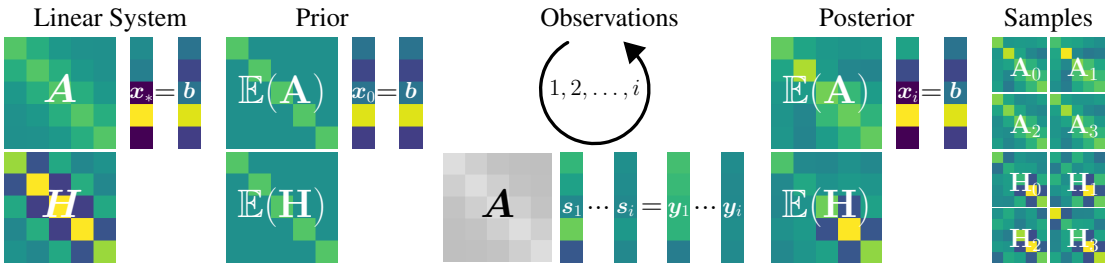


Figure 2.2: *Illustration of a probabilistic linear solver.* Given a prior for $\mathbf{A}$ or $\mathbf{H}$ modelling the linear operator $\boldsymbol{A}$ and its inverse $\boldsymbol{A}^{-1}$, posterior beliefs are inferred via observations $\boldsymbol{y}_i = \boldsymbol{A}\boldsymbol{s}_i$. This induces a distribution on the solution $\boldsymbol{x}_*$, quantifying computational uncertainty arising from finite computational resources. The plot shows $i = 3$ iterations of Algorithm 1 on a toy problem of dimension $n = 5$.

2.2 Probabilistic Linear Solvers

Let $\boldsymbol{A}\boldsymbol{x}_* = \boldsymbol{b}$ be a linear system with $\boldsymbol{A} \in \mathbb{R}^{n\times n}_{\text{sym}}$ positive definite and $\boldsymbol{b} \in \mathbb{R}^n$. *Probabilistic linear solvers* (PLS) [52, 66, 67] iteratively build a model for the linear operator $\boldsymbol{A}$, its inverse $\boldsymbol{H} = \boldsymbol{A}^{-1}$ or the solution $\boldsymbol{x}_*$, represented by random variables $\mathbf{A}, \mathbf{H}$ or $\mathbf{x}$. In the framework of probabilistic numerics [51, 53] such solvers can be seen as Bayesian agents performing *inference* via linear *observations* $\boldsymbol{Y} = \begin{pmatrix}\boldsymbol{y}_1 & \cdots & \boldsymbol{y}_i\end{pmatrix} \in \mathbb{R}^{n\times i}$ resulting from *actions* $\boldsymbol{S} = \begin{pmatrix}\boldsymbol{s}_1 & \cdots & \boldsymbol{s}_i\end{pmatrix} \in \mathbb{R}^{n\times i}$ given by an internal *policy* $\pi(\boldsymbol{s} \mid \mathbf{A}, \mathbf{H}, \mathbf{x}, \boldsymbol{A}, \boldsymbol{b})$. For a matrix-variate prior $p(\mathbf{A})$ or $p(\mathbf{H})$ encoding prior (generative) information, our solver computes posterior beliefs over the matrix, its inverse and the solution of the linear system. An illustration of a probabilistic linear solver is given in Figure 2.2.

Desiderata We begin by stipulating a fundamental set of desiderata for probabilistic linear solvers. To our knowledge, such a list has not been collated before. Connecting previously disjoint threads, the following presents a roadmap for the development of these methods. Probabilistic linear solvers modeling $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ must assume matrix-variate distributions which are expressive enough to capture structure and generative prior information either for $\boldsymbol{A}$ or its inverse. The distribution choice must also allow computationally efficient sampling and density evaluation. It should encode symmetry and positive definiteness and must be closed under positive linear combinations. Further, the two models for the system matrix or its inverse should be translatable into and consistent with each other. Actions $\boldsymbol{s}_i$ of a PLS should be model-based and induce a tractable distribution on linear observations $\boldsymbol{y}_i = \boldsymbol{A}\boldsymbol{s}_i$. Since probabilistic linear solvers are low-level procedures, their inference procedure must be computationally lightweight. Given (noise-corrupted) observations this requires tractable posteriors over $\mathbf{A}$, $\mathbf{H}$ and $\mathbf{x}$, which are calibrated in the sense that at convergence the true solution $\boldsymbol{x}_*$ represents a draw from the posterior $p(\mathbf{x} \mid \boldsymbol{Y}, \boldsymbol{S})$. Finally, such solvers need to allow preconditioning of the problem and ideally should return beliefs over non-linear properties of the system matrix extending the functionality of classic methods. These desiderata are summarized concisely in Table 2.1.

Table 2.1: *Desired properties of probabilistic linear solvers.* Symbols (✗, ∼, ✓) indicate which properties are encoded in Algorithm 1 and to what degree.

No.	Property	Formulation	
(1)	Distribution over matrices	$\mathbf{A} \sim \mathcal{D},\ p_{\mathcal{D}}(\mathbf{A})$	✓
(2)	Symmetry	$\mathbf{A} = \mathbf{A}^\intercal$ a.s.	✓
(3)	Positive definiteness	$\forall \boldsymbol{v} \neq 0 :\ \boldsymbol{v}^\intercal \mathbf{A} \boldsymbol{v} > 0$ a.s.	∼
(4)	Positive linear combination in same distribution family	$\forall \alpha_j > 0 :\ \sum_j \alpha_j \mathbf{A}_j \sim \mathcal{D}$	✓
(5)	Corresponding priors on the matrix and its inverse	$p(\mathbf{A}) \longleftrightarrow p(\mathbf{H})$	✓
(6)	Model-based policy	$\boldsymbol{s}_i \sim \pi(\boldsymbol{s} \mid \boldsymbol{A}, \boldsymbol{b}, \mathbf{A}, \mathbf{H}, \mathbf{x})$	✓
(7)	Matrix-vector product in tractable distribution family	$\mathbf{A}\boldsymbol{s} \sim \mathcal{D}'$	✓
(8)	Noisy observations	$p(\boldsymbol{Y} \mid \mathbf{A}, \boldsymbol{S}) = \mathcal{N}(\boldsymbol{Y}; \boldsymbol{AS}, \boldsymbol{\Lambda})$	✗
(9)	Tractable posterior	$p(\mathbf{A} \mid \boldsymbol{Y}, \boldsymbol{S})$ or $p(\mathbf{H} \mid \boldsymbol{Y}, \boldsymbol{S})$	✓
(10)	Calibrated uncertainty	$\boldsymbol{x}_* \sim \mathcal{N}(\mathbb{E}(\mathbf{x}), \mathrm{Cov}(\mathbf{x}))$	∼
(11)	Preconditioning	$\boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{x}_* = \boldsymbol{P}^{-1}\boldsymbol{b}$	✓
(12)	Distributions over non-linear derived quantities of $\boldsymbol{A}$	$\det(\mathbf{A}),\ \lambda(\mathbf{A}),\ \mathbf{A} = \mathbf{L}^\intercal\mathbf{L}, \ldots$	✗

2.2.1 Bayesian Inference Framework

Guided by these desiderata, we will now outline the inference framework for $\mathbf{A}, \mathbf{H}$ and $\mathbf{x}$ forming the base of the algorithm. The choice of matrix-variate prior distribution is limited by the desideratum that conditioning on linear observations $\boldsymbol{y}_i = \boldsymbol{A}\boldsymbol{s}_i$ must be tractable. This reduces the choice to stable distributions [68] and thus excludes candidates such as the Wishart, which has measure zero outside the cone of symmetric positive semi-definite matrices. For symmetric matrices, this essentially forces the use of the symmetric matrix-variate normal distribution, introduced in this context by Hennig [66]. Given $\boldsymbol{A}_0, \boldsymbol{W}_0^{\mathbf{A}} \in \mathbb{R}^{n \times n}_{\mathrm{sym}}$, assume a prior distribution

$$p(\mathbf{A}) = \mathcal{N}\big(\mathbf{A}; \boldsymbol{A}_0, \boldsymbol{W}_0^{\mathbf{A}} \otimes \boldsymbol{W}_0^{\mathbf{A}}\big),$$

where $\otimes$ denotes the symmetric Kronecker product [69].[1] The symmetric matrix-variate Gaussian induces a Gaussian distribution on linear observations. While it has non-zero measure only for symmetric matrices, its support is not the positive definite cone. However, positive definiteness can still be enforced post-hoc (see Proposition 2.1). We assume noise-free linear observations of the form $\boldsymbol{y}_i = \boldsymbol{A}\boldsymbol{s}_i$, leading to a Dirac likelihood

$$p(\boldsymbol{Y} \mid \mathbf{A}, \boldsymbol{S}) = \lim_{\varepsilon \to 0} \mathcal{N}\big(\boldsymbol{Y}; \boldsymbol{AS}, \varepsilon^2 \boldsymbol{I} \otimes \boldsymbol{I})\big) = \delta(\boldsymbol{Y} - \boldsymbol{AS}).$$

The posterior distribution follows from the properties of Gaussians [60] and has been investigated in detail in previous work [66, 67, 70]. It is given by $p(\mathbf{A} \mid \boldsymbol{S}, \boldsymbol{Y}) = \mathcal{N}(\mathbf{A}; \boldsymbol{A}_i, \boldsymbol{\Sigma}_i)$ with

$$\begin{aligned}
\boldsymbol{A}_i &= \boldsymbol{A}_0 + \boldsymbol{\Delta}_0^{\mathbf{A}} \boldsymbol{U}^\intercal + \boldsymbol{U}(\boldsymbol{\Delta}_0^{\mathbf{A}})^\intercal - \boldsymbol{U}\boldsymbol{S}^\intercal \boldsymbol{\Delta}_0^{\mathbf{A}} \boldsymbol{U}^\intercal \\
\boldsymbol{\Sigma}_i &= \boldsymbol{W}_0^{\mathbf{A}}(\boldsymbol{I}_n - \boldsymbol{S}\boldsymbol{U}^\intercal) \otimes \boldsymbol{W}_0^{\mathbf{A}}(\boldsymbol{I}_n - \boldsymbol{S}\boldsymbol{U}^\intercal)
\end{aligned}$$

[1] See Wenger and Hennig [1, Sec. S2, S3] for details on Kronecker-type products and matrix-variate normal distributions.

where $\boldsymbol{\Delta}_0^{\mathbf{A}} = \boldsymbol{Y} - \boldsymbol{A}_0\boldsymbol{S}$ and $\boldsymbol{U} = \boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S}(\boldsymbol{S}^\intercal\boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S})^{-1}$. We aim to construct a probabilistic model for the inverse $\boldsymbol{H} = \boldsymbol{A}^{-1}$ consistent with the model $\mathbf{A}$ as well. However, not even in the scalar case does the inverse of a Gaussian have finite mean. We ask instead what Gaussian model for $\mathbf{H}$ is as consistent as possible with our observational model for $\mathbf{A}$. For a prior of the form $p(\mathbf{H}) = \mathcal{N}\left(\mathbf{H}; \boldsymbol{H}_0, \boldsymbol{W}_0^{\mathbf{H}} \otimes \boldsymbol{W}_0^{\mathbf{H}}\right)$ and likelihood $p(\boldsymbol{S} \mid \mathbf{H}, \boldsymbol{Y}) = \delta(\boldsymbol{S} - \boldsymbol{H}\boldsymbol{Y})$, we analogously to the $\mathbf{A}$-model obtain a posterior distribution $p(\mathbf{H} \mid \boldsymbol{S}, \boldsymbol{Y}) = \mathcal{N}\left(\mathbf{H}; \boldsymbol{H}_i, \boldsymbol{\Sigma}_i^{\mathbf{H}}\right)$ with

$$\begin{aligned}\boldsymbol{H}_i &= \boldsymbol{H}_0 + \boldsymbol{\Delta}_0^{\mathbf{H}}(\boldsymbol{U}^{\mathbf{H}})^\intercal + \boldsymbol{U}^{\mathbf{H}}(\boldsymbol{\Delta}_0^{\mathbf{H}})^\intercal - \boldsymbol{U}^{\mathbf{H}}\boldsymbol{Y}^\intercal\boldsymbol{\Delta}_0^{\mathbf{H}}(\boldsymbol{U}^{\mathbf{H}})^\intercal \\ \boldsymbol{\Sigma}_i^{\mathbf{H}} &= \boldsymbol{W}_0^{\mathbf{H}}(\boldsymbol{I}_n - \boldsymbol{Y}(\boldsymbol{U}^{\mathbf{H}})^\intercal) \otimes \boldsymbol{W}_0^{\mathbf{H}}(\boldsymbol{I}_n - \boldsymbol{Y}(\boldsymbol{U}^{\mathbf{H}})^\intercal)\end{aligned}$$

where $\boldsymbol{\Delta}_0^{\mathbf{H}} = \boldsymbol{S} - \boldsymbol{H}_0\boldsymbol{Y}$ and $\boldsymbol{U}^{\mathbf{H}} = \boldsymbol{W}_0^{\mathbf{H}}\boldsymbol{Y}(\boldsymbol{Y}^\intercal\boldsymbol{W}_0^{\mathbf{H}}\boldsymbol{Y})^{-1}$. In Section 2.3 we will derive a covariance class, which establishes the correspondence between the two Gaussian viewpoints for the linear operator and its inverse and is consistent with our desiderata.

2.2.2 Algorithm

The above inference procedure leads to Algorithm 1. The degree to which the desiderata are encoded in our formulation of a PLS can be found in Table 2.1. We will now go into more detail about the policy, the choice of step size, stopping criteria and the implementation.

Algorithm 1: Probabilistic Linear Solver with Uncertainty Calibration

procedure ProbLinSolve($\boldsymbol{A}(\cdot), \boldsymbol{b}, \mathbf{A}, \mathbf{H}$) ▷ Prior for $\mathbf{A}$ or $\mathbf{H}$.
  $\boldsymbol{x}_0 \leftarrow \mathbb{E}(\mathbf{H})\,\boldsymbol{b}$ ▷ Initial guess for the solution.
  $\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$ ▷ Initial residual.
  while $\min(\sqrt{\mathrm{tr}(\mathrm{Cov}(\mathbf{x}))}, \|\boldsymbol{r}_{i-1}\|_2) > \max(\delta_{\mathrm{rtol}}\|\boldsymbol{b}\|_2, \delta_{\mathrm{atol}})$ **do** ▷ Stopping criterion.
    $\boldsymbol{s}_i \leftarrow \mathbb{E}(\mathbf{H})\,\boldsymbol{r}_{i-1}$ ▷ Compute action via policy.
    $\boldsymbol{y}_i \leftarrow \boldsymbol{A}\boldsymbol{s}_i$ ▷ Observation.
    $\alpha_i \leftarrow \frac{\boldsymbol{s}_i^\intercal\boldsymbol{r}_{i-1}}{\boldsymbol{s}_i^\intercal\boldsymbol{y}_i}$ ▷ Step size.
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_{i-1} + \alpha_i\boldsymbol{s}_i$ ▷ Update solution estimate.
    $\boldsymbol{r}_i \leftarrow \boldsymbol{r}_{i-1} - \alpha_i\boldsymbol{y}_i$ ▷ Update residual.
    $\mathbf{A} \leftarrow \mathrm{Infer}(\mathbf{A}, \boldsymbol{s}_i, \boldsymbol{y}_i)$ ▷ Compute posterior distributions.
    $\mathbf{H} \leftarrow \mathrm{Infer}(\mathbf{H}, \boldsymbol{s}_i, \boldsymbol{y}_i)$ ▷ (see Section 2.2.1)
    $\boldsymbol{\Phi}, \boldsymbol{\Psi} \leftarrow \mathrm{Calibrate}(\boldsymbol{S}, \boldsymbol{Y})$ ▷ Calibrate uncertainty.
  $\mathbf{x} \leftarrow \mathcal{N}(\boldsymbol{x}_i, \mathrm{Cov}(\mathbf{H}\boldsymbol{b}))$ ▷ Compute induced belief over solution.
  return $(\mathbf{x}, \mathbf{A}, \mathbf{H})$

Policy and Step Size In each iteration, our solver collects information about the linear operator $\boldsymbol{A}$ via actions $\boldsymbol{s}_i$ determined by the policy $\pi(\boldsymbol{s} \mid \mathbf{A}, \mathbf{H}, \mathbf{x}, \boldsymbol{A}, \boldsymbol{b})$. The next action

$$\boldsymbol{s}_i = \mathbb{E}(\mathbf{H})\,\boldsymbol{r}_{i-1} = \mathbb{E}(\mathbf{H})\,(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{i-1})$$

is chosen based on the current belief about the inverse and the current residual $\boldsymbol{r}_{i-1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{i-1}$. If $\mathbb{E}(\mathbf{H}) = \boldsymbol{A}^{-1}$, i.e. if the solver's estimate for the inverse equals the true inverse, then

Algorithm 1 converges in a single step since

$$\boldsymbol{x}_{i-1} + \boldsymbol{s}_i = \boldsymbol{x}_{i-1} + \mathbb{E}(\mathbf{H})\, \boldsymbol{r}_{i-1} = \boldsymbol{x}_{i-1} + \boldsymbol{A}^{-1}\boldsymbol{b} - \boldsymbol{x}_{i-1} = \boldsymbol{A}^{-1}\boldsymbol{b} = \boldsymbol{x}_*.$$

The step size minimizing the quadratic $q(\boldsymbol{x}_{i-1} + \alpha \boldsymbol{s}_i) = \frac{1}{2}(\boldsymbol{x}_{i-1} + \alpha \boldsymbol{s}_i)^\intercal \boldsymbol{A}(\boldsymbol{x}_{i-1} + \alpha \boldsymbol{s}_i) - \boldsymbol{b}^\intercal(\boldsymbol{x}_{i-1} + \alpha \boldsymbol{s}_i)$ along the action $\boldsymbol{s}_i$ is given by

$$\alpha_i = \arg\min_{\alpha} q(\boldsymbol{x}_{i-1} + \alpha \boldsymbol{s}_i) = \frac{\boldsymbol{s}_i^\intercal \boldsymbol{r}_{i-1}}{\boldsymbol{s}_i^\intercal \boldsymbol{A} \boldsymbol{s}_i}.$$

Stopping Criteria Classic linear solvers typically use stopping criteria based on the current residual of the form $\|\boldsymbol{r}_i\|_2 \leq \max(\delta_{\text{rtol}} \|\boldsymbol{b}\|_2, \delta_{\text{atol}})$ for relative and absolute tolerances δ_{rtol} and δ_{atol}. However, this residual may oscillate or even increase in all but the last step even if the error $\|\boldsymbol{x}_* - \boldsymbol{x}_i\|_2$ is monotonically decreasing [71, 72]. From a probabilistic point of view, we should stop if our posterior uncertainty is sufficiently small. Assuming the posterior covariance is calibrated, it holds that

$$\left(\mathbb{E}_{\boldsymbol{x}_*}(\|\boldsymbol{x}_* - \mathbb{E}(\mathbf{x})\|_2)\right)^2 \leq \mathbb{E}_{\boldsymbol{x}_*}\left(\|\boldsymbol{x}_* - \mathbb{E}(\mathbf{x})\|_2^2\right) = \mathrm{tr}(\mathrm{Cov}(\mathbf{x})).$$

Hence given calibration, we can bound the expected (relative) error between our estimate and the true solution by terminating when $\sqrt{\mathrm{tr}(\mathrm{Cov}(\mathbf{x}))} \leq \max(\delta_{\text{rtol}} \|\boldsymbol{b}\|_2, \delta_{\text{atol}})$. A probabilistic criterion is also necessary for an extension to the noisy setting, where classic convergence criteria become stochastic. However, probabilistic linear solvers typically suffer from miscalibration [73], an issue we will address in Section 2.3.

Implementation An implementation of Algorithm 1 is available as part of ProbNum, an open-source Python package implementing probabilistic numerical methods:

ProbNum  https://github.com/probabilistic-numerics/probnum

The mean and covariance up- and downdates in Section 2.2.1, when performed iteratively, are of low rank. In order to maintain numerical stability these updates can instead be performed for their respective Cholesky factors [74]. This also enables computationally efficient sampling or evaluation of probability density functions downstream.

2.2.3 Theoretical Properties

This section details some theoretical properties of our method such as its convergence behavior and computational complexity. In particular, we demonstrate that for a specific prior choice Algorithm 1 recovers the method of conjugate gradients as its solution estimate. We begin by establishing that our solver is a *conjugate directions method* and therefore converges in at most n steps in exact arithmetic.

Theorem 2.1 (Conjugate Directions Method)
Given a prior $p(\mathbf{H}) = \mathcal{N}\left(\mathbf{H}; \boldsymbol{H}_0, \boldsymbol{W}_0^{\mathbf{H}} \otimes \boldsymbol{W}_0^{\mathbf{H}}\right)$ *such that* $\boldsymbol{H}_0, \boldsymbol{W}_0^{\mathbf{H}} \in \mathbb{R}^{n \times n}_{\text{sym}}$ *are positive definite, then the actions* $\boldsymbol{s}_i$ *of Algorithm 1 are* $\boldsymbol{A}$*-conjugate, i.e. it holds that*

$$\boldsymbol{s}_j^\intercal \boldsymbol{A} \boldsymbol{s}_k = 0$$

for all $0 \leq j \neq k \leq i$.

Proof. See Appendix A.2.1 for a proof. □

We can obtain a better convergence rate by placing stronger conditions on the prior covariance class as outlined in Section 2.3. Given these assumptions, Algorithm 1 recovers the iterates of the (preconditioned) method of conjugate gradients and thus inherits its favorable convergence behavior (see e.g. Nocedal and Wright [65]).

Theorem 2.2 (Connection to the Conjugate Gradient Method)
Given a scalar prior mean $\boldsymbol{A}_0 = \boldsymbol{H}_0^{-1} = \alpha \boldsymbol{I}$ *with* $\alpha > 0$*, assume (2.3) and (2.4) hold, then the iterates* $\boldsymbol{x}_i$ *of Algorithm 1 are identical to the ones produced by the conjugate gradient method.*

Proof. See Appendix A.2.2. □

A common phenomenon observed when implementing conjugate gradient methods is that due to cancellation in the computation of the residuals, the search directions $\boldsymbol{s}_i$ lose $\boldsymbol{A}$-conjugacy [59, 75, 76]. In fact, they can become independent up to working precision for i large enough [76]. One way to combat this is to perform complete reorthogonalization of the search directions in each iteration as originally suggested by Lanczos [77]. Algorithm 1 does this *implicitly* via its choice of policy which depends on all previous search directions as opposed to just $\boldsymbol{s}_{i-1}$ for (naive) CG.

Computational Complexity The solver has time complexity $\mathcal{O}(in^2)$ for i iterations without uncertainty calibration. Compared to CG, inferring the posteriors in Section 2.2.1 adds an overhead of four outer products and four matrix-vector products per iteration, given (2.3) and (2.4). Uncertainty calibration outlined in Section 2.3 adds between $\mathcal{O}(1)$ and $\mathcal{O}(i^3)$ per iteration depending on the sophistication of the scheme. Already for moderate n, this is dominated by the iteration cost. In practice, means and covariances do not need to be formed in memory. Instead, they can be evaluated in a matrix-free fashion, if $\boldsymbol{S}$ and $\boldsymbol{Y}$ are stored. This leads to linear space complexity $\mathcal{O}(in)$.

2.2.4 Related Work

Numerical methods for the solution of linear systems have been studied in great detail since the last century. Standard texts [57–59, 65] give an in-depth overview. The conjugate gradient method recovered by our algorithm for a specific choice of prior was introduced by Hestenes and Stiefel [71]. Recently, randomization has been exploited to develop improved algorithms for large-scale problems arising from machine learning [78, 79]. The key difference to our approach is that we do not rely on sampling to approximate large-scale matrices, but instead perform probabilistic inference. Our approach is based on the framework of probabilistic numerics [51, 53] and is a natural continuation of previous work on probabilistic linear solvers. In historical order, Hennig and Kiefel [70] provided a probabilistic interpretation of Quasi-Newton methods, which was expanded upon in [66]. This work also relied on the symmetric matrix-variate Gaussian as used in our paper. Bartels and Hennig [80] estimate numerical error in approximate least-squares solutions by using a probabilistic model. More recently, Cockayne et al. [73] proposed a Bayesian conjugate gradient method performing inference on the solution of the system. This was connected to the matrix-based view by Bartels et al. [67].

2.3 Prior Covariance Class

Having outlined the proposed algorithm, this section derives a prior covariance class that satisfies nearly all desiderata, connects the two modes of prior information and allows for calibration of uncertainty by appropriately choosing the remaining degrees of freedom in the covariance. The third desideratum posited that $\mathbf{A}$ and $\mathbf{H}$ should be almost surely positive definite. This does not hold for the matrix-variate Gaussian. However, we can restrict the choice of admissable $\boldsymbol{W}_0^{\mathbf{A}}$ to act like $\boldsymbol{A}$ on $\mathrm{span}(\boldsymbol{S})$. This in turn induces a positive definite posterior mean.

Proposition 2.1 (Hereditary Positive Definiteness [70, 81])
Let $\boldsymbol{A}_0 \in \mathbb{R}^{n\times n}_{\mathrm{sym}}$ be positive definite. Assume the actions $\boldsymbol{S}$ are $\boldsymbol{A}$-conjugate and $\boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S} = \boldsymbol{Y}$, then $\boldsymbol{A}_i$ is symmetric positive definite.

Proof. This follows from Theorem 7.5 in Dennis and Moré [81] and is proved explicitly in Hennig and Kiefel [70]. For a proof in our setting see Appendix A.3.1. □

Prior information about the linear system usually concerns the matrix $\boldsymbol{A}$ itself and not its inverse, but the inverse is needed to infer the solution $\boldsymbol{x}_*$ of the linear problem. So a way to translate between a Gaussian distribution on $\mathbf{A}$ and $\mathbf{H}$ is crucial. Previous works generally committed to either one view or the other, potentially discarding available information. Below, we show that the two correspond, if we allow ourselves to constrain the space of possible models.

Definition 2.1
Let $\boldsymbol{A}_i$ and $\boldsymbol{H}_i$ be the means of $\mathbf{A}$ and $\mathbf{H}$ at step i. We say a prior induces *posterior*

correspondence if

$$\boldsymbol{A}_i^{-1} = \boldsymbol{H}_i \tag{2.1}$$

for all steps i of the solver. If only

$$\boldsymbol{A}_i^{-1}\boldsymbol{Y} = \boldsymbol{H}_i\boldsymbol{Y}, \tag{2.2}$$

we say that *weak posterior correspondence* holds.

The following theorem establishes a sufficient condition for weak posterior correspondence. For an asymmetric prior model, one can establish a stronger notion of posterior correspondence. A proof is included in the appendix in Appendix A.3.2.

Theorem 2.3 (Weak Posterior Correspondence)
Let $\boldsymbol{W}_0^{\mathbf{H}} \in \mathbb{R}^{n\times n}_{\text{sym}}$ *be positive definite. Assume* $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1}$*, and* $\boldsymbol{W}_0^{\mathbf{A}}, \boldsymbol{A}_0, \boldsymbol{W}_0^{\mathbf{H}}$ *satisfy*

$$\boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S} = \boldsymbol{Y}, \tag{2.3}$$

$$\boldsymbol{S}^\intercal(\boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{A}_0^{-1} - \boldsymbol{A}\boldsymbol{W}_0^{\mathbf{H}}) = \boldsymbol{0}. \tag{2.4}$$

Then weak posterior correspondence holds for the symmetric Kronecker covariance.

Proof. The proof relies on applying the matrix inversion lemma to the rank 2 mean update for $\boldsymbol{A}_i$ given in Section 2.2.1. See Appendix A.3.2 for a proof. □

Given the above, let $\boldsymbol{A}_0$ be a symmetric positive definite prior mean and $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1}$. Define the orthogonal projections $\boldsymbol{P}_{\boldsymbol{S}}^{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{S}(\boldsymbol{S}^\intercal\boldsymbol{A}\boldsymbol{S})^{-1}\boldsymbol{S}^\intercal\boldsymbol{A}$ and $\boldsymbol{P}_{\boldsymbol{Y}}^{\boldsymbol{H}_0} = \boldsymbol{A}_0^{-1}\boldsymbol{Y}(\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}\boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}$ with respect to the inner products induced by $\boldsymbol{A}$ and $\boldsymbol{A}_0^{-1}$, as well as $\boldsymbol{P}_{\boldsymbol{S}^\perp} = \boldsymbol{I} - \boldsymbol{S}(\boldsymbol{S}^\intercal\boldsymbol{S})^{-1}\boldsymbol{S}^\intercal$ and $\boldsymbol{P}_{\boldsymbol{Y}^\perp} = \boldsymbol{I} - \boldsymbol{Y}(\boldsymbol{Y}^\intercal\boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal$ projecting to the spaces $\operatorname{span}(\boldsymbol{S})^\perp$ and $\operatorname{span}(\boldsymbol{Y})^\perp$. We propose the following prior covariance class given by the prior covariance factors

$$\boldsymbol{W}_0^{\mathbf{A}} = \boldsymbol{P}_{\boldsymbol{S}}^{\boldsymbol{A}} + \boldsymbol{P}_{\boldsymbol{S}^\perp}\boldsymbol{\Phi}\boldsymbol{P}_{\boldsymbol{S}^\perp} \quad \text{and} \quad \boldsymbol{W}_0^{\mathbf{H}} = \boldsymbol{P}_{\boldsymbol{Y}}^{\boldsymbol{H}_0} + \boldsymbol{P}_{\boldsymbol{Y}^\perp}\boldsymbol{\Psi}\boldsymbol{P}_{\boldsymbol{Y}^\perp}, \tag{2.5}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{\Psi} \in \mathbb{R}^{n\times n}$ are degrees of freedom. This choice of covariance class satisfies Theorem 2.1, Proposition 2.1, Theorem 2.3 and for a scalar mean also Theorem 2.1. Therefore, it produces symmetric realizations, has symmetric positive semi-definite means, it links the matrix and the inverse view and at any given time only needs access to $\boldsymbol{v} \mapsto \boldsymbol{A}\boldsymbol{v}$ not $\boldsymbol{A}$ itself. It is also compatible with a preconditioner by simply transforming the problem.

This class can be interpreted as follows. The derived covariance factor $\boldsymbol{W}_0^{\mathbf{A}}$ acts like $\boldsymbol{A}$ on the space $\operatorname{span}(\boldsymbol{S})$ explored by the algorithm. On the remaining space, its uncertainty is additionally determined by the degrees of freedom in $\boldsymbol{\Phi}$. Likewise, our best guess for $\boldsymbol{A}^{-1}$ is $\boldsymbol{A}_0^{-1}$ on the space spanned by $\boldsymbol{Y}$. On the orthogonal space $\operatorname{span}(\boldsymbol{Y})^\perp$ the uncertainty is also influenced by $\boldsymbol{\Psi}$. The prior depends on actions and observations collected during a run of Algorithm 1, hence

one might call this an empirical Bayesian approach. This begs the question of how the algorithm is realizable for the proposed prior (2.5) given its dependence on future data. Notice that the posterior mean in Section 2.2.1 only depends on $\boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S} = \boldsymbol{Y}$ *not* on $\boldsymbol{W}_0^{\mathbf{A}}$ alone. Using (2.5), at iteration i we have $\boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S}_{1:i} = \boldsymbol{Y}_{1:i}$, i.e. the observations made up to this point. Similar reasoning applies to the inverse. Now, the posterior covariances do depend on $\boldsymbol{W}_0^{\mathbf{A}}$, respectively $\boldsymbol{W}_0^{\mathbf{H}}$ alone, but prior to convergence we only require $\mathrm{tr}(\mathrm{Cov}(\mathbf{x}))$ for the stopping criterion. We show in Appendix A.1.2 under the assumptions of Theorem 2.1 how to compute this at any iteration i independent of future actions and observations. Therefore prior to the convergence of Algorithm 1 *the covariance factors are never explicitly formed.*

Uncertainty Calibration Generally, the actions of Algorithm 1 identify eigenpairs $(\lambda_i, \boldsymbol{v}_i)$ in descending order of $\lambda_i \boldsymbol{v}_i^\intercal \boldsymbol{r}_0$ which is a well-known behavior of CG [65]. In part, since this dynamic of the underlying Krylov subspace method is not encoded in the prior, the solver in its current form is typically miscalibrated (see also [73]). While this non-linear information is challenging to include in the Gaussian framework, we can choose $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ in (2.5) to empirically calibrate uncertainty. This can be interpreted as a form of hyperparameter optimization similar to the optimization of kernel parameters in GP regression.

We would like to encode prior knowledge about the way $\boldsymbol{A}$ and $\boldsymbol{H}$ act in the respective orthogonal spaces $\mathrm{span}(\boldsymbol{S})^\perp$ and $\mathrm{span}(\boldsymbol{Y})^\perp$. For the Rayleigh quotient $R(\boldsymbol{A}, \boldsymbol{v}) = (\boldsymbol{v}^\intercal \boldsymbol{A}\boldsymbol{v})(\boldsymbol{v}^\intercal \boldsymbol{v})^{-1}$ it holds that $\lambda_{\min}(\boldsymbol{A}) \leq R(\boldsymbol{A}, \boldsymbol{v}) \leq \lambda_{\max}(\boldsymbol{A})$. Hence for vectors $\boldsymbol{v}$ lying in the respective null spaces of $\boldsymbol{S}$ and $\boldsymbol{Y}$ our uncertainty should be determined by the not yet explored eigenvalues $\lambda_{i+1}, \ldots, \lambda_n$ of $\boldsymbol{A}$ and $\boldsymbol{H}$. Without prior information about the eigenspaces, we choose $\boldsymbol{\Phi} = \phi\boldsymbol{I}$ and $\boldsymbol{\Psi} = \psi\boldsymbol{I}$. If a priori we know the respective spectra, a straightforward choice is

$$\phi = \psi^{-1} = \frac{1}{n-i}\sum_{j=i+1}^{n} \lambda_j(\boldsymbol{A}).$$

In the absence of prior spectral information, we can make use of already collected quantities during a run of Algorithm 1. We build a one-dimensional regression model $p(\log R_j \mid \boldsymbol{Y}, \boldsymbol{S})$ for the log-Rayleigh quotient $\log R(\boldsymbol{A}, \boldsymbol{s}_j)$. Such a model can encode the well-studied behavior of CG, whose Rayleigh coefficients rapidly decay at first, followed by a slower continuous decay [65]. Figure 2.3 illustrates this approach using a GP regression model. At convergence, we use the prediction of the Rayleigh quotient for the remaining $n - i$ dimensions by choosing

$$\phi = \psi^{-1} = \exp\left(\frac{1}{n-i}\sum_{j=i+1}^{n} \mathbb{E}(\log R_j \mid \mathbf{A}, \boldsymbol{S})\right),$$

i.e. uncertainty about actions in $\mathrm{span}(\boldsymbol{S})^\perp$ is calibrated to be the average Rayleigh quotient as an approximation to the spectrum. Depending on the application a simple or more complex model may be useful. For large problems, where generally $i \ll n$, more sophisticated schemes become computationally feasible. However, these do not necessarily need to be computationally demanding due to the simple nature of this one-dimensional regression problem. For example, approximate [82] or even exact GP regression [83] is possible in $\mathcal{O}(i)$ using a Kalman filter.

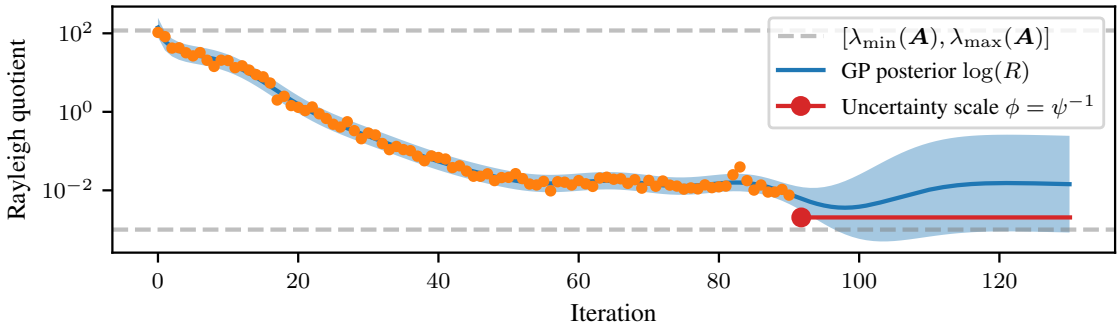


Figure 2.3: *Rayleigh regression.* Uncertainty calibration via one-dimensional GP regression with data $\{\log R(\boldsymbol{A}, \boldsymbol{s}_j)\}_{j=1}^{i}$ after $i = 91$ iterations of Algorithm 1 on an $n = 1000$ dimensional Matérn($\frac{3}{2}$) kernel matrix inversion problem. The degrees of freedom $\phi = \psi^{-1} > 0$ are set based on the average predicted Rayleigh quotient for the remaining $n - i = 909$ dimensions.

2.4 Experiments

This section demonstrates the functionality of Algorithm 1. We choose some – deliberately simple – example problems from machine learning and scientific computation, where the solver can be used to quantify uncertainty induced by finite computation, solve multiple consecutive linear systems, and propagate information between problems.

Gaussian Process Regression Suppose we want to infer a latent function from data $(\boldsymbol{X}, \boldsymbol{y})$ via GP regression [47], where $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{y} \in \mathbb{R}^n$. Given a prior $f \sim \mathcal{GP}(0, k)$ with kernel k, the posterior mean and marginal variance at $n_\diamond$ new inputs $\boldsymbol{X}_\diamond \in \mathbb{R}^{n_\diamond \times d}$ are given by

$$\begin{aligned} \mu_*(\boldsymbol{X}_\diamond) &= k(\boldsymbol{X}_\diamond, \boldsymbol{X})(\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y} \\ k(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond) &= k(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond) - k(\boldsymbol{X}_\diamond, \boldsymbol{X})(\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} k(\boldsymbol{X}, \boldsymbol{X}_\diamond), \end{aligned}$$

where $\boldsymbol{K} = k(\boldsymbol{X}, \boldsymbol{X}) \in \mathbb{R}^{n \times n}$ is the Gram matrix of the kernel and $k(\boldsymbol{X}, \boldsymbol{X}_\diamond) \in \mathbb{R}^{n \times n_\diamond}$. The bulk of computation during prediction arises from solving the linear system $(\boldsymbol{K} + \sigma^2 \boldsymbol{I})\boldsymbol{z} = \boldsymbol{b}$ for some right-hand side $\boldsymbol{b} \in \mathbb{R}^n$ repeatedly. When using a probabilistic linear solver for this task, we can quantify the uncertainty arising from finite computation as well as the belief of the solver about the shape of the GP at a set of new inputs. Figure 2.4 illustrates this. We can estimate the marginal variance of the GP without solving the linear system again by multiplying $k(\boldsymbol{X}, \boldsymbol{X}_\diamond)$ with the estimated inverse of $\boldsymbol{K} + \sigma^2 \boldsymbol{I}$. In large-scale applications, we can trade off computational expense for increased uncertainty arising from the numerical approximation and quantified by the probabilistic linear solver. By assessing the computational uncertainty arising from not exploring the full space, we can judge the quality of the estimated GP mean and marginal variance.

Gram Matrix Inversion Consider a linear problem $\boldsymbol{K}\boldsymbol{x}_* = \boldsymbol{b}$, where $\boldsymbol{K}$ is generated by a Mercer kernel. For a ν-times continuously differentiable kernel the eigenvalues $\lambda_n(\boldsymbol{K})$ decay approximately as $|\lambda_n| \in \mathcal{O}(n^{-\nu - \frac{1}{2}})$ [84]. We can make use of this generative prior information by

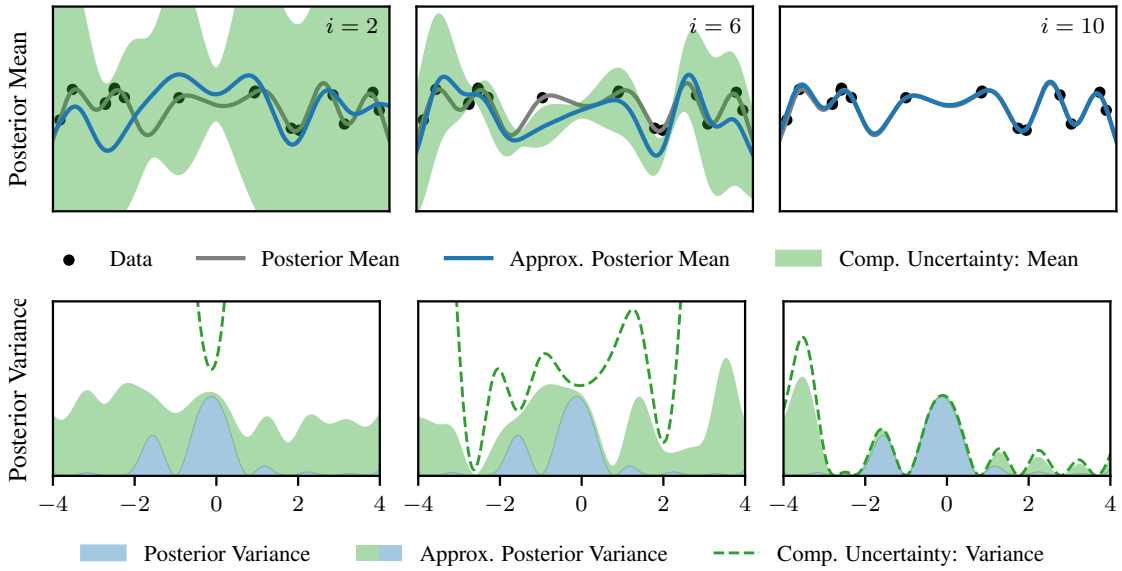


Figure 2.4: *Computational uncertainty in GP inference.* Posterior mean and variance of a GP computed using a probabilistic linear solver. Top: GP mean for a toy data set ($n = 16$) computed with an increasing number of iterations i of Algorithm 1. The approximated posterior mean (—) approaches the true mean with an increasing number of iterations i and the computational uncertainty (●) contracts. Bottom: Exact (●) and estimated posterior variance (● + ●) with computational uncertainty (○). The GP variance estimate is computed *without any additional solver iterations* by using the approximate inverse obtained from the approximation of the mean.

specifying a parametrized prior mean $\mu(n) = \log(\theta_0' n^{-\theta_1}) = \theta_0 - \theta_1 \log(n)$ for the log-Rayleigh quotient model. Typically, such Gram matrices are ill-conditioned and therefore $\hat{\boldsymbol{K}} = \boldsymbol{K} + \sigma^2 \boldsymbol{I}$ is used instead, implying $\lambda(\hat{\boldsymbol{K}})_i \geq \sigma^2$. In order to assess calibration we apply various differentiable kernels to the airline delay dataset from January 2020 [85]. We compute the log-ratio statistic $w(\boldsymbol{x}_*) = \frac{1}{2}\log(\mathrm{tr}(\mathrm{Cov}(\mathbf{x}))) - \log(\|\boldsymbol{x}_* - \mathbb{E}(\mathbf{x})\|_2)$ for no calibration, calibration via Rayleigh quotient GP regression using $\mu(n)$ as a prior mean, calibration by setting $\phi = \sigma^2$ and calibration using the average spectrum $\phi = \overline{\lambda}_{i+1:n}$. The average $\bar{w}$ for $10^5/n$ randomly sampled test problems is shown in Table 2.2.[2] Without any calibration, the solver is generally overconfident. All tested calibration procedures reverse this, resulting in more cautious uncertainty estimates. We observe that Rayleigh quotient regression overcorrects for larger problems. This is due to the fact that its model correctly predicts $\boldsymbol{K}$ to be numerically singular from the dominant Rayleigh quotients, however, it misses the information that the spectrum of $\hat{\boldsymbol{K}}$ is bounded from below by σ^2. If we know the (average) of the remaining spectrum, significantly better calibration can be achieved, but often this information is not available. Nonetheless, since in this setting the majority of eigenvalues satisfy $\lambda(\hat{\boldsymbol{K}})_i \approx \sigma^2$ by choosing $\phi = \psi^{-1} = \sigma^2$, we can get to the same degree of calibration. Therefore, we can improve the solver's uncertainty calibration at constant

[2]We decrease the number of samples with the dimension because forming *dense* kernel matrices in memory and computing their eigenvalues becomes computationally prohibitive – *not* because of the cost of our solver.

Table 2.2: *Uncertainty calibration for kernel Gram matrices.* Monte Carlo estimate $\bar{w} \approx \mathbb{E}_{\boldsymbol{x}_*}(w(\boldsymbol{x}_*))$ measuring calibration given $10^5/n$ sampled linear problems of the form $\hat{\boldsymbol{K}}\boldsymbol{x}_* = \boldsymbol{b}$ for each kernel and calibration method. For $\bar{w} \approx 0$ the solver is well calibrated, for $\bar{w} \gg 0$ underconfident and for $\bar{w} \ll 0$ overconfident.

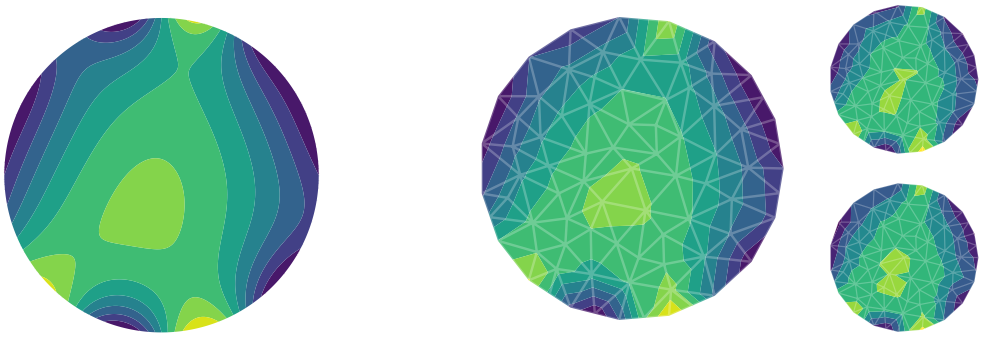
Kernel	n	None	Rayleigh regression	Lower bound σ^2	Average $\overline{\lambda}_{i+1:n}$
Matérn(3/2)	10^2	-5.99	-0.24	0.32	0.09
Matérn(3/2)	10^3	-1.93	7.53	4.26	4.19
Matérn(3/2)	10^4	3.87	17.16	8.48	8.47
Matérn(5/2)	10^2	-7.84	-1.01	-0.76	-0.80
Matérn(5/2)	10^3	-4.63	1.43	-0.80	-0.81
Matérn(5/2)	10^4	-4.34	10.81	0.80	0.80
RBF	10^2	-7.53	-0.70	-0.84	-0.87
RBF	10^3	-4.94	6.60	0.77	0.77
RBF	10^4	0.14	21.32	2.92	2.92

cost $\mathcal{O}(1)$ per iteration. For more general problems involving Gram matrices without damping, we may want to rely on Rayleigh regression instead.

Galerkin's Method for PDEs In the spirit of applying machine learning approaches to problems in the physical sciences [21], we use Algorithm 1 for the approximate solution of a PDE via Galerkin's method [64]. Consider the Dirichlet problem for the Poisson equation given by

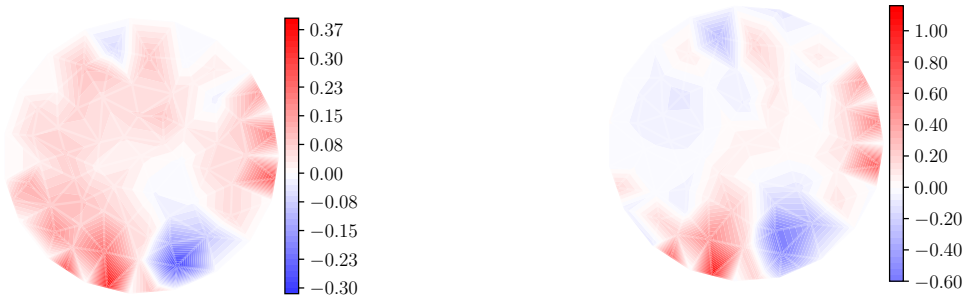
$$\begin{cases} -\Delta u(\boldsymbol{x}) = f(\boldsymbol{x}) & \boldsymbol{x} \in \Omega^\circ \\ u(\boldsymbol{x}) = u_{\partial\Omega}(\boldsymbol{x}) & \boldsymbol{x} \in \partial\Omega \end{cases}$$

where Ω is a connected open region with sufficiently regular boundary and $u_{\partial\Omega} : \partial\Omega \to \mathbb{R}$ defines the boundary conditions. One obtains an approximate solution by projecting the weak formulation of the PDE to a finite-dimensional subspace. This results in the *Galerkin equation* $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$, i.e. a linear system where $\boldsymbol{A}$ is the Gram matrix of the associated bilinear form. Figure 2.5 shows the induced uncertainty on the solution of the Dirichlet problem for $f(\boldsymbol{x}) = 15$ and $u_{\partial\Omega}(\boldsymbol{x}) = (\boldsymbol{x}_1^2 - 2\boldsymbol{x}_2)^2(1 + \sin(2\pi\boldsymbol{x}_1))$. The mesh and corresponding Gram matrix were computed using FEniCS [86]. We can exploit two properties of Algorithm 1 in this setting. First, if we need to solve multiple related problems $(\boldsymbol{A}_j, \boldsymbol{f}_j)_j$, by solving a single problem we obtain an estimate of the solution to all other problems. We can successively use the posterior over the inverse as a prior for the next problem. This approach is closely related to subspace recycling in numerical linear algebra [87, 88]. Second, suppose we first compute a solution in a low-dimensional subspace corresponding to a coarse discretization for computational efficiency. We can then leverage the estimated solution to extrapolate to an (adaptively) refined discretization based on the posterior uncertainty. In machine learning lingo these two approaches can be viewed as forms of *transfer learning*.



(a) Ground truth $\boldsymbol{u}$ (b) Estimated solution $\mathbb{E}(\mathbf{u})$ and samples $\mathbf{u}_i$

Figure 2.5: *Solving the Dirichlet problem with a probabilistic linear solver.* Figures 2.5(a) and 2.5(b) show the ground truth and mean of the solution computed with Algorithm 1 after $i = 23$ iterations along with samples from the posterior. The posterior on the coarse mesh can be used to assess uncertainty about the solution on a finer mesh.



(a) Signed error $\boldsymbol{u} - \mathbb{E}(\mathbf{u})$ (b) Whitened error $\mathrm{Cov}(\mathbf{u})^{-\frac{1}{2}}\,(\boldsymbol{u} - \mathbb{E}(\mathbf{u}))$

Figure 2.6: *Uncertainty calibration on the Dirichlet problem.* The signed error computed on the coarse mesh in Figure 2.6(a) shows that the approximation is better near the top boundary of Ω. Given perfect uncertainty calibration, the whitened error in Figure 2.6(b) is a sample from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The apparent structure in the plot and smaller-than-expected deviations in the upper part of Ω indicate the conservative confidence estimate of the solver.

2.5 Conclusion

In this work, we condensed a line of previous research on probabilistic linear algebra into a self-contained algorithm for the solution of linear problems in machine learning. We proposed first principles to constrain the space of possible generative models and derived a suitable covariance class. In particular, our proposed framework incorporates prior knowledge on the system matrix or its inverse and performs inference for both in a *consistent* fashion. Within our framework, we identified parameter choices that recover the iterates of conjugate gradients in the mean, but add calibrated uncertainty around them in a computationally lightweight manner. To our knowledge, our solver, available as part of the PROBNUM package, is the first practical implementation of this kind. In the final parts of this paper, we showcased applications like kernel matrix inversion, where prior spectral information can be used for uncertainty calibration and outlined example use cases for the propagation of computational uncertainty through chains of computations. Naturally, there are also limitations remaining. While our theoretical framework can incorporate noisy matrix-vector product evaluations into its inference procedure via a Gaussian likelihood, practically *tractable* inference in the inverse model is more challenging. Our solver also opens up new research directions. In particular, our outlined regression model on the Rayleigh quotient may lead to a probabilistic model of the eigenspectrum. Finally, the matrix-based view of probabilistic linear solvers could inform probabilistic approaches to matrix decompositions, analogous to the way Lanczos methods are used in the classical setting.

Chapter

3

Posterior and Computational Uncertainty in Gaussian Processes

Gaussian processes scale prohibitively with the size of the dataset. In response, many approximation methods have been developed, which inevitably introduce approximation error. This additional source of uncertainty, due to limited computation, is entirely ignored when using the approximate posterior. Therefore in practice, Gaussian process models are often as much about the approximation method as they are about the data. Here, we develop a new class of methods that provides consistent estimation of the combined uncertainty arising from *both* the finite number of data observed *and* the finite amount of computation expended. The most common GP approximations map to an instance in this class, such as methods based on the Cholesky factorization, conjugate gradients, and inducing points. For any method in this class, we prove (i) convergence of its posterior mean in the associated RKHS, (ii) decomposability of its combined posterior covariance into mathematical and computational covariances, and (iii) that the combined variance is a tight worst-case bound for the squared error between the method's

posterior mean and the latent function. Finally, we empirically demonstrate the consequences of ignoring computational uncertainty and show how implicitly modeling it improves generalization performance on benchmark datasets.

3.1 Introduction

Gaussian processes (GPs) are an expressive probabilistic model class, but their prohibitive scaling necessitates approximation [47]. A range of approximations based on kernel [89–97] or precision matrix [98–101] estimates, inducing point methods [102–109], and iterative solvers [3, 4, 110–114] have been proposed. These methods all use an affordable amount of computation to obtain an approximation of the *mathematical* posterior, which exists theoretically but cannot be accessed given limited computational resources. The approximate posterior is then used as a direct replacement of the mathematical posterior in downstream applications. Doing so, however, completely ignores the fact that we only expended a limited amount of compute. By analogy to the typical GP operation, where *limited data* induces modeling error captured by *mathematical uncertainty*, our work is motivated by the fact that *limited computation* induces approximation error that must be captured by *computational uncertainty*.

Here, we introduce IterGP, a class of methods that return a *combined uncertainty* that is the sum of mathematical and computational uncertainty. Figure 3.1 illustrates the difference between ignoring computational uncertainty and explicitly modeling it. We perform GP regression using a Matérn($\frac{3}{2}$) kernel on a toy dataset and compare SVGP (●) [109] to its analog in our framework, IterGP-PI (● + ●), for a fixed set of inducing points. The computational shortcuts of inducing point methods can lead to unavoidable biases in their posterior mean and covariance [115, 116]. As Figure 3.1 illustrates, SVGP may underestimate the marginal variance where inducing points do not coincide with data points. In contrast, IterGP is guaranteed to overestimate the mathematical uncertainty – with the difference precisely given by the computational uncertainty (●). Additionally, the computational uncertainty is a worst-case bound (—) on the error of the approximate posterior mean.

To be clear, this overestimation is desirable: IterGP is not a typical approximation in the sense that its combined posterior attempts to approximate the mathematical posterior. Rather, IterGP recognizes that the mathematical posterior exists, but we do not have access to it, given computational constraints. Finite compute is as true a source of posterior uncertainty as finite data. Taking this view seriously, the true goal of GPs in the limited compute regime should in fact be to track combined uncertainty. This intuition motivates IterGP and is formally a feature of our results. We show that, if you update your GP via computation, specifically matrix-vector multiplication, then the combined uncertainty of the IterGP algorithm is precisely the correct object to capture your belief (Theorem 3.2) – in the same way the mathematical posterior is the correct object given finite data and unlimited computation.

Formally, IterGP is a probabilistic numerical method [51–54]. It treats the (unknown) representer weights as a latent variable with a prior belief that, when marginalized out, corresponds to a

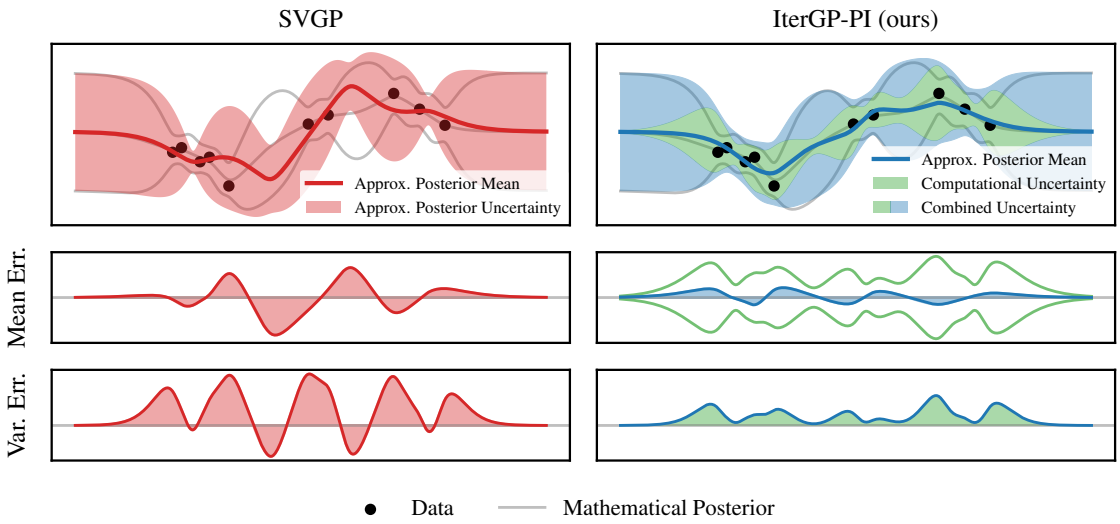


Figure 3.1: *Modeling computational uncertainty improves GP approximation.*

GP prior conditioned on no data. We then use a computational primitive (matrix-vector multiplication) that corresponds to tractable Bayesian updates on the representer weight distribution; that is, conditioning on computations performed on the data. The resulting belief can then be marginalized out to obtain a closed-form, tractable expression for the combined – mathematical plus computational – uncertainty. This uncertainty quantification can be done *exactly* in quadratic time and linear space complexity.

Our framework admits three key theoretical properties. First, common GP approximations such as the partial Cholesky, the method of conjugate gradients and inducing point methods (e.g. SVGP) map to a corresponding IterGP instance. Therefore, these approaches can either be directly extended or modified to properly account for computational uncertainty. Second, the approximate posterior mean of any method in our proposed class converges to the mathematical posterior mean in RKHS norm in at most n steps, where the convergence rate is determined by the choice of method (Theorem 3.1). Third, the combined uncertainty is a tight worst-case bound on the relative error between the approximate posterior mean and the latent function (Theorem 3.2). To the best of our knowledge, no existing GP approximation has this last property; an analogous guarantee only holds for exact GPs [44, Sec. 3.4].

Contribution This work introduces IterGP, which defines a new class of GP approximations that accounts for computational uncertainty arising from limited computation. Some IterGP instances extend classic methods with improved uncertainty quantification (Table 3.1). For any method in this class, we prove that the approximate posterior mean converges to the mathematical posterior mean (Theorem 3.1) and that the combined uncertainty is a tight worst-case bound on the relative distance to the latent function one is trying to learn (Theorem 3.2, Corollary 3.1). We demonstrate empirically that modeling computational uncertainty can either save computation or improve generalization on a set of regression benchmark datasets. In conclusion, we show that it is possible to exactly quantify the inevitable error of GP approximations at quadratic cost by propagating said error to the posterior in the form of computational uncertainty.

3.2 Computation-Aware Gaussian Process Inference

We aim to learn a latent function $h : \mathbb{X} \to \mathbb{R}$ given a training dataset $\boldsymbol{X} = \big(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\big) \in \mathbb{R}^{n \times d}$ of n inputs $\boldsymbol{x}_j \in \mathbb{X} \subset \mathbb{R}^d$ and corresponding outputs $\boldsymbol{y} = \big(y_1, \dots, y_n\big)^\intercal \in \mathbb{R}^n$.

Gaussian Processes A stochastic process $f \sim \mathcal{GP}(\mu, k)$ with mean function $\mu : \mathbb{R}^d \to \mathbb{R}$ and kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a *Gaussian process* (GP) if any collection of function values $\boldsymbol{f} = (f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_n))^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$ is jointly Gaussian with $\boldsymbol{\mu}_j = \mu(\boldsymbol{x}_j)$ and $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Assuming observation noise $\boldsymbol{y} \mid \boldsymbol{f} \sim \mathcal{N}\big(\boldsymbol{f}, \sigma^2 \boldsymbol{I}\big)$, the posterior distribution at test inputs $\boldsymbol{X}_\diamond$ is given by $\boldsymbol{f}_\diamond \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$ where the posterior mean and covariance functions are given by

$$\begin{aligned} \mu_*(\cdot) &= \mu(\cdot) + k(\cdot, \boldsymbol{X}) \overbrace{\hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})}^{\boldsymbol{v}_*}, \\ k_*(\cdot, \cdot) &= k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \hat{\boldsymbol{K}}^{-1} k(\boldsymbol{X}, \cdot) \end{aligned} \tag{3.1}$$

where $\hat{\boldsymbol{K}} \coloneqq \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$. Computing the *representer weights* $\boldsymbol{v}_* = \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$ exactly (as well as the posterior variance) is prohibitive given our limited computational budget.

Learning Representer Weights Consider the conditional distribution of the latent GP given its representer weights:

$$p(\boldsymbol{f}_\diamond \mid \boldsymbol{v}_*) = \mathcal{N}(\mu(\boldsymbol{X}_\diamond) + k(\boldsymbol{X}_\diamond, \boldsymbol{X}) \boldsymbol{v}_*, \;\; k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond)). \tag{3.2}$$

When $\boldsymbol{v}_*$ is known exactly, we recover (3.1). However, if we instead treat $\boldsymbol{v}_*$ as a random variable with the prior $p(\boldsymbol{v}_*) = \mathcal{N}\big(\boldsymbol{v}_*; \boldsymbol{0}, \hat{\boldsymbol{K}}^{-1}\big)$, then the resulting marginal $\int p(\boldsymbol{f}_\diamond \mid \boldsymbol{v}_*) p(\boldsymbol{v}_*) \, d\boldsymbol{v}_*$ recovers the GP prior $\mathcal{N}(\mu(\boldsymbol{X}_\diamond), k(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$. Our goal is to update this prior by iteratively applying the tractable computational primitive (i.e. matrix-vector multiplies). More specifically, each iteration conditions the current belief distribution $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_{i-1}, \boldsymbol{\Sigma}_{i-1})$ on a one-dimensional projection of the current *residual* $\boldsymbol{r}_{i-1} = (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}} \boldsymbol{v}_{i-1}$, where the projection is defined by an arbitrary vector $\boldsymbol{s}_i$:

$$\alpha_i \coloneqq \boldsymbol{s}_i^\intercal \boldsymbol{r}_{i-1} = \boldsymbol{s}_i^\intercal ((\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}} \boldsymbol{v}_{i-1}) = \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} (\boldsymbol{v}_* - \boldsymbol{v}_{i-1}). \tag{3.3}$$

The choice of *actions* $\boldsymbol{s}_i$, which intuitively weighs the approximation error of selected data points, defines different instances of our IterGP framework. Computing (3.3) requires a single matrix-vector multiplication. After computing α_i, we can perform an exact Bayesian update of $p(\boldsymbol{v}_*)$ via linear Gaussian identities. The updated belief about the representer weights $p(\boldsymbol{v}_*)$ conditioned on the observations α_i is given by $\mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$, with

$$\boldsymbol{v}_i = \boldsymbol{v}_{i-1} + \underbrace{\boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i}_{=: \boldsymbol{d}_i} \underbrace{(\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i)^{-1}}_{=: \eta_i} \underbrace{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} (\boldsymbol{v}_* - \boldsymbol{v}_{i-1})}_{= \alpha_i} = \boldsymbol{C}_i (\boldsymbol{y} - \boldsymbol{\mu}) \tag{3.4}$$

$$\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_{i-1} - \underbrace{\boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i}_{= \boldsymbol{d}_i} \underbrace{(\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i)^{-1}}_{= \eta_i} \underbrace{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1}}_{= \boldsymbol{d}_i^\intercal} = \hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i. \tag{3.5}$$

where $\boldsymbol{C}_i \coloneqq \sum_{j=1}^i \frac{1}{\eta_j} \boldsymbol{d}_j \boldsymbol{d}_j^\intercal = \boldsymbol{S}_i (\boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{S}_i)^{-1} \boldsymbol{S}_i^\intercal$ is a rank-i matrix (see Proposition B.1 for details). We can interpret $\boldsymbol{C}_i$ as an approximation to the precision matrix $\hat{\boldsymbol{K}}^{-1}$. With each

computation, the uncertainty about the representer weights contracts as $\boldsymbol{C}_i \to \hat{\boldsymbol{K}}^{-1} = \boldsymbol{\Sigma}_0$ as $i \to n$. After n iterations, $\boldsymbol{C}_n = \hat{\boldsymbol{K}}^{-1}$, meaning we fully recovered the representer weights with zero uncertainty. The consistent estimate for the representer weights is consequently $\boldsymbol{v}_i = \boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{\mu})$ giving an interpretation of the form of the posterior mean for the representer weights.

Combining Mathematical and Computational Uncertainty We now have a belief $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$ about the representer weights reflecting the expended computation. To account for this computational uncertainty, we treat the representer weights as a latent variable of the mathematical posterior by reparameterizing $p(\boldsymbol{f}_\diamond \mid \boldsymbol{y}) = p(\boldsymbol{f}_\diamond \mid \boldsymbol{v}_*)$ and then marginalizing. The resulting marginal considers all possible representer weights which would have resulted in the same computational observations and therefore *implicitly* adds the uncertainty coming from *the computation itself*. Since the posterior mean of a GP is a linear function of the representer weights, the marginal distribution is given by

$$p(\boldsymbol{f}_\diamond) = \int p(\boldsymbol{f}_\diamond \mid \boldsymbol{v}_*) p(\boldsymbol{v}_*)\, d\boldsymbol{v}_* = \mathcal{N}(\boldsymbol{f}_\diamond; \mu_i(\boldsymbol{X}_\diamond), k_i(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond)),$$

where

$$\begin{aligned} \mu_i(\cdot) &= \mu(\cdot) + k(\cdot, \boldsymbol{X})\boldsymbol{v}_i = k(\cdot, \boldsymbol{X})\boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{\mu}) \\ k_i(\cdot,\cdot) &= \underbrace{k(\cdot,\cdot) - k(\cdot, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \cdot)}_{\text{mathematical uncertainty}} + \underbrace{k(\cdot, \boldsymbol{X})\boldsymbol{\Sigma}_i k(\boldsymbol{X}, \cdot)}_{\text{computational uncertainty}} = \underbrace{k(\cdot,\cdot) - k(\cdot, \boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X}, \cdot)}_{\text{combined uncertainty}} \end{aligned}$$

since $\boldsymbol{\Sigma}_i = \hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i$.[1] As we perform more computation, the computational uncertainty reduces and we approach the mathematical uncertainty. While the *individual* terms are computationally prohibitive, the *combined* uncertainty can be evaluated cheaply since the approximate precision matrix $\boldsymbol{C}_i$ is of low rank. Figure 3.2 illustrates that computational uncertainty is large where there are data and we have not targeted computation yet. Different methods from our proposed class target computation in different parts of the input space. Where there is no data the prior is a good approximation of the posterior and therefore computational uncertainty is low.

Algorithm 2 computes an estimate of the representer weights $\boldsymbol{v}_i$ and the rank-i precision matrix approximation $\boldsymbol{C}_i$. A specific instance of IterGP is defined by a sequence of actions $\boldsymbol{s}_i$. To gain an intuition for how Algorithm 2 operates, it helps to interpret it as targeting a given computational budget towards certain data points defined by $\boldsymbol{s}_i$. Near data points $\boldsymbol{x}_j$ that are not targeted, i.e. $(\boldsymbol{s}_i)_j = 0$, computational uncertainty remains unchanged. In fact, data points $(\boldsymbol{x}_j, y_j)$ that are never targeted up to iteration i are not needed to compute $\mathcal{GP}(\mu_i, k_i)$, meaning that Algorithm 2 is *inherently online* and we can *observe data sequentially* without having to restart the algorithm (see Theorem B.5).

[1] While we derive the combined posterior from a probabilistic numerics perspective, we can alternatively interpret the posterior $\mathcal{GP}(\mu_i, k_i)$ as conditioning the GP prior f on linearly transformed data $\boldsymbol{S}_i^\intercal \boldsymbol{y} \mid \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{S}_i^\intercal \boldsymbol{f}, \sigma^2 \boldsymbol{S}_i^\intercal \boldsymbol{S}_i)$.

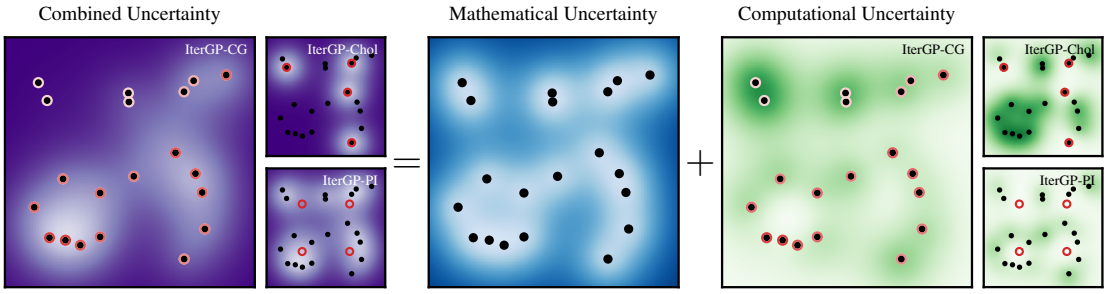


Figure 3.2: *Decomposition of the combined uncertainty.* The combined uncertainty (●) output by IterGP decomposes into the mathematical uncertainty (●) and computational uncertainty (●). After $i = 4$ iterations of Algorithm 2 computational uncertainty is small in parts of the input space where there either is no data (●) or computation was "targeted" (○). Which data points are targeted in each iteration and to what degree is defined by the magnitude of the action vector elements $(\boldsymbol{s}_i)_j$. Different instances of IterGP either reduce computational uncertainty locally (e.g. IterGP-Chol, IterGP-PI) or globally (e.g. IterGP-CG). After n iterations the mathematical uncertainty is recovered.

Algorithm 2: A Class of Computation-Aware Iterative Methods for GP Approximation

Input: Prior mean function μ, prior covariance function / kernel k, training inputs $\boldsymbol{X}$, targets $\boldsymbol{y}$
Output: (Combined) GP posterior $\mathcal{GP}(\mu_i, k_i)$

procedure ITERGP($\mu, k, \boldsymbol{X}, \boldsymbol{y}$)
  $(\mu_0, k_0) \leftarrow (\mu, k)$ ▷ Initialize mean and covariance function with prior.
  $\boldsymbol{\mu} \leftarrow \mu(\boldsymbol{X})$ ▷ Prior predictive mean.
  $\hat{\boldsymbol{K}} \leftarrow k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}$ ▷ Prior predictive kernel matrix.
  while not STOPPINGCRITERION() **do** ▷ Stopping criterion.
    $\boldsymbol{s}_i \leftarrow$ POLICY() ▷ Select action via policy (see Table 3.1 for examples).
    $\boldsymbol{r}_{i-1} \leftarrow (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}} \boldsymbol{v}_{i-1}$ ▷ Predictive residual.
    $\alpha_i \leftarrow \boldsymbol{s}_i^\intercal \boldsymbol{r}_{i-1}$ ▷ Observation via information operator.
    $\boldsymbol{d}_i \leftarrow \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i = (\boldsymbol{I} - \boldsymbol{C}_{i-1} \hat{\boldsymbol{K}}) \boldsymbol{s}_i$ ▷ Search direction.
    $\eta_i \leftarrow \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i = \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{d}_i$ ▷ Normalization constant.
    $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_{i-1} + \frac{1}{\eta_i} \boldsymbol{d}_i \boldsymbol{d}_i^\intercal$ ▷ Precision matrix approximation $\boldsymbol{C}_i \approx \hat{\boldsymbol{K}}^{-1}$.
    $\boldsymbol{Q}_i \leftarrow \boldsymbol{Q}_{i-1} + \frac{1}{\eta_i} \hat{\boldsymbol{K}} \boldsymbol{d}_i \boldsymbol{d}_i^\intercal \hat{\boldsymbol{K}}$ ▷ Kernel matrix approximation $\boldsymbol{Q}_i \approx \hat{\boldsymbol{K}}$.
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_{i-1} + \frac{\alpha_i}{\eta_i} \boldsymbol{d}_i$ ▷ Representer weights estimate.
    $\boldsymbol{\Sigma}_i \leftarrow \boldsymbol{\Sigma}_0 - \boldsymbol{C}_i$ ▷ Computational representer weights uncertainty.
  $p(\boldsymbol{v}_*) \leftarrow \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$ ▷ Belief about representer weights.
  $\mu_i(\cdot) \leftarrow \mu(\cdot) + k(\cdot, \boldsymbol{X}) \boldsymbol{v}_i$ ▷ Approximate posterior mean function.
  $k_i(\cdot, \cdot) \leftarrow k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \boldsymbol{C}_i k(\boldsymbol{X}, \cdot)$ ▷ Combined uncertainty.
  return $\mathcal{GP}(\mu_i, k_i)$

Greyed-out quantities are *not* needed to compute the combined posterior and are only included for exposition.

Table 3.1: *Instances of Algorithm 2, which map to commonly used GP approximations.*

Method	Actions $\boldsymbol{s}_i$	Classic Analog	Reference
IterGP-Chol	$\boldsymbol{e}_i$	(partial) Cholesky	Theorem B.1
IterGP-PBR	$\mathrm{ev}_i(\hat{\boldsymbol{K}})$	(partial) EVD / SVD	Theorem B.2
IterGP-CG	$\boldsymbol{s}_i^{\mathrm{PCG}}$ or $\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i$	(preconditioned) CG	Theorem B.3 and Corollary B.1
IterGP-PI	$k(\boldsymbol{X}, \boldsymbol{z}_i)$	$\approx$ Nyström (SoR, DTC), SVGP	Section 3.2.1 and Theorem B.4

3.2.1 Connection to Other GP Approximation Methods

IterGP extends the most commonly used GP approximations to include computational uncertainty, with at most quadratic cost (see Table 3.1 for a summary and Figure 3.2, Figure B.2 for illustration).

Cholesky Decomposition The (partial) Cholesky decomposition iteratively chooses data points or pivots $\boldsymbol{x}_i$ based on a given ordering. The resulting Cholesky factor is lower triangular and increases in rank each iteration, and a well-chosen ordering achieves fast convergence (cf. [117, Thm. 2.3]). If one chooses standard unit vectors $\boldsymbol{e}_i$ as actions corresponding to the selected datapoint per iteration, then Algorithm 2 recovers the partial Cholesky factorization exactly (Theorem B.1).

Conjugate Gradients CG [71] with preconditioning for GP inference has become increasingly popular [3, 4, 111–114, 118, 119]. Algorithm 2 recovers preconditioned CG exactly if we choose either preconditioned conjugate gradients or residuals as actions (see Theorem B.3 and Corollary B.1). In fact, Algorithm 2 can even construct its own diagonal-plus-low-rank preconditioner by first running a few iterations with an arbitrary policy and then using the byproducts of these iterations for the preconditioner. For example, if we run IterGP-Chol initially, we can construct an incomplete Cholesky preconditioner for subsequent CG iterations.

Inducing Point Methods Inducing point methods, such as variants of the Nyström approximation [103], i.e. subset of regressors (SoR) [102, 120] and deterministic training conditional (DTC) [105, 121], as well as SVGP [109] share a posterior mean, which by Theorem B.4 takes the form

$$\mu_{\mathrm{SVGP}}(\cdot) = q(\cdot, \boldsymbol{X})\boldsymbol{K}_{\boldsymbol{XZ}}(\boldsymbol{K}_{\boldsymbol{ZX}}(q(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I})\boldsymbol{K}_{\boldsymbol{XZ}})^{-1}\boldsymbol{K}_{\boldsymbol{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}) \tag{3.6}$$

where $\boldsymbol{Z} \in \mathbb{R}^{n \times i}$ is a set of inducing points and $q(\cdot, \cdot) = k(\cdot, \boldsymbol{Z})\boldsymbol{K}_{\boldsymbol{ZZ}}^{-1}k(\boldsymbol{Z}, \cdot)$. These approximations also have very closely related posterior covariance functions [107, 122]. If we choose actions $\boldsymbol{s}_i = k(\boldsymbol{X}, \boldsymbol{z}_i)$, by Proposition B.1, Algorithm 2 returns a posterior mean given by

$$\mu_i(\cdot) = k(\cdot, \boldsymbol{X})\boldsymbol{K}_{\boldsymbol{XZ}}(\underbrace{\boldsymbol{K}_{\boldsymbol{ZX}}(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I})\boldsymbol{K}_{\boldsymbol{XZ}}}_{\text{Gram matrix } \boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i})^{-1}\boldsymbol{K}_{\boldsymbol{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}). \tag{3.7}$$

Choosing such actions, given by kernel functions $k(\cdot, \boldsymbol{z}_i)$ centered at inducing points $\boldsymbol{z}_i$, reduces computational uncertainty in regions close to inducing points (see IterGP-PI in Figure 3.2),

where closeness is determined by the kernel. Comparing SVGP's and IterGP-PI's posterior mean provides a probabilistic numerical perspective on why even for small KL-divergence between the approximating distribution of SVGP and the true posterior, the mean estimate can be far from the true mean [116, Prop. 3.1]. As outlined in Section 3.2, (3.7) is a Bayesian update on the initially unknown representer weights $\boldsymbol{v}_* = \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$. The Gram matrix in (3.7) describes how surprising the computational observations $\boldsymbol{K}_{\boldsymbol{ZX}}(\boldsymbol{y}-\boldsymbol{\mu}) = \boldsymbol{S}_i^\intercal(\boldsymbol{y}-\boldsymbol{\mu}) = \boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{v}_*$ of the representer weights should be, given the prior uncertainty $\boldsymbol{\Sigma}_0$ about them. SVGP uses a similar form for the posterior mean (c.f. (3.6) and (3.7)), but the Gram matrix is "smaller" since $q(\boldsymbol{X}, \boldsymbol{X}) \preceq k(\boldsymbol{X}, \boldsymbol{X})$. This can be interpreted as inducing point methods being overconfident in their update of the representer weight estimates to achieve linear time complexity. As the inducing points approach the data points the two posterior mean functions μ_{SVGP} and μ_i become closer and are equivalent if the inducing points equal the training data.

3.2.2 The Cost of Computational Uncertainty

For arbitrary actions Algorithm 2 has higher cost than linear time GP approximations such as inducing point methods, due to its use of matrix-vector multiplication as the computational primitive to condition on the data. IterGP in the form given in Algorithm 2 performs three matrix-vector products per iteration resulting in a quadratic time complexity $\mathcal{O}(n^2 i)$ overall for i iterations. In this sense, Algorithm 2 represents a middle ground between the mathematical posterior—which incurs a cubic time complexity— and $\mathcal{O}(ni^2)$ approximations—which can only estimate their computational error through potentially loose theoretical bounds which may [e.g. 108, 109, 123] or may not be computable in less than $\mathcal{O}(n^3)$ [91, 117]. At any point during a run of Algorithm 2, computing the predictive mean on $n_\diamond$ new data points has cost $\mathcal{O}(n_\diamond n)$, while the marginal predictive (co-)variance can be evaluated in $\mathcal{O}(n_\diamond n i)$ since $\boldsymbol{C}_i$ is of rank i. Additionally, using Matheron's rule [124–126], sampling from the approximate posterior at $n_\diamond$ evaluation points also only requires $\mathcal{O}(n_\diamond n i)$ computation (assuming we can sample from the prior—see Appendix B.3.3). The objects required to make predictions and draw samples are the approximate representer weights $\boldsymbol{v}_i$ and low-rank precision matrix approximation $\boldsymbol{C}_i$ which both require $\mathcal{O}(ni)$ memory. Finally, the memory cost of Algorithm 2 is only linear in n since matrix multiplication $\boldsymbol{v} \mapsto \hat{\boldsymbol{K}}\boldsymbol{v}$ with the kernel matrix can be performed in a matrix-free fashion, i.e. without explicitly forming $\hat{\boldsymbol{K}}$ in memory [127].

3.2.3 Related Work

GP inference based on matrix-vector multiplies, particularly CG [71], has become popular recently [3, 4, 92, 111–114, 118]. Advances in specialized hardware have boosted their scalability without excessive memory footprint [113, 127]. Such iterative methods typically rely on preconditioning, which has been shown to significantly improve their performance [3, 4, 112]. Our method generalizes CG in this setting and thus retains the same benefits. At its core Algorithm 2 employs a (Bayesian) probabilistic numerical method [51–54], more specifically a probabilistic linear solver (PLS) [1, 66, 67, 73, 128, 129] applied to the linear system $\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{y}$. The fact that a

PLS using CG actions can recover CG in its posterior mean was observed previously [1, 66, 73]. Here, we extend this result to residual actions and preconditioning. Further, we also demonstrate the connection to the Cholesky and singular value decompositions. For randomized actions, the PLS as part of Algorithm 2 also recovers the randomized Kaczmarz method in its posterior mean [130–133]. Employing a PLS for GP approximation by updating beliefs over the kernel and precision matrix was suggested previously [1, 134]. Our work differs in that it updates a belief over the representer weights, as opposed to the kernel function or matrix, considers more general projections than just conjugate residuals, and, most importantly, provides a theoretically motivated combined posterior which can be computed exactly.

3.3 Theoretical Analysis

The main goals of our theoretical analysis will be to prove *convergence of IterGP's posterior mean* in norm (Theorem 3.1) and pointwise (Corollary 3.1) and to provide rigorous justification for the combined and computational uncertainty. Importantly, the *combined uncertainty is a tight worst-case bound on the relative distance to all potential latent functions* consistent with our (computational) observations (Theorem 3.2). We will also demonstrate a similar interpretation of the computational uncertainty as a bound on the relative error to the mathematical posterior mean (see (3.13) and (3.15)).

3.3.1 Estimation of Representer Weights

At the heart of Algorithm 2 is a probabilistic linear solver [1, 66, 67, 73] iteratively updating a belief about the representer weights. It constructs an expanding subspace $\operatorname{span}(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_i) = \operatorname{span}(\boldsymbol{d}_1, \ldots, \boldsymbol{d}_i)$ spanned by the actions in which the inverse $\hat{\boldsymbol{K}}^{-1}$ is perfectly identified. Each step $\boldsymbol{d}_i$ expanding this explored subspace is $\hat{\boldsymbol{K}}$-orthogonal to the previous ones.

> **Proposition 3.1** (Conjugate Direction Method)
> *Let the actions $\boldsymbol{s}_i$ of Algorithm 2 be linearly independent. Then Algorithm 2 is a conjugate direction method, i.e. it holds that $\boldsymbol{d}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{d}_j = 0$ for all $i \neq j$.*

Proof. Without loss of generality assume $i > j$. Then the result follows directly from Lemma B.1. □

Geometrically, Algorithm 2 iteratively projects the representer weights onto the expanding subspace $\operatorname{span}(\boldsymbol{S}_i)$ with respect to $\langle \cdot\,, \cdot \rangle_{\hat{\boldsymbol{K}}}$. We can use this intuition to understand the convergence of the representer weights estimate. The relative error $\rho(i)$ at iteration i is given by how small the "angle" between this subspace and the representer weights vector is.

Proposition 3.2 (Relative Error Bound for the Representer Weights)
For any choice of actions a relative error bound $\rho(i)$, s.t. $\|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\hat{\boldsymbol{K}}} \leq \rho(i)\|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}$ is given by

$$\rho(i) = (\bar{\boldsymbol{v}}_*^\intercal \underbrace{(\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}})}_{\text{projection onto } \operatorname{span}(\boldsymbol{S}_i)^{\perp_{\hat{\boldsymbol{K}}}}} \bar{\boldsymbol{v}}_*)^{\frac{1}{2}} \leq \lambda_{\max}(\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}) \leq 1 \tag{3.8}$$

where $\bar{\boldsymbol{v}}_* = \boldsymbol{v}_*/\|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}$. If the actions $\{\boldsymbol{s}_i\}_{i=1}^n$ are linearly independent, then $\rho(i) \leq \delta_{n=i}$.

Proof. See Appendix B.2.2. □

Proposition 3.2 guarantees convergence in at most n iterations, if the actions are chosen to be linearly independent, since $\boldsymbol{C}_i\hat{\boldsymbol{K}}$ is a $\hat{\boldsymbol{K}}$-orthogonal projection onto $\operatorname{span}(\boldsymbol{S}_i)$ (see Lemma B.1). Therefore, if our finite computational budget is large enough, we eventually recover the mathematical posterior. This is reflected by the contraction of the posterior over the representer weights (see Proposition B.2). The bound in Proposition 3.2 is tight without further assumptions on the actions since there exists an adversarial sequence of actions such that the first $(n-1)$ are in $\operatorname{span}(\boldsymbol{v}_*)^{\perp_{\hat{\boldsymbol{K}}}}$. Then the inverse is perfectly identified in that subspace, but $\boldsymbol{v}_i = \boldsymbol{C}_i\boldsymbol{y} = \boldsymbol{C}_i\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{0}$. In practice, one can derive tighter convergence bounds for specific sequences of actions. For example, for randomized actions, the bound depends on their distribution [131, 132]. If residuals $\boldsymbol{r}_i$ are chosen as actions, we obtain

$$\rho(i) = 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \text{ or } \rho(i) = \left(\frac{\lambda_{n-i}-\lambda_1}{\lambda_{n-i}+\lambda_1}\right) \tag{3.9}$$

since then Algorithm 2's estimate of the representer weights equals that of CG (Corollary B.1). Here κ is the condition number and λ_j the eigenvalues of either (i) the kernel matrix $\hat{\boldsymbol{K}}$ if $\boldsymbol{s}_i = \boldsymbol{r}_i$, or (ii) the preconditioned kernel matrix $\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\frac{\intercal}{2}}$ if $\boldsymbol{s}_i = \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i$.

3.3.2 Convergence in RKHS Norm of the Posterior Mean

Having established convergence of the representer weights estimate, we can use this result to prove convergence in norm of IterGP's posterior mean to the mathematical posterior at the same rate.

Theorem 3.1 (Convergence in RKHS Norm of the Posterior Mean Approximation)
Let $\mathcal{H}_k$ be the RKHS associated with kernel $k(\cdot,\cdot)$, $\sigma^2 > 0$ and let $\mu_* - \mu \in \mathcal{H}_k$ be the unique solution to the regularized empirical risk minimization problem

$$\operatorname*{arg\,min}_{f\in\mathcal{H}_k} \frac{1}{n}\Big(\sum_{j=1}^n (f(\boldsymbol{x}_j) - y_j + \mu(\boldsymbol{x}_j))^2 + \sigma^2\|f\|_{\mathcal{H}_k}^2\Big) \tag{3.10}$$

which is equivalent to the mathematical posterior mean up to shift by the prior μ [e.g.

47, Sec. 6.2]. Then for $i \in \{0, \dots, n\}$ the posterior mean $\mu_i(\cdot)$ computed by Algorithm 2 satisfies

$$\boxed{\|\mu_* - \mu_i\|_{\mathcal{H}_k} \leq \rho(i) c(\sigma^2) \|\mu_* - \mu_0\|_{\mathcal{H}_k}} \tag{3.11}$$

where $\mu_0 = \mu$ is the prior mean and the constant $c(\sigma^2) = \sqrt{1 + \frac{\sigma^2}{\lambda_{\min}(\boldsymbol{K})}} \to 1$ as $\sigma^2 \to 0$.

Proof. See Appendix B.2.3. □

Theorem 3.1 gives a bound on the RKHS-norm error between the posterior mean μ_i of IterGP and the mathematical posterior mean μ_*. If for the given prior kernel a bound on the RKHS-norm error $\|h - \mu_*\|_{\mathcal{H}_k}$ between the latent function h and the mathematical posterior mean μ_* is known, Theorem 3.1 can be directly used to bound the RKHS-norm error between IterGP's posterior mean and the latent function h via the triangle inequality: $\|h - \mu_i\|_{\mathcal{H}_k} \leq \underbrace{\|h - \mu_*\|_{\mathcal{H}_k}}_{\to 0 \text{ as } n \to \infty} + \underbrace{\|\mu_* - \mu_i\|_{\mathcal{H}_k}}_{\to 0 \text{ as } i \to n}$.

3.3.3 Combined and Computational Uncertainty as Worst Case Errors

While Theorem 3.1 shows convergence in norm for IterGP's posterior mean, the convergence rate $\rho(i)$ may contain expressions that cannot be evaluated at runtime with the limited computation at our disposal. For example, evaluating (3.9) for residual actions requires the computation of the kernel matrix spectrum. However, the combined uncertainty of IterGP is a tight bound on the *pointwise* relative error to all possible latent functions which would have resulted in the same computations.

Theorem 3.2 (Combined and Computational Uncertainty as Worst Case Errors)
Let $\sigma^2 \geq 0$ and let $k_i(\cdot,\cdot) = k_*(\cdot,\cdot) + k_i^{\text{comp}}(\cdot,\cdot)$ be the combined uncertainty computed by Algorithm 2. Then, for any $\boldsymbol{x} \in \mathcal{X}$ (assuming $\boldsymbol{x} \notin \boldsymbol{X}$ if $\sigma^2 > 0$) we have

$$\sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} \overbrace{\underbrace{g(\boldsymbol{x}) - \mu_*^g(\boldsymbol{x})}_{\text{error of math. post. mean}} + \underbrace{\mu_*^g(\boldsymbol{x}) - \mu_i^g(\boldsymbol{x})}_{\text{computational error}}}^{\text{error of approximate posterior mean}} = \sqrt{k_i(\boldsymbol{x},\boldsymbol{x}) + \sigma^2}, \quad \text{and} \tag{3.12}$$

$$\sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} \underbrace{\mu_*^g(\boldsymbol{x}) - \mu_i^g(\boldsymbol{x})}_{\text{computational error}} = \sqrt{k_i^{\text{comp}}(\boldsymbol{x},\boldsymbol{x})} \tag{3.13}$$

where $\mu_*^g(\cdot) = k(\cdot, \boldsymbol{X}) \hat{\boldsymbol{K}}^{-1} g(\boldsymbol{X})$ is the mathematical and $\mu_i^g(\cdot) = k(\cdot, \boldsymbol{X}) \boldsymbol{C}_i g(\boldsymbol{X})$ IterGP's posterior mean for the latent function $g \in \mathcal{H}_{k^\sigma}$. If $\sigma^2 = 0$, then the above also holds for $\boldsymbol{x} \in \boldsymbol{X}$.

Proof. See Appendix B.2.4. □

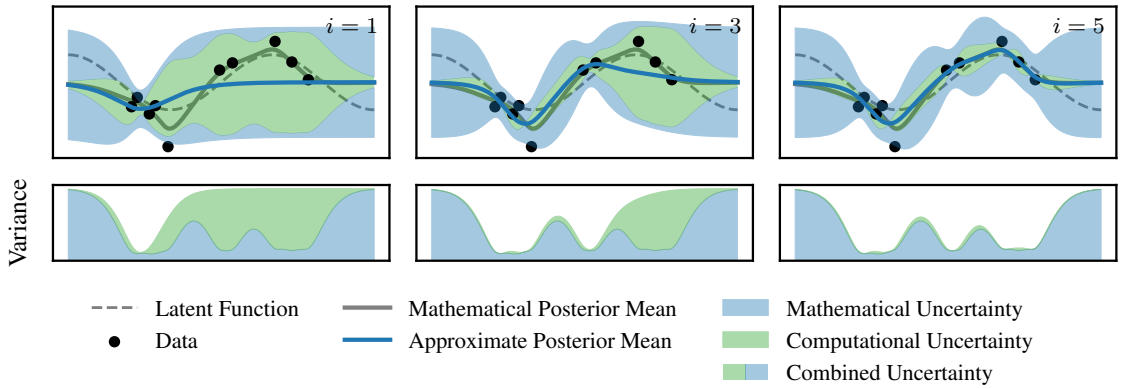


Figure 3.3: *Computational and combined uncertainty of IterGP as worst-case bounds.*[2]

Theorem 3.2 rigorously explains why the combined (mathematical + computational) uncertainty k_i is the correct object characterizing our belief about the latent function h, given that we are in the limited compute regime. In the same way that the mathematical uncertainty is a tight bound on the distance to all functions g which could have produced the data (see [44, Prop. 3.8]), the combined uncertainty is a tight bound on all functions g which would have produced the same computations.

3.3.4 Pointwise Convergence of the Posterior Mean

In particular, as Corollary 3.1 shows and Figure 3.3 illustrates, the computational uncertainty (🟢) is a pointwise bound on the relative distance to the mathematical posterior mean (3.15) and *the combined uncertainty* (🟢 + 🔵) *is a pointwise bound on the relative distance to the true latent function* (3.14).

Corollary 3.1 (Pointwise Convergence of the Posterior Mean)
Assume the conditions of Theorem 3.2 hold and assume the latent function $h \in \mathcal{H}_{k^\sigma}$. Let μ_* be the corresponding mathematical posterior mean and μ_i the posterior mean computed by Algorithm 2. Then it holds that

$$\frac{|h(\boldsymbol{x}) - \mu_i(\boldsymbol{x})|}{\|h\|_{\mathcal{H}_{k^\sigma}}} \leq \sqrt{k_i(\boldsymbol{x},\boldsymbol{x}) + \sigma^2}, \quad \textit{and} \tag{3.14}$$

$$\frac{|\mu_*(\boldsymbol{x}) - \mu_i(\boldsymbol{x})|}{\|h\|_{\mathcal{H}_{k^\sigma}}} \leq \sqrt{k_i^{\text{comp}}(\boldsymbol{x},\boldsymbol{x})}. \tag{3.15}$$

Proof. This follows from Theorem 3.2 by recognizing that $h/\|h\|_{\mathcal{H}_{k^\sigma}}$ has unit norm. □

[2] The combined (co-)variance decomposes into mathematical and computational covariances, as opposed to the combined standard deviation since $\sqrt{🟢 + 🔵} \neq \sqrt{🟢} + \sqrt{🔵}$. The bottom panel thus illustrates the variance decomposition. However, to better illustrate Theorem 3.2, in the upper panel, we plot the combined standard deviation $\sqrt{🟢 + 🔵}$ and computational standard deviation $\sqrt{🟢}$ within it, in line with standard GP plotting practice.

It is worth noting that Theorem 3.2 and Corollary 3.1 *do not hold for other GP approximations*. They explicitly rely on $\boldsymbol{C}_i\hat{\boldsymbol{K}}$ being the $\hat{\boldsymbol{K}}$-orthogonal projection onto the space spanned by the actions (see Lemma B.1). Since orthogonal projections are unique if another GP approximation is such a projection and therefore satisfies Theorem 3.2, it is in fact an instance of IterGP.

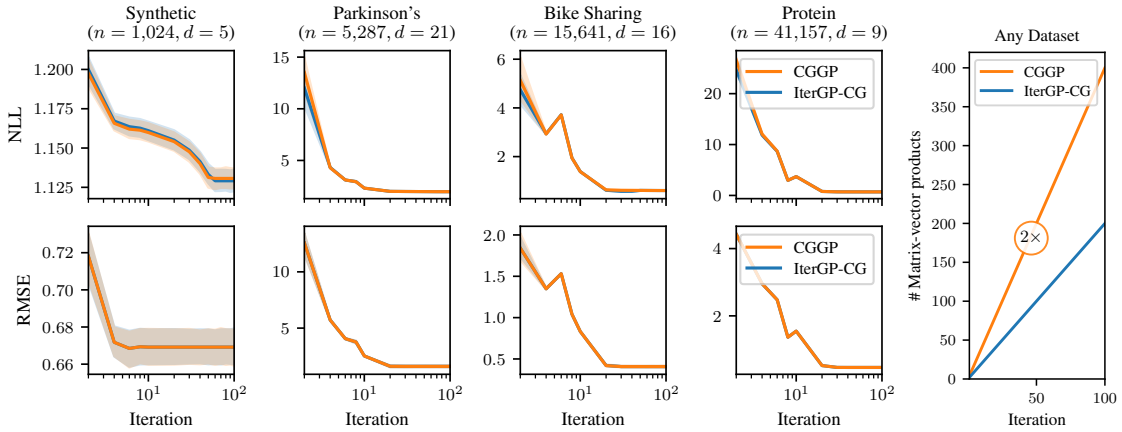
3.4 Experiments

To demonstrate the effects of quantifying computational uncertainty we perform GP regression on synthetic and benchmark datasets for the two most common GP approximations in the large-scale setting, SVGP [109] and CGGP [3], and their direct analogs from our class of methods. An implementation of Algorithm 2, based on KeOps [127] and ProbNum [55], is available at:

`https://github.com/JonathanWenger/itergp`

Experimental Setup We consider a synthetic dataset of iid uniformly sampled inputs $\boldsymbol{x}_j \in [-1, 1]^d$ with $y(\boldsymbol{x}) = \sin(\pi \boldsymbol{x}^\intercal \mathbf{1}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}\big(0, \sigma^2\big)$, as well as a range of UCI datasets [135] with training set sizes $n = 5,287$ to $57,247$, dimensions $d = 9$ to 26 and standardized features. All experiments were run on an NVIDIA GeForce RTX 2080 Ti graphics card. We perform GP regression using a zero mean prior and a Matérn($\frac{1}{2}$) kernel (for other kernels see Appendix B.4). All experiments were repeated 10 times with randomly sampled training and test splits of proportions 0.9 and 0.1. We report average metrics with 95% confidence intervals.

IterGP reduces the necessary computations for CG-based GP inference. We compare IterGP to the CG-based GP inference used in the GPyTorch library [3]. For all datasets, we select hyperparameters using the training procedure of Wenger et al. [4]. As we show in Theorem B.3, the posterior mean of IterGP with (conjugate) residual actions is exactly equivalent to performing CG to compute the representer weights. Therefore, both methods produce the exact same posterior mean estimate and thus achieve the same RMSE as a function of CG iterations (Figure 3.4, bottom). The primary difference between the two methods is in the posterior variance. The combined variance estimate of IterGP is essentially "free" in the sense that it reuses terms from the posterior mean calculation. In contrast, computing the posterior variance with CG requires $n_\diamond$ additional linear solves ($\hat{\boldsymbol{K}}^{-1}\boldsymbol{x}_{\diamond 1}, \ldots, \hat{\boldsymbol{K}}^{-1}\boldsymbol{x}_{\diamond n_\diamond}$). GPyTorch relies on the Lanczos Variance Estimate technique [136] which essentially warm-starts each of these solves by reusing quantities from the linear solve $\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \boldsymbol{x}_{\diamond 1})$. While this approach produces reliable variance estimates that converge to the true posterior variance, it requires additional computation: at least one set of additional CG iterations to compute $\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \boldsymbol{x}_{\diamond 1})$. In Figure 3.4(a) (top), we see that IterGP and GPyTorch's CGGP achieve nearly identical NLL, suggesting that both methods produce variances that yield similar generalization. The key difference between the methods is that 1) unlike CGGP, IterGP's variances exactly capture both mathematical and computational uncertainty, and 2) IterGP's variances require no additional solves, resulting in half as much computation as GPyTorch's CGGP implementation (see Figure 3.4(b)).



(a) Generalization on synthetic and UCI benchmark datasets. (b) Comp. Cost

Figure 3.4: *Generalization of CGGP and its closest IterGP analog.* (a) GP regression using a Matérn($\frac{1}{2}$) kernel on UCI datasets. The plot shows the average generalization error in terms of NLL and RMSE for an increasing number of solver iterations. The posterior mean of IterGP-CG and CGGP is identical, which explains the identical RMSE. However, CGGP performs additional computation for the posterior covariance as (b) illustrates, which is not needed since IterGP-CG has identical NLL.

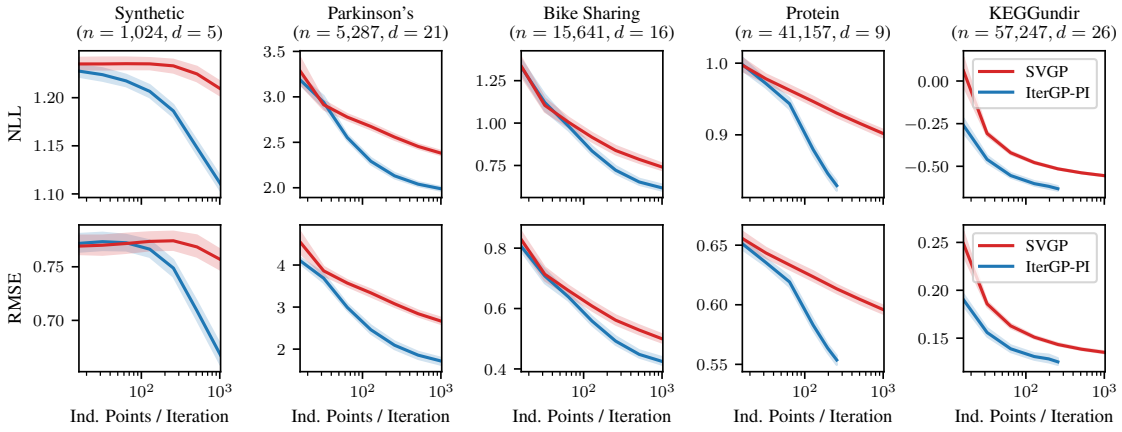


Figure 3.5: *Generalization of SVGP and its closest IterGP analog.* GP regression using a Matérn($\frac{1}{2}$) kernel on UCI datasets. The plot shows the average generalization error in terms of NLL and RMSE for an increasing number of identical inducing points. After a small number of inducing points relative to the size of the training data, IterGP has a significantly lower generalization error than SVGP.

Computational uncertainty improves generalization of inducing point methods. To understand the benefits of quantifying computational uncertainty, we compare the linear-time SVGP method (which does not quantify computational uncertainty) with the closest (quadratic-time) inducing point analog from our proposed IterGP framework (see Section 3.2.1). While the IterGP method is inherently more expensive than SVGP, our goal is simply to demonstrate that inducing points can yield far more accuracy if one has the budget to account for computational uncertainty. To that end, we compare SVGP against IterGP using the same set of randomly placed inducing points. We identify a set of kernel hyperparameters by optimizing the ELBO of SVGP on the training data, using these for both SVGP and IterGP. As Figure 3.5 shows, we find that across all datasets IterGP offers better RMSE and NLL than SVGP, despite the fact that the hyperparameters are chosen to favor SVGP. This suggests that the extra computation needed to quantify computational uncertainty can more "effectively" utilize a set of inducing points for predictive models.

3.5 Conclusion

Scalable GP approximations inevitably introduce error, leading to a worse model for the latent function in question. This work demonstrates that it is possible to account for *both* uncertainty arising from limited data *and* uncertainty arising from limited computation *exactly* – which as we show improves model performance. IterGP methods return this combined uncertainty which crucially represents a dataset-specific, pointwise worst-case bound on the error to the true latent function. At its core, IterGP performs repeated matrix-vector multiplication resulting in quadratic complexity. Since modern computing architectures (i.e. GPUs) have been specifically designed for this operation at scale, iterative approaches for GP approximation are becoming competitive with theoretically cheaper approximations, like inducing point methods [3, 113]. Finally, in addition to the general utility of IterGP, we expect this class of methods to be particularly useful in applications where accurate uncertainty quantification is important or, due to its inherently online nature, where data is acquired sequentially such as in active learning and Bayesian optimization.

Chapter

4

Preconditioning for Scalable GP Hyperparameter Optimization

Gaussian process hyperparameter optimization requires linear solves with, and log-determinants of, large kernel matrices. Iterative numerical techniques are becoming popular to scale to larger datasets, relying on the conjugate gradient method (CG) for the linear solves and stochastic trace estimation for the log-determinant. This work introduces new algorithmic and theoretical insights for preconditioning these computations. While preconditioning is well understood in the context of CG, we demonstrate that it can also accelerate convergence and reduce the variance of the estimates for the log-determinant and its derivative. We prove general probabilistic error

bounds for the preconditioned computation of the log-determinant, log-marginal likelihood and its derivatives. Additionally, we derive specific rates for a range of kernel-preconditioner combinations, showing that up to exponential convergence can be achieved. Our theoretical results enable provably efficient optimization of kernel hyperparameters, which we validate empirically on large-scale benchmark problems. There, our approach accelerates training by up to an order of magnitude.

4.1 Introduction

Gaussian processes (GPs) are a theoretically well-founded and powerful probabilistic model [47]. However, conditioning a GP on data is often computationally prohibitive for large datasets. This problem is amplified when optimizing kernel hyperparameters. Gradient-based optimization requires repeated evaluation of the log-marginal likelihood $\mathcal{L}$ and its derivatives. These computations both have cubic complexity in the size n of the data set.

GP Inference via Matrix-Vector Multiplication Recently, Krylov methods [59], based on iterative matrix-vector multiplication with the kernel matrix, have become popular for GP inference [3, 111–113, 137–139]. Methods primarily relying on matrix-vector products are advantageous. They can leverage structure in the kernel matrix [65], and importantly, they make effective use of modern hardware and parallelization [3, 113, 127]. When optimizing GP hyperparameters one needs to repeatedly solve linear systems with, and compute log-determinants of, the kernel matrix. Both can be done using Krylov methods. The linear systems are solved via the conjugate gradient method (CG) [71], which reduces the cost of kernel matrix solves from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2 i)$ for i iterations. The log-determinant can be approximated via stochastic trace estimation (STE) [140] combined with another Krylov method, the Lanczos algorithm [77] as suggested by Ubaru, Chen, and Saad [138]. Its derivative may also be estimated via STE combined with CG [3].

Challenges with this Approach Despite the advantages of combining Krylov methods with stochastic trace estimation, there are considerable challenges in practice. These essentially reduce to bias and variance of the numerical approximations. First, the convergence of CG depends on the conditioning of the kernel matrix, which can grow rapidly with n (e.g. for the RBF kernel). Many iterations may be needed to achieve a desired error, and stopping the solver early can result in biased solutions [119]. However, if a preconditioner – i.e. an approximation of the kernel matrix – is available, convergence can be accelerated substantially [59]. Second, stochastic approximations of the log-determinant and its derivative introduce variance into hyperparameter optimization. While the estimates are unbiased (assuming sufficient Krylov iterations), variance can significantly slow down optimization. Reducing variance either requires further approximation at the cost of more bias [114], or a larger number of samples ℓ which only reduces error at a rate of $\mathcal{O}(\ell^{-\frac{1}{2}})$ [141]. Now, while preconditioning is known to accelerate CG, it has not yet been explored for stochastic trace estimation in this context.

Contributions We demonstrate that, with only a small algorithmic modification, preconditioning can be exploited for highly efficient log-determinant estimation, and in turn GP hyperparameter optimization. We show that

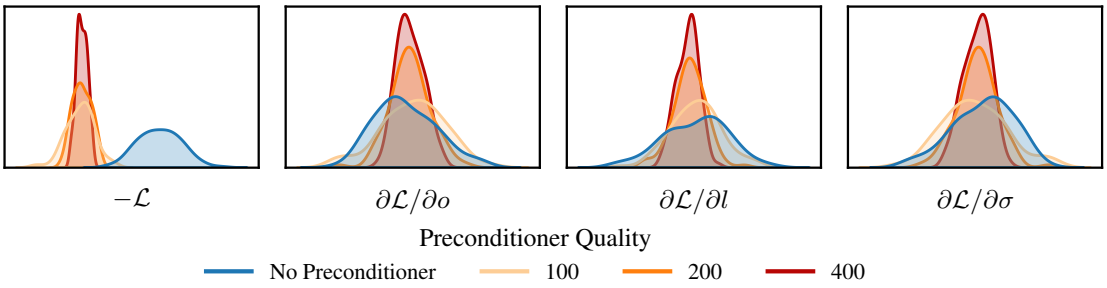


Figure 4.1: *Preconditioning reduces not only bias but also variance* in stochastic approximations to the log-marginal likelihood $\mathcal{L}$ and its derivatives. GP hyperparameter optimization for large datasets requires cheap estimates of the log-marginal likelihood and its gradient(s). Preconditioning makes these estimates more precise and less noisy as is shown here for increasing preconditioner quality on the "Elevators" dataset using a Matérn$(\frac{3}{2})$ kernel with the hyperparameters: outputscale o, lengthscale l and observation noise σ^2.

(a) *preconditioning reduces variance* – or equivalently accelerates convergence – of the stochastic estimate of the log-determinant and its derivative (Theorem 4.1).

We leverage this result, illustrated in Figure 4.1, to prove

(b) *stronger theoretical guarantees* for the computation of the log-determinant (Theorems 4.2 and 4.3) and log-marginal likelihood (Theorem 4.4) than previously known [3, 138] and a *novel error bound for the derivative* (Theorem 4.5).

To make these general results concrete, we derive

(c) *specific rates for important combinations of kernels and preconditioners* (Table 4.1), making preconditioner choice for GP inference rigorous rather than heuristic.

Finally, using our approach, we empirically observe

(d) *up to twelvefold speedup in training* of GP regression models applied to large-scale benchmark problems with up to $n \approx 325{,}000$ data points.

4.2 Background

We want to infer a latent relationship from an input space $\mathbb{X} \subset \mathbb{R}^d$ to an output space $\mathbb{Y} \subset \mathbb{R}$, given a dataset $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ of n training inputs $\boldsymbol{x}_j \in \mathbb{R}^d$ and outputs $\boldsymbol{y} \in \mathbb{R}^n$.

4.2.1 Gaussian Processes

A stochastic process $f \sim \mathcal{GP}(\mu, k)$ with mean function μ and kernel k is called a *Gaussian process* if $\boldsymbol{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$ is jointly Gaussian with mean $\boldsymbol{\mu}_j = \mu(\boldsymbol{x}_j)$

and covariance $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Assuming $\boldsymbol{y} \mid \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}, \sigma^2 \boldsymbol{I})$, the posterior distribution for test inputs $\boldsymbol{x}_\diamond$ is also Gaussian with

$$\begin{aligned}\mathbb{E}(\boldsymbol{f}_\diamond) &= \mu(\boldsymbol{x}_\diamond) + k(\boldsymbol{x}_\diamond, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}),\\ \mathrm{Cov}(\boldsymbol{f}_\diamond) &= k(\boldsymbol{x}_\diamond, \boldsymbol{x}_\diamond) - k(\boldsymbol{x}_\diamond, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \boldsymbol{x}_\diamond),\end{aligned}$$

where $\hat{\boldsymbol{K}} = \boldsymbol{K} + \sigma^2\boldsymbol{I}$. Without loss of generality, we assume $\mu = 0$ from now on.

Hyperparameter Optimization The computational bottleneck when optimizing kernel hyperparameters $\boldsymbol{\theta}$ is the repeated evaluation of the log-*marginal likelihood*

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\\ &= -\frac{1}{2}\big(\boldsymbol{y}^\intercal \hat{\boldsymbol{K}}^{-1}\boldsymbol{y} + \underbrace{\log\det(\hat{\boldsymbol{K}})}_{=\mathrm{tr}(\log(\hat{\boldsymbol{K}}))} + n\log(2\pi)\big)\end{aligned} \tag{4.1}$$

and its *derivative* with respect to the hyperparameters

$$\tfrac{\partial}{\partial\theta}\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{y}^\intercal \hat{\boldsymbol{K}}^{-1}\tfrac{\partial\hat{\boldsymbol{K}}}{\partial\theta}\hat{\boldsymbol{K}}^{-1}\boldsymbol{y} - \frac{1}{2}\,\mathrm{tr}(\hat{\boldsymbol{K}}^{-1}\tfrac{\partial\hat{\boldsymbol{K}}}{\partial\theta}). \tag{4.2}$$

Computing (4.1) and (4.2) via a Cholesky decomposition has complexity $\mathcal{O}(n^3)$ which is prohibitive for large n. In response, many methods for approximate GP inference were developed [92, 93, 108, 109, 136]. In contrast to these approaches, we avoid approximating the underlying Gaussian process model and instead leverage iterative numerical methods with controllable accuracy to scale to the large-scale setting similar to Gardner et al. [3] and Wang et al. [113]. To achieve this, we focus on efficient computation of the log-determinant in (4.1) and its derivative in (4.2) (see Section 4.3). This allows us to theoretically (Section 4.4) and empirically (Section 4.5) accelerate GP hyperparameter optimization.

4.2.2 Numerical Toolbox for Inference

We will use the following established numerical techniques.

Stochastic Trace Estimation (STE) The trace $\mathrm{tr}(\boldsymbol{A})$ of a matrix can be approximated by drawing ℓ independent random vectors $\boldsymbol{z}_j$ with $\mathbb{E}(\boldsymbol{z}_j) = \boldsymbol{0}$ and $\mathrm{Cov}(\sqrt{n}\boldsymbol{z}_j) = \boldsymbol{I}$ and computing *Hutchinson's estimator* [140]

$$\tau_\ell^{\mathrm{STE}}(\boldsymbol{A}) = \frac{n}{\ell}\sum_{j=1}^{\ell}\boldsymbol{z}_j^\intercal\boldsymbol{A}\boldsymbol{z}_j \approx \mathrm{tr}(\boldsymbol{A}). \tag{4.3}$$

Here, we additionally assume the random vectors are normalized $\boldsymbol{z}_j = \tilde{\boldsymbol{z}}_j/\|\tilde{\boldsymbol{z}}_j\|_2$ and that $\sqrt{n}(\boldsymbol{z}_1, \dots, \boldsymbol{z}_\ell)^\intercal \in \mathbb{R}^{\ell n}$ satisfies the convex concentration property (see Definition C.1). Normalization is necessary for Lanczos quadrature [see 142, Chap. 7.2] and concentration enables us to prove probabilistic error bounds. These assumptions are fulfilled by Rademacher-distributed random vectors $\tilde{\boldsymbol{z}}_j$ with entries $\{+1, -1\}$ and we conjecture also for vectors $\tilde{\boldsymbol{z}}_j \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Evaluating the log-marginal likelihood (4.1) requires computing $\mathrm{tr}(\log(\hat{\boldsymbol{K}}))$. To use Hutchinson's estimator, we need to efficiently compute ℓ quadratic terms $\{\boldsymbol{z}_j^\intercal\log(\hat{\boldsymbol{K}})\boldsymbol{z}_j\}_{j=1}^{\ell}$.

Stochastic Lanczos Quadrature (SLQ) Given a matrix function f, one can approximate bilinear forms $\boldsymbol{z}^\intercal f(\hat{\boldsymbol{K}})\boldsymbol{z}$ using quadrature [142, Chap. 7]. The nodes and weights of the quadrature rule can be computed efficiently via i iterations of the *Lanczos algorithm* [77] (or equivalently via CG [3]). The combination with Hutchinson's estimator

$$\tau_{\ell,i}^{\mathrm{SLQ}}(f(\hat{\boldsymbol{K}})) \approx \tau_{\ell}^{\mathrm{STE}}(f(\hat{\boldsymbol{K}})) \approx \mathrm{tr}(f(\hat{\boldsymbol{K}})), \tag{4.4}$$

is called *stochastic Lanczos quadrature* [138].

To compute the linear solves $\boldsymbol{v} \mapsto \hat{\boldsymbol{K}}^{-1}\boldsymbol{v}$ with the kernel matrix in (4.1) and (4.2), we use the *conjugate gradient method*.

Conjugate Gradient Method (CG) CG [71] is an iterative method for solving linear systems with symmetric positive definite matrices. It is particularly suited for large-scale systems since it is matrix-free and relies primarily on matrix-vector multiplication with $\hat{\boldsymbol{K}}$.

Preconditioning It is well-known that CG can be accelerated via a symmetric positive definite preconditioner $\hat{\boldsymbol{P}} \approx \hat{\boldsymbol{K}}$, by solving an equivalent linear system with matrix $\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\intercal}_{2} \approx \boldsymbol{I}$ [58]. CG's convergence is then determined by the condition number

$$\kappa \coloneqq \kappa(\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\intercal}_{2}) \ll \kappa(\hat{\boldsymbol{K}}) = \frac{|\lambda_{\max}(\hat{\boldsymbol{K}})|}{|\lambda_{\min}(\hat{\boldsymbol{K}})|}. \tag{4.5}$$

Suppose the approximation quality of a sequence of preconditioners $\{\hat{\boldsymbol{P}}_\ell\}_\ell$ indexed by ℓ[1] is given by

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq \mathcal{O}(g(\ell))\|\hat{\boldsymbol{K}}\|_F. \tag{4.6}$$

If $g(\ell) \to 0$ quickly, a small amount of precomputation can significantly accelerate CG, since by Lemma C.4

$$\kappa \leq (1 + \mathcal{O}(g(\ell))\|\hat{\boldsymbol{K}}\|_F)^2. \tag{4.7}$$

Preconditioners must be cheap to obtain and allow efficient linear solves $\boldsymbol{v} \mapsto \hat{\boldsymbol{P}}^{-1}\boldsymbol{v}$.[2] As an example, *diagonal-plus-low-rank* preconditioners $\hat{\boldsymbol{P}}_\ell = \sigma^2\boldsymbol{I} + \boldsymbol{L}_\ell\boldsymbol{L}_\ell^\intercal$, with $\boldsymbol{L}_\ell \in \mathbb{R}^{n\times\ell}$, admit linear solves in $\mathcal{O}(n\ell^2)$ via the matrix inversion lemma.

4.3 Log-Determinant Estimation

Our goal is to compute $\log\det(\hat{\boldsymbol{K}}) = \mathrm{tr}(\log(\hat{\boldsymbol{K}}))$ and its derivative via matrix-vector multiplication. As described, we can use stochastic trace estimation to do so. Now assume we additionally have access to a preconditioner $\hat{\boldsymbol{P}} \approx \hat{\boldsymbol{K}}$. As we will show, we can then not just accelerate the convergence of CG, but *also* more efficiently compute the forward and backward pass for the log-determinant.

[1] The use of ℓ for the number of random vectors and the preconditioner sequence is deliberate. Setting them to the same value enables variance reduction as we prove in Theorem 4.1.

[2] While CG (and Lanczos) assume an s.p.d. matrix, both can be implemented using only $\hat{\boldsymbol{P}}^{-1}$, not $\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\intercal}_{2}$.

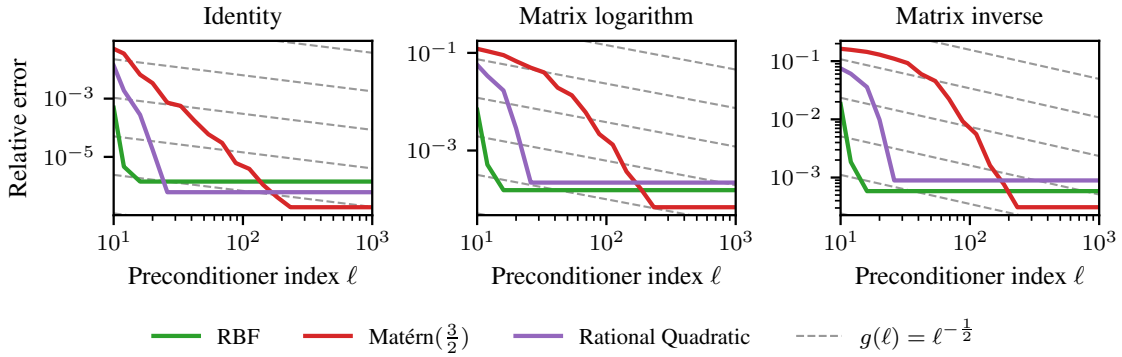


Figure 4.2: *Relative error of matrix functions.* The plot shows the relative error in Frobenius norm $\|f(\hat{\boldsymbol{K}}) - f(\boldsymbol{P}_\ell)\|_F / \|f(\hat{\boldsymbol{K}})\|_F$ of the approximation of a matrix function $f \in \{\mathrm{id}, \log, (\cdot)^{-1}\}$ applied to $\hat{\boldsymbol{K}}$ using a preconditioner $\boldsymbol{P}_\ell$. For analytic functions f the approximation via the preconditioner $f(\hat{\boldsymbol{P}}_\ell) \to f(\hat{\boldsymbol{K}})$ converges at the asymptotic rate of the preconditioner $\hat{\boldsymbol{P}}_\ell \to \hat{\boldsymbol{K}}$. Here, we use a partial Cholesky preconditioner on a synthetic dataset ($n = 1{,}000$).

By the properties of the matrix logarithm we can decompose the log-determinant into a *deterministic approximation* based on the preconditioner and a *residual trace* computed via stochastic trace estimation.[3] It holds by Lemma C.3, that

$$\log\det(\hat{\boldsymbol{K}}) = \log\det(\hat{\boldsymbol{P}}_\ell) + \mathrm{tr}(\underbrace{\log(\hat{\boldsymbol{K}}) - \log(\hat{\boldsymbol{P}}_\ell)}_{=\boldsymbol{\Delta}_{\log}}) \tag{4.8}$$

where $\mathrm{tr}(\boldsymbol{\Delta}_{\log}) = \mathrm{tr}\big(\log\big(\hat{\boldsymbol{P}}_\ell^{-\frac{1}{2}} \hat{\boldsymbol{K}} \hat{\boldsymbol{P}}_\ell^{-\intercal\frac{1}{2}}\big)\big)$ and we assume $\log\det(\hat{\boldsymbol{P}}_\ell)$ is efficient to compute. (4.8) has two crucial benefits we can exploit. First and foremost, the faster $\hat{\boldsymbol{P}}_\ell \to \hat{\boldsymbol{K}}$, i.e. $g(\ell) \to 0$, the less the stochastic approximation of $\mathrm{tr}(\boldsymbol{\Delta}_{\log})$ affects the estimate. Since its contribution to the overall error decreases the better $\log\det(\hat{\boldsymbol{P}}_\ell)$ approximates $\log\det(\hat{\boldsymbol{K}}_\ell)$, significantly fewer random vectors are needed to achieve a desired error with high probability. Second, we can now run Lanczos on the preconditioned matrix accelerating its convergence. As we will show later, one can also exploit (4.8) for the backward pass.

4.3.1 Variance-reduced Stochastic Trace Estimation

This intuitive argument for the log-determinant also holds generally, assuming a similar decomposition exists.

Theorem 4.1 (Variance-reduced Stochastic Trace Estimation)
Let $\hat{\boldsymbol{K}}, \hat{\boldsymbol{P}}_\ell \in \mathbb{R}^{n\times n}_{\mathrm{spd}}, \boldsymbol{\Delta}_f \in \mathbb{R}^{n\times n}$ *and* $f : \mathbb{R}^{n\times n}_{\mathrm{spd}} \to \mathbb{R}$ *such that* $\mathrm{tr}(f(\hat{\boldsymbol{K}})) = \mathrm{tr}(f(\hat{\boldsymbol{P}}_\ell)) + \mathrm{tr}(\boldsymbol{\Delta}_f)$*, and define the estimator* $\tau_* = \mathrm{tr}(f(\hat{\boldsymbol{P}}_\ell)) + \tau_\ell^{\mathrm{STE}}(\boldsymbol{\Delta}_f)$*. Now, assume there exist*

[3] Similar approaches have been suggested by Adams et al. [143] and Meyer et al. [144]. Our work is notably different in that it a) uses preconditioning, b) also considers the backward pass and c) gives stronger theoretical guarantees.

$c_{\boldsymbol{\Delta}} > 0$ *and* $g : \mathbb{N} \to (0, \infty)$ *such that*

$$\|\boldsymbol{\Delta}_f\|_F \leq c_{\boldsymbol{\Delta}} g(\ell) \|f(\hat{\boldsymbol{K}})\|_F. \tag{4.9}$$

Then there exists $c_{\boldsymbol{z}} > 0$ *dependent on the choice of random vectors, such that, if* $\ell \geq c_{\boldsymbol{z}} \log(\delta^{-1})$*, it holds with probability* $1 - \delta \in [\frac{1}{2}, 1)$ *that*

$$|\tau_* - \operatorname{tr}(f(\hat{\boldsymbol{K}}))| \leq \varepsilon_{\mathrm{STE}} \|f(\hat{\boldsymbol{K}})\|_F. \tag{4.10}$$

where for $C_1 = c_{\boldsymbol{\Delta}} \sqrt{c_{\boldsymbol{z}}}$ *the relative error is given by*

$$\varepsilon_{\mathrm{STE}}(\delta, \ell) = C_1 \sqrt{\log(\delta^{-1})} \ell^{-\frac{1}{2}} g(\ell). \tag{4.11}$$

Proof. See Appendix C.2. □

Notice that Theorem 4.1 assumes that the sequence $\{f(\hat{\boldsymbol{P}}_\ell)\}_\ell$ approximates $f(\hat{\boldsymbol{K}})$ sufficiently fast with ℓ in (4.9). Intuitively, if $\hat{\boldsymbol{P}}_\ell \to \hat{\boldsymbol{K}}$ quickly, one might expect the same for $f(\hat{\boldsymbol{P}}_\ell) \to f(\hat{\boldsymbol{K}})$ under certain conditions on f. Indeed, one obtains the same asymptotic rate $g(\ell)$ of the preconditioner $\hat{\boldsymbol{P}}_\ell$ for the approximation of $f(\hat{\boldsymbol{K}})$ by $f(\hat{\boldsymbol{P}})$ (see Proposition C.1). This is illustrated in Figure 4.2. Therefore, the error of the variance-reduced stochastic trace estimate is *determined by the quality* $g(\ell)$ *of the preconditioner*.

Comparison of Theorem 4.1 and Existing Results Consider the case where $f = \mathrm{id}$. If we are not using a preconditioner, i.e. $\hat{\boldsymbol{P}} = \boldsymbol{0}$ and thus $c_{\boldsymbol{\Delta}} = g(\ell) = 1$, we recover the well-known convergence rate $\mathcal{O}(\ell^{-\frac{1}{2}})$ of Hutchinson's estimator [141, 145]. If instead we choose a randomized low-rank approximation as a preconditioner with $g(\ell) = \ell^{-\frac{1}{2}}$, then Theorem 4.1 recovers the convergence rate $\varepsilon_{\mathrm{STE}} \in \mathcal{O}(\ell^{-1})$ of Hutch++ [144, 146, 147] as a special case. However, as we will show, using preconditioning one can achieve polynomial – even exponential – convergence rates for common kernels. Such a drastic improvement is possible since neither variant of Hutchinson's makes any assumptions about the kernel matrix, whereas preconditioners are designed to leverage structure.

4.3.2 Forward Pass

We can now analyze the error of the preconditioned stochastic log-determinant estimate. Combining Theorem 4.1 with Lanczos quadrature error analysis, the following holds.

Theorem 4.2 (Error Bound for $\log\det(\hat{\boldsymbol{K}})$)
Let $f = \log$, $\boldsymbol{\Delta}_{\log} = \log\left(\hat{\boldsymbol{P}}^{-\frac{1}{2}} \hat{\boldsymbol{K}} \hat{\boldsymbol{P}}^{-\frac{1}{2}}\right)$ *and assume the conditions of Theorem 4.1 hold.*

Then, with probability $1-\delta$*, it holds for* $\tau_*^{\log} = \log(\det(\hat{\boldsymbol{P}})) + \tau_{\ell,i}^{\mathrm{SLQ}}(\boldsymbol{\Delta}_{\log})$*, that*

$$|\tau_*^{\log} - \log\det(\hat{\boldsymbol{K}})| \leq (\varepsilon_{\mathrm{Lanczos}} + \varepsilon_{\mathrm{STE}})\|\log(\hat{\boldsymbol{K}})\|_F,$$

where the individual errors are bounded by

$$\varepsilon_{\mathrm{Lanczos}}(\kappa, i) \leq K_1 \left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i} \tag{4.12}$$

$$\varepsilon_{\mathrm{STE}}(\delta, \ell) \leq C_1\sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}}g(\ell) \tag{4.13}$$

and $K_1 = \frac{5\kappa\log(2(\kappa+1))}{2\|\log(\hat{\boldsymbol{K}})\|_F\sqrt{2\kappa+1}}$.

Proof. See Appendix C.3.2. □

Corollary 4.1
Assume the conditions of Theorem 4.2 hold. If the number of random vectors ℓ *satisfies* (4.11) *with* $\varepsilon_{\mathrm{STE}} = \frac{\varepsilon}{2}$ *and we run*

$$i \geq \frac{\sqrt{3}}{4}\sqrt{\kappa}\log\left(2K_1\varepsilon^{-1}\right) \tag{4.14}$$

iterations of Lanczos, then it holds that

$$\boxed{\mathrm{P}\left(|\tau_*^{\log} - \log\det(\hat{\boldsymbol{K}})| \leq \varepsilon\|\log(\hat{\boldsymbol{K}})\|_F\right) \geq 1-\delta.}$$

Proof. See Appendix C.3.2. □

We note two major improvements over the bound by Ubaru, Chen, and Saad [138, Corollary 4.5]. First, the number of Lanczos steps now depends on the condition number κ of the *preconditioned* matrix, implying faster convergence. Second, depending on the preconditioner quality $g(\ell)$, we need significantly fewer random vectors by Theorem 4.1.

4.3.3 Backward Pass

By differentiating through (4.8), we obtain a decomposition into a *deterministic approximation* based on the preconditioner and a *residual trace* for the backward pass. For $\boldsymbol{\Delta}_{\mathrm{inv}\partial} = \hat{\boldsymbol{K}}^{-1}\frac{\partial\hat{\boldsymbol{K}}}{\partial\theta} - \hat{\boldsymbol{P}}^{-1}\frac{\partial\hat{\boldsymbol{P}}}{\partial\theta}$, we have

$$\frac{\partial}{\partial\theta}\log\det(\hat{\boldsymbol{K}}) = \mathrm{tr}(\hat{\boldsymbol{P}}^{-1}\frac{\partial\hat{\boldsymbol{P}}}{\partial\theta}) + \mathrm{tr}(\boldsymbol{\Delta}_{\mathrm{inv}\partial}), \tag{4.15}$$

Therefore the stochastic trace estimator

$$\tau_{\ell,i}^{\mathrm{SCG}}(\boldsymbol{\Delta}_{\mathrm{inv}\partial}) \approx \mathrm{tr}(\boldsymbol{\Delta}_{\mathrm{inv}\partial}) \tag{4.16}$$

requires solves $\boldsymbol{z}_j^\intercal \hat{\boldsymbol{K}}^{-1} \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta} \boldsymbol{z}_j$ and $\boldsymbol{z}_j^\intercal \hat{\boldsymbol{P}}^{-1} \frac{\partial \hat{\boldsymbol{P}}}{\partial \theta} \boldsymbol{z}_j$. The former can be computed with i iterations of preconditioned CG, while the latter is simply a solve with the preconditioner. The deterministic term $\operatorname{tr}\left(\hat{\boldsymbol{P}}^{-1} \frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}\right)$ is efficient to calculate for many types of preconditioners. For example, if $\hat{\boldsymbol{P}}$ is a diagonal-plus-low-rank preconditioner it can be computed in $\mathcal{O}(n\ell^2)$ (see Appendix C.3.3). Using Theorem 4.1, we obtain a probabilistic error bound for the derivative estimate.

Theorem 4.3 (Error Bound for $\operatorname{tr}\left(\hat{\boldsymbol{K}}^{-1} \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\right)$)
Let $f(\hat{\boldsymbol{K}}) = \hat{\boldsymbol{K}}^{-1} \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}$, $\boldsymbol{\Delta}_{\text{inv}\partial} = \hat{\boldsymbol{K}}^{-1} \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta} - \hat{\boldsymbol{P}}^{-1} \frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}$ and assume the conditions of Theorem 4.1 hold. If we solve $\hat{\boldsymbol{K}}^{-1} \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta} \boldsymbol{z}_j$ with i iterations of preconditioned CG, initialized at $\mathbf{0}$ or better, then it holds with probability $1 - \delta$ for $\tau_*^{\text{inv}\partial} = \operatorname{tr}\left(\hat{\boldsymbol{P}}^{-1} \frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}\right) + \tau_{\ell,i}^{\text{SCG}}(\boldsymbol{\Delta}_{\text{inv}\partial})$, that

$$\left|\tau_*^{\text{inv}\partial} - \operatorname{tr}\left(\hat{\boldsymbol{K}}^{-1} \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\right)\right| \leq (\varepsilon_{\text{CG}'} + \varepsilon_{\text{STE}}) \|\boldsymbol{K}^{-1} \frac{\partial \boldsymbol{K}}{\partial \theta}\|_F,$$

where the individual errors are bounded by

$$\varepsilon_{\text{CG}'}(\kappa, i) \leq K_2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \tag{4.17}$$

$$\varepsilon_{\text{STE}}(\delta, \ell) \leq C_1 \sqrt{\log(\delta^{-1})} \ell^{-\frac{1}{2}} g(\ell) \tag{4.18}$$

and $K_2 = 2\sqrt{\kappa(\hat{\boldsymbol{K}})} n$.

Proof. See Appendix C.3.3. □

Corollary 4.2
Assume the conditions of Theorem 4.3 hold. If the number of random vectors ℓ satisfies (4.11) with $\varepsilon_{\text{STE}} = \frac{\varepsilon}{2}$, and we run

$$i \geq \tfrac{1}{2}\sqrt{\kappa} \log(2K_2 \varepsilon^{-1}) \tag{4.19}$$

iterations of CG, then

$$\boxed{\mathrm{P}\left(\left|\tau_*^{\text{inv}\partial} - \operatorname{tr}\left(\hat{\boldsymbol{K}}^{-1} \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\right)\right| \leq \varepsilon \|\hat{\boldsymbol{K}}^{-1} \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_F\right) \geq 1 - \delta.}$$

Proof. See Appendix C.3.3. □

4.4 Efficient GP Hyperparameter Optimization

Having established an efficient way to compute the forward and backward pass for the log-determinant, we can use these results to accelerate GP hyperparameter optimization by fully

exploiting preconditioning not just for the linear solves, but *also* for the log-determinant and its derivative.

4.4.1 Log-Marginal Likelihood

We obtain a bound on the log-marginal likelihood by combining Theorem 4.2 with standard CG convergence analysis.

Theorem 4.4 (Error Bound for the log-Marginal Likelihood)
Assume the conditions of Theorem 4.2 hold and we solve $\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{y}$ via preconditioned CG initialized at $\boldsymbol{v}_0$ and terminated after i iterations. Then with probability $1-\delta$, the error in the estimate $\eta = -\frac{1}{2}(\boldsymbol{y}^\intercal \boldsymbol{v}_i + \tau_*^{\log} + n\log(2\pi))$ of the log-marginal likelihood $\mathcal{L}$ satisfies

$$|\eta - \mathcal{L}| \leq \varepsilon_{\mathrm{CG}} + \tfrac{1}{2}(\varepsilon_{\mathrm{Lanczos}} + \varepsilon_{\mathrm{STE}})\|\log(\hat{\boldsymbol{K}})\|_F,$$

where the individual errors are bounded by

$$\varepsilon_{\mathrm{CG}}(\kappa, i) \leq K_3 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \tag{4.20}$$

$$\varepsilon_{\mathrm{Lanczos}}(\kappa, i) \leq K_1 \left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i} \tag{4.21}$$

$$\varepsilon_{\mathrm{STE}}(\delta, \ell) \leq C_1 \sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}} g(\ell) \tag{4.22}$$

for $K_3 = \sqrt{\kappa(\hat{\boldsymbol{K}})}\|\boldsymbol{y}\|_2\|\boldsymbol{v}_0 - \boldsymbol{v}_*\|_2$.

Proof. See Appendix C.4.1. □

4.4.2 Derivative of the Log-Marginal Likelihood

Similarly, we can leverage Theorem 4.3 for the derivative.

Theorem 4.5 (Error Bound for the Derivative)
Assume the conditions of Theorem 4.3 hold and we solve $\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{y}$ via preconditioned CG initialized at $\mathbf{0}$ or better and terminated after i iterations. Then with probability $1-\delta$, the error in the estimate $\phi = \frac{1}{2}(\boldsymbol{v}_i^\intercal \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{v}_i - \tau_*^{\mathrm{inv}\partial})$ of the derivative of the log-marginal likelihood $\frac{\partial}{\partial\theta}\mathcal{L}$ satisfies

$$|\phi - \tfrac{\partial}{\partial\theta}\mathcal{L}| \leq \varepsilon_{\mathrm{CG}} + \tfrac{1}{2}(\varepsilon_{\mathrm{CG}'} + \varepsilon_{\mathrm{STE}})\|\hat{\boldsymbol{K}}^{-1}\tfrac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\|_F,$$

where the individual errors are bounded by

$$\varepsilon_{\mathrm{CG}}(\kappa, i) \leq K_4 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \tag{4.23}$$

$$\varepsilon_{\mathrm{CG}'}(\kappa, i) \leq K_2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \tag{4.24}$$

$$\varepsilon_{\mathrm{STE}}(\delta, \ell) \leq C_1 \sqrt{\log(\delta^{-1})} \ell^{-\frac{1}{2}} g(\ell) \tag{4.25}$$

for $K_4 = 6\kappa(\hat{\boldsymbol{K}}) \max(\|\boldsymbol{v}_*\|_2, \|\boldsymbol{v}_*\|_2^3) \|\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_2$.

Proof. See Appendix C.4.2. □

Table 4.1: *Error rates for combinations of kernels and preconditioners.* The rate $g(\ell)$ measures how fast a sequence of preconditioners $\{\hat{\boldsymbol{P}}_\ell\}_\ell$ approaches the kernel matrix $\hat{\boldsymbol{K}}$ constructed from data $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. Thus it determines both the convergence speed of Krylov methods and the preconditioned stochastic trace estimator. Or, equivalently, the faster $g(\ell) \to 0$ the fewer CG iterations i and random vectors ℓ are needed to approximate the log-marginal likelihood and its gradient. See Appendix C.5 for the corresponding proofs.

Kernel	d	Preconditioner	$g(\ell)$	Condition
any	$\mathbb{N}$	none	1	
any	$\mathbb{N}$	truncated SVD	$\ell^{-\frac{1}{2}}$	
any	$\mathbb{N}$	random. SVD	$\ell^{-\frac{1}{2}} + \mathcal{O}(\ell^{\frac{1}{4}} s^{-\frac{1}{4}})$	w/ high prob. for s samples
any	$\mathbb{N}$	random. Nyström	$\ell^{-\frac{1}{2}} + \mathcal{O}(\ell^{\frac{1}{4}} s^{-\frac{1}{4}})$	w/ high prob. for s samples
any	$\mathbb{N}$	RFF	$\ell^{-\frac{1}{2}}$	w/ high prob.
RBF	1	partial Cholesky	$\exp(-c\ell)$	for some $c > 0$
RBF	$\mathbb{N}$	QFF	$\exp(-b\ell^{\frac{1}{d}})$	for some $b > 0$ if $\ell^{\frac{1}{d}} > 2\gamma^{-2}$
Matérn(ν)	$\mathbb{N}$	partial Cholesky	$\ell^{-(\frac{2\nu}{d}+1)}$	$2\nu \in \mathbb{N}$, maximin ordering [117]
Matérn(ν)	1	QFF	$\ell^{-(s(\nu)+1)}$	where $s(\nu) \in \mathbb{N}$
mod. Matérn(ν)	$\mathbb{N}$	QFF	$\ell^{-\frac{s(\nu)+1}{d}}$	where $s(\nu) \in \mathbb{N}$
additive	$\mathbb{N}$	any	$dg(\ell)$	all summands have rate $g(\ell)$
any	$\mathbb{N}$	any kernel approx.	$g(\ell)$	$\exists$ uniform convergence bound

4.4.3 Preconditioner Choice

Our theoretical convergence results fundamentally depend on how quickly the preconditioner approximates the kernel matrix, either directly via $g(\ell)$ or indirectly via the condition number improvement (4.7). This leaves the question of which preconditioners should be chosen in practice and what rates $g(\ell)$ they attain. In Table 4.1, we give an extensive list of kernel-preconditioner combinations with associated rates (see Appendix C.5 for proofs). This includes the commonly

used RBF and Matérn(ν) kernels for which the Cholesky [148] and QFF [149] preconditioners result in exponential and polynomial convergence rates, respectively. For STE in this context, this is a substantial improvement over the rate of Hutchinson's estimator $\mathcal{O}(\ell^{-\frac{1}{2}})$ [141, 145, 150] and HUTCH++ with $\mathcal{O}(\ell^{-1})$ [144, 146, 147]. Depending on the problem this can mean a difference of tens vs. thousands of random vectors. To the best of our knowledge, for the use of CG in GP inference, only the one-dimensional RBF kernel and partial Cholesky preconditioner have been previously analyzed theoretically [3]. In contrast, Table 4.1 gives convergence rates for arbitrary d-dimensional kernels and multiple preconditioners. Our results also apply to any kernel approximation with a uniform convergence bound (such as RFF [91]). All the while for many, e.g. diagonal-plus-low-rank preconditioners, the amount of precomputation needed *amortizes with more data*, i.e. the cost of preconditioning becomes negligible the larger the dataset.

4.4.4 Algorithms

The analysis above leads to Algorithms 3 and 4 computing $\mathcal{L}$ and $\frac{\partial}{\partial\theta}\mathcal{L}$ for GP hyperparameter optimization.[4] Our algorithms are similar to those presented in prior work by Gardner et al. [3], Cutajar et al. [112], and Ubaru, Chen, and Saad [138], yet crucially they leverage preconditioning for faster CG convergence *and* variance reduction of the log-determinant estimate and its derivative. In the following, $\mathrm{CG}(\hat{\boldsymbol{K}}, \boldsymbol{y}, \hat{\boldsymbol{P}}, i)$ denotes a CG solve of $\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{y}$ with preconditioner $\hat{\boldsymbol{P}}$ run for i iterations. Here, we equivalently use CG instead of Lanczos, as suggested by Gardner et al. [3].

Algorithm 3: log-Marginal Likelihood

Input: Targets $\boldsymbol{y}$, kernel matrix $\hat{\boldsymbol{K}}$, preconditioner $\hat{\boldsymbol{P}}$, # of random vectors ℓ, CG iterations i
Output: Estimate of log-marginal likelihood $\mathcal{L}(\theta)$

procedure LOGMARGLIKELIHOOD($\boldsymbol{y}, \hat{\boldsymbol{K}}, \hat{\boldsymbol{P}}, \ell, i$)
  $\boldsymbol{v}_i \leftarrow \mathrm{CG}(\hat{\boldsymbol{K}}, \boldsymbol{y}, \hat{\boldsymbol{P}}, i)$ ▷ Approximate representer weights.
  $\tau^{\log}_{\hat{\boldsymbol{P}}} \leftarrow \log\det(\hat{\boldsymbol{P}})$ ▷ Log-determinant of the preconditioner.
  for $j = 1, \ldots, \ell$ **do** ▷ Loop is embarrassingly parallel.
    $\boldsymbol{z}_j \leftarrow \tilde{\boldsymbol{z}}_j / \|\tilde{\boldsymbol{z}}_j\|_2$ for rand. vector $\tilde{\boldsymbol{z}}_j$ ▷ Sample random vector and normalize.
    $\boldsymbol{T} \leftarrow \mathrm{CG}(\hat{\boldsymbol{K}}, \boldsymbol{z}_j, \hat{\boldsymbol{P}}, i)$ ▷ Run Lanczos process (equivalent to CG).
    $[\boldsymbol{W}, \boldsymbol{\lambda}] \leftarrow \mathrm{EIGENDECOMP}(\boldsymbol{T})$ ▷ Since $\boldsymbol{T}$ is tridiagonal, eigendecomp. costs $\mathcal{O}(i^2)$.
    $\omega_k \leftarrow (\boldsymbol{e}_1^\intercal \boldsymbol{w}_k)^2$ for $k = 0, \ldots, i$ ▷ Compute quadrature weights.
    $\gamma_j \leftarrow \sum_{k=0}^{i} \omega_k \log(\lambda_k)$ ▷ Quadrature estimate of quadratic form $\boldsymbol{z}_j^\intercal \boldsymbol{\Delta}_{\log} \boldsymbol{z}_j$
  $\tau^{\log}_* \leftarrow \tau^{\log}_{\hat{\boldsymbol{P}}} + \frac{n}{\ell}\sum_{j=1}^{\ell}\gamma_j$ ▷ Variance-reduced log-determinant estimate.
  return $-\frac{1}{2}(\boldsymbol{y}^\intercal \boldsymbol{v}_i + \tau^{\log}_* + n\log(2\pi))$ ▷ Estimate of log-marginal likelihood $\mathcal{L}(\theta)$.

[4] While presented sequentially for clarity, in practice one would pre-sample all random vectors and run a single call of (parallelized) CG with multiple right-hand sides, as in Gardner et al. [3] since the loops are embarrassingly parallel.

Algorithm 4: Derivative of the log-Marginal Likelihood

Input: Targets $\boldsymbol{y}$, kernel matrix $\hat{\boldsymbol{K}}$, preconditioner $\hat{\boldsymbol{P}}$, kernel matrix derivative $\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}$, preconditioner derivative $\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}$, # of random vectors ℓ, CG iterations i

Output: Estimate of derivative of the log-marginal likelihood $\frac{\partial}{\partial \theta}\mathcal{L}(\boldsymbol{\theta})$

procedure DERIVATIVE($\boldsymbol{y}, \hat{\boldsymbol{K}}, \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}, \hat{\boldsymbol{P}}, \frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}, \ell, i$)
- $\boldsymbol{v}_i \leftarrow \mathrm{CG}(\hat{\boldsymbol{K}}, \boldsymbol{y}, \hat{\boldsymbol{P}}, i)$ ▷ Approximate representer weights.
- $\tau_{\hat{\boldsymbol{P}}}^{\mathrm{inv}\partial} \leftarrow \mathrm{tr}(\hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta})$ ▷ Preconditioner estimate of $\mathrm{tr}(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta})$.
- **for** $j = 1, \ldots, \ell$ **do** ▷ Loop is embarrassingly parallel.
  - $\boldsymbol{z}_j \leftarrow \tilde{\boldsymbol{z}}_j / \|\tilde{\boldsymbol{z}}_j\|_2$ for rand. vector $\tilde{\boldsymbol{z}}_j$ ▷ Sample random vector and normalize.
  - $\boldsymbol{w}_j \leftarrow \mathrm{CG}(\hat{\boldsymbol{K}}, \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{z}_j, \hat{\boldsymbol{P}}, i)$ ▷ CG estimate of $\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{z}_j$
  - $\tilde{\boldsymbol{w}}_j \leftarrow \hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}\boldsymbol{z}_j$ ▷ Preconditioner estimate of $\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{z}_j$
  - $\gamma_j \leftarrow \boldsymbol{z}_j^\intercal(\boldsymbol{w}_j - \tilde{\boldsymbol{w}}_j)$ ▷ Estimate of quadratic form $\boldsymbol{z}_j^\intercal \boldsymbol{\Delta}_{\mathrm{inv}\partial}\boldsymbol{z}_j$.
- $\tau_*^{\mathrm{inv}\partial} \leftarrow \tau_{\hat{\boldsymbol{P}}}^{\mathrm{inv}\partial} + \frac{n}{\ell}\sum_{j=1}^{\ell}\gamma_j$ ▷ Variance-reduced estimate of $\mathrm{tr}(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta})$.
- **return** $\frac{1}{2}(\boldsymbol{v}_i^\intercal \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{v}_i - \tau_*^{\mathrm{inv}\partial})$ ▷ Estimate of derivative $\frac{\partial}{\partial \theta}\mathcal{L}(\boldsymbol{\theta})$.

Computational Complexity Algorithm 3 has complexity $\mathcal{O}(n^2 i\ell + c_{\log\det})$ and Algorithm 4 has complexity $\mathcal{O}((n^2 i + c_{\mathrm{solve}})\ell + c_{\mathrm{tr\,inv}\partial})$, where $c_{(\cdot)}$ denotes the cost of an operation with the preconditioner.[5] Assuming $i, \ell \ll n$, this is asymptotically faster than Cholesky-based inference with complexity $\mathcal{O}(n^3)$. Due to the reduction to matrix-vector multiplication, if $\boldsymbol{v} \mapsto \hat{\boldsymbol{K}}\boldsymbol{v}$ is more efficient than $\mathcal{O}(n^2)$ (e.g. for structured or sparse matrices) the complexity reduces further. Finally, the **for**-loops are embarrassingly parallel, giving additional speedup in practice.

4.4.5 Related Work

Krylov methods have been used for GP inference since the work of Gibbs [110]. While these methods were primarily relegated to structured GPs that afford fast matrix-vector products [92, 118, 151], they have seen growing use as a general purpose method, especially when coupled with specialized, parallel hardware [3, 111, 113, 114, 137]. Preconditioners can be used to accelerate and stabilize the necessary linear solves with the kernel matrix [112, 152–155]. To compute the log-determinant of the kernel matrix, some recent works propose variance-free (but biased) estimates [e.g. 114], though many works compute this term by combining STE [140, 141, 145, 156] with SLQ [3, 138, 139, 142, 157]. Our work builds on ideas for variance-reduced stochastic trace estimation [143, 144, 146, 147], but, by leveraging preconditioning, requires significantly fewer random vectors than existing approaches. When applied to GP hyperparameter optimization, we obtain stronger theoretical guarantees for the forward pass than previously known [138] and novel guarantees for the backward pass. Finally, our results on preconditioners for kernel matrices (Table 4.1) give a rigorous foundation to their use for GPs as proposed by Cutajar et al. [112] and others.

[5] For diagonal-plus-low-rank preconditioners, such as the partial Cholesky, c_{solve}, $c_{\log\det}$, and $c_{\mathrm{tr\,inv}\partial}$ are in $\mathcal{O}(n\ell^2)$ by the matrix inversion and determinant lemmas.

4.5 Experiments

We validate our theoretical findings empirically via GP hyperparameter optimization on synthetic and benchmark datasets with and without preconditioning. We find that

(a) *preconditioning reduces bias and variance* in the forward and backward pass, resulting in
(b) *better search directions* and *fewer likelihood and gradient evaluations for the line search*.

This allows the use of rapidly converging optimizers, and *accelerates training significantly.*

Experimental Setup We consider a one-dimensional synthetic dataset of $n = 1{,}000$ iid standard normal samples, as well as a range of UCI datasets [135] with training set sizes ranging from $n = 12{,}449$ to $326{,}155$ (see Table 4.2). All experiments were performed on single NVIDIA GPUs, a GeForce RTX 2080 and Titan RTX, respectively. We perform GP regression using an RBF and Matérn($\frac{3}{2}$) kernel with output scale o, lengthscales l_j – one per input dimension – and noise σ^2. Hyperparameters were optimized with L-BFGS using an Armijo-Wolfe line search and early stopping via a validation set. We use a partial Cholesky preconditioner throughout. An implementation of our method is available as part of GPyTorch [3] at:

GPyTorch https://github.com/cornellius-gp/gpytorch

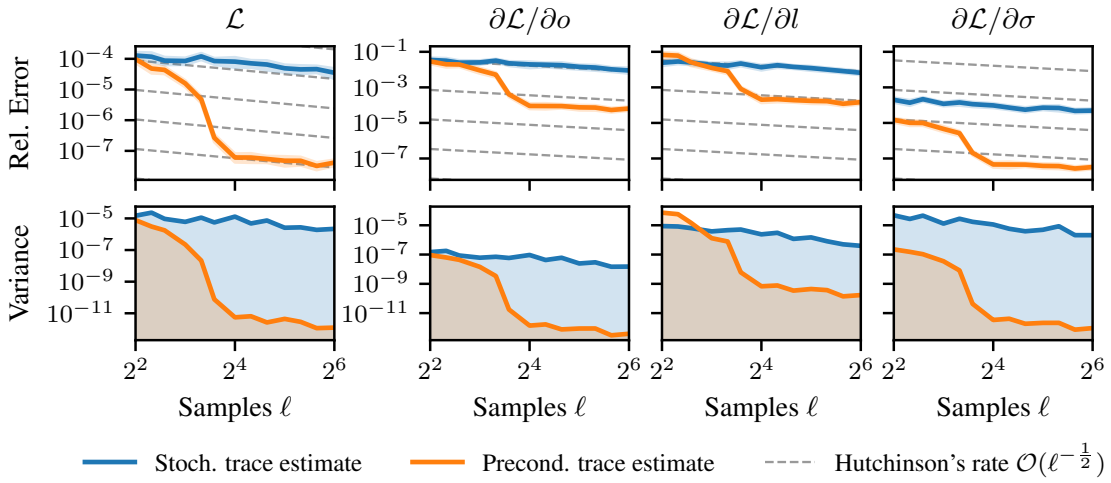
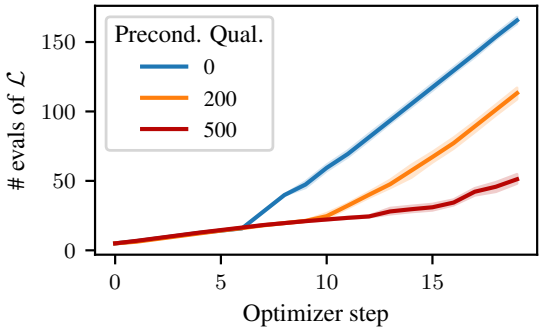


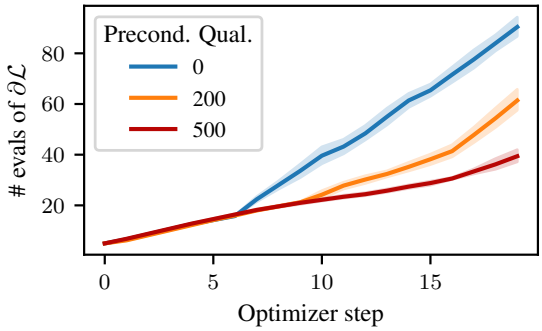
Figure 4.3: *Bias and variance of the estimators for the* log*-marginal likelihood* $\mathcal{L}$ *and its derivatives.* The relative error and variance decrease faster with the number of random vectors ℓ when using a preconditioner $\hat{\boldsymbol{P}}_\ell$. The decrease rate $\mathcal{O}(\ell^{-\frac{1}{2}}g(\ell))$, determined by the preconditioner, significantly improves upon the standard Hutchinson's rate $\mathcal{O}(\ell^{-\frac{1}{2}})$.

Preconditioning reduces bias & variance in $\mathcal{L}$ and $\frac{\partial}{\partial\theta}\mathcal{L}$ Figure 4.3 shows the relative error of the marginal log-likelihood and its derivatives on synthetic data. Already for $\ell \geq 16$ random samples *bias and variance are reduced by several orders of magnitude*. We observe an exponential decrease and then a return to the standard Hutchinson's rate of $\mathcal{O}(\ell^{-\frac{1}{2}})$. After $\ell = 16$ iterations

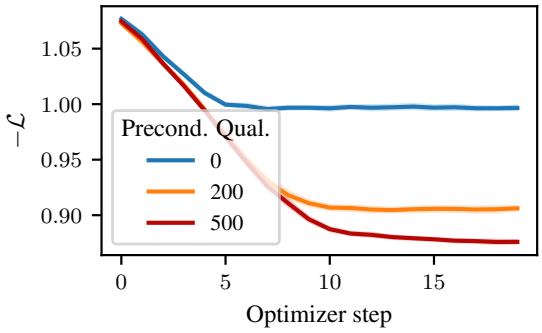
the algorithm computing the preconditioner has reached a specified tolerance and terminates, invalidating the approximation quality assumption (4.6) for $\ell > 16$. Similar observations hold for the Matérn and RatQuad kernel (see Table C.1 and Figure C.1). As predicted by Theorem 4.1 and illustrated by Figure 4.3, the variance reduction is determined by the preconditioner. For higher dimensions, the rate $g(\ell)$ slows (see Table 4.1), which in turn reduces the bias and variance reduction via our method (see Table C.1). However, on real datasets, we still see strong variance reduction via our method, possibly since real data often lies on a low-dimensional manifold.



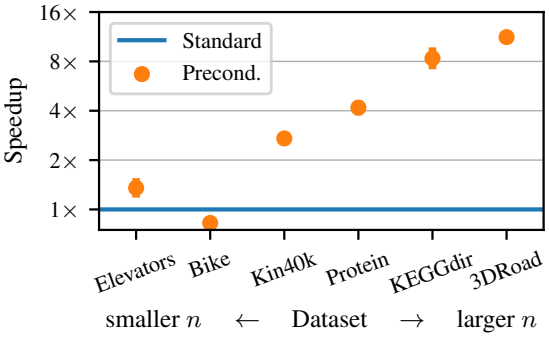
(a) Objective eval. during line search on "Protein".



(b) Gradient eval. during line search on "Protein".



(c) Training loss for "Protein".



(d) Speedup on UCI datasets.

Figure 4.4: *Preconditioning reduces noise and in turn accelerates hyperparameter optimization.* Variance reduction improves optimization via better search directions and fewer evaluations of (a) the optimization objective $\mathcal{L}$ and (b) the gradient $\frac{\partial}{\partial\theta_j}\mathcal{L}$ for the line search. (c) Training loss decreases with preconditioner quality, as shown for the "Protein" dataset. (d) The reduction in loss and gradient evaluations and of noise in the gradients results in an order of magnitude speedup on UCI datasets.

Preconditioning accelerates hyperparameter optimization On datasets from the UCI repository, we find that preconditioning results in lower training loss $-\mathcal{L}(\boldsymbol{\theta})$ (illustrated in Figure 4.4(c)) on almost all datasets and essentially identical generalization error (see Table 4.2). Reducing stochasticity via preconditioning significantly lowers the number of $\mathcal{L}$ and $\frac{\partial}{\partial\theta}\mathcal{L}$ evaluations for the line search during optimization (see Figures 4.4(a) and 4.4(b)) and results in less noisy search directions. In fact, the noise in the loss and gradients caused by stochastic trace estimation

previously necessitated the use of slower converging, but more noise-robust optimizers [113], such as Adam [158]. As these experiments show, our variance-reduced estimators make the use of L-BFGS possible, which significantly outperforms Adam (c.f. Table 4.2 and Table C.3). These combined effects are due to preconditioning *accelerate training up to twelvefold*, as Figure 4.4(d) shows. We observe that the speedup increases with the size of the dataset. This is partly explained by the amortizing cost of computing and applying the preconditioner with increasing n.

Table 4.2: *Hyperparameter optimization on UCI datasets.* GP regression using a Matérn($\frac{3}{2}$) kernel and partial Cholesky preconditioner of size 500 with $\ell = 50$ random samples. Hyperparameters were optimized with L-BFGS for at most 20 steps using early stopping. All results, but "3DRoad", are averaged over 10 runs. Differences larger than one standard deviation are in bold font.

Dataset	n	d	$-\mathcal{L}_{\text{train}} \downarrow$		$-\mathcal{L}_{\text{test}} \downarrow$		RMSE $\downarrow$		Runtime	
			Basic	Precon.	Basic	Precon.	Basic	Precon.	Basic	Precon.
Elevators	12 449	18	0.465	**0.438**	0.402	0.402	0.348	0.348	53.0s	**39.2s**
Bike	13 034	17	−0.998	−0.999	−0.993	−0.988	0.045	0.045	**30.6s**	37.1s
Kin40k	30 000	8	−0.334	**−0.433**	−0.314	−0.314	**0.093**	0.095	3.1m	**44.6s**
Protein	34 297	9	0.996	**0.927**	0.887	0.884	0.572	**0.558**	14.9m	**42.5s**
KEGGdir	36 620	20	−0.950	**−1.004**	−0.946	−0.949	0.086	0.086	24.2m	**2.9m**
3DRoad	326 155	3	0.773	**0.128**	1.436	**1.169**	0.298	**0.127**	22.8h	**2.0h**

4.6 Conclusion

When interpreting structural knowledge about the kernel matrix as a form of prior information for a numerical method, one might reasonably hope that it can be leveraged to accelerate Gaussian process hyperparameter optimization. Preconditioning is a way to encode and exploit such structure. As we showed, it can be used to great effect – not only for the solution of linear systems – but importantly *also* for stochastic approximation of the log-determinant and its derivative. Our convergence results combined with the rates for kernel-preconditioner pairs in Table 4.1 rigorously explain why preconditioning has been observed empirically to be so effective for large-scale GP inference [3, 112, 113]. Our work implies that software packages for Gaussian processes, which make use of Krylov methods for inference, should not use a fixed preconditioner. Instead, the preconditioner should be automatically chosen based on the specified model. While we derive a range of such kernel and preconditioner combinations, better preconditioners likely exist for certain kernels or types of data. Other scientific fields invest substantial research effort into the design of preconditioners, e.g. for PDEs [159]. Our work strongly suggests that, similarly, developing specialized preconditioners is a promising approach to accelerate and scale Gaussian processes.

Chapter

5

Conclusion

Modern decision-making systems rely on accurate machine learning models, which quantify their inherent uncertainty. In practice, the performance and reliability of such models are not only dictated by data but also by computational resources. It is therefore essential to develop efficient numerical approximations *and* to account for their inherent impact on a model's predictions.

5.1 Summary

This thesis presents new techniques in probabilistic numerical linear algebra that are tailored to the numerical tasks that arise in probabilistic machine learning. When applied to Gaussian processes, the proposed algorithms automatically exploit structure in the model to accelerate inference, as well as hyperparameter optimization, and importantly can propagate their approximation error back to the model. This enables *exact* uncertainty quantification that not only integrates the uncertainty from observing limited data but also the uncertainty from performing only a limited amount of computation. All proposed algorithms reduce to the fundamental primitive of matrix-vector multiplication, which makes them both generally applicable and uniquely suited to modern parallel hardware such as GPUs. As a result of this work, Gaussian processes can scale to larger datasets without compromising their ability to quantify uncertainty – a fundamental prerequisite for optimal decision-making.

Software The methods developed in this thesis are not only of theoretical interest but have also directly influenced the open-source software landscape attesting to their practical relevance. Our work on probabilistic linear solvers has led to the creation of the first comprehensive software package for probabilistic numerical methods, a crucial step towards their wider adoption [55]. Moreover, our novel approach to accelerate Gaussian process hyperparameter optimization via

preconditioning [4] now defines the default large-scale training procedure in GPyTorch, the most widely used Python library for Gaussian processes [3].

5.2 Future Research

The methodological developments in probabilistic numerical linear algebra presented in this thesis open up several promising research directions. In particular, whenever a Gaussian process model is employed for an a priori known downstream task, the proposed approximation methods can be explicitly tailored toward it. This potentially enables significant speedup by trading off increased uncertainty for computational efficiency, with little impact on the downstream task. Such approaches could be particularly beneficial in research areas such as Bayesian optimization, active learning, Bayesian deep learning and the data-driven solution of differential equations.

Prior Knowledge and Policy Design The design of specialized preconditioners for commonly used kernels is a direct avenue for future work. Once implemented in a software library, these can programmatically be selected based on a user's model specification and then directly serve as prior knowledge for a probabilistic linear solver, accelerating its convergence. Similarly, the policy of a probabilistic linear solver opens up a vast design space, that so far has been left largely unexplored. Current solver policies are mostly inspired by classic linear solvers, which are designed to solve generic linear systems. Instead, specialized policies which are informed by a downstream task promise to enable significant computational speedup. For example, in Gaussian process hyperparameter optimization, we not only need to solve a linear system but also estimate a log-determinant, which is not reflected in a generic policy. Other examples include Bayesian optimization and active learning, where the goal is not to globally learn the latent function, but only in specific regions of the input space. Policies exploiting the trade-off between enhanced precision in regions of interest and increased uncertainty elsewhere to achieve superior computational efficiency represent an exciting line of future research.

Non-Gaussianity and Bayesian Deep Learning Current probabilistic linear solvers fundamentally rely on the fact that Gaussians are closed under conditioning on linear observations. While many numerical methods trace back to the primitive of matrix-vector multiplication, there are often non-linear transformations that invalidate the assumption of Gaussianity. This raises the question of whether one can design similar algorithms quantifying computational uncertainty under the assumption of non-Gaussian likelihoods, or even for generic Bayesian inference with a potentially different computational primitive than matrix-vector multiplication. An application for which an extension of IterGP to non-Gaussian likelihoods would directly be useful is Bayesian deep learning. Performing a Laplace approximation to equip a neural network with uncertainty quantification posthoc can equivalently be viewed as a Gaussian process model with a prior defined via the neural tangent kernel [160]. Therefore efficient, computation-aware, inference methods for Gaussian process classification directly translate into efficient numerical methods for Bayesian deep learning. This is particularly useful since Gaussian process inference scales with the size of the dataset, rather than with the dimension of the parameter space which is considerably larger for overparametrized networks.

Differential Equations and Dynamical Systems Ordinary and partial differential equations are important mechanistic models in the physical sciences that often describe dynamical systems. Differential equations are typically solved via discretization which gives rise to (sequences of) linear systems with characteristic structure. As demonstrated by Pförtner et al. [46] the ideas presented in this work readily extend to the design of computation-aware solvers for differential equations. In the PDE setting scaling considerations are particularly important since an accurate simulation requires high-fidelity discretizations resulting in systems that easily reach hundreds of thousands of dimensions. If a differential equation describes a dynamical system, time discretization often results in sequences of linear systems with closely related system matrices or right-hand sides. Finding a way to adaptively recycle computation between time steps by appropriately updating a belief is an exciting direction for future research, that promises significant computational savings as well as accurate error quantification. Such an approach could be applied to discrete gradient flow approximations, to computation-aware sequence modeling or to efficiently solve spatio-temporal partial differential equations.

Appendix

A

Appendix of Chapter 2

A.1 Probabilistic Linear Solvers

A.1.1 Matrix-variate Prior Construction

From a practical point of view, it is important to be able to construct a prior for $\mathbf{A}$ and $\mathbf{H}$ from an initial guess $\boldsymbol{x}_0$ for the solution. This reduces down to finding $\boldsymbol{A}_0$ and $\boldsymbol{H}_0$ symmetric positive definite, such that $\boldsymbol{A}_0 = \boldsymbol{H}_0^{-1}$ and $\boldsymbol{x}_0 = \boldsymbol{H}_0\boldsymbol{b}$ for the covariance class derived in Section 2.3. We provide a computationally efficient construction of such a prior here.

Proposition A.1
Let $\boldsymbol{x}_0 \in \mathbb{R}^n$ *and* $\boldsymbol{b} \in \mathbb{R}^n \setminus \{0\}$. *Assume* $\boldsymbol{x}_0^\intercal \boldsymbol{b} > 0$, *then for* $\alpha < \frac{\boldsymbol{b}^\intercal \boldsymbol{x}_0}{\boldsymbol{b}^\intercal \boldsymbol{b}}$,

$$\boldsymbol{H}_0 = \alpha \boldsymbol{I} + \frac{1}{(\boldsymbol{x}_0 - \alpha \boldsymbol{b})^\intercal \boldsymbol{b}} (\boldsymbol{x}_0 - \alpha \boldsymbol{b})(\boldsymbol{x}_0 - \alpha \boldsymbol{b})^\intercal$$

is symmetric positive definite and $\boldsymbol{H}_0\boldsymbol{b} = \boldsymbol{x}_0$*. Further, it holds that*

$$\boldsymbol{A}_0 = \boldsymbol{H}_0^{-1} = \alpha^{-1}\boldsymbol{I} - \frac{\alpha^{-1}}{(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal \boldsymbol{x}_0}(\boldsymbol{x}_0 - \alpha\boldsymbol{b})(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal.$$

If $\boldsymbol{x}_0^\intercal \boldsymbol{b} < 0$ *or* $\boldsymbol{x}_0^\intercal \boldsymbol{b} = 0$*, then for* $\boldsymbol{x}_1 = -\boldsymbol{x}_0$ *or* $\boldsymbol{x}_1 = \frac{\boldsymbol{b}^\intercal \boldsymbol{b}}{\boldsymbol{b}^\intercal \boldsymbol{A}\boldsymbol{b}}\boldsymbol{b}$ *respectively, it holds that* $\|\boldsymbol{x}_1 - \boldsymbol{x}_*\|_{\boldsymbol{A}}^2 < \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_{\boldsymbol{A}}^2$*, i.e.* $\boldsymbol{x}_1$ *is a strictly better initialization than* $\boldsymbol{x}_0$*.*

Proof. Let $\boldsymbol{H}_0$ as above. Then $\boldsymbol{H}_0\boldsymbol{b} = \alpha\boldsymbol{b} + \boldsymbol{x}_0 - \alpha\boldsymbol{b} = \boldsymbol{x}_0$. The second term of the sum in the form of $\boldsymbol{H}_0$ is of rank 1. Its non-zero eigenvalue is given by

$$\lambda = \frac{1}{(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal \boldsymbol{b}}(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal(\boldsymbol{x}_0 - \alpha\boldsymbol{b}) = \frac{1}{\boldsymbol{x}_0^\intercal \boldsymbol{b} - \alpha\boldsymbol{b}^\intercal \boldsymbol{b}}\|\boldsymbol{x}_0 - \alpha\boldsymbol{b}\|_2^2 \geq 0$$

since by assumption $\boldsymbol{x}_0^\intercal \boldsymbol{b} > 0$ and $\alpha < \frac{\boldsymbol{b}^\intercal \boldsymbol{x}_0}{\boldsymbol{b}^\intercal \boldsymbol{b}}$. Now by Weyl's theorem it holds that $\lambda_{\min}(\boldsymbol{A}) + \lambda_{\min}(\boldsymbol{E}) \leq \lambda_{\min}(\boldsymbol{A} + \boldsymbol{E})$ and therefore $\boldsymbol{H}_0$ is positive definite. By the matrix inversion lemma we have for $\gamma = \frac{\alpha^{-1}}{(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal \boldsymbol{b}}$ that

$$\begin{aligned}
\boldsymbol{A}_0 = \boldsymbol{H}_0^{-1} &= \alpha^{-1}(\boldsymbol{I} - \frac{\gamma}{1 + \gamma\|\boldsymbol{x}_0 - \alpha\boldsymbol{b}\|_2^2}(\boldsymbol{x}_0 - \alpha\boldsymbol{b})(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal) \\
&= \alpha^{-1}\boldsymbol{I} - \frac{\alpha^{-2}}{(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal \boldsymbol{b} + \alpha^{-1}\|\boldsymbol{x}_0 - \alpha\boldsymbol{b}\|_2^2}(\boldsymbol{x}_0 - \alpha\boldsymbol{b})(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal \\
&= \alpha^{-1}\boldsymbol{I} - \frac{\alpha^{-1}}{(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal \boldsymbol{x}_0}(\boldsymbol{x}_0 - \alpha\boldsymbol{b})(\boldsymbol{x}_0 - \alpha\boldsymbol{b})^\intercal.
\end{aligned}$$

Finally, we obtain

$$\|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_{\boldsymbol{A}}^2 = (\boldsymbol{x}_0 - \boldsymbol{A}^{-1}\boldsymbol{b})^\intercal \boldsymbol{A}(\boldsymbol{x}_0 - \boldsymbol{A}^{-1}\boldsymbol{b}) = \boldsymbol{x}_0^\intercal \boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b}^\intercal \boldsymbol{A}^{-1}\boldsymbol{b} - 2\boldsymbol{b}^\intercal \boldsymbol{x}_0.$$

Therefore if either $\boldsymbol{x}_0^\intercal \boldsymbol{b} < 0$ or $\boldsymbol{x}_0^\intercal \boldsymbol{b} = 0$, then $\boldsymbol{x}_1 = -\boldsymbol{x}_0$ or $\boldsymbol{x}_1 = \frac{\boldsymbol{b}^\intercal \boldsymbol{b}}{\boldsymbol{b}^\intercal \boldsymbol{A}\boldsymbol{b}}\boldsymbol{b}$, respectively are closer to $\boldsymbol{x}_*$ in $\boldsymbol{A}$ norm by positive definiteness of $\boldsymbol{A}$. This concludes the proof. □

A.1.2 Stopping Criteria

In addition to the classic stopping criteria $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i\|_2 \leq \max(\delta_{\text{rtol}}\|\boldsymbol{b}\|_2, \delta_{\text{atol}})$ it is natural from a probabilistic viewpoint to use the induced posterior covariance of $\mathbf{x}$. Let $\boldsymbol{M} \in \mathbb{R}_{\text{sym}}^{n \times n}$ be a positive-definite matrix, then by linearity and the cyclic property of the trace it holds that

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{x}_*}\left(\|\boldsymbol{x}_* - \mathbb{E}(\mathbf{x})\|_{\boldsymbol{M}}^2\right) &= \mathbb{E}_{\boldsymbol{x}_*}((\boldsymbol{x}_* - \mathbb{E}(\mathbf{x}))^\intercal \boldsymbol{M}(\boldsymbol{x}_* - \mathbb{E}(\mathbf{x}))) \\
&= \operatorname{tr}(\mathbb{E}_{\boldsymbol{x}_*}((\boldsymbol{x}_* - \mathbb{E}(\mathbf{x}))^\intercal \boldsymbol{M}(\boldsymbol{x}_* - \mathbb{E}(\mathbf{x})))) \\
&= \mathbb{E}_{\boldsymbol{x}_*}(\operatorname{tr}((\boldsymbol{x}_* - \mathbb{E}(\mathbf{x}))^\intercal \boldsymbol{M}(\boldsymbol{x}_* - \mathbb{E}_{\mathbf{x}}()))) \\
&= \mathbb{E}_{\boldsymbol{x}_*}(\boldsymbol{M}\operatorname{tr}((\boldsymbol{x}_* - \mathbb{E}(\mathbf{x}))(\boldsymbol{x}_* - \mathbb{E}(\mathbf{x}))^\intercal)) \\
&= \operatorname{tr}(\boldsymbol{M}\,\mathbb{E}_{\boldsymbol{x}_*}((\boldsymbol{x}_* - \mathbb{E}(\mathbf{x}))(\boldsymbol{x}_* - \mathbb{E}(\mathbf{x}))^\intercal))
\end{aligned}$$

$$
\begin{aligned}
&= \operatorname{tr}(\boldsymbol{M}(\operatorname{Cov}(\boldsymbol{x}_* - \mathbb{E}(\mathbf{x})) + (\mathbb{E}_{\boldsymbol{x}_*}(\boldsymbol{x}_*) - \mathbb{E}(\mathbf{x}))(\mathbb{E}_{\boldsymbol{x}_*}(\boldsymbol{x}_*) - \mathbb{E}(\mathbf{x}))^\intercal)) \\
&= \operatorname{tr}(\boldsymbol{M}\operatorname{Cov}(\boldsymbol{x}_*)) + \|\mathbb{E}_{\boldsymbol{x}_*}(\boldsymbol{x}_*) - \mathbb{E}(\mathbf{x})\|_{\boldsymbol{M}}^2.
\end{aligned}
$$

Assuming calibration holds, i.e. $\boldsymbol{x}_* \sim \mathcal{N}(\mathbb{E}(\mathbf{x}), \operatorname{Cov}(\mathbf{x}))$, we can bound the (relative) error by terminating when $\operatorname{tr}(\boldsymbol{M}\operatorname{Cov}(\mathbf{x})) \leq \max(\delta_{\text{rtol}}\|\boldsymbol{b}\|, \delta_{\text{atol}})$ either in l_2-norm for $\boldsymbol{M} = \boldsymbol{I}$ or in $\boldsymbol{A}$-norm for $\boldsymbol{M} = \boldsymbol{A}$.

We can efficiently evaluate the required $\operatorname{tr}(\boldsymbol{M}\operatorname{Cov}(\mathbf{x}))$ without ever forming $\operatorname{Cov}(\mathbf{x})$ in memory from already computed quantities. At iteration i we have

$$
\operatorname{Cov}(\mathbf{x}) = \operatorname{Cov}(\mathbf{H}\boldsymbol{b}) = \frac{1}{2}(\boldsymbol{W}_i^{\mathbf{H}}(\boldsymbol{b}^\intercal \boldsymbol{W}_i^{\mathbf{H}}\boldsymbol{b}) + (\boldsymbol{W}_i^{\mathbf{H}}\boldsymbol{b})(\boldsymbol{W}_i^{\mathbf{H}}\boldsymbol{b})^\intercal)
$$

and therefore

$$
\operatorname{tr}(\boldsymbol{M}\operatorname{Cov}(\mathbf{x})) = \frac{1}{2}\big((\boldsymbol{b}^\intercal \boldsymbol{W}_i^{\mathbf{H}}\boldsymbol{b})\operatorname{tr}(\boldsymbol{M}\boldsymbol{W}_i^{\mathbf{H}}) + (\boldsymbol{W}_i^{\mathbf{H}}\boldsymbol{b})^\intercal \boldsymbol{M}(\boldsymbol{W}_i^{\mathbf{H}}\boldsymbol{b})\big).
$$

Given the update for the covariance of the inverse view, we obtain the following recursion for its trace

$$
\operatorname{tr}(\boldsymbol{M}\boldsymbol{W}_i^{\mathbf{H}}) = \operatorname{tr}(\boldsymbol{M}\boldsymbol{W}_{k-1}^{\mathbf{H}}) - \frac{1}{\boldsymbol{y}_i^\intercal \boldsymbol{W}_{k-1}^{\mathbf{H}}\boldsymbol{y}_i}\operatorname{tr}((\boldsymbol{W}_{k-1}^{\mathbf{H}}\boldsymbol{y}_i)^\intercal \boldsymbol{M}(\boldsymbol{W}_{k-1}^{\mathbf{H}}\boldsymbol{y}_i)).
$$

Computing the trace in this iterative fashion adds at most three matrix-vector products and three inner products for arbitrary $\boldsymbol{M}$ all other quantities are computed for the covariance update anyhow. For our proposed covariance class (2.5) we obtain for $\boldsymbol{M} = \boldsymbol{I}$, $\boldsymbol{\Psi} = \psi\boldsymbol{I}$ and $\boldsymbol{P}_{\boldsymbol{Y}^\perp} = \boldsymbol{I} - \boldsymbol{Y}(\boldsymbol{Y}^\intercal\boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal$ that

$$
\begin{aligned}
\operatorname{tr}(\boldsymbol{W}_0^{\mathbf{H}}) &= \operatorname{tr}(\boldsymbol{A}_0^{-1}\boldsymbol{Y}(\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}\boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1} + \boldsymbol{P}_{\boldsymbol{Y}^\perp}\boldsymbol{\Psi}\boldsymbol{P}_{\boldsymbol{Y}^\perp}) \\
&= \operatorname{tr}((\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}\boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}\boldsymbol{A}_0^{-1}\boldsymbol{Y}) + \psi\operatorname{tr}(\boldsymbol{P}_{\boldsymbol{Y}^\perp}\boldsymbol{P}_{\boldsymbol{Y}^\perp}) \\
&= \operatorname{tr}((\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}\boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}\boldsymbol{A}_0^{-1}\boldsymbol{Y}) + \psi\operatorname{tr}(\boldsymbol{P}_{\boldsymbol{Y}^\perp}) \\
&= \operatorname{tr}((\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}\boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal\boldsymbol{A}_0^{-1}\boldsymbol{A}_0^{-1}\boldsymbol{Y}) + \psi(n-i),
\end{aligned}
$$

which for a scalar prior mean $\boldsymbol{A}_0 = \alpha\boldsymbol{I}$ reduces to $\operatorname{tr}(\boldsymbol{W}_0^{\mathbf{H}}) = \alpha^{-1}i + \psi(n-i)$.

A.1.3 Low-rank Updates to Cholesky Factorizations

In order to maintain numerical stability when performing low-rank updates to symmetric positive definite matrices, as is the case in Algorithm 1 for the mean and covariance estimates, it is advantageous to use a representation based on the Cholesky decomposition. One can perform the rank-2 update for the mean estimate and the rank-1 downdate for the covariance in Section 2.2.1 in each iteration of the algorithm for their respective Cholesky factors instead (see also Seeger [74]). The rank-2 update can be seen as a combination of a rank-1 up- and downdate since

$$
\boldsymbol{u}\boldsymbol{v}^\intercal + \boldsymbol{v}\boldsymbol{u}^\intercal = \frac{1}{2}((\boldsymbol{u}+\boldsymbol{v})(\boldsymbol{u}+\boldsymbol{v})^\intercal - (\boldsymbol{u}-\boldsymbol{v})(\boldsymbol{u}-\boldsymbol{v})^\intercal).
$$

Similar updates arise in Quasi-Newton methods for the approximate (inverse) Hessian [65]. Having Cholesky factors of the covariance available has the additional advantage that downstream sampling or the evaluation of the probability density function is computationally cheap.

A.2 Theoretical Properties

In this section, we provide detailed proofs for the theoretical results on convergence and the connection of Algorithm 1 to the method of conjugate gradients. We restate each theorem here as a reference to the reader. We begin by proving an intermediate result giving an interpretation to the posterior mean of $\mathbf{A}$ and $\mathbf{H}$ at each step of the method.

Proposition A.2 (Subspace Equivalency)
Let $\boldsymbol{A}_i$ and $\boldsymbol{H}_i$ be the posterior means defined as in Section 2.2.1 and assume $\boldsymbol{A}_0$ and $\boldsymbol{H}_0$ are symmetric. Then for $1 \leq i \leq n$ it holds that

$$\boldsymbol{A}_i \boldsymbol{S} = \boldsymbol{Y} \quad \text{and} \quad \boldsymbol{H}_i \boldsymbol{Y} = \boldsymbol{S}, \tag{A.1}$$

i.e. $\boldsymbol{A}_i$ and $\boldsymbol{H}_i$ act like $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ on the spaces spanned by the actions $\boldsymbol{S}$, respectively the observations $\boldsymbol{Y}$.

Proof. Since $\boldsymbol{A}_0$ and $\boldsymbol{H}_0$ are symmetric so are the expressions $\boldsymbol{\Delta}_\mathbf{A} \boldsymbol{S}$ and $\boldsymbol{\Delta}_\mathbf{H}^\intercal \boldsymbol{Y}$. We have that

$$\begin{aligned}
\boldsymbol{A}_i \boldsymbol{S} &= (\boldsymbol{A}_0 + \boldsymbol{\Delta}_\mathbf{A} \boldsymbol{U}_\mathbf{A}^\intercal + \boldsymbol{U}_\mathbf{A} \boldsymbol{\Delta}_\mathbf{A}^\intercal - \boldsymbol{U}_\mathbf{A} \boldsymbol{S}^\intercal \boldsymbol{\Delta}_\mathbf{A} \boldsymbol{U}_\mathbf{A}^\intercal) \boldsymbol{S} \\
&= \boldsymbol{A}_0 \boldsymbol{S} + \boldsymbol{\Delta}_\mathbf{A} \boldsymbol{I} + \boldsymbol{U}_\mathbf{A} \boldsymbol{\Delta}_\mathbf{A}^\intercal \boldsymbol{S} - \boldsymbol{U}_\mathbf{A} \boldsymbol{S}^\intercal \boldsymbol{\Delta}_\mathbf{A} \boldsymbol{I} \\
&= \boldsymbol{A}_0 \boldsymbol{S} + \boldsymbol{Y} - \boldsymbol{A}_0 \boldsymbol{S} \\
&= \boldsymbol{Y}.
\end{aligned}$$

In the case of the inverse model we obtain

$$\begin{aligned}
\boldsymbol{H}_i \boldsymbol{Y} &= (\boldsymbol{H}_0 + \boldsymbol{\Delta}_\mathbf{H} \boldsymbol{U}_\mathbf{H}^\intercal + \boldsymbol{U}_\mathbf{H} \boldsymbol{\Delta}_\mathbf{H}^\intercal - \boldsymbol{U}_\mathbf{H} \boldsymbol{Y}^\intercal \boldsymbol{\Delta}_\mathbf{H} \boldsymbol{U}_\mathbf{H}^\intercal) \boldsymbol{Y} \\
&= \boldsymbol{H}_0 \boldsymbol{Y} + \boldsymbol{\Delta}_\mathbf{H} \boldsymbol{I} + \boldsymbol{U}_\mathbf{H} \boldsymbol{\Delta}_\mathbf{H}^\intercal \boldsymbol{Y} - \boldsymbol{U}_\mathbf{H} \boldsymbol{Y}^\intercal \boldsymbol{\Delta}_\mathbf{H} \boldsymbol{I} \\
&= \boldsymbol{H}_0 \boldsymbol{Y} + \boldsymbol{S} - \boldsymbol{H}_0 \boldsymbol{Y} \\
&= \boldsymbol{S}.
\end{aligned}$$

□

A.2.1 Conjugate Directions Method

Theorem 2.1 (Conjugate Directions Method)
Given a prior $p(\mathbf{H}) = \mathcal{N}\big(\mathbf{H}; \boldsymbol{H}_0, \boldsymbol{W}_0^\mathbf{H} \otimes \boldsymbol{W}_0^\mathbf{H}\big)$ such that $\boldsymbol{H}_0, \boldsymbol{W}_0^\mathbf{H} \in \mathbb{R}_{\text{sym}}^{n \times n}$ are positive definite, then the actions $\boldsymbol{s}_i$ of Algorithm 1 are $\boldsymbol{A}$-conjugate, i.e. it holds that

$$\boldsymbol{s}_j^\intercal \boldsymbol{A} \boldsymbol{s}_k = 0$$

for all $0 \leq j \neq k \leq i$.

Proof. Since $\boldsymbol{H}_0$ is assumed to be symmetric, the form of the posterior mean in Section 2.2.1 implies that $\boldsymbol{H}_i$ is symmetric for all $1 \leq i \leq n$. We will show conjugacy is by induction. To that end, consider the base case $i = 2$. We have

$$\begin{aligned}
\boldsymbol{s}_2^\intercal \boldsymbol{A} \boldsymbol{s}_1 &= \boldsymbol{r}_1^\intercal \boldsymbol{H}_1 \boldsymbol{A} \boldsymbol{s}_1 \\
&= (\boldsymbol{r}_0^\intercal - \alpha_1 \boldsymbol{y}_1^\intercal) \boldsymbol{H}_1 \boldsymbol{A} \boldsymbol{s}_1 \\
&= \left(\boldsymbol{r}_0^\intercal \boldsymbol{H}_1 - \frac{\boldsymbol{s}_1^\intercal \boldsymbol{r}_0}{\boldsymbol{s}_1^\intercal \boldsymbol{y}_1} \boldsymbol{y}_1^\intercal \boldsymbol{H}_1 \right) \boldsymbol{y}_1 \\
&= \boldsymbol{r}_0^\intercal \boldsymbol{s}_1 - \boldsymbol{s}_1^\intercal \boldsymbol{r}_0 = 0
\end{aligned}$$

where we used (A.1) and the definition of α_j in Algorithm 1. Now for the induction step, assume that $\boldsymbol{s}_j^\intercal \boldsymbol{A} \boldsymbol{s}_k = 0$ for all $j \neq k$ such that $1 \leq j, k \leq i$. We obtain for $1 \leq j \leq i$ that

$$\begin{aligned}
\boldsymbol{s}_{i+1}^\intercal \boldsymbol{A} \boldsymbol{s}_j &= \boldsymbol{r}_i^\intercal \boldsymbol{H}_i \boldsymbol{A} \boldsymbol{s}_j \\
&= \left(\boldsymbol{r}_0 - \sum_{\ell=1}^{i} \alpha_\ell \boldsymbol{y}_\ell \right)^\intercal \boldsymbol{H}_i \boldsymbol{y}_j \\
&= \boldsymbol{r}_0^\intercal \boldsymbol{s}_j - \sum_{\ell=1}^{i} \alpha_\ell \boldsymbol{y}_\ell^\intercal \boldsymbol{s}_j \\
&= \boldsymbol{r}_0^\intercal \boldsymbol{s}_j - \alpha_j \boldsymbol{y}_j^\intercal \boldsymbol{s}_j \\
&= \boldsymbol{r}_0^\intercal \boldsymbol{s}_j - \boldsymbol{s}_j^\intercal \boldsymbol{r}_{j-1} \\
&= \boldsymbol{r}_0^\intercal \boldsymbol{s}_j - \boldsymbol{s}_j^\intercal \left(\sum_{\ell=1}^{j-1} \boldsymbol{r}_0 - \alpha_\ell \boldsymbol{y}_\ell \right) \\
&= \boldsymbol{r}_0^\intercal \boldsymbol{s}_j - \boldsymbol{s}_j^\intercal \boldsymbol{r}_0 = 0
\end{aligned}$$

where we used the update equation of the residual $\boldsymbol{r}_j$ in Algorithm 1, the definition of α_j, the induction hypothesis and (A.1). This proves the statement. □

A.2.2 Relationship to the Conjugate Gradient Method

Theorem 2.2 (Connection to the Conjugate Gradient Method)
Given a scalar prior mean $\boldsymbol{A}_0 = \boldsymbol{H}_0^{-1} = \alpha \boldsymbol{I}$ with $\alpha > 0$, assume (2.3) and (2.4) hold, then the iterates $\boldsymbol{x}_i$ of Algorithm 1 are identical to the ones produced by the conjugate gradient method.

Proof. The proof outlined here is closely related to the proofs connecting Quasi-Newton methods to the conjugate gradient method [66, 161], but makes different assumptions on the prior distribution.

We begin by recognizing that the choice of step length α_i in Algorithm 1 is identical to the one in the conjugate gradient method [65]. Hence, it suffices to show that $\boldsymbol{s}_i \propto \boldsymbol{s}_i^{\mathrm{CG}}$. Theorem 2.1

established that Algorithm 1 is a conjugate directions method. Now by assumption $\boldsymbol{A}_0 = \alpha \boldsymbol{I}$ and $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1}$, therefore $\boldsymbol{s}_1 = \alpha \boldsymbol{I} \boldsymbol{r}_0 \propto \boldsymbol{r}_0 = \boldsymbol{s}_1^{\mathrm{CG}}$. It suffices show that $\boldsymbol{s}_i$ lies in the Krylov space $\mathcal{K}_i(\boldsymbol{A}, \boldsymbol{r}_0) = \{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{i-1}\boldsymbol{r}_0\}$ for all $0 < i \leq n$. This completes the argument, since $\mathcal{K}_i(\boldsymbol{A}, \boldsymbol{r}_0)$ is an i-dimensional subspace of $\mathbb{R}^n$ and thus $\boldsymbol{A}$-conjugacy uniquely determines the search directions up to scaling, as $\boldsymbol{A}$ is positive definite.

To complete the proof we proceed as follows. The posterior mean of the inverse model $\boldsymbol{H}_{i-1}$ at step $i-1$ maps an arbitrary vector $\boldsymbol{v} \in \mathbb{R}^n$ to $\operatorname{span}(\boldsymbol{H}_0\boldsymbol{v}, \boldsymbol{H}_0\boldsymbol{Y}_{1:i-1}, \boldsymbol{S}_{1:i-1}, \boldsymbol{W}_0^{\mathbf{H}}\boldsymbol{Y}_{1:i-1})$. This follows directly from its form in given in Section 2.2.1. By assumption $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1} = \alpha^{-1}\boldsymbol{I}$, therefore using (2.3) and (2.4) we have $\operatorname{span}(\boldsymbol{W}_0^{\mathbf{H}}\boldsymbol{Y}_{1:i-1}) = \operatorname{span}(\boldsymbol{Y}_{1:i-1})$. This implies $\boldsymbol{H}_{i-1}$ maps to $\operatorname{span}(\boldsymbol{v}, \boldsymbol{S}_{1:i-1}, \boldsymbol{Y}_{1:i-1})$ and thus $\boldsymbol{s}_i \in \operatorname{span}(\boldsymbol{r}_{i-1}, \boldsymbol{S}_{1:i-1}, \boldsymbol{Y}_{1:i-1})$. We will now show that $\operatorname{span}(\boldsymbol{r}_{i-1}, \boldsymbol{S}_{1:i-1}, \boldsymbol{Y}_{1:i-1}) \subset \mathcal{K}_i(\boldsymbol{A}, \boldsymbol{r}_0)$ by induction, completing the argument.

We begin with the base case. Since $\boldsymbol{H}_0$ is assumed to be scalar, we have $\boldsymbol{s}_1 \propto \boldsymbol{r}_0 \in \mathcal{K}_0(\boldsymbol{A}, \boldsymbol{r}_0)$ and therefore $\boldsymbol{y}_1 = \boldsymbol{A}\boldsymbol{s}_1$ and $\boldsymbol{r}_1 = \boldsymbol{r}_0 - \alpha_1 \boldsymbol{y}_1$ are in $\mathcal{K}_1(\boldsymbol{A}, \boldsymbol{r}_0)$. For the induction step assume $\operatorname{span}(\boldsymbol{r}_{i-1}, \boldsymbol{S}_{1:i-1}, \boldsymbol{Y}_{1:i-1}) \subset \mathcal{K}_i(\boldsymbol{A}, \boldsymbol{r}_0)$. The definition of the policy of Algorithm 1 gives

$$\boldsymbol{s}_i = -\,\mathbb{E}(\mathbf{H})\,\boldsymbol{r}_{i-1} \propto \boldsymbol{H}_{i-1}\boldsymbol{r}_{i-1} \in \operatorname{span}(\boldsymbol{r}_{i-1}, \boldsymbol{S}_{1:i-1}, \boldsymbol{Y}_{1:i-1}) \subset \mathcal{K}_i(\boldsymbol{A}, \boldsymbol{r}_0),$$

where we used the induction hypothesis. This implies that $\boldsymbol{y}_i = \boldsymbol{A}\boldsymbol{s}_i \in \mathcal{K}_{i+1}(\boldsymbol{A}, \boldsymbol{r}_0)$ and $\boldsymbol{r}_i = \boldsymbol{r}_{i-1} - \alpha_i \boldsymbol{y}_i \in \mathcal{K}_{i+1}(\boldsymbol{A}, \boldsymbol{r}_0)$ by the definition of the Krylov space. Therefore, $\operatorname{span}(\boldsymbol{r}_i, \boldsymbol{S}_{1:i}, \boldsymbol{Y}_{1:i}) \subset \mathcal{K}_{i+1}(\boldsymbol{A}, \boldsymbol{r}_0)$. This completes the proof. □

A.3 Prior Covariance Class

A.3.1 Hereditary Positive-Definiteness

Proposition 2.1 (Hereditary Positive Definiteness [70, 81])
Let $\boldsymbol{A}_0 \in \mathbb{R}^{n\times n}_{\mathrm{sym}}$ be positive definite. Assume the actions $\boldsymbol{S}$ are $\boldsymbol{A}$-conjugate and $\boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S} = \boldsymbol{Y}$, then $\boldsymbol{A}_i$ is symmetric positive definite.

Proof. This is shown in Hennig and Kiefel [70]. We give an identical proof in our notation as a reference to the reader. By Theorem 7.5 in Dennis and Moré [81] it holds that if $\boldsymbol{A}_i$ is positive definite and $\boldsymbol{s}_{i+1}^{\intercal}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{s}_{i+1} \neq 0$, then $\boldsymbol{A}_{i+1}$ is positive definite if and only if $\det(\boldsymbol{A}_{i+1}) > 0$. By the matrix determinant lemma and the recursive formulation of the posterior we have

$$\det(\boldsymbol{A}_{i+1}) = \det(\boldsymbol{A}_i)\Bigg(\frac{1}{(\boldsymbol{s}_{i+1}^{\intercal}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{s}_{i+1})^2}\big((\boldsymbol{y}_{i+1}^{\intercal}\boldsymbol{A}_i^{-1}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{s}_{i+1})^2 - (\boldsymbol{y}_{i+1}^{\intercal}\boldsymbol{A}_i^{-1}\boldsymbol{y}_{i+1}) \cdot (\boldsymbol{s}_{i+1}^{\intercal}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{A}_i^{-1}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{s}_{i+1}) + (\boldsymbol{s}_{i+1}^{\intercal}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{A}_i^{-1}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{s}_{i+1})(\boldsymbol{y}_{i+1}^{\intercal}\boldsymbol{s}_{i+1})\big)\Bigg)$$

Hence it suffices to show that

$$0 < (\boldsymbol{y}_{i+1}^{\intercal}\boldsymbol{A}_i^{-1}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{s}_{i+1})^2 - (\boldsymbol{y}_{i+1}^{\intercal}\boldsymbol{A}_i^{-1}\boldsymbol{y}_{i+1})(\boldsymbol{s}_{i+1}^{\intercal}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{A}_i^{-1}\boldsymbol{W}_i^{\mathbf{A}}\boldsymbol{s}_{i+1})$$

$$+ (\boldsymbol{s}_{i+1}^\intercal \boldsymbol{W}_i^{\mathbf{A}} \boldsymbol{A}_i^{-1} \boldsymbol{W}_i^{\mathbf{A}} \boldsymbol{s}_{i+1})(\boldsymbol{y}_{i+1}^\intercal \boldsymbol{s}_{i+1}),$$

which simplifies to

$$\boldsymbol{y}_{i+1}^\intercal \boldsymbol{A}_i^{-1} \boldsymbol{y}_{i+1} - \frac{(\boldsymbol{y}_{i+1}^\intercal \boldsymbol{A}_i^{-1} \boldsymbol{W}_i^{\mathbf{A}} \boldsymbol{s}_{i+1})^2}{\boldsymbol{s}_{i+1}^\intercal \boldsymbol{W}_i^{\mathbf{A}} \boldsymbol{A}_i^{-1} \boldsymbol{W}_i^{\mathbf{A}} \boldsymbol{s}_{i+1}} < \boldsymbol{y}_{i+1}^\intercal \boldsymbol{s}_{i+1}$$

Now by $\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{S} = \boldsymbol{Y}$, we have $\boldsymbol{W}_i^{\mathbf{A}} \boldsymbol{s}_{i+1} = \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{s}_{i+1} = \boldsymbol{y}_{i+1}$ and the above reduces to

$$0 < \boldsymbol{s}_{i+1}^\intercal \boldsymbol{A} \boldsymbol{s}_{i+1},$$

which is fulfilled by the assumption that $\boldsymbol{A}$ is positive definite. Thus $\boldsymbol{A}_{i+1}$ is positive definite. Symmetry follows immediately from the form of the posterior mean. □

A.3.2 Posterior Correspondence

Definition 2.1
Let $\boldsymbol{A}_i$ and $\boldsymbol{H}_i$ be the means of $\mathbf{A}$ and $\mathbf{H}$ at step i. We say a prior induces *posterior correspondence* if

$$\boldsymbol{A}_i^{-1} = \boldsymbol{H}_i \tag{2.1}$$

for all steps i of the solver. If only

$$\boldsymbol{A}_i^{-1} \boldsymbol{Y} = \boldsymbol{H}_i \boldsymbol{Y}, \tag{2.2}$$

we say that *weak posterior correspondence* holds.

Matrix-variate Normal Prior

We begin by establishing posterior correspondence in the case of general matrix-variate normal priors. We first prove a general non-constructive condition and close with a sufficient condition for correspondence which limits the possible choices of covariance factors to a specific class.

Lemma A.1 (General Correspondence)
Let $1 \leq i \leq n$, $\boldsymbol{W}_0^{\mathbf{A}}, \boldsymbol{W}_0^{\mathbf{H}}$ *symmetric positive-definite and assume* $\boldsymbol{A}_0^{-1} = \boldsymbol{H}_0$, *then* (2.1) *holds if and only if*

$$\mathbf{0} = (\boldsymbol{A}\boldsymbol{S} - \boldsymbol{A}_0\boldsymbol{S}) \left[(\boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} \boldsymbol{A}\boldsymbol{S})^{-1} \boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} - (\boldsymbol{S}^\intercal \boldsymbol{A}^\intercal \boldsymbol{W}_0^{\mathbf{H}} \boldsymbol{A}\boldsymbol{S})^{-1} \boldsymbol{S}^\intercal \boldsymbol{A}^\intercal \boldsymbol{W}_0^{\mathbf{H}} \right].$$

Proof. By the matrix inversion lemma we have

$$\begin{aligned} 0 &= \boldsymbol{A}_i^{-1} - \boldsymbol{H}_i \\ &= \left(\boldsymbol{A}_0 + (\boldsymbol{Y} - \boldsymbol{A}_0\boldsymbol{S})(\boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{S})^{-1} \boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \right)^{-1} \end{aligned}$$

$$
\begin{aligned}
&- \boldsymbol{H}_0 - (\boldsymbol{S} - \boldsymbol{H}_0\boldsymbol{Y})(\boldsymbol{Y}^\intercal \boldsymbol{W}_0^{\mathbf{H}} \boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal \boldsymbol{W}_0^{\mathbf{H}} \\
&= \boldsymbol{A}_0^{-1} - \boldsymbol{A}_0^{-1}(\boldsymbol{Y} - \boldsymbol{A}_0\boldsymbol{S})(\boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{S} + \boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1}(\boldsymbol{Y} - \boldsymbol{A}_0\boldsymbol{S}))^{-1}\boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} \\
&\quad - \boldsymbol{A}_0^{-1} - \boldsymbol{A}_0^{-1}(\boldsymbol{A}_0\boldsymbol{S} - \boldsymbol{Y})(\boldsymbol{Y}^\intercal \boldsymbol{W}_0^{\mathbf{H}} \boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal \boldsymbol{W}_0^{\mathbf{H}} \\
&= -\boldsymbol{A}_0^{-1}(\boldsymbol{Y} - \boldsymbol{A}_0\boldsymbol{S}) \left[(\boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1}\boldsymbol{Y})^{-1}\boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} - (\boldsymbol{Y}^\intercal \boldsymbol{W}_0^{\mathbf{H}} \boldsymbol{Y})^{-1}\boldsymbol{Y}^\intercal \boldsymbol{W}_0^{\mathbf{H}}\right],
\end{aligned}
$$

where we used the assumption $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1}$. Left-multiplying with $-\boldsymbol{A}_0$ and using $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{S}$ completes the proof. □

Corollary A.1 (Correspondence at Convergence)
Let $i = n$, $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1}$ and assume $\boldsymbol{S}$ has full rank, i.e. the linear solver has performed n linearly independent actions, then (2.1) holds for any symmetric positive-definite choice of $\boldsymbol{W}_0^{\mathbf{A}}$ and $\boldsymbol{W}_0^{\mathbf{H}}$.

Proof. By assumption, $\boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1}$ and $\boldsymbol{S}^\intercal \boldsymbol{A}^\intercal \boldsymbol{W}_0^{\mathbf{H}}$ are invertible. Then by Lemma A.1 the correspondence condition (2.1) holds. □

Theorem A.1 (Sufficient Condition for Correspondence)
Let $1 \leq i \leq n$ arbitrary and assume $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1}$. Assume $\boldsymbol{W}_0^{\boldsymbol{A}}, \boldsymbol{A}_0, \boldsymbol{W}_0^{\boldsymbol{H}}$ satisfy

$$0 = \boldsymbol{S}^\intercal (\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} - \boldsymbol{A}^\intercal \boldsymbol{W}_0^{\mathbf{H}}) \tag{A.2}$$

or equivalently let $\boldsymbol{B}_{\langle \boldsymbol{S}\rangle^\perp} \in \mathbb{R}^{n \times i}$ be a basis of the orthogonal space $\langle \boldsymbol{S}\rangle^\perp$ spanned by the actions. For $\boldsymbol{\Phi} \in \mathbb{R}^{(n-i)\times n}$ arbitrary, if

$$\boldsymbol{W}_0^{\mathbf{H}} = \boldsymbol{A}^{-\intercal}(\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} - \boldsymbol{B}_{\langle \boldsymbol{S}\rangle^\perp}\boldsymbol{\Phi}) \tag{A.3}$$

and the commutation relations

$$[\boldsymbol{A}_0, \boldsymbol{A}] = \boldsymbol{0} \tag{A.4}$$
$$[\boldsymbol{W}_0^{\mathbf{A}}, \boldsymbol{A}] = \boldsymbol{0} \tag{A.5}$$
$$[\boldsymbol{B}_{\langle \boldsymbol{S}\rangle^\perp}\boldsymbol{\Phi}, \boldsymbol{A}] = \boldsymbol{0} \tag{A.6}$$

are fulfilled, then $\boldsymbol{W}_0^{\mathbf{H}}$ is symmetric and (2.1) holds.

Proof. By assumption $\boldsymbol{W}_0^{\mathbf{A}}$ is symmetric positive-definite and (A.2) is equivalent to $\boldsymbol{S}^\intercal \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} = \boldsymbol{S}^\intercal \boldsymbol{A}^\intercal \boldsymbol{W}_0^{\mathbf{H}}$, which implies (A.1). Now, assumption (A.2) is equivalent to columns of the difference $\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} - \boldsymbol{A}^\intercal \boldsymbol{W}_0^{\mathbf{H}}$ lying in $\langle \boldsymbol{S}\rangle^\perp$, i.e. we can choose a basis $\boldsymbol{B}_{\langle \boldsymbol{S}\rangle^\perp}$ and coefficient matrix $\boldsymbol{\Phi}$ such that

$$\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} - \boldsymbol{A}^\intercal \boldsymbol{W}_0^{\mathbf{H}} = \boldsymbol{B}_{\langle \boldsymbol{S}\rangle^\perp}\boldsymbol{\Phi}.$$

Rearranging the above gives (A.3). With the commutation relations and

$$[\boldsymbol{A}, \boldsymbol{B}] = \mathbf{0} \iff [\boldsymbol{A}^{-1}, \boldsymbol{B}] = \mathbf{0} \iff [\boldsymbol{A}, \boldsymbol{B}^{-1}] = \mathbf{0} \iff [\boldsymbol{A}^{-1}, \boldsymbol{B}^{-1}] = \mathbf{0}$$

it holds that

$$(\boldsymbol{W}_0^{\mathbf{H}})^\intercal = \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} \boldsymbol{A}^{-1} - \boldsymbol{B}_{\langle \boldsymbol{S} \rangle^\perp} \boldsymbol{\Phi} \boldsymbol{A}^{-1} = \boldsymbol{A}^{-\intercal} \boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} - \boldsymbol{A}^{-\intercal} \boldsymbol{B}_{\langle \boldsymbol{S} \rangle^\perp} \boldsymbol{\Phi} = \boldsymbol{W}_0^{\mathbf{H}}$$

hence $\boldsymbol{W}_0^{\mathbf{H}}$ is symmetric. Finally, by Lemma A.1 posterior mean correspondence (2.1) holds. □

If we want to ensure correspondence for all iterations, (A.6) is trivially satisfied. The question now becomes what form can $\boldsymbol{A}_0$ and $\boldsymbol{W}_0^{\mathbf{A}}$ take in order to ensure symmetric $\boldsymbol{W}_0^{\mathbf{H}}$. This comes down to finding matrices which commute with $\boldsymbol{A}$.

Lemma A.2 (Commuting Matrices of a Symmetric Matrix)
Let $r \in \mathbb{N}$, $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symmetric. Assume $\boldsymbol{M}$ has the form

$$\boldsymbol{M} = \mathfrak{p}_r(\boldsymbol{A}) = \sum_{i=0}^{r} c_i \boldsymbol{A}^i$$

for a set of coefficients $c_i \in \mathbb{R}$, then $\boldsymbol{M}$ and $\boldsymbol{A}$ commute. If $\boldsymbol{A}$ has n distinct eigenvalues, $\boldsymbol{M}$ is diagonalizable and $[\boldsymbol{M}, \boldsymbol{A}] = \mathbf{0}$, then

$$\boldsymbol{M} = \mathfrak{p}_{n-1}(\boldsymbol{A}),$$

i.e. $\boldsymbol{M}$ is a polynomial in $\boldsymbol{A}$ of degree at most $n - 1$.

Proof. The first result follows immediately since

$$\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A} = \mathfrak{p}_r(\boldsymbol{A}) \boldsymbol{A} = \sum_{i=0}^{r} c_i \boldsymbol{A}^{i+1} = \boldsymbol{A} \mathfrak{p}_r(\boldsymbol{A}) = \boldsymbol{A} \boldsymbol{W}_0^{\mathbf{A}}.$$

Assume now that $\boldsymbol{A}$ has n distinct eigenvalues $\lambda_0, \dots, \lambda_{n-1}$, $\boldsymbol{M}$ is diagonalizable and $\boldsymbol{M}$ and $\boldsymbol{A}$ commute. Now, if and only if $[\boldsymbol{A}, \boldsymbol{M}] = 0$, then $\boldsymbol{A}$ and $\boldsymbol{M}$ are simultaneously diagonalizable by Theorem 5.2 in Conrad [162], i.e. we can find a common basis in which both $\boldsymbol{A}$ and $\boldsymbol{M}$ are represented by diagonal matrices. Hence, the set of matrices commuting with $\boldsymbol{A}$ forms an n-dimensional subspace $\mathcal{U}_n \subset \mathbb{R}^{n \times n}$. Now, by the first part of this proof $\{\boldsymbol{I}, \boldsymbol{A}, \dots, \boldsymbol{A}^{n-1}\} \subset \mathcal{U}_n$. It remains to be shown, that this set forms a basis of $\mathcal{U}_n$. By isomorphism of finite dimensional vector spaces this is equivalent to proving that

$$\{\boldsymbol{b}_0, \boldsymbol{b}_1, \dots, \boldsymbol{b}_{n-1}\} \coloneqq \left\{ \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_{n-1} \end{pmatrix}, \dots, \begin{pmatrix} \lambda_0^{n-1} \\ \vdots \\ \lambda_{n-1}^{n-1} \end{pmatrix} \right\}$$

forms a basis of $\mathbb{R}^n$. It suffices to show that all $\boldsymbol{b}_i$ are independent. Assume the contrary, then $\sum_{i=0}^{n-1} \alpha_i \boldsymbol{b}_i = 0$ for some $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{R}$, such that not all $\alpha_i = 0$. This implies that the polynomial $\sum_{i=0}^{n-1} \alpha_i x^i$ has n zeros $\lambda_0, \dots, \lambda_{n-1}$. This contradicts the fundamental theorem of algebra, concluding the proof. □

The above suggests that tractable choices of $\boldsymbol{A}_0$ and $\boldsymbol{W}_0^{\mathbf{A}}$ for the non-symmetric matrix-variate prior, which imply symmetric $\boldsymbol{W}_0^{\mathbf{H}}$, are of polynomial form in $\boldsymbol{A}$.

> **Example A.1** (Posterior Correspondence Covariance Class)
> Tractable choices of the prior parameters in the $\mathbf{A}$ view, which satisfy posterior correspondence and the commutation relations are for example
>
> $$\boldsymbol{A}_0 = c_0 \boldsymbol{I} \qquad \text{and} \qquad \boldsymbol{W}_0^{\mathbf{A}} = \sum_{j=1}^{n-1} c_j \boldsymbol{A}^j,$$
>
> where $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1}$ with $c_j \in \mathbb{R}$. Motivated by $\operatorname{tr}(\boldsymbol{A}) \overset{!}{=} \operatorname{tr}(\boldsymbol{A}_0)$ an initial choice could be $c_0 = n^{-1} \operatorname{tr}(\boldsymbol{A})$.

In practice, we do not actually require $\boldsymbol{W}_0^{\mathbf{A}}$. We only ever need access to $\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{S}$.

Symmetric Matrix-variate Normal Prior

We now turn to the symmetric model, which we assumed throughout Section 2.2. We prove Theorem 2.3, the main result of this section demonstrating *weak posterior correspondence* for the symmetric Kronecker covariance, by employing the matrix inversion lemma for the posterior mean $\boldsymbol{A}_i$.

> **Theorem 2.3** (Weak Posterior Correspondence)
> *Let $\boldsymbol{W}_0^{\mathbf{H}} \in \mathbb{R}^{n \times n}_{\text{sym}}$ be positive definite. Assume $\boldsymbol{H}_0 = \boldsymbol{A}_0^{-1}$, and $\boldsymbol{W}_0^{\mathbf{A}}, \boldsymbol{A}_0, \boldsymbol{W}_0^{\mathbf{H}}$ satisfy*
>
> $$\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{S} = \boldsymbol{Y}, \tag{2.3}$$
> $$\boldsymbol{S}^\intercal (\boldsymbol{W}_0^{\mathbf{A}} \boldsymbol{A}_0^{-1} - \boldsymbol{A} \boldsymbol{W}_0^{\mathbf{H}}) = \boldsymbol{0}. \tag{2.4}$$
>
> *Then weak posterior correspondence holds for the symmetric Kronecker covariance.*

Proof. Without loss of generality $\boldsymbol{S}^\intercal \boldsymbol{A} \boldsymbol{S} = \boldsymbol{I}$, i.e. only the direction of the action matters in Algorithm 1, not its magnitude. This can be seen from the forms of $\boldsymbol{A}_i$ and $\boldsymbol{H}_i$ in Section 2.2.1. Any positive factor $\alpha > 0$ of $\boldsymbol{s}_i$ cancels in the update expressions. We aim to show that $\boldsymbol{A}_i^{-1} \boldsymbol{Y} = \boldsymbol{H}_i \boldsymbol{Y}$. Expanding the right hand side we have using (A.1), that $\boldsymbol{H}_i \boldsymbol{Y} = \boldsymbol{S}$. Then

by Lemma S3 and Lemma S6 of Wenger and Hennig [1] and $\boldsymbol{S}^\intercal \boldsymbol{A} \boldsymbol{S} = \boldsymbol{I}$, the left hand side evaluates to

$$\begin{aligned}
\boldsymbol{A}_i^{-1} \boldsymbol{Y} &= (\boldsymbol{A}_0^{-1} - \boldsymbol{F}) \boldsymbol{Y} \\
&= (\boldsymbol{A}_0^{-1} - \boldsymbol{A}_0^{-1} \boldsymbol{A} \boldsymbol{S} (\boldsymbol{I} + \boldsymbol{\Pi})^{-1} \boldsymbol{S}^\intercal \boldsymbol{A} \boldsymbol{A}_0^{-1} + \boldsymbol{S} \boldsymbol{S}^\intercal) \boldsymbol{A} \boldsymbol{S} \\
&= \boldsymbol{A}_0^{-1} \boldsymbol{A} \boldsymbol{S} - \boldsymbol{A}_0^{-1} \boldsymbol{A} \boldsymbol{S} + \boldsymbol{S} \\
&= \boldsymbol{S} \\
&= \boldsymbol{H}_i \boldsymbol{Y}.
\end{aligned}$$

This concludes the proof. □

This theorem shows that for a certain choice of symmetric matrix-variate normal prior the estimated inverse of the matrix $\boldsymbol{H}_i$ corresponds to the inverse of the estimated matrix $\boldsymbol{A}_i^{-1}$. It also shows that both act like $\boldsymbol{A}^{-1}$ on the space spanned by $\boldsymbol{Y}$, consistent with the interpretation of them representing the best guess for the inverse $\boldsymbol{A}^{-1}$.

Appendix

B

Appendix of Chapter 3

B.1 Connections to Other GP Approximations

B.1.1 Pivoted Cholesky Decomposition

Theorem B.1 (Pivoted Cholesky Decomposition)
Let $(j_i)_{i=1}^n$ be a set of indices defining the pivot elements of the pivoted Cholesky decomposition and $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ the corresponding permutation matrix. Assume the actions of

Algorithm 2 are given by the standard unit vectors $\boldsymbol{s}_i = \boldsymbol{P}\boldsymbol{e}_i = \boldsymbol{e}_{j_i}$, *i.e.*

$$(\boldsymbol{s}_i)_j = (\boldsymbol{e}_{j_i})_j = \begin{cases} 1 & \text{if } j = j_i \\ 0 & \text{otherwise} \end{cases}. \tag{B.1}$$

Then Algorithm 2 recovers the pivoted Cholesky decomposition, i.e. it holds for all $i \in \{0, \dots, n\}$ *that*

$$\boldsymbol{P}^\intercal \boldsymbol{Q}_i \boldsymbol{P} = \boldsymbol{L}_i \boldsymbol{L}_i^\intercal, \tag{B.2}$$

where $\boldsymbol{L}_i \in \mathbb{R}^{n \times i}$ *is the (partial) Cholesky factor of* $\boldsymbol{P}^\intercal \hat{\boldsymbol{K}} \boldsymbol{P}$ *as computed by Algorithm 5.*

Proof. We proceed by induction. Assume (B.2) holds after i iterations of Algorithm 2. For the base case $i = 0$, it holds by assumption that $\boldsymbol{P}^\intercal \boldsymbol{Q}_0 \boldsymbol{P} = \boldsymbol{P}^\intercal \hat{\boldsymbol{K}} \boldsymbol{C}_0 \hat{\boldsymbol{K}} \boldsymbol{P} = \boldsymbol{0}$. Now for the induction step $i \to i+1$, we have

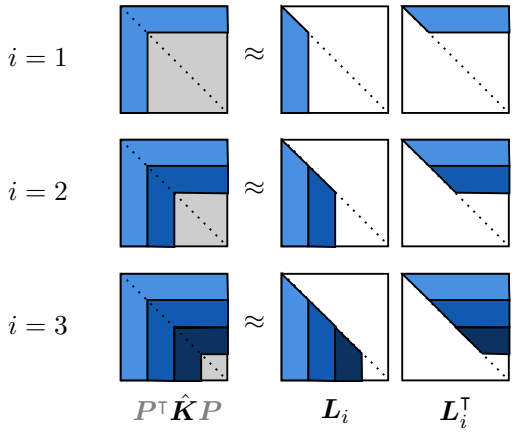
$$\begin{aligned}
\frac{1}{\eta_{i+1}} \hat{\boldsymbol{K}} \boldsymbol{d}_i \boldsymbol{d}_i^\intercal \hat{\boldsymbol{K}} &= \frac{1}{\eta_{i+1}} \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_i \hat{\boldsymbol{K}} \boldsymbol{s}_{i+1} \boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_i \hat{\boldsymbol{K}} \\
&= \frac{1}{\eta_{i+1}} \hat{\boldsymbol{K}} (\boldsymbol{\Sigma}_0 - \boldsymbol{C}_i) \hat{\boldsymbol{K}} \boldsymbol{s}_{i+1} \boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}} (\boldsymbol{\Sigma}_0 - \boldsymbol{C}_i) \hat{\boldsymbol{K}} \\
&= \frac{1}{\eta_{i+1}} (\hat{\boldsymbol{K}} - \boldsymbol{Q}_i) \boldsymbol{s}_{i+1} \boldsymbol{s}_{i+1}^\intercal (\hat{\boldsymbol{K}} - \boldsymbol{Q}_i) \\
&\overset{\text{IH}}{=} \frac{1}{\eta_{i+1}} (\hat{\boldsymbol{K}} - \boldsymbol{P}\boldsymbol{L}_i \boldsymbol{L}_i^\intercal \boldsymbol{P}^\intercal) \boldsymbol{s}_{i+1} \boldsymbol{s}_{i+1}^\intercal (\hat{\boldsymbol{K}} - \boldsymbol{P}\boldsymbol{L}_i \boldsymbol{L}_i^\intercal \boldsymbol{P}^\intercal) \\
&= \frac{(\hat{\boldsymbol{K}} - \boldsymbol{P}\boldsymbol{L}_i \boldsymbol{L}_i^\intercal \boldsymbol{P}^\intercal) \boldsymbol{P}\boldsymbol{e}_{i+1}}{\sqrt{\boldsymbol{e}_{i+1}^\intercal \boldsymbol{P}^\intercal (\hat{\boldsymbol{K}} - \boldsymbol{P}\boldsymbol{L}_i \boldsymbol{L}_i^\intercal \boldsymbol{P}^\intercal) \boldsymbol{P}\boldsymbol{e}_{i+1}}} \frac{\boldsymbol{e}_{i+1}^\intercal \boldsymbol{P}^\intercal (\hat{\boldsymbol{K}} - \boldsymbol{P}\boldsymbol{L}_i \boldsymbol{L}_i^\intercal \boldsymbol{P}^\intercal)}{\sqrt{\boldsymbol{e}_{i+1}^\intercal \boldsymbol{P}^\intercal (\hat{\boldsymbol{K}} - \boldsymbol{P}\boldsymbol{L}_i \boldsymbol{L}_i^\intercal \boldsymbol{P}^\intercal) \boldsymbol{P}\boldsymbol{e}_{i+1}}} \\
&= \frac{\boldsymbol{P}(\boldsymbol{P}^\intercal \hat{\boldsymbol{K}} \boldsymbol{P} - \boldsymbol{L}_i \boldsymbol{L}_i^\intercal) \boldsymbol{e}_{i+1}}{\sqrt{\boldsymbol{e}_{i+1}^\intercal (\boldsymbol{P}^\intercal \hat{\boldsymbol{K}} \boldsymbol{P} - \boldsymbol{L}_i \boldsymbol{L}_i^\intercal) \boldsymbol{e}_{i+1}}} \frac{\boldsymbol{e}_{i+1}^\intercal (\boldsymbol{P}^\intercal \hat{\boldsymbol{K}} \boldsymbol{P} - \boldsymbol{L}_i \boldsymbol{L}_i^\intercal) \boldsymbol{P}^\intercal}{\sqrt{\boldsymbol{e}_{i+1}^\intercal (\boldsymbol{P}^\intercal \hat{\boldsymbol{K}} \boldsymbol{P} - \boldsymbol{L}_i \boldsymbol{L}_i^\intercal) \boldsymbol{e}_{i+1}}} \\
&= \boldsymbol{P}\boldsymbol{l}_{i+1} \boldsymbol{l}_{i+1}^\intercal \boldsymbol{P}^\intercal.
\end{aligned}$$

where $\boldsymbol{l}_{i+1}$ is given by Algorithm 5. Combining this with one more use of the induction hypothesis we obtain

$$\begin{aligned}
\boldsymbol{P}^\intercal \boldsymbol{Q}_{i+1} \boldsymbol{P} &= \boldsymbol{P}^\intercal \boldsymbol{Q}_i \boldsymbol{P} + \frac{1}{\eta_{i+1}} \boldsymbol{P}^\intercal \hat{\boldsymbol{K}} \boldsymbol{d}_{i+1} \boldsymbol{d}_{i+1}^\intercal \hat{\boldsymbol{K}} \boldsymbol{P} \\
&= \boldsymbol{L}_i \boldsymbol{L}_i^\intercal + \boldsymbol{l}_{i+1} \boldsymbol{l}_{i+1}^\intercal = \begin{pmatrix} \boldsymbol{L}_i & \boldsymbol{l}_{i+1} \end{pmatrix} \begin{pmatrix} \boldsymbol{L}_i^\intercal \\ \boldsymbol{l}_{i+1}^\intercal \end{pmatrix} = \boldsymbol{L}_{i+1} \boldsymbol{L}_{i+1}^\intercal
\end{aligned}$$

This proves the claim.

□



Algorithm 5: (Pivoted) Cholesky Decomposition

Input: Kernel matrix $\hat{K}$, permutation matrix P
Output: Triangular L_i, s.t. $L_i L_i^\intercal \approx P^\intercal \hat{K} P$

procedure CHOLESKY($\hat{K}, P$)
  $A \leftarrow P^\intercal \hat{K} P$
  for $i \in \{1, \dots, n\}$ **do**
    $l_i \leftarrow A_{:i} / \sqrt{A_{ii}}$
    $A \leftarrow A - l_i l_i^\intercal = P^\intercal \hat{K} P - L_i L_i^\intercal$
    $L_i = \begin{pmatrix} L_{i-1} & l_i \end{pmatrix}$
  return L_i

Figure B.1: *Cholesky decomposition.* Every column added to the lower triangular Cholesky factor L defines the ith "right angle ruler"-pattern in $P^\intercal \hat{K} P$. The bottom right matrix in gray given by $P^\intercal \hat{K} P - L_i L_i^\intercal = P^\intercal \hat{K} P - \sum_{j=1}^{i} l_j l_j^\intercal$ changes every iteration.

B.1.2 Singular / Eigenvalue Decomposition

Theorem B.2 (Singular / Eigenvalue Decomposition)
Let the actions $s_i = u_i$ of Algorithm 2 be given by the eigenvectors u_i of $\hat{K}$ in arbitrary order. Then for $i \in \{1, \dots, n\}$ it holds that

$$C_i = U_i \Lambda_i^{-1} U_i^\intercal = \mathrm{SVD}_i(\hat{K}^{-1})$$
$$Q_i = U_i \Lambda_i U_i^\intercal = \mathrm{SVD}_i(\hat{K}),$$

where $U = \begin{pmatrix} u_1, \dots, u_i \end{pmatrix} \in \mathbb{R}^{n \times i}$ and $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_i) \in \mathbb{R}^{i \times i}$ is the diagonal matrix of eigenvalues of $\hat{K}$ with the order given by the order of the actions.

Proof. It holds by assumption and (B.21), that

$$C_i = S_i (S_i^\intercal \hat{K} S_i)^{-1} S_i^\intercal = U_i (U_i^\intercal \hat{K} U_i)^{-1} U_i^\intercal = U_i \Lambda_i^{-1} U_i^\intercal,$$

as well as

$$Q_i = \hat{K} C_i \hat{K} = \hat{K} U_i \Lambda_i^{-1} U_i^\intercal \hat{K} = U_i \Lambda_i \Lambda_i^{-1} \Lambda_i U_i^\intercal = U_i \Lambda_i U_i^\intercal$$

This proves the claim. □

B.1.3 Conjugate Gradient Method

Algorithm 6: Preconditioned Conjugate Gradient Method [71]

Input: Kernel matrix $\hat{\boldsymbol{K}}$, targets $\boldsymbol{y}$, prior mean $\boldsymbol{\mu}$, preconditioner $\hat{\boldsymbol{P}}$
Output: Representer weights $\boldsymbol{v}_i \approx \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

procedure CG($\hat{\boldsymbol{K}}, \boldsymbol{y} - \boldsymbol{\mu}, \hat{\boldsymbol{P}}$)
  $\boldsymbol{v}_0 \leftarrow \boldsymbol{0}$
  $\boldsymbol{s}_0 \leftarrow \boldsymbol{0}$
  while $\|\boldsymbol{r}_i\|_2 > \max(\delta_{\text{rtol}}\|\boldsymbol{y}\|_2, \delta_{\text{atol}})$ **and** $i < i_{\max}$ **do**
    $\boldsymbol{r}_{i-1} \leftarrow (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}}\boldsymbol{v}_{i-1}$
    $\boldsymbol{s}_i \leftarrow \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1} - \frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1})^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i-1}}{\boldsymbol{s}_{i-1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i-1}}\boldsymbol{s}_{i-1}$
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_{i-1} + \frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1})^\intercal \boldsymbol{r}_{i-1}}{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i}\boldsymbol{s}_i$
  return $\boldsymbol{v}_i$

Theorem B.3 (Preconditioned Conjugate Gradient Method)
Let $\hat{\boldsymbol{P}} \in \mathbb{R}^{n\times n}$ be a symmetric positive definite preconditioner. Assume the actions of Algorithm 2 are given by

$$\begin{aligned} \boldsymbol{s}_1^{\text{CG}} &= \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_0 \\ \boldsymbol{s}_i^{\text{CG}} &= \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1} - \frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1})^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i-1}}{\boldsymbol{s}_{i-1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i-1}}\boldsymbol{s}_{i-1} \end{aligned} \tag{B.3}$$

the preconditioned conjugate gradient method. Then Algorithm 2 recovers preconditioned CG initialized at $\boldsymbol{v}_0^{\text{CG}} = \boldsymbol{0}$, i.e. it holds for $i \in \{1, \dots, n\}$ that

$$\boldsymbol{s}_i = \boldsymbol{d}_i = \boldsymbol{s}_i^{\text{CG}} \tag{B.4}$$
$$\boldsymbol{v}_i = \boldsymbol{v}_i^{\text{CG}} \tag{B.5}$$
$$\boldsymbol{r}_{i-1} = \boldsymbol{r}_{i-1}^{\text{CG}}. \tag{B.6}$$

Proof. By assumption $\boldsymbol{s}_i = \boldsymbol{s}_i^{\text{CG}}$ for all i. We prove the remaining claims by induction. For the base case we have by assumption $\boldsymbol{d}_0 = \boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{s}_0 = \boldsymbol{s}_0 = \boldsymbol{s}_0^{\text{CG}}$ and $\boldsymbol{v}_0 = \boldsymbol{0} = \boldsymbol{v}_0^{\text{CG}}$. Now for the induction step $i \to i+1$ assume the hypotheses (B.4), (B.5) and (B.6) hold $\forall j \leq i$. Using the properties of CG it holds for $j' < i$ that

$$\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{j'} = 0 \tag{B.7}$$
$$(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \boldsymbol{s}_{j'} = 0 \tag{B.8}$$
$$(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \boldsymbol{r}_{j'} = 0 \tag{B.9}$$
$$\langle \boldsymbol{s}_1, \dots, \boldsymbol{s}_i \rangle = \langle \boldsymbol{r}_0, \hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}\boldsymbol{r}_0, \dots, (\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}})^{i-1}\boldsymbol{r}_0 \rangle = \langle \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_0, \dots, \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1} \rangle \tag{B.10}$$

We now first show $\hat{\boldsymbol{K}}$-conjugacy of the actions in iteration $i+1$. We have for $j \leq i$ that

$$\begin{aligned}\boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}} \boldsymbol{s}_j &= \big(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i - \frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i}{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i}\boldsymbol{s}_i\big)^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_j \\ &= (\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_j - \frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i}{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i}\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_j\end{aligned}$$

Now if $j = i$, clearly $\boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_j = \boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i = 0$. If $j < i$, we have using (B.10), that

$$\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}\boldsymbol{s}_j \in \langle \hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}\boldsymbol{r}_0, \ldots, (\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}})^j \boldsymbol{r}_0\rangle \subset \langle \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_0, \ldots, \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_j\rangle. \tag{B.11}$$

Therefore we obtain for $j < i$, that

$$\boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_j \overset{(B.7)}{=} \boldsymbol{r}_i^\intercal \hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}\boldsymbol{s}_j \overset{(B.11)}{=} \boldsymbol{r}_i^\intercal \left(\sum_{\ell=1}^{j} \gamma_\ell \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_\ell\right) \overset{(B.9)}{=} 0. \tag{B.12}$$

Thus in combination we have

$$\forall j \in \{1, \ldots, i\}: \quad \boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_j = 0. \tag{B.13}$$

Now for the search direction we have

$$\begin{aligned}\boldsymbol{d}_{i+1} = \boldsymbol{\Sigma}_i \hat{\boldsymbol{K}}\boldsymbol{s}_{i+1} &= \left(\boldsymbol{\Sigma}_0 - \sum_{j=1}^{i} \frac{\boldsymbol{d}_j \boldsymbol{d}_j^\intercal}{\eta_j}\right)\hat{\boldsymbol{K}}\boldsymbol{s}_{i+1} \\ &= \boldsymbol{s}_{i+1} - \sum_{j=1}^{i} \frac{\boldsymbol{d}_j^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i+1}}{\eta_j}\boldsymbol{d}_j \overset{(B.4)}{=} \boldsymbol{s}_{i+1} - \sum_{j=1}^{i} \frac{\boldsymbol{s}_j^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i+1}}{\eta_j}\boldsymbol{d}_j \\ &\overset{(B.13)}{=} \boldsymbol{s}_{i+1}.\end{aligned} \tag{B.14}$$

Further, we have for the solution estimate, that $\boldsymbol{v}_{i+1} = \boldsymbol{v}_i + \boldsymbol{d}_{i+1}\frac{\alpha_{i+1}}{\eta_{i+1}}$. It holds that

$$\begin{aligned}\alpha_{i+1} = \boldsymbol{s}_{i+1}^\intercal \boldsymbol{r}_i &= \big(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i - \frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i}{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i}\boldsymbol{s}_i\big)^\intercal \boldsymbol{r}_i \\ &= (\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \boldsymbol{r}_i - \sum_{j=}^{i} c_j (\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{j-1})^\intercal \boldsymbol{r}_i \overset{(B.9)}{=} (\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \boldsymbol{r}_i\end{aligned}$$

as well as

$$\eta_{i+1} = \boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{\Sigma}_i \hat{\boldsymbol{K}}\boldsymbol{s}_{i+1} = \boldsymbol{d}_{i+1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i+1} \overset{(B.14)}{=} \boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i+1}$$

Combining the above and recalling Algorithm 6, we obtain

$$\boldsymbol{v}_{i+1} = \boldsymbol{v}_i + \boldsymbol{d}_{i+1}\frac{\alpha_{i+1}}{\eta_{i+1}} = \boldsymbol{v}_i + \boldsymbol{d}_{i+1}\frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i)^\intercal \boldsymbol{r}_i}{\boldsymbol{s}_{i+1}^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i+1}} = \boldsymbol{v}_{i+1}^{\mathrm{CG}}.$$

Finally, the residual is computed identically in Algorithm 2 as in Algorithm 6, giving

$$\boldsymbol{r}_i = (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}}\boldsymbol{v}_i = (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}}\boldsymbol{v}_i^{\mathrm{CG}} = \boldsymbol{r}_i^{\mathrm{CG}}.$$

This proves the claims. □

Corollary B.1 (Preconditioned Gradient Actions as CG Actions)
Choosing actions

$$\boldsymbol{s}_i = \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1} \tag{B.15}$$

in Theorem B.3 instead also reproduces the preconditioned conjugate gradient method, i.e. it holds for $i \in \{1, \dots, n\}$ *that*

$$\boldsymbol{d}_i = \boldsymbol{s}_i^{\mathrm{CG}} \tag{B.16}$$
$$\boldsymbol{v}_i = \boldsymbol{v}_i^{\mathrm{CG}} \tag{B.17}$$
$$\boldsymbol{r}_{i-1} = \boldsymbol{r}_{i-1}^{\mathrm{CG}}. \tag{B.18}$$

Proof. It suffices to show that $\boldsymbol{d}_i = \boldsymbol{s}_i^{\mathrm{CG}}$. The rest of the argument is then identical to the proof of Theorem B.3. We prove the claim by induction. For the base case by assumption $\boldsymbol{s}_1 = \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_0 = \boldsymbol{s}_1^{\mathrm{CG}}$. Now for the induction step $i \to i+1$, assume that $\boldsymbol{d}_j = \boldsymbol{s}_j$ for all $j \leq i$, then

$$\begin{aligned}
\boldsymbol{d}_{i+1} &= \boldsymbol{\Sigma}_i \hat{\boldsymbol{K}} \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i \\
&= (\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}})\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i \\
&= \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i - \boldsymbol{D}_i(\boldsymbol{D}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{D}_i)^{-1}\boldsymbol{D}_i^\intercal \hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i && \text{By (B.21).} \\
&\overset{\text{IH}}{=} \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i - \boldsymbol{S}_i^{\mathrm{CG}}((\boldsymbol{S}_i^{\mathrm{CG}})^\intercal \hat{\boldsymbol{K}}\boldsymbol{S}_i^{\mathrm{CG}})^{-1}(\boldsymbol{S}_i^{\mathrm{CG}})^\intercal \hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i
\end{aligned}$$

Now by the same argument as in (B.12) in the proof of Theorem B.3 we have for all $j < i$ that $\boldsymbol{r}_i^\intercal \hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}\boldsymbol{s}_j^{\mathrm{CG}} = 0$. Therefore

$$\begin{aligned}
&= \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i - \boldsymbol{s}_i^{\mathrm{CG}}((\boldsymbol{s}_i^{\mathrm{CG}})^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i^{\mathrm{CG}})^{-1}(\boldsymbol{s}_i^{\mathrm{CG}})^\intercal \hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i \\
&= \boldsymbol{s}_{i+1}^{\mathrm{CG}} && \text{By (B.3).}
\end{aligned}$$

This proves the claim. □

Corollary B.2 (Deflated Conjugate Gradient Method)
Let the first $0 < \ell < n$ *actions* $(\boldsymbol{s}_i)_{i=1}^{\ell}$ *of Algorithm 2 be linearly independent and the remaining ones be given by* $\boldsymbol{s}_i = \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i$*, where* $\hat{\boldsymbol{P}} \approx \hat{\boldsymbol{K}}$ *is a preconditioner. Then Algorithm 2 is equivalent to the preconditioned deflated CG algorithm [163, Alg. 3.6] with deflation subspace* $\operatorname{span}(\boldsymbol{S}_\ell)$.

Proof. By the form of preconditioned deflated CG given in Algorithm 3.6 of Saad et al. [163] and Corollary B.1, it suffices to show that the residual $\boldsymbol{r}_\ell$ satisfies $\boldsymbol{S}_\ell^\intercal \boldsymbol{r}_\ell = \boldsymbol{0}$ and that for all $i > \ell$, it holds that

$$\boldsymbol{s}_i^{\mathrm{defCG}} = \boldsymbol{d}_i = (\boldsymbol{I} - \boldsymbol{C}_{i-1}\hat{\boldsymbol{K}})\boldsymbol{s}_i.$$

Now it holds by Lemma B.2 and (B.21), that

$$\boldsymbol{S}_\ell^\intercal \boldsymbol{r}_\ell = \boldsymbol{S}_\ell^\intercal (\boldsymbol{I} - \hat{\boldsymbol{K}}\boldsymbol{C}_\ell)(\boldsymbol{y} - \boldsymbol{\mu}) = \underbrace{\boldsymbol{S}_\ell^\intercal (\boldsymbol{I} - \hat{\boldsymbol{K}}\boldsymbol{S}_\ell(\boldsymbol{S}_\ell^\intercal \hat{\boldsymbol{K}}\boldsymbol{S}_\ell)^{-1}\boldsymbol{S}_\ell^\intercal)}_{=\boldsymbol{0}}(\boldsymbol{y} - \boldsymbol{\mu}) = \boldsymbol{0}.$$

This proves the first claim. Now, by Saad et al. [163, Alg. 3.6], the search directions $(\boldsymbol{s}_i^{\text{defCG}})_{i=\ell+1}^n$ of preconditioned deflated CG are given by

$$\begin{aligned}
\boldsymbol{s}_i^{\text{defCG}} &= \boldsymbol{s}_i^{\text{CG}} - \boldsymbol{S}_\ell(\boldsymbol{S}_\ell^\intercal \hat{\boldsymbol{K}}\boldsymbol{S}_\ell)^{-1}\boldsymbol{S}_\ell^\intercal \hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i \\
&= (\boldsymbol{I} - \boldsymbol{C}_{\ell+1:(i-1)}\hat{\boldsymbol{K}})\boldsymbol{s}_i - \boldsymbol{S}_\ell(\boldsymbol{S}_\ell^\intercal \hat{\boldsymbol{K}}\boldsymbol{S}_\ell)^{-1}\boldsymbol{S}_\ell^\intercal \hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i \qquad \text{Corollary B.1} \\
&= (\boldsymbol{I} - \boldsymbol{C}_{\ell+1:(i-1)}\hat{\boldsymbol{K}})\boldsymbol{s}_i - \boldsymbol{C}_\ell \hat{\boldsymbol{K}}\boldsymbol{s}_i \\
&= (\boldsymbol{I} - (\boldsymbol{C}_{\ell+1:(i-1)} - \boldsymbol{C}_\ell)\hat{\boldsymbol{K}})\boldsymbol{s}_i \\
&= (\boldsymbol{I} - \boldsymbol{C}_{i-1}\hat{\boldsymbol{K}})\boldsymbol{s}_i \\
&= \boldsymbol{d}_i
\end{aligned}$$

This proves the claim. □

Remark B.1 (Preconditioning and Algorithm 2)
Iterative methods typically have convergence rates depending on the condition number of the system matrix. One successful strategy in practice to accelerate convergence is to use a preconditioner $\hat{\boldsymbol{P}} \approx \hat{\boldsymbol{K}}$ [65]. A preconditioner needs to be cheap to compute and allow efficient matrix-vector multiplication $\boldsymbol{v} \mapsto \hat{\boldsymbol{P}}^{-1}\boldsymbol{v}$. Now, Algorithm 2 implicitly constructs and applies a *deflation-based preconditioner*, which are defined via a deflation subspace to be projected out [164]. In Algorithm 2 this is precisely the already explored space $\operatorname{span}(\boldsymbol{S}_i) = \operatorname{span}(\boldsymbol{D}_i)$ spanned by the actions. Therefore, if we run a mixed strategy, meaning first choosing actions that define a certain subspace and then choose residual actions, we recover the *deflated conjugate gradient method* [163] (see Corollary B.2 for a proof). Alternatively, one can also use byproducts of the iteration of Algorithm 2 to construct a diagonal-plus-low-rank preconditioner of the form $\hat{\boldsymbol{P}} = \sigma^2\boldsymbol{I} + \boldsymbol{U}\boldsymbol{U}^\intercal \approx \hat{\boldsymbol{K}}$ where $\boldsymbol{U} = \boldsymbol{K}\boldsymbol{D}_i \operatorname{diag}(\eta_1, \dots, \eta_i) \in \mathbb{R}^{n \times i}$. Therefore, again if running a mixed strategy, one can first construct a preconditioner and then accelerate the convergence of subsequent CG iterations. In this sense one can double-dip in terms of preconditioning (conjugate) gradient iterations by combining these two techniques *at essentially no overhead*.

B.1.4 Inducing Point Methods

Theorem B.4 (Approximate Posterior Mean of Nyström, SoR, DTC and SVGP)
Let $\boldsymbol{Z} \in \mathbb{R}^{n \times m}$ be a set of distinct inducing inputs such that $\operatorname{rank}(\boldsymbol{K}_{\boldsymbol{XZ}}) = m \leq n$. Then the posterior mean of the Nyström variants subset of regressors (SoR) and deterministic

training conditional (DTC) is identical to the one of SVGP and given by

$$\begin{aligned}\mu(\cdot) &= k(\cdot, \boldsymbol{Z})(\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}} + \sigma^2\boldsymbol{K_{ZZ}})^{-1}\boldsymbol{K_{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}) \\ &= q(\cdot, \boldsymbol{X})\boldsymbol{K_{XZ}}(\boldsymbol{K_{ZX}}(q(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I})\boldsymbol{K_{XZ}})^{-1}\boldsymbol{K_{ZX}}(\boldsymbol{y} - \boldsymbol{\mu})\end{aligned} \tag{B.19}$$

Proof. By eqns. (16b) and (20b) of Quiñonero-Candela and Rasmussen [107] the posterior mean of SoR and DTC is identical and given by

$$\mu(\cdot) = k(\cdot, \boldsymbol{Z})(\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}} + \sigma^2\boldsymbol{K_{ZZ}})^{-1}\boldsymbol{K_{ZX}}(\boldsymbol{y} - \boldsymbol{\mu})$$

Now, by Theorem 5 of Wild, Kanagawa, and Sejdinovic [122] the posterior mean of SVGP for a fixed set of inducing points is equivalent to the Nyström approximation, which takes the form above. Recognizing that $\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}} \in \mathbb{R}^{m\times m}$ is invertible, it holds that

$$\begin{aligned}\mu(\cdot) &= k(\cdot, \boldsymbol{Z})(\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}} + \sigma^2\boldsymbol{K_{ZZ}})^{-1}\boldsymbol{K_{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}) \\ &= k(\cdot, \boldsymbol{Z})\boldsymbol{K_{ZZ}^{-1}}(\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}}\boldsymbol{K_{ZZ}^{-1}} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{K_{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}) \\ &= k(\cdot, \boldsymbol{Z})\boldsymbol{K_{ZZ}^{-1}}\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}}((\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}}\boldsymbol{K_{ZZ}^{-1}} + \sigma^2\boldsymbol{I})\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}})^{-1}\boldsymbol{K_{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}) \\ &= k(\cdot, \boldsymbol{Z})\boldsymbol{K_{ZZ}^{-1}}\boldsymbol{K_{ZX}}\boldsymbol{K_{XZ}}(\boldsymbol{K_{ZX}}(\boldsymbol{K_{XZ}}\boldsymbol{K_{ZZ}^{-1}}\boldsymbol{K_{ZX}} + \sigma^2\boldsymbol{I})\boldsymbol{K_{XZ}})^{-1}\boldsymbol{K_{ZX}}(\boldsymbol{y} - \boldsymbol{\mu}) \\ &= q(\cdot, \boldsymbol{X})\boldsymbol{K_{XZ}}(\boldsymbol{K_{ZX}}(q(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I})\boldsymbol{K_{XZ}})^{-1}\boldsymbol{K_{ZX}}(\boldsymbol{y} - \boldsymbol{\mu})\end{aligned}$$

This proves the claim. □

B.2 Theoretical Results and Proofs

B.2.1 Properties of Algorithm 1

Lemma B.1 (Geometric Properties of Algorithm 2)
Let $i \in \{1, \ldots, n\}$, and assume $\boldsymbol{\Sigma}_0$ is chosen such that $\boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{s}_j = \boldsymbol{s}_j$ for all $j \leq i$ (e.g. $\boldsymbol{\Sigma}_0 = \hat{\boldsymbol{K}}^{-1}$). Then it holds for the quantities computed by Algorithm 2 that

$$\operatorname{span}(\boldsymbol{S}_i) = \operatorname{span}(\boldsymbol{D}_i) \tag{B.20}$$

$$\boldsymbol{C}_i = \boldsymbol{D}_i(\boldsymbol{D}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{D}_i)^{-1}\boldsymbol{D}_i^\intercal = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal \tag{B.21}$$

$$\boldsymbol{C}_i\hat{\boldsymbol{K}} \text{ is the } \hat{\boldsymbol{K}}\text{-orthogonal projection onto } \operatorname{span}(\boldsymbol{D}_i) \tag{B.22}$$

$$\boldsymbol{\Sigma}_i\hat{\boldsymbol{K}} \text{ is the } \hat{\boldsymbol{K}}\text{-orthogonal projection onto } \operatorname{span}(\boldsymbol{D}_i)^{\perp_{\hat{\boldsymbol{K}}}} \tag{B.23}$$

$$\boldsymbol{d}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{d}_j = 0 \qquad \text{for all } j < i \tag{B.24}$$

where $\boldsymbol{S}_i = \begin{pmatrix}\boldsymbol{s}_1 \cdots \boldsymbol{s}_i\end{pmatrix} \in \mathbb{R}^{n\times i}$ and $\boldsymbol{D}_i = \begin{pmatrix}\boldsymbol{d}_1 \cdots \boldsymbol{d}_i\end{pmatrix} \in \mathbb{R}^{n\times i}$.

Proof. We prove the claims by induction. We begin with the base case $i = 1$.

By assumption it holds that $\boldsymbol{S}_1 = \boldsymbol{s}_1 = \boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{s}_1 = \boldsymbol{d}_1 = \boldsymbol{D}_1$. Now by Algorithm 2, we have $\boldsymbol{C}_1 = \frac{1}{\eta_1}\boldsymbol{d}_1\boldsymbol{d}_1^\intercal$, which with the above proves (B.21). By the batched form (B.21) of $\boldsymbol{C}_i$, the statements (B.22) and (B.23) follow immediately. Finally, $\hat{\boldsymbol{K}}$-orthogonality for a single search direction holds trivially.

Now for the induction step $i \to i+1$. Assume that (B.20), (B.21), (B.22), (B.23) and (B.24) hold for iteration i. Then we have that

$$\boldsymbol{d}_{i+1} = \boldsymbol{\Sigma}_i\hat{\boldsymbol{K}}\boldsymbol{s}_{i+1} = \boldsymbol{s}_{i+1} - \boldsymbol{C}_i\hat{\boldsymbol{K}}\boldsymbol{s}_{i+1} \overset{\text{(B.21)}}{=} \boldsymbol{s}_{i+1} - \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{s}_{i+1} \in \operatorname{span}(\boldsymbol{S}_{i+1})$$

By the induction hypothesis the above also implies $\operatorname{span}(\boldsymbol{S}_{i+1}) = \operatorname{span}(\boldsymbol{D}_{i+1})$. This proves (B.20). Next, we have by the induction hypotheses (B.21) and (B.24) that

$$\begin{aligned}
\boldsymbol{C}_{i+1} &= \boldsymbol{C}_i + \frac{1}{\eta}\boldsymbol{d}_{i+1}\boldsymbol{d}_{i+1}^\intercal \\
&= \boldsymbol{D}_i(\boldsymbol{D}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{D}_i)^{-1}\boldsymbol{D}_i^\intercal + \frac{1}{\eta_{i+1}}\boldsymbol{d}_{i+1}\boldsymbol{d}_{i+1}^\intercal \\
&= \sum_{k=1}^{i+1}\frac{1}{\eta_k}\boldsymbol{d}_k\boldsymbol{d}_k^\intercal \\
&= \boldsymbol{D}_{i+1}(\boldsymbol{D}_{i+1}^\intercal\hat{\boldsymbol{K}}\boldsymbol{D}_{i+1})^{-1}\boldsymbol{D}_{i+1}^\intercal
\end{aligned}$$

This proves the first equality of (B.21). For the second, first recognize that an orthogonal projection onto a linear subspace $\operatorname{span}(\boldsymbol{A})$ with respect to the $\boldsymbol{B}$-inner product is given by $\boldsymbol{P}_{\boldsymbol{A}} = \boldsymbol{A}(\boldsymbol{A}^\intercal\boldsymbol{B}\boldsymbol{A})^{-1}\boldsymbol{A}^\intercal\boldsymbol{B}$. The projection onto its $\boldsymbol{B}$-orthogonal subspace is given by $\boldsymbol{P}_{\boldsymbol{A}^{\perp_{\boldsymbol{B}}}} = \boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{A}}$. Therefore (B.22) and (B.23) follow directly from the above argument. Now since projection onto a subspace is unique and independent of the choice of basis, we have by $\operatorname{span}(\boldsymbol{D}_{i+1}) = \operatorname{span}(\boldsymbol{S}_{i+1})$ that

$$\boldsymbol{C}_i\hat{\boldsymbol{K}} = \boldsymbol{P}_{\boldsymbol{D}_{i+1}} = \boldsymbol{P}_{\boldsymbol{S}_{i+1}} = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}$$

Now since $\hat{\boldsymbol{K}}$ is non-singular, the second equality of (B.21) follows. Finally, we will prove $\hat{\boldsymbol{K}}$-orthogonality of the search directions. Let $j < i+1$, then it holds that

$$\boldsymbol{d}_{i+1}^\intercal\hat{\boldsymbol{K}}\boldsymbol{d}_j = (\underbrace{\boldsymbol{\Sigma}_i\hat{\boldsymbol{K}}\boldsymbol{s}_{i+1}}_{\in\operatorname{span}(\boldsymbol{S}_i)^{\perp_{\hat{\boldsymbol{K}}}}})^\intercal\hat{\boldsymbol{K}}\underbrace{\boldsymbol{d}_j}_{\in\operatorname{span}(\boldsymbol{S}_i)} = 0$$

by (B.20) and (B.23). This completes the proof. □

Corollary B.3
Let $i \in \{1, \dots, n\}$. It holds for $\boldsymbol{C}_i\hat{\boldsymbol{K}}$, the $\hat{\boldsymbol{K}}$-orthogonal projection onto $\boldsymbol{S}_i$, that

$$(\boldsymbol{C}_i\hat{\boldsymbol{K}})^2 = \boldsymbol{C}_i\hat{\boldsymbol{K}} \tag{B.25}$$
$$\boldsymbol{C}_i\hat{\boldsymbol{K}}\boldsymbol{C}_i = \boldsymbol{C}_i \tag{B.26}$$

Further for $\boldsymbol{H}_i = \boldsymbol{\Sigma}_i\hat{\boldsymbol{K}} = \boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}$ the $\hat{\boldsymbol{K}}$-orthogonal projection onto $\boldsymbol{S}_i^{\perp_{\hat{K}}}$, we have

$$\boldsymbol{H}_i^2 = \boldsymbol{H}_i \tag{B.27}$$
$$\boldsymbol{H}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{H}_i = \boldsymbol{H}_i^\intercal\hat{\boldsymbol{K}} = \hat{\boldsymbol{K}}\boldsymbol{H}_i \tag{B.28}$$

Proof. By Lemma B.1, it holds that $\boldsymbol{C}_i = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal$. Therefore

$$\boldsymbol{C}_i\hat{\boldsymbol{K}}\boldsymbol{C}_i = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal = \boldsymbol{C}_i.$$

This proves (B.26) and (B.25). Define $\boldsymbol{H}_i = \boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}$, then

$$\boldsymbol{H}_i\boldsymbol{H}_i = (\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}})(\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}) = \boldsymbol{I} - 2\boldsymbol{C}_i\hat{\boldsymbol{K}} + (\boldsymbol{C}_i\hat{\boldsymbol{K}})^2 = \boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}} = \boldsymbol{H}_i$$

as well as

$$\begin{aligned}\boldsymbol{H}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{H}_i &= (\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}})^\intercal\hat{\boldsymbol{K}}(\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}) = (\hat{\boldsymbol{K}} - \hat{\boldsymbol{K}}\boldsymbol{C}_i\hat{\boldsymbol{K}})(\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}) \\ &= \hat{\boldsymbol{K}} - 2\hat{\boldsymbol{K}}\boldsymbol{C}_i\hat{\boldsymbol{K}} + \hat{\boldsymbol{K}}(\boldsymbol{C}_i\hat{\boldsymbol{K}})^2 \\ &= \hat{\boldsymbol{K}} - \hat{\boldsymbol{K}}\boldsymbol{C}_i\hat{\boldsymbol{K}} = \boldsymbol{H}_i^\intercal\hat{\boldsymbol{K}} = \hat{\boldsymbol{K}}\boldsymbol{H}_i.\end{aligned}$$

□

Lemma B.2
Let $\boldsymbol{\Sigma}_0 = \hat{\boldsymbol{K}}^{-1}$, then it holds that

$$\boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{\mu}) = \boldsymbol{v}_i, \tag{B.29}$$
$$\boldsymbol{\Sigma}_i(\boldsymbol{y} - \boldsymbol{\mu}) = \boldsymbol{v}_* - \boldsymbol{v}_i. \tag{B.30}$$

Proof. We prove the statement by induction. By assumption $\boldsymbol{C}_0(\boldsymbol{y} - \boldsymbol{\mu}) = \boldsymbol{v}_0$. Now assume (B.29) holds. Then for $i \to i+1$, we have

$$\boldsymbol{C}_{i+1}(\boldsymbol{y} - \boldsymbol{\mu}) = (\boldsymbol{C}_i + \frac{1}{\eta_{i+1}}\boldsymbol{d}_{i+1}\boldsymbol{d}_{i+1}^\intercal)(\boldsymbol{y} - \boldsymbol{\mu}) \overset{\text{IH}}{=} \boldsymbol{v}_i + \frac{\boldsymbol{d}_{i+1}^\intercal(\boldsymbol{y} - \boldsymbol{\mu})}{\eta_{i+1}}\boldsymbol{d}_{i+1}$$

Now by the update to the representer weights in Algorithm 2 it suffices to show that $\alpha_{i+1} = \boldsymbol{d}_{i+1}^\intercal(\boldsymbol{y} - \boldsymbol{\mu})$. We have

$$\begin{aligned}\boldsymbol{d}_{i+1}^\intercal(\boldsymbol{y} - \boldsymbol{\mu}) &= (\boldsymbol{\Sigma}_i\hat{\boldsymbol{K}}\boldsymbol{s}_{i+1})^\intercal(\boldsymbol{y} - \boldsymbol{\mu}) = \boldsymbol{s}_{i+1}^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_i(\boldsymbol{y} - \boldsymbol{\mu}) \\ &= \boldsymbol{s}_{i+1}^\intercal\hat{\boldsymbol{K}}(\hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i)(\boldsymbol{y} - \boldsymbol{\mu}) \overset{\text{IH}}{=} \boldsymbol{s}_{i+1}^\intercal((\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}}\boldsymbol{v}_i) = \boldsymbol{s}_{i+1}^\intercal\boldsymbol{r}_i = \alpha_i.\end{aligned}$$

□

Lemma B.3
Let $\boldsymbol{\Sigma}_0 = \hat{\boldsymbol{K}}^{-1}$, $\boldsymbol{C}_0 = \mathbf{0}$ and consequently $\boldsymbol{v}_0 = \mathbf{0}$, then it holds for the residual at iteration $i \in \{1, \dots, n\}$ that

$$\begin{aligned} \boldsymbol{r}_{i-1} &= \hat{\boldsymbol{K}}(\boldsymbol{v}_* - \boldsymbol{v}_{i-1}) && \text{(B.31)} \\ &= \hat{\boldsymbol{K}}\boldsymbol{\Sigma}_{i-1}\hat{\boldsymbol{K}}\boldsymbol{v}_* && \text{(B.32)} \\ &= (\hat{\boldsymbol{K}} - \boldsymbol{Q}_{i-1})\boldsymbol{v}_*. && \text{(B.33)} \end{aligned}$$

Proof. It holds by definition, that

$$\boldsymbol{r}_{i-1} = (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}}\boldsymbol{v}_{i-1} = \hat{\boldsymbol{K}}\boldsymbol{v}_* - \hat{\boldsymbol{K}}\boldsymbol{v}_{i-1} = \hat{\boldsymbol{K}}(\boldsymbol{v}_* - \boldsymbol{v}_{i-1}).$$

Further we have by (B.30), that

$$= \hat{\boldsymbol{K}}\boldsymbol{\Sigma}_{i-1}(\boldsymbol{y} - \boldsymbol{\mu}) = \hat{\boldsymbol{K}}\boldsymbol{\Sigma}_{i-1}\hat{\boldsymbol{K}}\boldsymbol{v}_*,$$

and finally, by the definition of the kernel matrix approximation in Algorithm 2, we obtain

$$= \hat{\boldsymbol{K}}(\hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_{i-1})\hat{\boldsymbol{K}}\boldsymbol{v}_* = (\hat{\boldsymbol{K}} - \boldsymbol{Q}_{i-1})\boldsymbol{v}_*.$$

□

Proposition B.1 (Batch of Observations)
Let $\boldsymbol{\Sigma}_0$ such that $\boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{s}_j = \boldsymbol{s}_j$ for all $j \in \{1, \dots, i\}$. Then after i iterations the posterior over the representer weights in (3.4) is equivalent to the one computed for a batch of observations, i.e.

$$\begin{aligned} \boldsymbol{v}_i &= \boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal(\boldsymbol{y} - \boldsymbol{\mu}) \\ \boldsymbol{\Sigma}_i &= \boldsymbol{\Sigma}_0 - \boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0 \end{aligned}$$

Proof. This can be seen as a direct consequence of recursively applying Bayes' theorem

$$p(\boldsymbol{v}_* \mid \{\alpha_i\}_{i=1}^m, \{\boldsymbol{s}_i\}_{i=1}^m) = \frac{p(\alpha_m \mid \boldsymbol{s}_m, \boldsymbol{v}_*)p(\boldsymbol{v}_* \mid \{\alpha_i\}_{i=1}^{m-1}, \{\boldsymbol{s}_i\}_{i=1}^{m-1})}{\int p(\alpha_m \mid \boldsymbol{s}_m, \boldsymbol{v}_*)p(\boldsymbol{v}_* \mid \{\alpha_i\}_{i=1}^{m-1}, \{\boldsymbol{s}_i\}_{i=1}^{m-1})d\boldsymbol{v}_*}.$$

However, here we also give a geometric proof based on the projection property of the precision matrix approximation $\boldsymbol{C}_i$. By using (B.21) and the assumption on $\boldsymbol{\Sigma}_0$ we have that

$$\begin{aligned} \boldsymbol{C}_i = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal &= \boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal \\ &= \boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0 \end{aligned}$$

This proves that

$$\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_0 - \boldsymbol{C}_i = \boldsymbol{\Sigma}_0 - \boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{\Sigma}_0$$

Now by (B.29) it holds that $\boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{\mu}) = \boldsymbol{v}_i$. This proves the claim. □

Proposition B.2 (Posterior Contraction)
Let $\boldsymbol{S}_i \in \mathbb{R}^{n\times i}$ be the actions chosen by Algorithm 2, then its posterior contracts as

$$\operatorname{tr}(\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_0^{-1}) = \operatorname{tr}(\boldsymbol{\Sigma}_i\hat{\boldsymbol{K}}) = n - \operatorname{rank}(\boldsymbol{S}_i).$$

Proof. Since $\boldsymbol{\Sigma}_0 = \hat{\boldsymbol{K}}^{-1}$, we have by (B.21), that

$$\begin{aligned}
\operatorname{tr}(\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_0^{-1}) &= \operatorname{tr}((\boldsymbol{\Sigma}_0 - \boldsymbol{C}_i)\hat{\boldsymbol{K}}) \\
&= \operatorname{tr}(\boldsymbol{I}_n - \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^\dagger\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}) \\
&= \operatorname{tr}(\boldsymbol{I}_n) - \operatorname{tr}(\underbrace{\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^\dagger}_{\in\mathbb{R}^{i\times i}}) \\
&= n - \operatorname{rank}(\boldsymbol{S}_i)
\end{aligned}$$

Now, if the actions $\boldsymbol{S}_i$ are chosen linearly independent, then $\operatorname{rank}(\boldsymbol{S}_i) = i$. □

Theorem B.5 (Online GP Approximation with Algorithm 1)
Let $n, n' \in \mathbb{N}$ and consider training data sets $\boldsymbol{X} \in \mathbb{R}^{n\times d}, \boldsymbol{y} \in \mathbb{R}^n$ and $\boldsymbol{X}' \in \mathbb{R}^{n'\times d}, \boldsymbol{y}' \in \mathbb{R}^{n'}$. Consider two sequences of actions $(\boldsymbol{s}_i)_{i=1}^n \in \mathbb{R}^n$ and $(\tilde{\boldsymbol{s}}_i)_{i=1}^{n+n'} \in \mathbb{R}^{n+n'}$ such that for all $i \in \{1,\dots,n\}$, it holds that

$$\tilde{\boldsymbol{s}}_i = \begin{pmatrix} \boldsymbol{s}_i \\ \boldsymbol{0} \end{pmatrix} \tag{B.34}$$

Then the posterior returned by Algorithm 2 for the dataset $(\boldsymbol{X}, \boldsymbol{y})$ using actions $\boldsymbol{s}_i$ is identical to the posterior returned by Algorithm 2 for the extended dataset using actions $\tilde{\boldsymbol{s}}_i$, i.e. it holds for any $i \in \{1,\dots,n\}$, that

$$\textsc{IterGP}(\mu, k, \boldsymbol{X}, \boldsymbol{y}, (\boldsymbol{s}_i)_i) = (\mu_i, k_i) = (\tilde{\mu}_i, \tilde{k}_i) = \textsc{IterGP}\left(\mu, k, \begin{pmatrix}\boldsymbol{X}\\ \boldsymbol{X}'\end{pmatrix}, \begin{pmatrix}\boldsymbol{y}\\ \boldsymbol{y}'\end{pmatrix}, (\tilde{\boldsymbol{s}}_i)_i\right).$$

Proof. Define $\tilde{\boldsymbol{X}} = \begin{pmatrix}\boldsymbol{X}\\ \boldsymbol{X}'\end{pmatrix}$ and $\tilde{\boldsymbol{y}} = \begin{pmatrix}\boldsymbol{y}\\ \boldsymbol{y}'\end{pmatrix}$. We begin by showing that the search directions of both methods satisfy

$$\boldsymbol{d}_i' = \begin{pmatrix}\boldsymbol{d}_i\\ \boldsymbol{0}\end{pmatrix}. \tag{B.35}$$

We proceed by induction. For $i = 0$ it holds by definition of Algorithm 2 and (B.34) that

$$\tilde{\boldsymbol{d}}_0 = \tilde{\boldsymbol{s}}_0 = \begin{pmatrix}\boldsymbol{s}_0\\ \boldsymbol{0}\end{pmatrix} = \begin{pmatrix}\boldsymbol{d}_0\\ \boldsymbol{0}\end{pmatrix}. \tag{B.36}$$

For the induction step $i \to i+1$, assume that (B.35) holds for $j \in \{1,\dots,i\}$. Then, we have

$$\tilde{\boldsymbol{d}}_{i+1} = \tilde{\boldsymbol{\Sigma}}_{i-1}(k(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}}) + \sigma^2\boldsymbol{I}_{n+n'})\tilde{\boldsymbol{s}}_{i+1}$$

$$
\begin{aligned}
&= (\boldsymbol{I}_{n+n'} - \tilde{\boldsymbol{C}}_i(k(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}}) + \sigma^2 \boldsymbol{I}_{n+n'}))\tilde{\boldsymbol{s}}_{i+1} \\
&= \tilde{\boldsymbol{s}}_{i+1} - \sum_{j=1}^{i} \frac{1}{\tilde{\eta}_j} \tilde{\boldsymbol{d}}_j (\tilde{\boldsymbol{d}}_j)^\intercal (k(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}}) + \sigma^2 \boldsymbol{I}_{n+n'}) \tilde{\boldsymbol{s}}_{i+1} \\
&\overset{\text{IH}}{=} \begin{pmatrix} \boldsymbol{s}_{i+1} \\ \boldsymbol{0} \end{pmatrix} - \sum_{j=1}^{i} \frac{1}{\tilde{\eta}_j} \begin{pmatrix} \boldsymbol{d}_j \\ \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_j^\intercal & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} k(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{I}_n & k(\boldsymbol{X}, \boldsymbol{X}') \\ k(\boldsymbol{X}', \boldsymbol{X}) & k(\boldsymbol{X}', \boldsymbol{X}') + \boldsymbol{I}_{n'} \end{pmatrix} \begin{pmatrix} \boldsymbol{s}_{i+1} \\ \boldsymbol{0} \end{pmatrix} \\
&= \begin{pmatrix} \boldsymbol{s}_{i+1} - \sum_{j=1}^{i} \frac{1}{\eta_j} \boldsymbol{d}_j (\boldsymbol{d}_j)^\intercal \hat{\boldsymbol{K}} \boldsymbol{s}_{i+1} \\ \boldsymbol{0} \end{pmatrix} \\
&= \begin{pmatrix} \boldsymbol{d}_{i+1} \\ \boldsymbol{0} \end{pmatrix}
\end{aligned}
$$

where we used that $\tilde{\eta}_j = \tilde{\boldsymbol{s}}_j^\intercal (k(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}}) + \sigma^2 \boldsymbol{I}_{n+n'}) \tilde{\boldsymbol{d}}_j = \boldsymbol{s}_j^\intercal \hat{\boldsymbol{K}} \boldsymbol{d}_j = \eta_j$. This proves (B.35). Now recognize that

$$
\begin{aligned}
\tilde{\alpha}_j = \tilde{\boldsymbol{s}}_j^\intercal \tilde{\boldsymbol{r}}_j = \tilde{\boldsymbol{s}}_j^\intercal (\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{\mu}} - \tilde{\boldsymbol{K}} \tilde{\boldsymbol{C}}_i (\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{\mu}})) = \tilde{\boldsymbol{s}}_j^\intercal (\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{\mu}} - (\tilde{\boldsymbol{K}} + \sigma^2 \boldsymbol{I}) \sum_{\ell=1}^{j} \frac{1}{\tilde{\eta}_\ell} \tilde{\boldsymbol{d}}_\ell \tilde{\boldsymbol{d}}_\ell^\intercal (\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{\mu}})) \\
= \boldsymbol{s}_j^\intercal (\boldsymbol{y} - \boldsymbol{\mu}) - \sum_{\ell=1}^{j} \frac{1}{\eta_\ell} \boldsymbol{s}_j^\intercal \hat{\boldsymbol{K}} \boldsymbol{d}_\ell \boldsymbol{d}_\ell^\intercal (\boldsymbol{y} - \boldsymbol{\mu}) = \boldsymbol{s}_j^\intercal (\boldsymbol{y} - \boldsymbol{\mu} - \hat{\boldsymbol{K}} \boldsymbol{C}_j (\boldsymbol{y} - \boldsymbol{\mu})) = \boldsymbol{s}_j^\intercal \boldsymbol{r}_j = \alpha_j
\end{aligned}
$$

Therefore, we have that

$$
\tilde{\mu}_i(\cdot) = \mu(\cdot) + k(\cdot, \tilde{\boldsymbol{X}}) \tilde{\boldsymbol{v}}_i = \mu(\cdot) + k(\cdot, \tilde{\boldsymbol{X}}) \sum_{j=1}^{i} \frac{\tilde{\alpha}_j}{\tilde{\eta}_j} \tilde{\boldsymbol{d}}_j = \mu(\cdot) + k(\cdot, \boldsymbol{X}) \boldsymbol{v}_i = \mu_i(\cdot)
$$

as well as

$$
\begin{aligned}
\tilde{k}_i(\cdot, \cdot) &= k(\cdot, \cdot) - k(\cdot, \tilde{\boldsymbol{X}}) \tilde{\boldsymbol{C}}_i k(\tilde{\boldsymbol{X}}, \cdot) = k(\cdot, \cdot) - k(\cdot, \tilde{\boldsymbol{X}}) \sum_{j=1}^{i} \frac{1}{\tilde{\eta}_j} \tilde{\boldsymbol{d}}_j (\tilde{\boldsymbol{d}}_j)^\intercal k(\tilde{\boldsymbol{X}}, \cdot) \\
&= k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \sum_{j=1}^{i} \frac{1}{\eta_j} \boldsymbol{d}_j \boldsymbol{d}_j^\intercal k(\boldsymbol{X}, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \boldsymbol{C}_i k(\boldsymbol{X}, \cdot) = k_i(\cdot, \cdot).
\end{aligned}
$$

□

Remark B.2 (Streaming Gaussian Processes)
Theorem B.5 shows that any variant of IterGP can be used in the online setting where data arrives sequentially *while* the algorithm is running. If we assume data points arrive one at a time, choose unit vector actions (IterGP-Chol) and perform one iteration of Algorithm 2 after each data point, then Algorithm 2 simply computes the mathematical GP posterior.

B.2.2 Approximation of Representer Weights

Proposition 3.2 (Relative Error Bound for the Representer Weights)
For any choice of actions a relative error bound $\rho(i)$, s.t. $\|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\hat{\boldsymbol{K}}} \leq \rho(i)\|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}$ is given by

$$\rho(i) = (\bar{\boldsymbol{v}}_*^\intercal \underbrace{(\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}})}_{\text{projection onto } \operatorname{span}(\boldsymbol{S}_i)^{\perp_{\hat{\boldsymbol{K}}}}} \bar{\boldsymbol{v}}_*)^{\frac{1}{2}} \leq \lambda_{\max}(\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}) \leq 1 \tag{3.8}$$

where $\bar{\boldsymbol{v}}_* = \boldsymbol{v}_*/\|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}$. If the actions $\{\boldsymbol{s}_i\}_{i=1}^n$ are linearly independent, then $\rho(i) \leq \delta_{n=i}$.

Proof. Define $\boldsymbol{H}_i = \boldsymbol{\Sigma}_i\hat{\boldsymbol{K}} = \boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}$. We have by Lemma B.2, that

$$\|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\hat{\boldsymbol{K}}}^2 = \|\boldsymbol{H}_i\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}^2 = (\boldsymbol{H}_i\boldsymbol{v}_*)^\intercal\hat{\boldsymbol{K}}\boldsymbol{H}_i\boldsymbol{v}_* \overset{(B.28)}{=} \boldsymbol{v}_*{}^\intercal\boldsymbol{H}_i\boldsymbol{v}_* = \bar{\boldsymbol{v}}_*^\intercal\boldsymbol{H}_i\bar{\boldsymbol{v}}_*\|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}^2$$

This proves the first equality of Proposition 3.2. Further it holds that

$$\begin{aligned}\|\boldsymbol{H}_i\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}} &= \|\hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{H}_i\boldsymbol{v}_*\|_2 = \|(\boldsymbol{I} - \hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{C}_i\hat{\boldsymbol{K}}^{\frac{1}{2}})\hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{v}_*\|_2 \leq \|\boldsymbol{I} - \hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{C}_i\hat{\boldsymbol{K}}^{\frac{1}{2}}\|_2\|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}} \\ &= \lambda_{\max}(\boldsymbol{I} - \hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{C}_i\hat{\boldsymbol{K}}^{\frac{1}{2}})\|\boldsymbol{v}_*\|_{\hat{\boldsymbol{K}}}.\end{aligned}$$

Now by Weyl's inequality and the fact that $\hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{C}_i\hat{\boldsymbol{K}}^{\frac{1}{2}}$ is positive semi-definite, it holds that

$$\lambda_{\max}(\boldsymbol{H}_i) = \lambda_{\max}(\boldsymbol{I} - \hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{C}_i\hat{\boldsymbol{K}}^{\frac{1}{2}}) \leq \lambda_{\max}(\boldsymbol{I}) - \lambda_{\min}(\hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{C}_i\hat{\boldsymbol{K}}^{\frac{1}{2}}) \leq 1.$$

Now, recall that similar matrices $\boldsymbol{A}$ and $\boldsymbol{B} = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}$ have the same eigenvalues. Therefore

$$\boldsymbol{I} - \hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{C}_i\hat{\boldsymbol{K}}^{\frac{1}{2}} = \hat{\boldsymbol{K}}^{\frac{1}{2}}(\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}})\hat{\boldsymbol{K}}^{-\frac{1}{2}}$$

and $\boldsymbol{I} - \boldsymbol{C}_i\hat{\boldsymbol{K}}$ have the same eigenvalues. Finally, since by (B.23) $\boldsymbol{H}_i$ is a projection onto $\operatorname{span}(\boldsymbol{S}_i)^{\perp_{\hat{\boldsymbol{K}}}}$, it has full rank at iteration n if the actions are linearly independent and therefore $\lambda_{\max}(\boldsymbol{H}_n) = 1$. This proves the claim. □

B.2.3 Convergence Analysis of the Posterior Mean Approximation

Theorem 3.1 (Convergence in RKHS Norm of the Posterior Mean Approximation)
Let $\mathcal{H}_k$ be the RKHS associated with kernel $k(\cdot,\cdot)$, $\sigma^2 > 0$ and let $\mu_* - \mu \in \mathcal{H}_k$ be the unique solution to the regularized empirical risk minimization problem

$$\arg\min_{f\in\mathcal{H}_k} \frac{1}{n}\Big(\sum_{j=1}^n (f(\boldsymbol{x}_j) - y_j + \mu(\boldsymbol{x}_j))^2 + \sigma^2\|f\|_{\mathcal{H}_k}^2\Big) \tag{3.10}$$

which is equivalent to the mathematical posterior mean up to shift by the prior μ [e.g. 47, Sec. 6.2]. Then for $i \in \{0,\dots,n\}$ the posterior mean $\mu_i(\cdot)$ computed by Algorithm 2 satisfies

$$\boxed{\|\mu_* - \mu_i\|_{\mathcal{H}_k} \leq \rho(i)c(\sigma^2)\|\mu_* - \mu_0\|_{\mathcal{H}_k}} \tag{3.11}$$

where $\mu_0 = \mu$ is the prior mean and the constant $c(\sigma^2) = \sqrt{1 + \frac{\sigma^2}{\lambda_{\min}(\boldsymbol{K})}} \to 1$ as $\sigma^2 \to 0$.

Proof. Let $\rho(i)$ such that $\|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\hat{\boldsymbol{K}}} \leq \rho(i)\|\boldsymbol{v}_* - \boldsymbol{v}_0\|_{\hat{\boldsymbol{K}}}$, where $\boldsymbol{v}_0 = \mathbf{0}$. Then, we have for $i \in \{0, \ldots, n\}$, that

$$\begin{aligned}
\|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\boldsymbol{K}}^2 &\leq \|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\hat{\boldsymbol{K}}}^2 \\
&\leq \rho(i)^2 \|\boldsymbol{v}_* - \boldsymbol{v}_0\|_{\hat{\boldsymbol{K}}}^2 \\
&= \rho(i)^2 \big(\|\boldsymbol{v}_* - \boldsymbol{v}_0\|_{\boldsymbol{K}}^2 + \sigma^2 \frac{1}{\lambda_{\min}(\boldsymbol{K})} \underbrace{\lambda_{\min}(\boldsymbol{K}) \|\boldsymbol{v}_* - \boldsymbol{v}_0\|_2^2}_{\leq \|\boldsymbol{v}_* - \boldsymbol{v}_0\|_{\boldsymbol{K}}^2} \big) \\
&\leq \rho(i)^2 \left(1 + \frac{\sigma^2}{\lambda_{\min}(\boldsymbol{K})} \right) \|\boldsymbol{v}_* - \boldsymbol{v}_0\|_{\boldsymbol{K}}^2
\end{aligned}$$

Now by assumption

$$\mu_i(\cdot) = \mu(\cdot) + \sum_{j=1}^{n} (\boldsymbol{v}_i)_j k(\cdot, \boldsymbol{x}_j) = \mu(\cdot) + k(\cdot, \boldsymbol{X}) \boldsymbol{C}_i \boldsymbol{y}.$$

By the reproducing property we obtain for $\Delta = \boldsymbol{v}_* - \boldsymbol{v}_i$ that

$$\begin{aligned}
\|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\boldsymbol{K}}^2 &= \Delta^\intercal \boldsymbol{K} \Delta \\
&= \sum_{\ell=1}^{n} \sum_{j=1}^{n} \Delta_\ell \Delta_j k(\boldsymbol{x}_\ell, \boldsymbol{x}_j) \\
&= \sum_{\ell=1}^{n} \sum_{j=1}^{n} \Delta_\ell \Delta_j \langle k(\cdot, \boldsymbol{x}_\ell), k(\cdot, \boldsymbol{x}_j) \rangle_{\mathcal{H}_k} && k \text{ is the reproducing kernel of } \mathcal{H}_k \\
&= \langle \sum_{\ell=1}^{n} \Delta_\ell k(\cdot, \boldsymbol{x}_\ell), \sum_{j=1}^{n} \Delta_j k(\cdot, \boldsymbol{x}_j) \rangle_{\mathcal{H}_k} \\
&= \| \sum_{\ell=1}^{n} \Delta_\ell k(\cdot, \boldsymbol{x}_\ell) \|_{\mathcal{H}_k}^2 \\
&= \| \sum_{\ell=1}^{n} (\boldsymbol{v}_*)_\ell k(\cdot, \boldsymbol{x}_\ell) - \sum_{\ell=1}^{n} (\boldsymbol{v}_i)_\ell k(\cdot, \boldsymbol{x}_\ell) \|_{\mathcal{H}_k}^2 \\
&= \|\mu_* - \mu_i\|_{\mathcal{H}_k}^2 && \text{See Theorem 3.4 in Kanagawa et al. [44]}
\end{aligned}$$

Combining the above and setting $c(\sigma^2) = 1 + \frac{\sigma^2}{\lambda_{\min}(\boldsymbol{K})}$ we obtain

$$\|\mu_* - \mu_i\|_{\mathcal{H}_k} = \|\boldsymbol{v}_* - \boldsymbol{v}_i\|_{\boldsymbol{K}} \leq \rho(i) c(\sigma^2) \|\boldsymbol{v}_* - \boldsymbol{v}_0\|_{\boldsymbol{K}} = \rho(i) c(\sigma^2) \|\mu_* - \mu_0\|_{\mathcal{H}_k}.$$

□

B.2.4 Combined Uncertainty as Worst Case Error

Theorem 3.2 (Combined and Computational Uncertainty as Worst Case Errors)
Let $\sigma^2 \geq 0$ and let $k_i(\cdot,\cdot) = k_*(\cdot,\cdot) + k_i^{\text{comp}}(\cdot,\cdot)$ be the combined uncertainty computed by Algorithm 2. Then, for any $\boldsymbol{x} \in \mathcal{X}$ (assuming $\boldsymbol{x} \notin \boldsymbol{X}$ if $\sigma^2 > 0$) we have

$$\sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} \overbrace{\underbrace{g(\boldsymbol{x}) - \mu_*^g(\boldsymbol{x})}_{\text{error of math. post. mean}} + \underbrace{\mu_*^g(\boldsymbol{x}) - \mu_i^g(\boldsymbol{x})}_{\text{computational error}}}^{\text{error of approximate posterior mean}} = \sqrt{k_i(\boldsymbol{x},\boldsymbol{x}) + \sigma^2}, \quad \textit{and} \tag{3.12}$$

$$\sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} \underbrace{\mu_*^g(\boldsymbol{x}) - \mu_i^g(\boldsymbol{x})}_{\text{computational error}} = \sqrt{k_i^{\text{comp}}(\boldsymbol{x},\boldsymbol{x})} \tag{3.13}$$

where $\mu_*^g(\cdot) = k(\cdot,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}g(\boldsymbol{X})$ is the mathematical and $\mu_i^g(\cdot) = k(\cdot,\boldsymbol{X})\boldsymbol{C}_i g(\boldsymbol{X})$ IterGP's posterior mean for the latent function $g \in \mathcal{H}_{k^\sigma}$. If $\sigma^2 = 0$, then the above also holds for $\boldsymbol{x} \in \boldsymbol{X}$.

Proof. Let $\boldsymbol{x}_0 = \boldsymbol{x}$, $c_0 = 1$ and $c_j = -(\boldsymbol{C}_i k^\sigma(\boldsymbol{X},\boldsymbol{x}))_j$ for $j = 1,\dots n$, where $k^\sigma(\cdot,\cdot) \coloneqq k(\cdot,\cdot) + \sigma^2\delta(\cdot,\cdot)$. Then by Lemma 3.9 of Kanagawa et al. [44], it holds that

$$\begin{aligned}
\left(\sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} (g(\boldsymbol{x}) - \mu_i^g(\boldsymbol{x}))\right)^2 &= \left(\sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} \sum_{j=0}^{n} c_j g(\boldsymbol{x}_j)\right)^2 \\
&= \|k^\sigma(\cdot,\boldsymbol{x}_0) - \sum_{j=1}^{n} k(\boldsymbol{x},\boldsymbol{x}_j)\boldsymbol{C}_i k^\sigma(\cdot,\boldsymbol{x}_j)\|_{\mathcal{H}_{k^\sigma}}^2 \\
&= \|k^\sigma(\cdot,\boldsymbol{x}) - k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i k^\sigma(\boldsymbol{X},\cdot)\|_{\mathcal{H}_{k^\sigma}}^2 \\
&= \langle k^\sigma(\cdot,\boldsymbol{x}), k^\sigma(\cdot,\boldsymbol{x})\rangle_{\mathcal{H}_{k^\sigma}} - 2\langle k^\sigma(\cdot,\boldsymbol{x}), k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i k^\sigma(\boldsymbol{X},\cdot)\rangle_{\mathcal{H}_{k^\sigma}} \\
&\quad + \langle k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i k^\sigma(\boldsymbol{X},\cdot), k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i k^\sigma(\boldsymbol{X},\cdot)\rangle_{\mathcal{H}_{k^\sigma}}
\end{aligned}$$

Now by the reproducing property, it follows that

$$= k^\sigma(\boldsymbol{x},\boldsymbol{x}) - 2k^\sigma(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i k^\sigma(\boldsymbol{X},\boldsymbol{x}) + k^\sigma(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i k^\sigma(\boldsymbol{X},\boldsymbol{X})\boldsymbol{C}_i k^\sigma(\boldsymbol{X},\boldsymbol{x})$$

If $\sigma^2 > 0$ and $\boldsymbol{x} \neq \boldsymbol{x}_j$ or if $\sigma^2 = 0$, it holds that $k^\sigma(\boldsymbol{x},\boldsymbol{X}) = k(\boldsymbol{x},\boldsymbol{X})$. Further by definition $k^\sigma(\boldsymbol{X},\boldsymbol{X}) = \hat{\boldsymbol{K}}$ and finally by (B.26), it holds that $\boldsymbol{C}_i\hat{\boldsymbol{K}}\boldsymbol{C}_i = \boldsymbol{C}_i$. Therefore we have

$$\begin{aligned}
&= k(\boldsymbol{x},\boldsymbol{x}) + \sigma^2 - 2k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X},\boldsymbol{x}) + k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i\hat{\boldsymbol{K}}\boldsymbol{C}_i k(\boldsymbol{X},\boldsymbol{x}) \\
&= k(\boldsymbol{x},\boldsymbol{x}) - k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X},\boldsymbol{x}) + \sigma^2 \\
&= k_i(\boldsymbol{x},\boldsymbol{x}) + \sigma^2
\end{aligned}$$

We prove (3.13) by an analogous argument. Choose $c_j \coloneqq ((\hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i)k^\sigma(\boldsymbol{X},\boldsymbol{x}))_j$. We have

$$\left(\sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} (\mu_*^g(\boldsymbol{x}) - \mu_i^g(\boldsymbol{x}))\right)^2 = \left(\sup_{g \in \mathcal{H}_{k^\sigma} : \|g\|_{\mathcal{H}_{k^\sigma}} \leq 1} \sum_{j=0}^{n} c_j g(\boldsymbol{x}_j)\right)^2$$

$$
\begin{aligned}
&= \|\sum_{j=1}^{n} k(\boldsymbol{x}, \boldsymbol{x}_j)(\hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i)k^\sigma(\cdot, \boldsymbol{x}_j)\|^2_{\mathcal{H}_{k^\sigma}} \\
&= \|k(\boldsymbol{x}, \boldsymbol{X})(\hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i)k^\sigma(\boldsymbol{X}, \cdot)\|^2_{\mathcal{H}_{k^\sigma}} \\
&= k^\sigma(\boldsymbol{x}, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}\hat{\boldsymbol{K}}\hat{\boldsymbol{K}}^{-1}k^\sigma(\boldsymbol{X}, \boldsymbol{x}) \\
&\qquad - 2k^\sigma(\boldsymbol{x}, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}\hat{\boldsymbol{K}}\boldsymbol{C}_i k^\sigma(\boldsymbol{X}, \boldsymbol{x}) \\
&\qquad + k^\sigma(\boldsymbol{x}, \boldsymbol{X})\boldsymbol{C}_i\hat{\boldsymbol{K}}\boldsymbol{C}_i k^\sigma(\boldsymbol{X}, \boldsymbol{x})
\end{aligned}
$$

Again, we use that $k^\sigma(\boldsymbol{x}, \boldsymbol{X}) = k(\boldsymbol{x}, \boldsymbol{X})$ by assumption and (B.26). Therefore

$$
\begin{aligned}
&= k(\boldsymbol{x}, \boldsymbol{X})(\hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i)k(\boldsymbol{X}, \boldsymbol{x}) \\
&= k_i^{\text{comp}}(\boldsymbol{x}, \boldsymbol{x})
\end{aligned}
$$

This concludes the proof. □

B.3 Implementation of IterGP

B.3.1 Policy Choice

As illustrated in Figure 3.2, the choice of policy of Algorithm 2 determines where computation in input space is targeted and therefore where the combined posterior contracts first. However, the policy also determines whether the error in the posterior mean or (co-)variance are predominantly reduced first, as Figure B.2 shows (cf. IterGP-Chol and IterGP-PBR). Therefore the policy choice is application-dependent. If we are primarily interested in the predictive mean, we might select residual actions (IterGP-CG). If downstream we are making use of the predictive uncertainty, we might want to contract uncertainty globally as quickly as possible at the expense of predictive accuracy (IterGP-PI). Such a choice is not unique to IterGP, but necessary whenever we select a GP approximation. What IterGP adds is computation-aware, meaningful uncertainty quantification in the sense of Corollary 3.1 no matter the choice of policy.

B.3.2 Stopping Criterion

In our implementation of Algorithm 2 we use the following two stopping criteria. Our computational budget can be directly controlled by specifying a *maximum number of iterations*, since each iteration of IterGP needs the same number of matrix-vector multiplies. Alternatively, we terminate if the *absolute or relative norm of the residual* are sufficiently small, i.e. if

$$
\|\boldsymbol{r}_i\|_2 < \delta_{\text{abstol}} \qquad \text{or} \qquad \|\boldsymbol{r}_i\|_2 < \delta_{\text{reltol}}\|\boldsymbol{y}\|_2. \tag{B.37}
$$

Of course other choices are possible. From a probabilistic numerics standpoint one may want to terminate once the combined marginal uncertainty at the training data is sufficiently small relative to the observation noise.

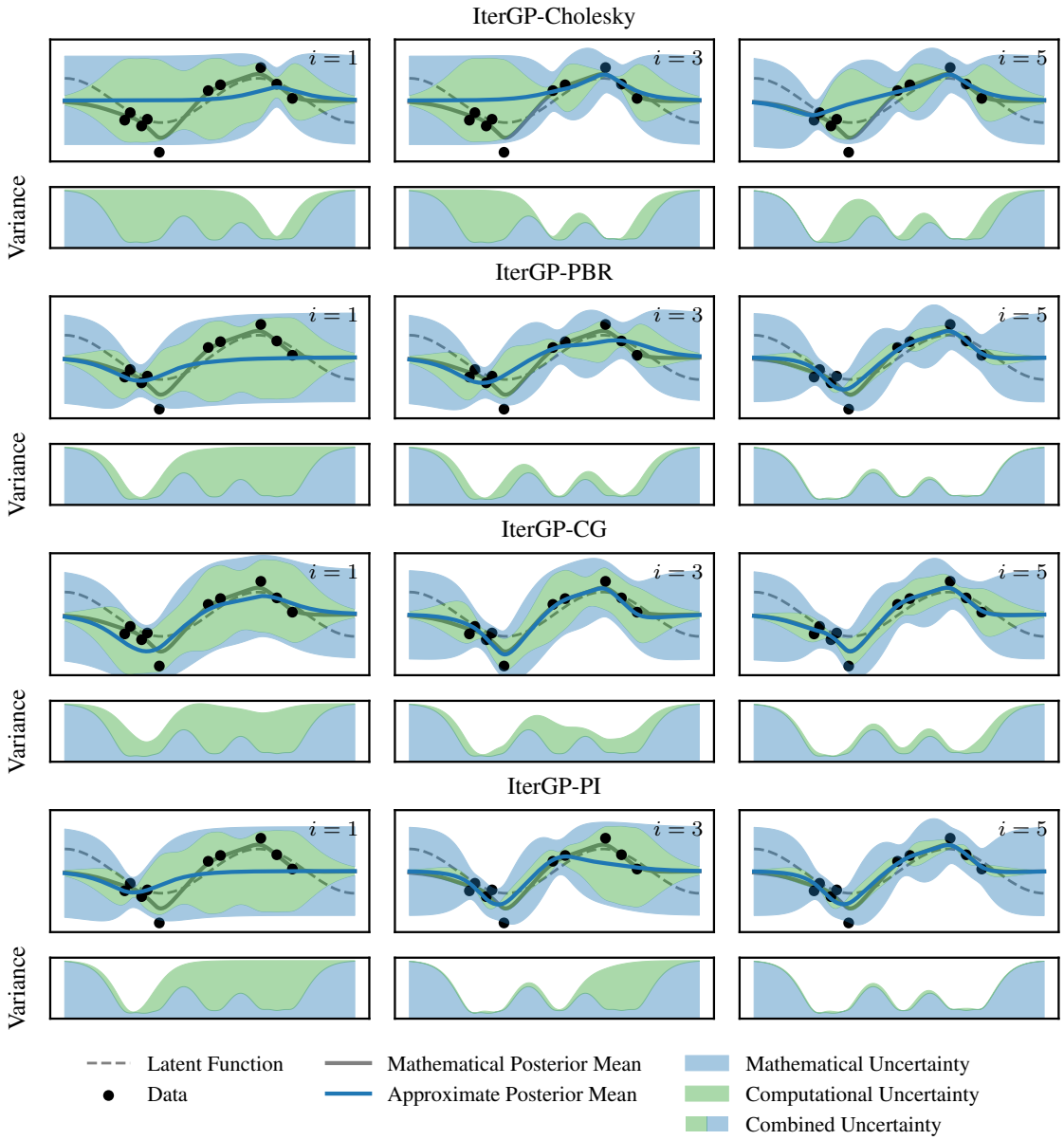


Figure B.2: *Illustration of IterGP analogs of commonly used GP approximations.*

B.3.3 Efficient Sampling from the Combined Posterior

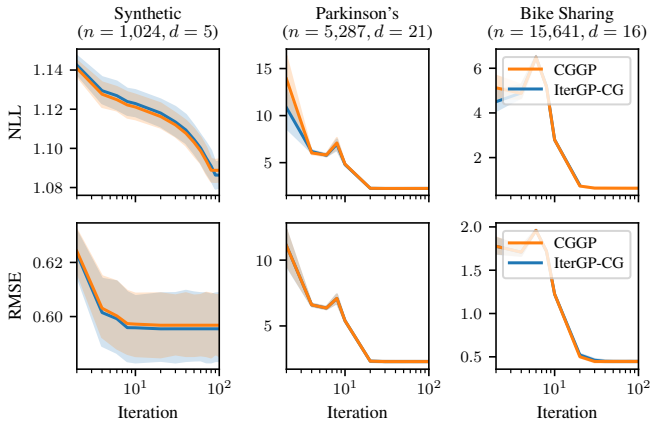
Sampling from an exact GP posterior has cubic cost $\mathcal{O}(n_\diamond^3)$ in the number of evaluation points $n_\diamond$, which is prohibitive for many useful downstream applications such as numerical integration over the posterior using Monte-Carlo methods. Wilson et al. [125, 126] recently showed how to make use of *Matheron's rule* [124, 165, 166] to efficiently sample from a GP posterior by sampling from the prior and then performing a pathwise update. We can directly make use of this strategy since Algorithm 2 computes a low-rank approximation to the precision matrix. Assume we are given a draw $f'_{\text{prior}} \in \mathcal{H}_k^\theta$ from the prior[1] such that $\boldsymbol{y}' \sim \mathcal{N}\left(f'_{\text{prior}}(\boldsymbol{X}), \sigma^2 \boldsymbol{I}\right)$ constitutes a draw from the prior predictive. Then

$$f'(\cdot) = f'_{\text{prior}}(\cdot) + k(\cdot, \boldsymbol{X})\boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{y}') \tag{B.38}$$

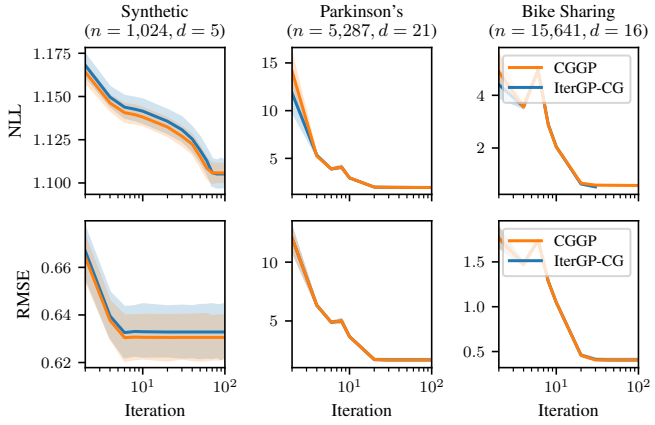
is a draw from the combined posterior by Matheron's rule, which we can evaluate in $\mathcal{O}(n_\diamond n i)$ for $n_\diamond$ evaluation points, since $\boldsymbol{C}_i$ has rank i.

[1] In infinite-dimensional reproducing kernel Hilbert spaces samples $f \sim \mathcal{GP}(\mu, k)$ from a Gaussian process almost surely do not lie in the RKHS $\mathcal{H}_k$ [Cor. 4.10, 44]. However, there exists $f' \in \mathcal{H}_k^\theta$ in a larger RKHS $\mathcal{H}_k^\theta \supset \mathcal{H}_k$ such that $f'(\boldsymbol{x}) = f(\boldsymbol{x})$ with probability 1 [Thm. 4.12, 44].

B.4 Additional Experimental Results

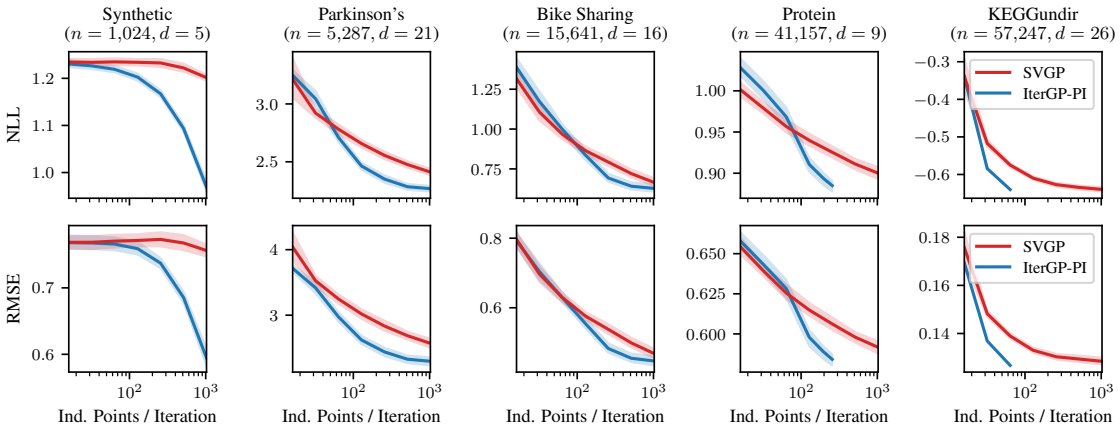


(a) RBF kernel

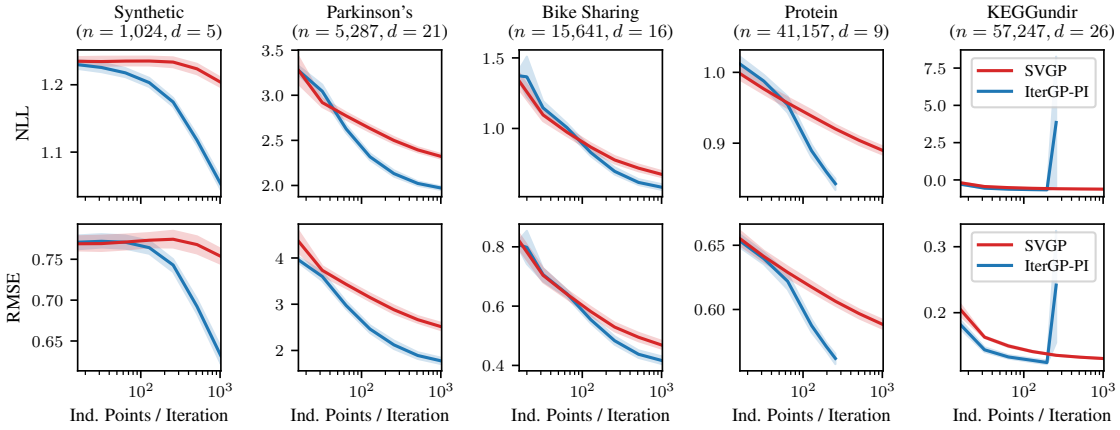


(b) Matérn($\frac{3}{2}$) kernel

Figure B.3: *Generalization of CGGP and its closest IterGP analog.* GP regression using an RBF and Matérn($\frac{3}{2}$) kernel on UCI datasets. The plot shows the average generalization error in terms of NLL and RMSE for an increasing number of solver iterations. The posterior mean of IterGP-CG and CGGP is identical, which explains the identical RMSE.



(a) RBF kernel



(b) Matérn($\frac{3}{2}$) kernel

Figure B.4: *Generalization of SVGP and its closest IterGP analog.* GP regression using an RBF and Matérn($\frac{3}{2}$) kernel on UCI datasets. The plot shows the average generalization error in terms of NLL and RMSE for an increasing number of identical inducing points. After a small number of inducing points relative to the size of the training data, IterGP has significantly lower generalization error than SVGP. For the "KEGGundir" dataset after ≈ 128 iterations we observe numerical instability in some runs when computing the combined posterior of IterGP using a Matérn($\frac{3}{2}$) kernel.

Appendix

C

Appendix of Chapter 4

C.1 Background on Krylov Methods

C.1.1 Conjugate Gradient Method

Theorem C.1 (Convergence Rate of Preconditioned CG [58])
Let $\boldsymbol{A}, \boldsymbol{P} \in \mathbb{R}^{n\times n}$ be symmetric positive definite. The error of the conjugate gradient method with preconditioner $\boldsymbol{P}$ after $i \in \mathbb{N}$ steps is given by

$$\|\boldsymbol{x}_k - \boldsymbol{x}\|_{\boldsymbol{A}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\boldsymbol{x}_0 - \boldsymbol{x}\|_{\boldsymbol{A}} \tag{C.1}$$

and in euclidean norm by

$$\|\boldsymbol{x}_k - \boldsymbol{x}\|_2 \leq 2\sqrt{\kappa(\boldsymbol{A})}\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\boldsymbol{x}_0 - \boldsymbol{x}\|_2 \tag{C.2}$$

where $\kappa = \kappa(\boldsymbol{P}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{P}^{-\frac{\intercal}{2}})$ is the condition number of the preconditioned system matrix.

Proof. Preconditioned CG is equivalent to running CG on the transformed problem

$$\tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \boldsymbol{P}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{P}^{-\frac{\intercal}{2}}\tilde{\boldsymbol{x}} = \boldsymbol{P}^{-\frac{1}{2}}\boldsymbol{b}$$

with the substitution $\tilde{\boldsymbol{x}} = \boldsymbol{P}^{\frac{\intercal}{2}}\boldsymbol{x}$. By Trefethen and Bau [58], the convergence rate of CG on the problem is given by

$$\|\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}\|_{\tilde{\boldsymbol{A}}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\tilde{\boldsymbol{x}}_0 - \tilde{\boldsymbol{x}}\|_{\tilde{\boldsymbol{A}}}$$

The first equation follows by recognizing that

$$\|\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}\|^2_{\tilde{\boldsymbol{A}}} = (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}})^\intercal \boldsymbol{P}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{P}^{-\frac{\intercal}{2}}(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}) = (\boldsymbol{x}_i - \boldsymbol{x})\boldsymbol{A}(\boldsymbol{x}_i - \boldsymbol{x}) = \|\boldsymbol{x}_i - \boldsymbol{x}\|^2_{\boldsymbol{A}}.$$

Now it holds by the min-max principle, that

$$\begin{aligned}\sqrt{\lambda_{\min}(\tilde{\boldsymbol{A}})}\|\boldsymbol{x}_k - \boldsymbol{x}\|_2 \leq \|\boldsymbol{x}_i - \boldsymbol{x}\|_{\boldsymbol{A}} &\leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\boldsymbol{x}_0 - \boldsymbol{x}\|_{\boldsymbol{A}} \\ &\leq 2\sqrt{\lambda_{\max}(\boldsymbol{A})}\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\boldsymbol{x}_0 - \boldsymbol{x}\|_2.\end{aligned}$$

□

Corollary C.1
Let $\varepsilon \in (0, 1]$, then preconditioned CG has relative error $\|\boldsymbol{x}_k - \boldsymbol{x}\|_{\boldsymbol{A}} \leq \varepsilon\|\boldsymbol{x}_0 - \boldsymbol{x}\|_{\boldsymbol{A}}$ after

$$i \geq \frac{\sqrt{\kappa}}{2}\log(2\varepsilon^{-1}) \tag{C.3}$$

iterations, where κ is the condition number of the preconditioned system matrix. In euclidean norm $\|\cdot\|_2$ relative error ε is achieved after

$$i \geq \frac{\sqrt{\kappa}}{2} \log(2\sqrt{\kappa(\boldsymbol{A})}\varepsilon^{-1}) \tag{C.4}$$

iterations.

Proof. It holds by Lemma C.7 and the assumption on the number of iterations m, that

$$2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m \leq 2\exp(-\frac{2}{\sqrt{\kappa}}m) \leq 2\exp(-\log(\frac{2}{\varepsilon})) = \varepsilon$$

Using Theorem C.1 proves the statement. The proof for the euclidean norm is analogous. □

C.1.2 Lanczos Algorithm

The *Lanczos algorithm* [77] is a Krylov method, which for a symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ iteratively builds an approximate tridiagonalization

$$\boldsymbol{A} \approx \tilde{\boldsymbol{Q}}\tilde{\boldsymbol{T}}\tilde{\boldsymbol{Q}}$$

where $\tilde{\boldsymbol{Q}} \in \mathbb{R}^{n\times i}$ orthonormal and $\tilde{\boldsymbol{T}} \in \mathbb{R}^{i\times i}$ tridiagonal. For an initial probe vector $\boldsymbol{b} \in \mathbb{R}^n$, Gram-Schmidt orthogonalization is applied to the Krylov subspace basis. The orthogonalized vectors form $\tilde{\boldsymbol{Q}}$, while the Gram-Schmidt coefficients form $\tilde{\boldsymbol{T}}$. This low-rank approximation becomes an exact tridiagonalization $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^\intercal$ for $i = n$. The Lanczos process is often used to compute (approximate) eigenvalues and eigenvectors, which is done by computing an eigendecomposition of the tridiagonal matrix $\tilde{\boldsymbol{T}}$ at cost $\mathcal{O}(i^2)$. The tridiagonal matrix $\tilde{\boldsymbol{T}}$ can also be formed by running CG on the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and by collecting the step lengths α_j and conjugacy corrections β_j used in the solution and search direction updates [159, Section 6.7.3].

C.1.3 Stochastic Lanczos Quadrature

One can approximate $\operatorname{tr}(f(\boldsymbol{A}))$ for symmetric positive definite $\boldsymbol{A}$ via *stochastic Lanczos quadrature* (SLQ) [138, 142] by combining Hutchinson's estimator with quadrature and the Lanczos algorithm. It holds that

$$\operatorname{tr}(f(\boldsymbol{A})) \approx \tau_\ell^{\mathrm{STE}}(f(\boldsymbol{A})) = \frac{n}{\ell}\sum_{j=1}^{\ell} \boldsymbol{z}_j^\intercal f(\boldsymbol{A})\boldsymbol{z}_j \approx \frac{n}{\ell}\sum_{j=1}^{\ell} I_i^{(j)} = \tau_{\ell,i}^{\mathrm{SLQ}}(f(\boldsymbol{A}))$$

The quadratic terms $\boldsymbol{z}_j^\intercal f(\boldsymbol{A})\boldsymbol{z}_j$ are approximated by quadrature $I_i^{(j)}$ where the weights and nodes of the quadrature rule are computed via i iterations of the Lanczos algorithm. For the $\log$-determinant the following bound for the error incurred by Lanczos quadrature holds.

Corollary C.2 (Section 4.3 of Ubaru, Chen, and Saad [138])
Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be symmetric positive definite with condition number $\kappa = \kappa(\boldsymbol{A})$. Then it holds that

$$\left|\tau_{\ell}^{\mathrm{STE}}(\log(\boldsymbol{A})) - \tau_{\ell,i}^{\mathrm{SLQ}}(\log(\boldsymbol{A}))\right| \leq K \left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i} \tag{C.5}$$

where $K = \frac{5\kappa \log(2(\kappa+1))}{2\sqrt{2\kappa+1}}$.

C.2 Stochastic Trace Estimation

Definition C.1 (Convex Concentration Property [167])
Let $\boldsymbol{x} \in \mathbb{R}^n$ be a random vector. We say $\boldsymbol{x}$ has the *convex concentration property* (c.c.p.) with constant $K \in \mathbb{R}$ if for every 1-Lipschitz convex function $\phi : \mathbb{R}^n \to \mathbb{R}$, we have $\mathbb{E}(|\phi(\boldsymbol{x})|) < \infty$ and for every $t > 0$,

$$\mathrm{P}(|\phi(\boldsymbol{x}) - \mathbb{E}(\phi(\boldsymbol{x}))| \geq t) \leq 2\exp(-\frac{t^2}{K^2}).$$

Examples of random vectors satisfying the convex concentration property are vectors

- with independent and almost surely bounded entries $|x_j| \leq 1$, where $K = 2\sqrt{2}$ [168];
- Gaussian random vectors $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$, where $K^2 = 2\|\boldsymbol{\Sigma}\|_2$ [169]; and
- uniformly distributed random vectors on the sphere $\sqrt{n}\mathbb{S}^{n-1}$, where $K = 2$ [169].

Remark C.1
Note, that since Rademacher random vectors have iid entries $\{+1, -1\}$, they satisfy $\|\tilde{\boldsymbol{z}}_j\|_2 = \sqrt{n}$. In particular, it holds that $\sqrt{n}\boldsymbol{z}_j = \tilde{\boldsymbol{z}}_j$. Therefore the random vectors $\sqrt{n}\boldsymbol{z}_j$ and $\boldsymbol{z}' = \sqrt{n}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell)^\intercal \in \mathbb{R}^{\ell n}$ all have independent entries bounded by 1 and thus satisfy the convex concentration property with $K = 2\sqrt{2}$.

Theorem C.2 (Hanson-Wright Inequality for Random Vectors with the Convex Concentration Property [170])
Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{x} \in \mathbb{R}^n$ a zero-mean random vector with the convex concentration property with constant K. Then for all $t > 0$, it holds that

$$\mathrm{P}(|\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} - \mathbb{E}(\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x})| \geq t) \leq 2\exp(-c\min\left(\frac{t^2}{\|\boldsymbol{A}\|_F^2}, \frac{t}{\|\boldsymbol{A}\|_2}\right)) \tag{C.6}$$

where $c = c(K) > 0$ is a constant only dependent on the distribution of the random vectors.

Proof. By Theorem 2.5 of Adamczak [170] we have

$$\mathrm{P}(|\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} - \mathbb{E}(\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x})| > t) \leq 2\exp(-\frac{1}{c'}\min\left(\frac{t^2}{2K^4\|\boldsymbol{A}\|_F^2}, \frac{t}{K^2\|\boldsymbol{A}\|_2}\right))$$

where $c' > 0$ is a universal constant. Now it holds that

$$\min\left(\frac{t^2}{2K^4\|\boldsymbol{A}\|_F^2}, \frac{t}{K^2\|\boldsymbol{A}\|_2}\right) \geq \frac{1}{K^2\max(2K^2,1)}\min\left(\frac{t^2}{\|\boldsymbol{A}\|_F^2}, \frac{t}{\|\boldsymbol{A}\|_2}\right)$$

Choosing $c = \frac{1}{c'K^2\max(2K^2,1)}$ concludes the proof. □

Lemma C.1
Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\ell \in \mathbb{N}$. Consider ℓ random vectors $\tilde{\boldsymbol{z}}_j \in \mathbb{R}^n$ with zero mean and unit covariance, such that for $\boldsymbol{z}_j = \tilde{\boldsymbol{z}}_j/\|\tilde{\boldsymbol{z}}_j\|_2$ the stacked random vector $\sqrt{n}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell)^\intercal \in \mathbb{R}^{\ell n}$ has the convex concentration property.[a] Then there exists $c_{\boldsymbol{z}} > 0$ such that if $\ell \geq c_{\boldsymbol{z}}\log(\delta^{-1})$, then Hutchinson's trace estimator τ_ℓ^{STE} satisfies

$$\mathrm{P}\left(|\tau_\ell^{\mathrm{STE}}(\boldsymbol{A}) - \mathrm{tr}(\boldsymbol{A})| \leq \sqrt{c_{\boldsymbol{z}}\log(\delta^{-1})\ell^{-1}}\|\boldsymbol{A}\|_F\right) \geq 1 - \delta.$$

[a] See Remark C.1 for an explanation of why this is satisfied for Rademacher random vectors.

Proof. The proof strategy used here is the same as in Meyer et al. [144, Lemma 2] with a different assumption on the distribution of the random vectors. To begin, define

$$\boldsymbol{A}' = \begin{pmatrix} \boldsymbol{A} & 0 & \ldots & 0 \\ 0 & \boldsymbol{A} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \boldsymbol{A} \end{pmatrix} \in \mathbb{R}^{\ell n\times \ell n} \quad \text{and} \quad \boldsymbol{z}' = \sqrt{n}\begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \vdots \\ \boldsymbol{z}_\ell \end{pmatrix} \in \mathbb{R}^{\ell n}.$$

By assumption the random vector $\boldsymbol{z}'$ has the convex concentration property and therefore Theorem C.2 holds. We obtain

$$\mathrm{P}\left(|(\boldsymbol{z}')^\intercal \boldsymbol{A}'\boldsymbol{z}' - \mathbb{E}((\boldsymbol{z}')^\intercal \boldsymbol{A}'\boldsymbol{z}')| \geq t\right) \leq 2\exp(-c\cdot\min\left(\frac{t^2}{\|\boldsymbol{A}'\|_F^2}, \frac{t}{\|\boldsymbol{A}'\|_2}\right)). \tag{C.7}$$

Now, we have $(\boldsymbol{z}')^\intercal \boldsymbol{A}'\boldsymbol{z}' = n\sum_{j=1}^{\ell}\boldsymbol{z}_j^\intercal \boldsymbol{A}\boldsymbol{z}_j = \ell\tau_\ell^{\mathrm{STE}}(\boldsymbol{A})$ and

$$\begin{aligned}\mathbb{E}((\boldsymbol{z}')^\intercal \boldsymbol{A}'\boldsymbol{z}') &= n\sum_{j=1}^{\ell}\mathbb{E}\left(\boldsymbol{z}_j^\intercal \boldsymbol{A}\boldsymbol{z}_j\right) = n\ell\,\mathbb{E}\left(\mathrm{tr}(\boldsymbol{z}_j^\intercal \boldsymbol{A}\boldsymbol{z}_j)\right) = n\ell\,\mathbb{E}\left(\mathrm{tr}(\boldsymbol{A}\boldsymbol{z}_j\boldsymbol{z}_j^\intercal)\right) \\ &= n\ell\,\mathrm{tr}(\boldsymbol{A}\,\mathbb{E}\left(\boldsymbol{z}_j\boldsymbol{z}_j^\intercal\right)) = n\ell\,\mathrm{tr}(\boldsymbol{A}\,\mathrm{Cov}(\boldsymbol{z}_j)) = \ell\,\mathrm{tr}(\boldsymbol{A}\,\mathrm{Cov}\left(\sqrt{n}\boldsymbol{z}_j\right)) = \ell\,\mathrm{tr}(\boldsymbol{A})\end{aligned}$$

Therefore by setting $t = \sqrt{\frac{\log(2\delta^{-1})}{c}\ell}\|\boldsymbol{A}\|_F$, we obtain

$$\mathrm{P}\left(\ell|\tau_\ell^{\mathrm{STE}}(\boldsymbol{A}) - \mathrm{tr}(\boldsymbol{A})| \geq \sqrt{\frac{\log(2\delta^{-1})}{c}\ell}\|\boldsymbol{A}\|_F\right)$$
$$\leq 2\exp(-c \cdot \min\left(\frac{\log(2\delta^{-1})}{c}\frac{\ell\|\boldsymbol{A}\|_F^2}{\|\boldsymbol{A}'\|_F^2}, \sqrt{\frac{\log(2\delta^{-1})}{c}\ell}\frac{\|\boldsymbol{A}\|_F}{\|\boldsymbol{A}'\|_2}\right))$$

Further, it holds that $\|\boldsymbol{A}'\|_F^2 = \ell\|\boldsymbol{A}\|_F^2$ and $\|\boldsymbol{A}'\|_2 = \|\boldsymbol{A}\|_2$, thus we have

$$= 2\exp(-\min\left(\log(2\delta^{-1}), \sqrt{c\log(2\delta^{-1})\ell}\frac{\|\boldsymbol{A}\|_F}{\|\boldsymbol{A}\|_2}\right)).$$

Now assume $\ell \geq \frac{1}{c}\log(2\delta^{-1})$. Then since $\|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_F$, the minimum is given by

$$\min\left(\log(2\delta^{-1}), \sqrt{c\log(2\delta^{-1})\ell}\frac{\|\boldsymbol{A}\|_F}{\|\boldsymbol{A}\|_2}\right) = \log(2\delta^{-1}).$$

Further setting $c_{\boldsymbol{z}} = 2c^{-1}$, it holds that

$$\ell \geq c_{\boldsymbol{z}}\log(\delta^{-1}) = 2\log(\delta^{-1})c^{-1} \geq \log(2\delta^{-1})c^{-1}$$

since $0 < \delta \leq \frac{1}{2}$. Combining the above we obtain

$$\mathrm{P}\left(\ell|\tau_\ell^{\mathrm{STE}}(\boldsymbol{A}) - \mathrm{tr}(\boldsymbol{A})| \geq \sqrt{c_{\boldsymbol{z}}\log(\delta^{-1})\ell}\|\boldsymbol{A}\|_F\right) \leq 2\exp(-\log(2\delta^{-1})) = \delta,$$

which is equivalent to

$$\mathrm{P}\left(|\tau_\ell^{\mathrm{STE}}(\boldsymbol{A}) - \mathrm{tr}(\boldsymbol{A})| \leq \sqrt{c_{\boldsymbol{z}}\log(\delta^{-1})\ell^{-1}}\|\boldsymbol{A}\|_F\right) \geq 1 - \delta.$$

This proves the statement. □

Theorem 4.1 (Variance-reduced Stochastic Trace Estimation)
Let $\hat{\boldsymbol{K}}, \hat{\boldsymbol{P}}_\ell \in \mathbb{R}_{\mathrm{spd}}^{n\times n}, \boldsymbol{\Delta}_f \in \mathbb{R}^{n\times n}$ *and* $f : \mathbb{R}_{\mathrm{spd}}^{n\times n} \to \mathbb{R}$ *such that* $\mathrm{tr}(f(\hat{\boldsymbol{K}})) = \mathrm{tr}(f(\hat{\boldsymbol{P}}_\ell)) + \mathrm{tr}(\boldsymbol{\Delta}_f)$*, and define the estimator* $\tau_* = \mathrm{tr}(f(\hat{\boldsymbol{P}}_\ell)) + \tau_\ell^{\mathrm{STE}}(\boldsymbol{\Delta}_f)$*. Now, assume there exist* $c_{\boldsymbol{\Delta}} > 0$ *and* $g : \mathbb{N} \to (0, \infty)$ *such that*

$$\|\boldsymbol{\Delta}_f\|_F \leq c_{\boldsymbol{\Delta}}g(\ell)\|f(\hat{\boldsymbol{K}})\|_F. \tag{4.9}$$

Then there exists $c_{\boldsymbol{z}} > 0$ *dependent on the choice of random vectors, such that, if* $\ell \geq c_{\boldsymbol{z}}\log(\delta^{-1})$*, it holds with probability* $1 - \delta \in [\frac{1}{2}, 1)$ *that*

$$|\tau_* - \mathrm{tr}(f(\hat{\boldsymbol{K}}))| \leq \varepsilon_{\mathrm{STE}}\|f(\hat{\boldsymbol{K}})\|_F. \tag{4.10}$$

where for $C_1 = c_{\boldsymbol{\Delta}}\sqrt{c_{\boldsymbol{z}}}$ the relative error is given by

$$\varepsilon_{\mathrm{STE}}(\delta, \ell) = C_1\sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}}g(\ell). \tag{4.11}$$

Proof. By assumption $|\tau_* - \operatorname{tr}(f(\hat{\boldsymbol{K}}))| = |\tau_\ell^{\mathrm{STE}}(\boldsymbol{\Delta}_f) - \operatorname{tr}(\boldsymbol{\Delta}_f)|$. By Lemma C.1 it holds with probability $\geq 1-\delta$, that

$$\begin{aligned}
|\tau_\ell^{\mathrm{STE}}(\boldsymbol{\Delta}_f) - \operatorname{tr}(\boldsymbol{\Delta}_f)| &\leq \sqrt{c_{\boldsymbol{z}}\log(\delta^{-1})\ell^{-1}}\|\boldsymbol{\Delta}_f\|_F \\
&\leq \sqrt{c_{\boldsymbol{z}}\log(\delta^{-1})}c_{\boldsymbol{\Delta}_f}\ell^{-\frac{1}{2}}g(\ell)\|f(\hat{\boldsymbol{K}})\|_F \qquad \text{Assumption (4.9).} \\
&= \varepsilon_{\mathrm{STE}}(\delta, \ell)\|f(\hat{\boldsymbol{K}})\|_F
\end{aligned}$$

This concludes the proof. □

Corollary C.3
Let $\varepsilon \in (0, 1]$ be a desired error. If the conditions of Theorem 4.1 hold and the number of random vectors ℓ satisfies

$$\ell^{\frac{1}{2}}g(\ell)^{-1} \geq C_1\varepsilon^{-1}\sqrt{\log(\delta^{-1})}, \tag{C.8}$$

then it holds that

$$\boxed{\mathrm{P}(|\tau_* - \operatorname{tr}(\boldsymbol{A})| \leq \varepsilon\|\boldsymbol{A}\|_F) \geq 1-\delta.}$$

Proof. Follows from Theorem 4.1 given (C.8). □

C.3 Log-Determinant Estimation

C.3.1 Approximation of a Matrix Function

Lemma C.2 (Lipschitz Continuity)
Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ be symmetric. Assume $f : \Omega \to \mathbb{R}$ is globally Lipschitz continuous with Lipschitz constant $L > 0$ on the combined spectrum $\Omega = \lambda(\boldsymbol{A}) \cup \lambda(\boldsymbol{B}) \subset \mathbb{R}$, then there exists $c_p > 0$ such that

$$\|f(\boldsymbol{A}) - f(\boldsymbol{B})\|_p \leq c_p L\|\boldsymbol{A} - \boldsymbol{B}\|_p, \tag{C.9}$$

where $\|\cdot\|_p$ denotes any matrix norm. In particular $c_2 = 1$ and $c_F = \sqrt{n}$.

Proof. Since $\boldsymbol{A}, \boldsymbol{B}$ are symmetric, they are normal. By Kittaneh [171], it holds that

$$\|f(\boldsymbol{A}) - f(\boldsymbol{B})\|_2 \leq L\|\boldsymbol{A} - \boldsymbol{B}\|_2.$$

The result now follows by equivalence of norms on finite-dimensional spaces. For the Frobenius norm we have $\frac{1}{\sqrt{n}}\|\boldsymbol{M}\|_F \leq \|\boldsymbol{M}\|_2 \leq \|\boldsymbol{M}\|_F$, and therefore $c_F = \sqrt{n}$.

□

Proposition C.1
Let $\hat{\boldsymbol{K}} \in \mathbb{R}^{n\times n}$ be symmetric positive definite and assume f is analytic in a domain containing the spectrum $\lambda(\hat{\boldsymbol{K}})$. Let $\{\hat{\boldsymbol{P}}_\ell\}_\ell$ be a sequence of preconditioners with approximation quality (4.6). Then it holds that

$$\|f(\hat{\boldsymbol{K}}) - f(\hat{\boldsymbol{P}}_\ell)\|_F \leq c(n, \hat{\boldsymbol{K}}, f)g(\ell)\|f(\hat{\boldsymbol{K}})\|_F \tag{C.10}$$

where $c(n, \hat{\boldsymbol{K}}, f) = \frac{L\|\hat{\boldsymbol{K}}\|_F}{c_{f(\lambda)}}$, $L > 0$ is the Lipschitz constant of f and

$$c_{f(\lambda)} = \max\{\min_j |f(\lambda_j(\hat{\boldsymbol{K}}))|, \frac{\max_j |f(\lambda_j(\hat{\boldsymbol{K}}))|}{\sqrt{n}}\}.$$

Proof. It holds that

$$\|f(\hat{\boldsymbol{K}})\|_F = \sqrt{\sum_{j=1}^{n} f(\lambda_j)^2} \geq \begin{cases} \sqrt{n \min_j f(\lambda_j)^2} = \sqrt{n}\min_j |f(\lambda_j)| \\ \|f(\hat{\boldsymbol{K}})\|_2 = \sigma_{\max}(f(\hat{\boldsymbol{K}})) = \sqrt{\lambda_{\max}(f(\hat{\boldsymbol{K}})^2)} = \max_j |f(\lambda_j)| \end{cases}$$

and therefore $\|f(\hat{\boldsymbol{K}})\|_F \geq \sqrt{n}c_{f(\lambda)}$. Since f is analytic and therefore Lipschitz, it holds that

$$\begin{aligned}
\|f(\hat{\boldsymbol{K}}) - f(\hat{\boldsymbol{P}}_\ell)\|_F &\leq L\sqrt{n}\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F && \text{Lemma C.2} \\
&\leq L\sqrt{n}g(\ell)\|\hat{\boldsymbol{K}}\|_F && \text{Preconditioner quality (4.6)} \\
&= L\sqrt{n}g(\ell)\frac{\|\hat{\boldsymbol{K}}\|_F}{\|f(\hat{\boldsymbol{K}})\|_F}\|f(\hat{\boldsymbol{K}})\|_F \\
&\leq L\sqrt{n}g(\ell)\frac{\|\hat{\boldsymbol{K}}\|_F}{\sqrt{n}c_{f(\lambda)}}\|f(\hat{\boldsymbol{K}})\|_F \\
&\leq \frac{L\|\hat{\boldsymbol{K}}\|_F}{c_{f(\lambda)}}g(\ell)\|f(\hat{\boldsymbol{K}})\|_F.
\end{aligned}$$

This proves the claim. □

C.3.2 Approximation of the Log-Determinant

Lemma C.3 (Decomposition of the log-determinant)
For $\hat{\boldsymbol{K}}, \hat{\boldsymbol{P}} \in \mathbb{R}^{n\times n}$ symmetric positive definite, it holds that

$$\log\det(\hat{\boldsymbol{K}}) = \log\det(\hat{\boldsymbol{P}}) + \operatorname{tr}(\log(\hat{\boldsymbol{K}}) - \log(\hat{\boldsymbol{P}})) \tag{C.11}$$

$$= \log\det(\hat{\boldsymbol{P}}) + \operatorname{tr}(\log(\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\intercal\frac{1}{2}})). \tag{C.12}$$

Proof. For symmetric positive definite matrices $\boldsymbol{A}, \boldsymbol{B}$, the matrix logarithm satisfies the following

$$\begin{aligned}\log\det(\boldsymbol{A}) &= \operatorname{tr}(\log(\boldsymbol{A})),\\ \operatorname{tr}(\log(\boldsymbol{A}\boldsymbol{B})) &= \operatorname{tr}(\log(\boldsymbol{A})) + \operatorname{tr}(\log(\boldsymbol{B})),\\ \log(\boldsymbol{A}^{-1}) &= -\log(\boldsymbol{A}).\end{aligned}$$

Using the above properties, we obtain

$$\begin{aligned}\log\det(\hat{\boldsymbol{K}}) &= \operatorname{tr}(\log(\hat{\boldsymbol{P}}\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}))\\ &= \operatorname{tr}(\log(\hat{\boldsymbol{P}})) - \operatorname{tr}(\log(\hat{\boldsymbol{P}})) + \operatorname{tr}(\log(\hat{\boldsymbol{K}}))\\ &= \log\det(\hat{\boldsymbol{P}}) + \operatorname{tr}(\log(\hat{\boldsymbol{K}}) - \log(\hat{\boldsymbol{P}}))\end{aligned}$$

Now since $\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}$ and $\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\intercal\frac{1}{2}}$ are similar, they have the same determinant. Therefore we have

$$\operatorname{tr}(\log(\hat{\boldsymbol{K}}) - \log(\hat{\boldsymbol{P}})) = \operatorname{tr}(\log(\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}})) = \log\det(\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\intercal\frac{1}{2}}).$$

This completes the proof. □

Theorem 4.2 (Error Bound for $\log\det(\hat{\boldsymbol{K}})$)
Let $f = \log$, $\boldsymbol{\Delta}_{\log} = \log\left(\hat{\boldsymbol{P}}^{-\frac{1}{2}}\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-\intercal\frac{1}{2}}\right)$ and assume the conditions of Theorem 4.1 hold. Then, with probability $1-\delta$, it holds for $\tau_*^{\log} = \log(\det(\hat{\boldsymbol{P}})) + \tau_{\ell,i}^{\mathrm{SLQ}}(\boldsymbol{\Delta}_{\log})$, that

$$|\tau_*^{\log} - \log\det(\hat{\boldsymbol{K}})| \leq (\varepsilon_{\mathrm{Lanczos}} + \varepsilon_{\mathrm{STE}})\|\log(\hat{\boldsymbol{K}})\|_F,$$

where the individual errors are bounded by

$$\varepsilon_{\mathrm{Lanczos}}(\kappa, i) \leq K_1 \left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i} \tag{4.12}$$

$$\varepsilon_{\mathrm{STE}}(\delta, \ell) \leq C_1\sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}}g(\ell) \tag{4.13}$$

and $K_1 = \frac{5\kappa\log(2(\kappa+1))}{2\|\log(\hat{\boldsymbol{K}})\|_F\sqrt{2\kappa+1}}$.

Proof. Using the decomposition (4.8), we have

$$\begin{aligned}|\tau_*^{\log} - \log\det(\hat{\boldsymbol{K}})| &= |\tau_{\ell,i}^{\mathrm{SLQ}}(\boldsymbol{\Delta}_{\log}) - \operatorname{tr}(\boldsymbol{\Delta}_{\log})|\\ &\leq |\operatorname{tr}(\boldsymbol{\Delta}_{\log}) - \tau_\ell^{\mathrm{STE}}(\boldsymbol{\Delta}_{\log})| + |\tau_\ell^{\mathrm{STE}}(\boldsymbol{\Delta}_{\log}) - \tau_{\ell,i}^{\mathrm{SLQ}}(\boldsymbol{\Delta}_{\log})|\\ &= \underbrace{|\operatorname{tr}(\log(\hat{\boldsymbol{K}})) - (\operatorname{tr}(\log(\hat{\boldsymbol{P}})) + \tau_\ell^{\mathrm{STE}}(\boldsymbol{\Delta}_{\log}))|}_{e_{\mathrm{STE}}} + \underbrace{|\tau_\ell^{\mathrm{STE}}(\boldsymbol{\Delta}_{\log}) - \tau_{\ell,i}^{\mathrm{SLQ}}(\boldsymbol{\Delta}_{\log})|}_{e_{\mathrm{Lanczos}}}.\end{aligned}$$

Now the individual absolute errors are bounded as follows. By the error bound for stochastic trace estimation in Theorem 4.1, we have

$$e_{\mathrm{STE}} \leq \varepsilon_{\mathrm{STE}}(\delta,\ell)\|\log(\hat{\boldsymbol{K}})\|_F = C_1\sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}}g(\ell)\|\log(\hat{\boldsymbol{K}})\|_F$$

and by Corollary C.2, it follows that

$$e_{\mathrm{Lanczos}} \leq K\left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i} = K_1\left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i}\|\log(\hat{\boldsymbol{K}})\|_F.$$

□

Corollary 4.1
Assume the conditions of Theorem 4.2 hold. If the number of random vectors ℓ satisfies (4.11) with $\varepsilon_{\mathrm{STE}} = \frac{\varepsilon}{2}$ and we run

$$i \geq \tfrac{\sqrt{3}}{4}\sqrt{\kappa}\log\left(2K_1\varepsilon^{-1}\right) \tag{4.14}$$

iterations of Lanczos, then it holds that

$$\boxed{\mathrm{P}\left(|\tau_*^{\log} - \log\det(\hat{\boldsymbol{K}})| \leq \varepsilon\|\log(\hat{\boldsymbol{K}})\|_F\right) \geq 1-\delta.}$$

Proof. By assumption Theorem 4.1 is satisfied and therefore $\varepsilon_{\mathrm{STE}} = \frac{\varepsilon}{2}$ with probability $1-\delta$. Now for the error of Lanczos it holds by Theorem 4.2 in combination with Lemma C.7, that

$$\begin{aligned}
\varepsilon_{\mathrm{Lanczos}} &\leq K_1\left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i} \\
&\leq K_1\exp(-\frac{4}{\sqrt{2\kappa+1}}i) && \text{Lemma C.7} \\
&\leq K_1\exp(-\frac{\sqrt{3\kappa}}{\sqrt{2\kappa+1}}\log(2K_1\varepsilon^{-1})) && \text{By assumption (4.14).} \\
&\leq K_1\exp(-\log(2K_1\varepsilon^{-1})) \\
&= \frac{\varepsilon}{2}
\end{aligned}$$

The result now follows by Theorem 4.2. □

C.3.3 Approximation of the Derivative of the Log-Determinant

Computation of $\mathrm{tr}(\hat{\boldsymbol{P}}^{-1}\frac{\partial\hat{\boldsymbol{P}}}{\partial\theta})$ Algorithm 3 and Algorithm 4 primarily rely on matrix-vector multiplication, except for computation of $\tau_{\hat{\boldsymbol{P}}}^{\mathrm{inv}\partial} = \mathrm{tr}(\hat{\boldsymbol{P}}^{-1}\frac{\partial\hat{\boldsymbol{P}}}{\partial\theta})$. Efficient computation of this term depends on the structure of $\hat{\boldsymbol{P}}^{-1}$. If $\hat{\boldsymbol{P}}$ is the pivoted-Cholesky preconditioner, or any other

diagonal-plus-low-rank preconditioner $\sigma^2 \boldsymbol{I} + \boldsymbol{L}_\ell \boldsymbol{L}_\ell^\intercal$, we can rewrite this term using the matrix inversion lemma

$$\begin{aligned}\operatorname{tr}(\hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}) &= \sigma^{-2}\operatorname{tr}(\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}) - \sigma^{-2}\operatorname{tr}(\boldsymbol{L}_\ell \left(\sigma^2 \boldsymbol{I} + \boldsymbol{L}_\ell^\intercal \boldsymbol{L}_\ell\right)^{-1} \boldsymbol{L}_\ell^\intercal \frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}) \\ &= \sigma^{-2}\sum_{j=1}^{n} \frac{\partial \hat{\boldsymbol{P}}_{jj}}{\partial \theta} - \sigma^{-2}\left(\left(\boldsymbol{L}_\ell \left(\sigma^2 \boldsymbol{I} + \boldsymbol{L}_\ell^\intercal \boldsymbol{L}_\ell\right)^{-1}\right) \circ \left(\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}\boldsymbol{L}_\ell\right)\right)\mathbf{1}, \qquad \text{(C.13)}\end{aligned}$$

where $\circ$ denotes elementwise multiplication. The second term requires ℓ matrix-vector multiplies with $\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}$ and $\mathcal{O}(n\ell^2)$ additional work. The first term is simply the derivative of the kernel diagonal which will take $\mathcal{O}(n)$ time. Similar efficient procedures exist for other types of preconditioners, such as when $\hat{\boldsymbol{P}}^{-1}$ has a banded structure.

Theorem 4.3 (Error Bound for $\operatorname{tr}\big(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\big)$)
Let $f(\hat{\boldsymbol{K}}) = \hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}$, $\boldsymbol{\Delta}_{\text{inv}\partial} = \hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta} - \hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}$ and assume the conditions of Theorem 4.1 hold. If we solve $\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{z}_j$ with i iterations of preconditioned CG, initialized at $\mathbf{0}$ or better, then it holds with probability $1-\delta$ for $\tau_*^{\text{inv}\partial} = \operatorname{tr}\big(\hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}\big) + \tau_{\ell,i}^{\text{SCG}}(\boldsymbol{\Delta}_{\text{inv}\partial})$, that

$$\left|\tau_*^{\text{inv}\partial} - \operatorname{tr}\big(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\big)\right| \leq (\varepsilon_{\text{CG}'} + \varepsilon_{\text{STE}})\|\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta}\|_F,$$

where the individual errors are bounded by

$$\varepsilon_{\text{CG}'}(\kappa, i) \leq K_2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \qquad (4.17)$$

$$\varepsilon_{\text{STE}}(\delta, \ell) \leq C_1\sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}}g(\ell) \qquad (4.18)$$

and $K_2 = 2\sqrt{\kappa(\hat{\boldsymbol{K}})}n$.

Proof. Using the decomposition (4.15), we have

$$\begin{aligned}|\tau_*^{\text{inv}\partial} - \operatorname{tr}(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta})| &= |\tau_{\ell,i}^{\text{SCG}}(\boldsymbol{\Delta}_{\text{inv}\partial}) - \operatorname{tr}(\boldsymbol{\Delta}_{\text{inv}\partial})| \\ &\leq |\operatorname{tr}(\boldsymbol{\Delta}_{\text{inv}\partial}) - \tau_\ell^{\text{STE}}(\boldsymbol{\Delta}_{\text{inv}\partial})| + |\tau_\ell^{\text{STE}}(\boldsymbol{\Delta}_{\text{inv}\partial}) - \tau_{\ell,i}^{\text{SCG}}(\boldsymbol{\Delta}_{\text{inv}\partial})| \\ &= \underbrace{|\operatorname{tr}(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}) - (\operatorname{tr}(\hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}) + \tau_\ell^{\text{STE}}(\boldsymbol{\Delta}_{\text{inv}\partial}))|}_{e_{\text{STE}}} + \underbrace{|\tau_\ell^{\text{STE}}(\boldsymbol{\Delta}_{\text{inv}\partial}) - \tau_{\ell,i}^{\text{SCG}}(\boldsymbol{\Delta}_{\text{inv}\partial})|}_{e_{\text{CG}}}.\end{aligned}$$

Now the individual absolute errors are bounded as follows. By the error bound for stochastic trace estimation in Theorem 4.1, we have

$$e_{\text{STE}} \leq \varepsilon_{\text{STE}}(\delta, \ell)\|\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_F = C_1\sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}}g(\ell)\|\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_F.$$

Now, let $\boldsymbol{w}_j = \hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{z}_j$, $\tilde{\boldsymbol{w}}_j = \hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}\boldsymbol{z}_j$ and $\boldsymbol{w}_{i,j} \approx \boldsymbol{w}_j$ be the solution computed via preconditioned CG with i iterations. Then we have by Theorem C.1, that

$$
\begin{aligned}
e_{\mathrm{CG}} &= |\frac{n}{\ell}\sum_{j=1}^{\ell} \boldsymbol{z}_j^\intercal(\boldsymbol{w}_{i,j} - \tilde{\boldsymbol{w}}_j - (\boldsymbol{w}_j - \tilde{\boldsymbol{w}}_j))| \\
&\leq \frac{n}{\ell}\sum_{j=1}^{\ell} \underbrace{\|\boldsymbol{z}_j\|_2}_{=1}\|\boldsymbol{w}_{i,j} - \boldsymbol{w}_j\|_2 \\
&\leq \frac{n}{\ell}\sum_{j=1}^{\ell} 2\sqrt{\kappa(\hat{\boldsymbol{K}})}\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\boldsymbol{w}_{0,j} - \boldsymbol{w}_j\|_2 \qquad \text{CG convergence by Theorem C.1.} \\
&\leq 2n\sqrt{\kappa(\hat{\boldsymbol{K}})}\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\boldsymbol{w}_j\|_2 \qquad \text{CG initialized at } \boldsymbol{w}_{0,j} = \mathbf{0} \text{ or better.} \\
&\leq K_2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{z}_j\|_F \\
&\leq K_2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_F
\end{aligned}
$$

This completes the argument. □

Corollary 4.2
Assume the conditions of Theorem 4.3 hold. If the number of random vectors ℓ satisfies (4.11) with $\varepsilon_{\mathrm{STE}} = \frac{\varepsilon}{2}$, and we run

$$
i \geq \tfrac{1}{2}\sqrt{\kappa}\log(2K_2\varepsilon^{-1}) \tag{4.19}
$$

iterations of CG, then

$$
\boxed{\mathrm{P}\left(\left|\tau_*^{\mathrm{inv}\partial} - \mathrm{tr}\left(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\right)\right| \leq \varepsilon\|\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_F\right) \geq 1-\delta.}
$$

Proof. By assumption Theorem 4.1 is satisfied and therefore $\varepsilon_{\mathrm{STE}} = \frac{\varepsilon}{2}$ with probability $1-\delta$. Now for the error of CG, it holds by Theorem 4.3 in combination with Lemma C.7, that

$$
\begin{aligned}
\varepsilon_{\mathrm{CG}} &\leq K_2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \\
&\leq K_2\exp(-\frac{2i}{\sqrt{\kappa}}) \qquad \text{Lemma C.7} \\
&\leq K_2\exp(-\log(2K_2\varepsilon^{-1})) \qquad \text{Assumption (4.19)} \\
&\leq \frac{\varepsilon}{2}.
\end{aligned}
$$

The result now follows by Theorem 4.3. □

C.4 GP Hyperparameter Optimization

C.4.1 Approximation of the Log-Marginal Likelihood

Theorem 4.4 (Error Bound for the log-Marginal Likelihood)
Assume the conditions of Theorem 4.2 hold and we solve $\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{y}$ via preconditioned CG initialized at $\boldsymbol{v}_0$ and terminated after i iterations. Then with probability $1-\delta$, the error in the estimate $\eta = -\frac{1}{2}(\boldsymbol{y}^\intercal \boldsymbol{v}_i + \tau_*^{\log} + n\log(2\pi))$ of the log-marginal likelihood $\mathcal{L}$ satisfies

$$|\eta - \mathcal{L}| \leq \varepsilon_{\mathrm{CG}} + \tfrac{1}{2}(\varepsilon_{\mathrm{Lanczos}} + \varepsilon_{\mathrm{STE}})\|\log(\hat{\boldsymbol{K}})\|_F,$$

where the individual errors are bounded by

$$\varepsilon_{\mathrm{CG}}(\kappa, i) \leq K_3 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \tag{4.20}$$

$$\varepsilon_{\mathrm{Lanczos}}(\kappa, i) \leq K_1 \left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i} \tag{4.21}$$

$$\varepsilon_{\mathrm{STE}}(\delta, \ell) \leq C_1 \sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}} g(\ell) \tag{4.22}$$

for $K_3 = \sqrt{\kappa(\hat{\boldsymbol{K}})}\|\boldsymbol{y}\|_2\|\boldsymbol{v}_0 - \boldsymbol{v}_*\|_2$.

Proof. It holds by assumption that

$$\begin{aligned}
|\eta - \mathcal{L}| &= \frac{1}{2}|\boldsymbol{y}^\intercal \boldsymbol{v}_i + \tau_*^{\log} - (\boldsymbol{y}^\intercal \hat{\boldsymbol{K}}^{-1}\boldsymbol{y} + \log\det(\hat{\boldsymbol{K}}))| \\
&\leq \frac{1}{2}\big(\underbrace{|\boldsymbol{y}^\intercal \boldsymbol{v}_i - \boldsymbol{y}^\intercal \boldsymbol{v}_*|}_{e_{\mathrm{CG}}} + \underbrace{|\tau_*^{\log} - \log\det(\hat{\boldsymbol{K}})|}_{e_{\mathrm{SLQ}}}\big).
\end{aligned}$$

For the error of CG when solving $\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{y}$, we have by Theorem C.1

$$\begin{aligned}
e_{\mathrm{CG}} &= |\boldsymbol{y}^\intercal \boldsymbol{v}_i - \boldsymbol{y}^\intercal \boldsymbol{v}_*| \\
&\leq \|\boldsymbol{y}\|_2 \|\boldsymbol{v}_i - \boldsymbol{v}_*\|_2 \\
&\leq \|\boldsymbol{y}\|_2 2\sqrt{\kappa(\hat{\boldsymbol{K}})}\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\boldsymbol{v}_0 - \boldsymbol{v}_*\|_2 = 2\varepsilon_{\mathrm{CG}},
\end{aligned}$$

and for the absolute error in the log-determinant estimate via preconditioned stochastic Lanczos quadrature, we obtain by Theorem 4.2, that

$$e_{\mathrm{SLQ}} \leq (\varepsilon_{\mathrm{Lanczos}} + \varepsilon_{\mathrm{STE}})\|\log(\hat{\boldsymbol{K}})\|_F.$$

This proves the statement. □

Corollary C.4
Assume the conditions of Theorem 4.4 hold. If the number of random vectors ℓ satisfies (4.11) with $\varepsilon_{\mathrm{STE}} = \varepsilon$, and we run $i \geq \max(i_{\mathrm{CG}}, i_{\mathrm{Lanczos}})$ iterations of CG and Lanczos, where

$$i_{\mathrm{CG}} \geq \frac{1}{2}\sqrt{\kappa}\log(2K_3\varepsilon^{-1}), \tag{C.14}$$

$$i_{\mathrm{Lanczos}} \geq \frac{\sqrt{3}}{4}\sqrt{\kappa}\log\left(K_1\varepsilon^{-1}\right), \tag{C.15}$$

then it holds that

$$\boxed{\mathrm{P}(|\eta - \mathcal{L}| \leq \varepsilon(1 + \|\log(\hat{\boldsymbol{K}})\|_F)) \geq 1 - \delta.}$$

Proof. We begin with the error of CG, it holds by Theorem 4.4 in combination with Lemma C.7, that

$$\begin{aligned}
\varepsilon_{\mathrm{CG}} &\leq K_3\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \\
&\leq K_3\exp(-\frac{2i}{\sqrt{\kappa}}) && \text{Lemma C.7} \\
&\leq K_3\exp(-\log(2K_3\varepsilon^{-1})) && \text{Assumption (C.14)} \\
&= \frac{\varepsilon}{2}.
\end{aligned}$$

Now for the error of the estimate of the log-determinant. By assumption Theorem 4.1 is satisfied and therefore $\varepsilon_{\mathrm{STE}} = \varepsilon$ with probability $1 - \delta$. For the error of Lanczos, it holds by Theorem 4.4, that

$$\begin{aligned}
\varepsilon_{\mathrm{Lanczos}} &\leq K_1\left(\frac{\sqrt{2\kappa+1}-1}{\sqrt{2\kappa+1}+1}\right)^{2i} \\
&\leq K_1\exp(-\frac{4}{\sqrt{2\kappa+1}}i) && \text{Lemma C.7} \\
&\leq K_1\exp(-\frac{\sqrt{3\kappa}}{\sqrt{2\kappa+1}}\log(K_1\varepsilon^{-1})) && \text{Assumption (C.15).} \\
&\leq K_1\exp(-\log(K_1\varepsilon^{-1})) \\
&= \varepsilon
\end{aligned}$$

The result now follows by Theorem 4.4. □

C.4.2 Approximation of the Derivative of the Log-Marginal Likelihood

Theorem 4.5 (Error Bound for the Derivative)
Assume the conditions of Theorem 4.3 hold and we solve $\hat{\boldsymbol{K}}\boldsymbol{v}_* = \boldsymbol{y}$ via preconditioned CG initialized at $\mathbf{0}$ or better and terminated after i iterations. Then with probability $1-\delta$, the error in the estimate $\phi = \frac{1}{2}(\boldsymbol{v}_i^\intercal \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{v}_i - \tau_*^{\text{inv}\partial})$ of the derivative of the log*-marginal likelihood $\frac{\partial}{\partial\theta}\mathcal{L}$ satisfies*

$$|\phi - \tfrac{\partial}{\partial\theta}\mathcal{L}| \leq \varepsilon_{\text{CG}} + \tfrac{1}{2}(\varepsilon_{\text{CG}'} + \varepsilon_{\text{STE}})\|\hat{\boldsymbol{K}}^{-1}\tfrac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\|_F,$$

where the individual errors are bounded by

$$\varepsilon_{\text{CG}}(\kappa, i) \leq K_4 \left(\tfrac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \tag{4.23}$$

$$\varepsilon_{\text{CG}'}(\kappa, i) \leq K_2 \left(\tfrac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \tag{4.24}$$

$$\varepsilon_{\text{STE}}(\delta, \ell) \leq C_1\sqrt{\log(\delta^{-1})}\ell^{-\frac{1}{2}}g(\ell) \tag{4.25}$$

for $K_4 = 6\kappa(\hat{\boldsymbol{K}})\max(\|\boldsymbol{v}_*\|_2, \|\boldsymbol{v}_*\|_2^3)\|\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\|_2$.

Proof. It holds that

$$\begin{aligned}
|\phi - \frac{\partial}{\partial\theta}\mathcal{L}| &= \frac{1}{2}|\boldsymbol{v}_i^\intercal \frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\boldsymbol{v}_i - \tau_*^{\text{inv}\partial} - \left(\boldsymbol{y}^\intercal \hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\hat{\boldsymbol{K}}^{-1}\boldsymbol{y} - \text{tr}(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta})\right)| \\
&\leq \Bigg(\underbrace{|\boldsymbol{v}_i^\intercal \frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\boldsymbol{v}_i - \boldsymbol{v}_*{}^\intercal \frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\boldsymbol{v}_*|}_{e_{\text{CG}}} + \underbrace{|\tau_*^{\text{inv}\partial} - \text{tr}(\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta})|}_{e_{\text{CGSTE}}}\Bigg)
\end{aligned}$$

Now by Theorem 4.3, we have

$$e_{\text{CGSTE}} \leq (\varepsilon_{\text{CG}'} + \varepsilon_{\text{STE}})\|\hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\|_F.$$

For the absolute error of the quadratic term, it holds that

$$\begin{aligned}
e_{\text{CG}} &= |\|\boldsymbol{v}_*\|^2_{\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}} - \|\boldsymbol{v}_i - \boldsymbol{v}_* + \boldsymbol{v}_*\|^2_{\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}}| \\
&\leq |\|\boldsymbol{v}_*\|^2_{\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}} - (\|\boldsymbol{v}_i - \boldsymbol{v}_*\|_{\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}} + \|\boldsymbol{v}_*\|_{\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}})^2| \\
&= \|\boldsymbol{v}_i - \boldsymbol{v}_*\|_{\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}} + 2\|\boldsymbol{v}_i - \boldsymbol{v}_*\|_{\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}}\|\boldsymbol{v}_*\|_{\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}} \\
&\leq \|\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\|_2(\|\boldsymbol{v}_i - \boldsymbol{v}_*\|_2^2 + 2\|\boldsymbol{v}_i - \boldsymbol{v}_*\|_2\|\boldsymbol{v}_*\|_2) \\
&\leq \|\frac{\partial \hat{\boldsymbol{K}}}{\partial\theta}\|_2 g(\|\boldsymbol{v}_i - \boldsymbol{v}_*\|_2)(1 + 2\|\boldsymbol{v}_*\|_2)
\end{aligned}$$

for $g(t) = \max(t, t^2)$. Now it holds by Theorem C.1 and monotoncity of g, that

$$\leq \|\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_2 g\left(2\sqrt{\kappa(\hat{\boldsymbol{K}})}\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|\boldsymbol{v}_0 - \boldsymbol{v}_*\|_2\right)(1 + 2\|\boldsymbol{v}_*\|_2)$$

Since for $a \leq 1$, it holds that $g(at) \leq ag(t)$ and for $a > 1 : g(at) \leq a^2 g(t)$, we have

$$\begin{aligned}
&\leq \|\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_2 4\kappa(\hat{\boldsymbol{K}})\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i g(\|\boldsymbol{v}_0 - \boldsymbol{v}_*\|_2)(1 + 2\|\boldsymbol{v}_*\|_2) \\
&\leq \|\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\|_2 4\kappa(\hat{\boldsymbol{K}})\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i 3\max(\|\boldsymbol{v}_*\|_2, \|\boldsymbol{v}_*\|_2^3) \\
&= 2K_4\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i
\end{aligned}$$

where we used that CG was initialized at $\boldsymbol{v}_0 = \boldsymbol{0}$ or better. □

Corollary C.5
Assume the conditions of Theorem 4.5 hold. If the number of random vectors ℓ satisfies (4.11) with $\varepsilon_{\text{STE}} = \varepsilon$, and we run $i \geq \max(i_{\text{CG}}, i_{\text{CG}'})$ iterations of CG, where

$$i_{\text{CG}} \geq \frac{1}{2}\sqrt{\kappa}\log(2K_4\varepsilon^{-1}), \tag{C.16}$$

$$i_{\text{CG}'} \geq \frac{1}{2}\sqrt{\kappa}\log(K_2\varepsilon^{-1}), \tag{C.17}$$

then it holds that

$$\boxed{\mathrm{P}\left(|\phi - \tfrac{\partial}{\partial\theta}\mathcal{L}(\boldsymbol{\theta})| \leq \varepsilon(1 + \|\boldsymbol{K}^{-1}\tfrac{\partial \boldsymbol{K}}{\partial \theta}\|_F)\right) \geq 1 - \delta.}$$

Proof. We begin with the error of CG's estimate of $\boldsymbol{y}^\intercal \hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\hat{\boldsymbol{K}}^{-1}\boldsymbol{y}$, it holds by Theorem 4.5 in combination with Lemma C.7, that

$$\begin{aligned}
\varepsilon_{\text{CG}} &\leq K_4\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \\
&\leq K_4\exp(-\frac{2i}{\sqrt{\kappa}}) && \text{Lemma C.7} \\
&\leq K_4\exp(-\log(2K_4\varepsilon^{-1})) && \text{Assumption (C.16)} \\
&= \frac{\varepsilon}{2}.
\end{aligned}$$

Now for the error of the stochastic trace estimator. By assumption Theorem 4.1 is satisfied and therefore $\varepsilon_{\text{STE}} = \varepsilon$ with probability $1 - \delta$. For the error of CG used in the stochastic trace

estimate, we obtain by Theorem 4.5

$$\begin{aligned}
\varepsilon_{\mathrm{CG}'} &\leq K_2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^i \\
&\leq K_2 \exp(-\frac{2i}{\sqrt{\kappa}}) && \text{Lemma C.7} \\
&\leq K_2 \exp(-\log(K_2 \varepsilon^{-1})) && \text{Assumption (C.17)} \\
&= \varepsilon.
\end{aligned}$$

The result now follows by Theorem 4.5. □

C.5 Preconditioning

Lemma C.4 (Condition Number and Preconditioner Quality)
Let $\hat{\boldsymbol{K}}, \hat{\boldsymbol{P}}_\ell \in \mathbb{R}^{n \times n}$ symmetric positive-definite such that (4.6) holds and assume that there exists $c > 0$ such that $c \geq \max(\lambda_{\min}(\hat{\boldsymbol{P}}), \lambda_{\min}(\hat{\boldsymbol{K}}))$ for all ℓ. Then it holds that

$$\kappa = \kappa(\hat{\boldsymbol{P}}_\ell^{-\frac{1}{2}} \hat{\boldsymbol{K}} \hat{\boldsymbol{P}}_\ell^{-\frac{1}{2}}) \leq (1 + \mathcal{O}(g(\ell)) \|\hat{\boldsymbol{K}}\|_F)^2 \tag{C.18}$$

Proof. Part of the strategy for this proof is adapted from Gardner et al. [3]. First note, that the matrices $\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}$, $\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-1}$ and $\hat{\boldsymbol{P}}_\ell^{-\frac{1}{2}} \hat{\boldsymbol{K}} \hat{\boldsymbol{P}}_\ell^{-\frac{1}{2}}$ are similar, and thus have the same eigenvalues. Now, we have:

$$\begin{aligned}
\kappa(\hat{\boldsymbol{P}}_\ell^{-\frac{1}{2}} \hat{\boldsymbol{K}} \hat{\boldsymbol{P}}_\ell^{-\frac{1}{2}}) &= \frac{\lambda_{\max}(\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}})}{\lambda_{\min}(\hat{\boldsymbol{K}}\hat{\boldsymbol{P}}^{-1})} \\
&= \|\hat{\boldsymbol{P}}^{-1}\hat{\boldsymbol{K}}\|_2 \|\hat{\boldsymbol{P}}\hat{\boldsymbol{K}}^{-1}\|_2 \\
&= \|\hat{\boldsymbol{P}}^{-1}(\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}} + \hat{\boldsymbol{P}})\|_2 \|(\hat{\boldsymbol{P}} - \hat{\boldsymbol{K}} + \hat{\boldsymbol{K}})\hat{\boldsymbol{K}}^{-1}\|_2 \\
&= \|1 + \hat{\boldsymbol{P}}^{-1}(\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}})\|_2 \|1 - \hat{\boldsymbol{K}}^{-1}(\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}})\|_2
\end{aligned}$$

Applying Cauchy-Schwarz and the triangle inequality, we obtain:

$$\begin{aligned}
&\leq (1 + \|\hat{\boldsymbol{P}}^{-1}\|_2 \|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}\|_2)(1 + \|\hat{\boldsymbol{K}}^{-1}\|_2 \|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}\|_2) \\
&\leq (1 + c\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}\|_F)^2 \\
&\leq (1 + \mathcal{O}(g(\ell)) \|\hat{\boldsymbol{K}}\|_F)^2
\end{aligned}$$

□

Note since typically $\hat{\boldsymbol{P}} = \sigma^2 \boldsymbol{I} + \boldsymbol{P}$ with $\lambda_{\min}(\boldsymbol{P}) \approx 0$ and $\lambda_{\min}(\hat{\boldsymbol{K}}) \leq \sigma^2 + \lambda_{\min}(\boldsymbol{K})$, where $\lambda_{\min}(\boldsymbol{K})$ small for most kernels, usually $c \approx \sigma^2$.

C.5.1 Additive Kernels

Lemma C.5 (Additive Kernels)
Let $k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=1}^{d} k_j(\boldsymbol{x}, \boldsymbol{y})$ be an additive kernel and $\{(\hat{\boldsymbol{P}}_\ell)_j\}_{j=1}^{d}$ a set of preconditioners indexed by ℓ, such that for all $j = 1, \ldots, d$, we have

$$\|\hat{\boldsymbol{K}}_j - (\hat{\boldsymbol{P}}_\ell)_j\|_F \leq c_j g(\ell) \|\hat{\boldsymbol{K}}_j\|_F. \tag{C.19}$$

Then it holds for $\hat{\boldsymbol{P}}_\ell = \sum_{j=1}^{d} (\hat{\boldsymbol{P}}_\ell)_j$ and $c = \max_j c_j$ that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c d g(\ell) \|\hat{\boldsymbol{K}}\|_F \tag{C.20}$$

Proof. It holds by assumption, that

$$\begin{aligned}
\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F &\leq \sum_{j=1}^{d} \|\hat{\boldsymbol{K}}_j - (\hat{\boldsymbol{P}}_\ell)_j\|_F && \text{Cauchy-Schwarz} \\
&\leq \sum_{j=1}^{d} c_j g(\ell) \|\hat{\boldsymbol{K}}_j\|_F \\
&\leq \max_j c_j g(\ell) \sum_{j=1}^{d} \sqrt{\sum_{j=1}^{n} \lambda_i(\hat{\boldsymbol{K}}_j)^2} \\
&\leq c g(\ell) \sum_{j=1}^{d} \sqrt{\sum_{j=1}^{n} \lambda_i(\hat{\boldsymbol{K}})^2} && \boldsymbol{A}, \boldsymbol{B} \text{ spd} \implies \lambda_i(\boldsymbol{A}) \leq \lambda_i(\boldsymbol{A} + \boldsymbol{B}) \\
&\leq c g(\ell) d \|\hat{\boldsymbol{K}}\|_F
\end{aligned}$$

□

C.5.2 Kernels with a Uniformly Converging Approximation

Lemma C.6 (Preconditioner Quality from Uniform Convergence)
Let $\Omega \subset \mathbb{R}^d$ be the data domain and $k(\cdot, \cdot)$ a positive-definite kernel such that for all $\boldsymbol{x} \in \Omega$ it holds that $k(\boldsymbol{x}, \boldsymbol{x}) \leq o^2$. Let $P_\ell(\cdot, \cdot)$ be a kernel approximation, such that a uniform convergence bound of the form

$$\sup_{\boldsymbol{x}, \boldsymbol{y} \in \Omega} |k(\boldsymbol{x}, \boldsymbol{y}) - P_\ell(\boldsymbol{x}, \boldsymbol{y})| \leq g(\ell) c_{unif}(d, \Omega, k) \tag{C.21}$$

holds. Then it holds for the preconditioner $\hat{\boldsymbol{P}}_\ell = \sigma^2 \boldsymbol{I} + P_\ell(\boldsymbol{X}, \boldsymbol{X})$, that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq g(\ell) c(d, \Omega, k, n) \|\hat{\boldsymbol{K}}\|_F, \tag{C.22}$$

where $c(d, \Omega, k, n) = \frac{\sqrt{n}}{o^2} c_{unif}(d, \Omega, k)$.

Proof. It holds that

$$
\begin{aligned}
\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F = \|\boldsymbol{K} - \boldsymbol{P}_\ell\|_F &= \sqrt{\sum_{i,j=1}^{n} (k(\boldsymbol{x}_i, \boldsymbol{x}_j) - P_\ell(\boldsymbol{x}_i, \boldsymbol{x}_j))^2} \\
&\leq \sqrt{\sum_{i,j=1}^{n} (g(\ell) c_{\text{unif}}(d, \Omega, k))^2} && \text{uniform convergence bound (C.21)} \\
&= n g(\ell) c_{\text{unif}}(d, \Omega, k) \\
&= g(\ell)\sqrt{n}\left(\sum_{i=1}^{n} \frac{k(\boldsymbol{x}_i, \boldsymbol{x}_i)^2}{k(\boldsymbol{x}_i, \boldsymbol{x}_i)^2}\right)^{\frac{1}{2}} && k \text{ bounded} \\
&\leq g(\ell)\frac{\sqrt{n}}{o^2}\left(\sum_{i=1}^{n} \boldsymbol{K}_{ij}^2\right)^{\frac{1}{2}} \\
&= g(\ell) c(d, \Omega, k, n) \|\boldsymbol{K}\|_F \\
&\leq g(\ell) c(d, \Omega, k, n) \|\hat{\boldsymbol{K}}\|_F
\end{aligned}
$$

□

C.5.3 Partial Cholesky Decomposition

Proposition C.2 (Partial Cholesky Approximation Quality)
Let $k(\boldsymbol{x}, \boldsymbol{y}) = k(\|\boldsymbol{x} - \boldsymbol{y}\|)$ *be a stationary kernel with output scale* $o^2 = k(0) > 0$. *Assume the kernel matrix spectrum decays at least exponentially, i.e.* $\lambda_j(\boldsymbol{K}) \leq c \exp(-bj)$ *for* $c > 0$ *and* $b > \log(4)$. *Then the Cholesky preconditioner* $\hat{\boldsymbol{P}}_\ell = \sigma^2 \boldsymbol{I} + \boldsymbol{L}_\ell \boldsymbol{L}_\ell^\intercal$ *satisfies*

$$
\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq \frac{c\sqrt{n}}{o^2} \exp(-b'\ell) \|\boldsymbol{K}\|_F \tag{C.23}
$$

where $b' = b - \log(4) > 0$.

Proof. Since $k(\cdot, \cdot)$ is a positive definite kernel, the choice $o^2 = k(0)$ is no restriction. Now, it holds that

$$
\begin{aligned}
& \lambda_j \leq c \exp(-bj) = c \exp(-(b' + \log(4))j) = c \exp(-b'j) 4^{-j} \\
\iff\ & 4^j \lambda_j \leq c \exp(-b'j).
\end{aligned}
$$

Therefore by Theorem 3.2 of Harbrecht, Peters, and Schneider [172], we have $\operatorname{tr}(\boldsymbol{K} - \boldsymbol{L}_\ell \boldsymbol{L}_\ell^\intercal) \leq cn \exp(-b'\ell)$. Now it holds since $\boldsymbol{K} - \boldsymbol{L}_\ell \boldsymbol{L}_\ell^\intercal$ positive definite, that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F = \|\boldsymbol{K} - \boldsymbol{L}_\ell \boldsymbol{L}_\ell^\intercal\|_F \leq \operatorname{tr}(\boldsymbol{K} - \boldsymbol{L}_\ell \boldsymbol{L}_\ell^\intercal) \leq cn \exp(-b'\ell),$$

and with $\boldsymbol{K}_{jj} = o^2$ that

$$n = \sqrt{n}\left(\sum_{j=1}^{n} \frac{\boldsymbol{K}_{jj}^2}{o^4}\right)^{\frac{1}{2}} \leq \frac{\sqrt{n}}{o^2}\left(\sum_{i,j=1}^{n} \boldsymbol{K}_{ij}^2\right)^{\frac{1}{2}} = \frac{\sqrt{n}}{o^2}\|\boldsymbol{K}\|_F.$$

This concludes the argument. □

C.5.4 Quadrature Fourier Features (QFF)

Proposition C.3 (QFF Approximation Quality)
Assume $\Omega = [0,1]^d$, k a kernel with Fourier transform $p(\omega) = \exp(-\frac{1}{2}\sum_{j=1}^{d} \omega_j^2 \gamma_j^2)$ such that Assumption 1 of Mutnỳ and Krause [149] is satisfied and let $\hat{\boldsymbol{P}}_\ell = \sigma^2 \boldsymbol{I} + \boldsymbol{P}_\ell$, where $\boldsymbol{P}_\ell$ is the QFF approximated kernel matrix. Let $\ell^{\frac{1}{d}} > \frac{2}{\gamma^2}$, then for $b = \frac{1}{2}(\log(4) - 1)$, it holds that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c(d,n,k) \exp(-b\ell^{\frac{1}{d}})\|\hat{\boldsymbol{K}}\|_F. \tag{C.24}$$

Proof. By Theorem 1 of Mutnỳ and Krause [149], replacing $\ell = (2\bar{m})^d$ it holds that

$$\begin{aligned}
\sup_{\boldsymbol{x},\boldsymbol{y}\in\Omega} |k(\boldsymbol{x},\boldsymbol{y}) - P_\ell(\boldsymbol{x},\boldsymbol{y})| &\leq d2^{d-1}\sqrt{\frac{\pi}{2}}\left(\frac{e}{2\gamma^2 \ell^{\frac{1}{d}}}\right)^{\frac{1}{2}\ell^{\frac{1}{d}}} \\
&\leq c(d)\left(\frac{e}{4}\right)^{\frac{1}{2}\ell^{\frac{1}{d}}} && \ell^{\frac{1}{d}} > 2\gamma^{-2} \\
&= c(d)\exp(\frac{1}{2}\ell^{\frac{1}{d}} - \log(4)\frac{1}{2}\ell^{\frac{1}{d}}) \\
&= c(d)\exp(-\frac{1}{2}(\log(4)-1)\ell^{\frac{1}{d}}) \\
&= c(d)\exp(-b\ell^{\frac{1}{d}})
\end{aligned}$$

Now by Lemma C.6, we have

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c(d,n)\exp(-b\ell^{\frac{1}{d}})\|\hat{\boldsymbol{K}}\|_F, \tag{C.25}$$

where $c(d,n) = \sqrt{n}c(d)$ by Assumption 1 of Mutnỳ and Krause [149], which assumes $k(\boldsymbol{x},\boldsymbol{y}) \leq 1$. □

Proposition C.4 (General QFF Approximation Quality)
Assume $\Omega = [0,1]^d$, k a kernel with Fourier transform $p(\omega)$ such that Assumption 1 of Mutnỳ and Krause [149] is satisfied and $f_\delta(\phi) = p(\cot(\phi))\frac{\cos(\delta\cot(\phi))}{\sin(\phi)^2}$ is $(s-1)$-times absolutely continuous. Let $\hat{\boldsymbol{P}}_\ell = \sigma^2\boldsymbol{I} + \boldsymbol{P}_\ell$, where $\boldsymbol{P}_\ell$ is the QFF approximated kernel matrix. Then it holds that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c(d,\Omega,k,n)\ell^{-\frac{s+1}{d}}\|\hat{\boldsymbol{K}}\|_F. \tag{C.26}$$

Proof. By Theorem 4 of [149], replacing $\ell = (2\bar{m})^d$, it holds that

$$\begin{aligned}\sup_{\boldsymbol{x},\boldsymbol{y}\in\Omega}|k(\boldsymbol{x},\boldsymbol{y}) - P_\ell(\boldsymbol{x},\boldsymbol{y})| &\leq d2^{d-1}\frac{(s+2)^{s+1}}{s!}\max_{\delta\in\Omega}\mathrm{TV}(f_\delta^{(s)})2^{s+1}\ell^{-\frac{s+1}{d}} \\ &= c(d,\Omega,k)\ell^{-\frac{s+1}{d}}\end{aligned}$$

Now by Lemma C.6, we have

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c(d,\Omega,k,n)\ell^{-\frac{s+1}{d}}\|\hat{\boldsymbol{K}}\|_F.$$

□

Remark C.2 (Modified Matérn Kernel)
Proposition C.4 is satisfied for example for the modified Matérn(ν) kernel, defined via its spectral density

$$p(\omega) = \prod_{j=1}^{d}\frac{1}{(1+\gamma_j^2\omega_j^2)^{\nu+\frac{1}{2}}}. \tag{C.27}$$

See the appendix of Mutnỳ and Krause [149] for a detailed definition and motivation.

C.5.5 Truncated Singular Value Decomposition

Proposition C.5 (Truncated SVD Approximation Quality)
Let $\boldsymbol{K}$ be a kernel matrix and $\boldsymbol{P}_\ell = \boldsymbol{V}_\ell\boldsymbol{\Lambda}_\ell\boldsymbol{V}_\ell$ its truncated singular value decomposition consisting of the eigenvectors of the largest ℓ eigenvalues. Then it holds for $\hat{\boldsymbol{P}} = \sigma^2\boldsymbol{I} + \boldsymbol{P}_\ell$, that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c(n)\ell^{-\frac{1}{2}}\|\hat{\boldsymbol{K}}\|_F. \tag{C.28}$$

where $c(n) = \sqrt{n}$.

Proof. Since the optimal rank-ℓ approximation in Frobenius norm is given by the truncated SVD [173], we have for $\lambda_1(\boldsymbol{K}) \geq \cdots \geq \lambda_n(\boldsymbol{K})$, that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F^2 = \|\boldsymbol{K} - \boldsymbol{P}_\ell\|_F^2 = \sum_{j=\ell+1}^{n} \lambda_j(\boldsymbol{K})^2$$

Now since $\lambda_{\ell+1}(\boldsymbol{K}) \leq \frac{1}{\ell}\sum_{j=1}^{\ell} \lambda_j(\boldsymbol{K}) \leq \frac{1}{\ell}\operatorname{tr}(\boldsymbol{K})$, we have

$$\begin{aligned} &\leq \lambda_{\ell+1}(\boldsymbol{K}) \sum_{j=\ell+1}^{n} \lambda_j(\boldsymbol{K}) \leq \frac{1}{\ell}\operatorname{tr}(\boldsymbol{K})^2 \\ &= \frac{1}{\ell}\|\boldsymbol{\lambda}\|_1^2 \leq \frac{n}{\ell}\|\boldsymbol{\lambda}\|_2^2 = \frac{n}{\ell}\|\boldsymbol{K}\|_F^2 \leq \frac{n}{\ell}\|\hat{\boldsymbol{K}}\|_F^2 \end{aligned}$$

where $\boldsymbol{\lambda} = (\lambda_1(\boldsymbol{K}), \ldots, \lambda_n(\boldsymbol{K}))^\intercal \in \mathbb{R}^n$. □

C.5.6 Randomized Singular Value Decomposition

Proposition C.6 (Randomized SVD Approximation Quality)
Let $\boldsymbol{K}$ be a kernel matrix and $\boldsymbol{P}_\ell = \boldsymbol{H}_\ell \boldsymbol{H}_\ell^\intercal \boldsymbol{K}$ its randomized singular value decomposition constructed via the LINEARTIMESVD *algorithm [174] with $s \in \mathbb{N}$ samples drawn according to probabilities $\{p_j\}_{j=1}^n$. Then for $\hat{\boldsymbol{P}} = \sigma^2 \boldsymbol{I} + \boldsymbol{P}_\ell$, it holds with probability $1-\delta$, that*

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c(n)(\ell^{-\frac{1}{2}} + \mathcal{O}(\ell^{\frac{1}{4}} s^{-\frac{1}{4}}))\|\hat{\boldsymbol{K}}\|_F, \tag{C.29}$$

where $c(n) = \sqrt{n}$.

Proof. By Drineas, Kannan, and Mahoney [174, Theorem 4] it holds with probability $1-\delta$, that

$$\begin{aligned} \|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F^2 &= \|\boldsymbol{K} - \boldsymbol{P}_\ell\|_F^2 \\ &\leq \|\boldsymbol{K} - \boldsymbol{K}_\ell\|_F^2 + c(p_j, \delta)\ell^{\frac{1}{2}} s^{-\frac{1}{2}} \|\boldsymbol{K}\|_F^2 \end{aligned}$$

By the same argument as in the proof of Proposition C.5 for the error of the optimal rank-ℓ approximation $\|\boldsymbol{K} - \boldsymbol{K}_\ell\|_F^2$, we obtain

$$\begin{aligned} &\leq \frac{n}{\ell}\|\boldsymbol{K}\|_F^2 + c(p_j, \delta)\ell^{\frac{1}{2}} s^{-\frac{1}{2}} \|\boldsymbol{K}\|_F^2 \\ &\leq n(\ell^{-1} + \mathcal{O}(\ell^{\frac{1}{2}} s^{-\frac{1}{2}}))\|\hat{\boldsymbol{K}}\|_F^2 \end{aligned}$$

This completes the proof. □

C.5.7 Randomized Nyström Method

Proposition C.7 (Randomized Nyström Approximation Quality)
Let $\boldsymbol{K}$ be a kernel matrix and $\boldsymbol{P}_\ell = \boldsymbol{C}\boldsymbol{W}_\ell^{+}\boldsymbol{C}^\intercal$ its randomized Nyström approximation constructed via Algorithm 3 of Drineas and Mahoney [104] with $s \in \mathbb{N}$ columns drawn according to probabilities $p_j = \frac{\boldsymbol{K}_{jj}^2}{\sum_{j=1}^n \boldsymbol{K}_{jj}^2}$. Then for $\hat{\boldsymbol{P}} = \sigma^2\boldsymbol{I} + \boldsymbol{P}_\ell$, it holds with probability $1-\delta$, that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c(n)(\ell^{-\frac{1}{2}} + \mathcal{O}(\ell^{\frac{1}{4}} s^{-\frac{1}{4}}))\|\hat{\boldsymbol{K}}\|_F, \tag{C.30}$$

where $c(n) = \sqrt{n}$.

Proof. By Drineas and Mahoney [104, Theorem 3] it holds with probability $1-\delta$, that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F = \|\boldsymbol{K} - \boldsymbol{P}_\ell\|_F \leq \|\boldsymbol{K} - \boldsymbol{K}_\ell\|_F + c(\delta)\ell^{\frac{1}{4}} s^{-\frac{1}{4}} \sum_{j=1}^n \boldsymbol{K}_{jj}^2$$

By the same argument as in the proof of Proposition C.5 for the error of the optimal rank-ℓ approximation $\|\boldsymbol{K} - \boldsymbol{K}_\ell\|_F$, we obtain

$$\begin{aligned}
&\leq n^{\frac{1}{2}}\ell^{-\frac{1}{2}}\|\boldsymbol{K}\|_F^2 + c(\delta)\ell^{\frac{1}{4}} s^{-\frac{1}{4}} \sum_{j=1}^n \boldsymbol{K}_{jj}^2 \\
&\leq n^{\frac{1}{2}}\ell^{-\frac{1}{2}}\|\boldsymbol{K}\|_F^2 + c(\delta)\ell^{\frac{1}{4}} s^{-\frac{1}{4}} \sqrt{\sum_{j=1}^n \boldsymbol{K}_{jj}^2}\|\boldsymbol{K}\|_F \\
&\leq n^{\frac{1}{2}}(\ell^{-\frac{1}{2}} + \mathcal{O}(\ell^{\frac{1}{4}} s^{-\frac{1}{4}}))\|\hat{\boldsymbol{K}}\|_F
\end{aligned}$$

This completes the proof. □

C.5.8 Random Fourier Features (RFF)

Proposition C.8 (RFF Approximation Quality)
Let $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ be a positive-definite kernel with compact data domain $\Omega \subset \mathbb{R}^d$. Let $\boldsymbol{P}_\ell = \boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^\intercal$ be the random Fourier feature approximation [91], where $\boldsymbol{Z}_\ell \in \mathbb{R}^{n\times\ell}$. Then for $\hat{\boldsymbol{P}}_\ell = \sigma^2\boldsymbol{I} + \boldsymbol{P}_\ell$ it holds with probability $1-\delta$, that

$$\|\hat{\boldsymbol{K}} - \hat{\boldsymbol{P}}_\ell\|_F \leq c(d, \Omega, k, \delta, n)\ell^{-\frac{1}{2}}\|\hat{\boldsymbol{K}}\|_F. \tag{C.31}$$

Proof. By Theorem 1 of Sriperumbudur and Szabó [175] with probability $1-\delta$, we obtain the uniform convergence bound

$$\sup_{\boldsymbol{x},\boldsymbol{y}\in\Omega} |k(\boldsymbol{x}, \boldsymbol{y}) - P_\ell(\boldsymbol{x}, \boldsymbol{y})| \leq c_{\text{unif}}(d, \Omega, k, \delta)\ell^{-\frac{1}{2}}. \tag{C.32}$$

Now, applying Lemma C.6 completes the proof. □

C.6 Technical Results

Lemma C.7
Let $x \in \mathbb{R}$ such that $x > 1$, then it holds that

$$\frac{x-1}{x+1} \leq \exp(-\frac{2}{x}). \tag{C.33}$$

Proof. By Mitrinovic and Vasic [176, Section 3.6.18] it holds for $y > 0$, that

$$\frac{\log(y+1)}{y} \geq \frac{2}{2+y}$$

Substituting $y = \frac{x+1}{x-1} - 1$ we obtain

$$\log(\frac{x+1}{x-1}) \geq \frac{2(\frac{x+1}{x-1} - 1)}{1 + \frac{x+1}{x-1}} = \frac{2(x+1-x+1)}{2x} = \frac{2}{x} \iff \frac{x+1}{x-1} \geq \exp(\frac{2}{x})$$

This proves the claim. □

C.7 Additional Experimental Results

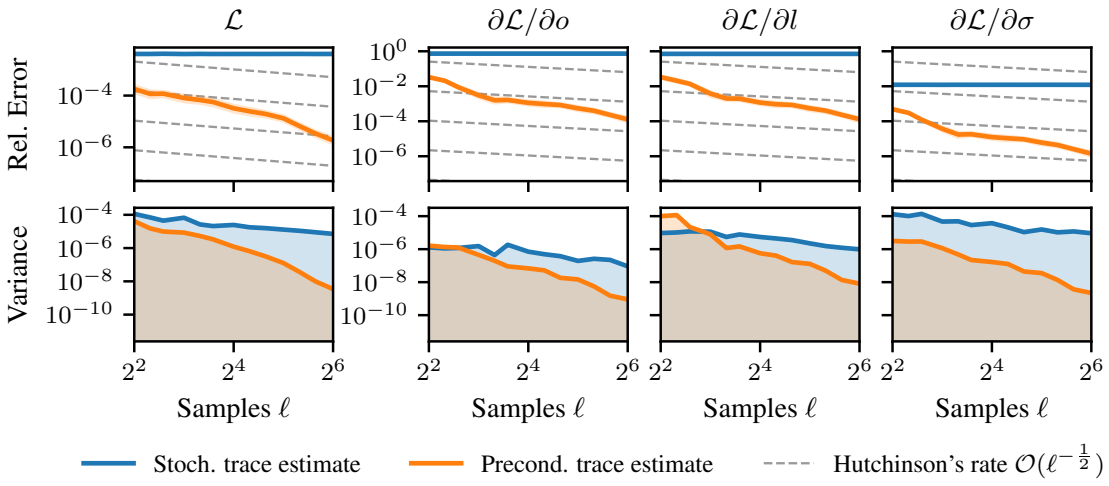
C.7.1 Synthetic Data

We report bias and variance of the stochastic estimators for the log-maginal likelihood and its derivatives for the exponentiated quadratic, Matérn($\frac{3}{2}$) and rational quadratic kernel on a synthetic dataset of size $n = 1{,}000$ with varying dimensionality $d \in \{1, 2, 3\}$ in Table C.1. Using $\ell = 128$ random samples with a preconditioner of the same size and $i = 25$ Lanczos / CG iterations, bias and variance are reduced by several orders of magnitude across different kernels. Note, that the variance reduction tends to decline with dimensionality, even though this is not necessarily universal across kernels. We show the bias and variance reduction for increasing number of random samples, respectively preconditioner quality in Figure C.1.

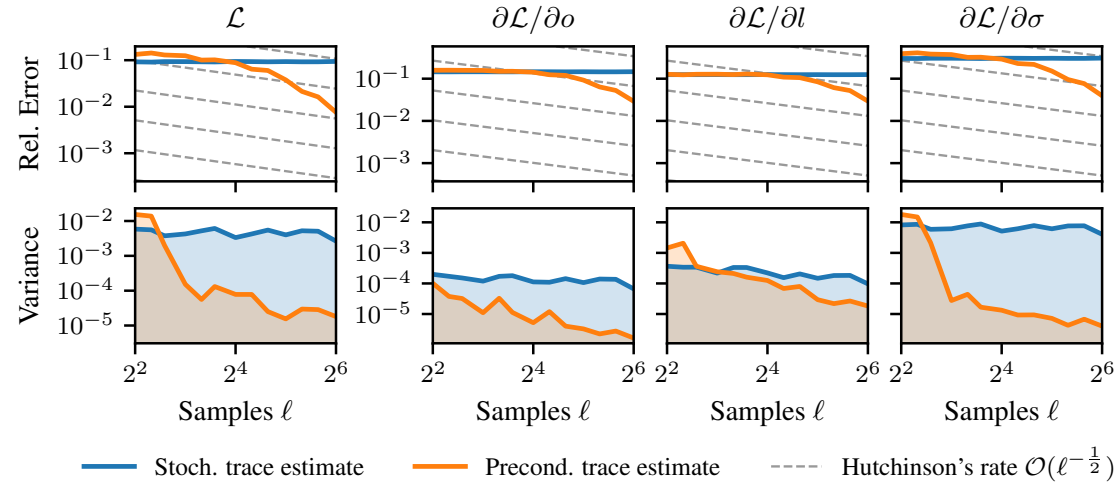
C.7.2 UCI Datasets

For the experiments we conducted on UCI datasets, we report the full experimental results with their deviation across 10 runs in Table C.2. Test errors with and without preconditioning did not differ by more than two standard deviations. However, model evaluations of the optimizer were significantly reduced when using a preconditioner of size 500, leading to substantial speedup. Note, that the experiment on the "3DRoad" dataset was only carried out once due to the prohibitive runtime without preconditioning.

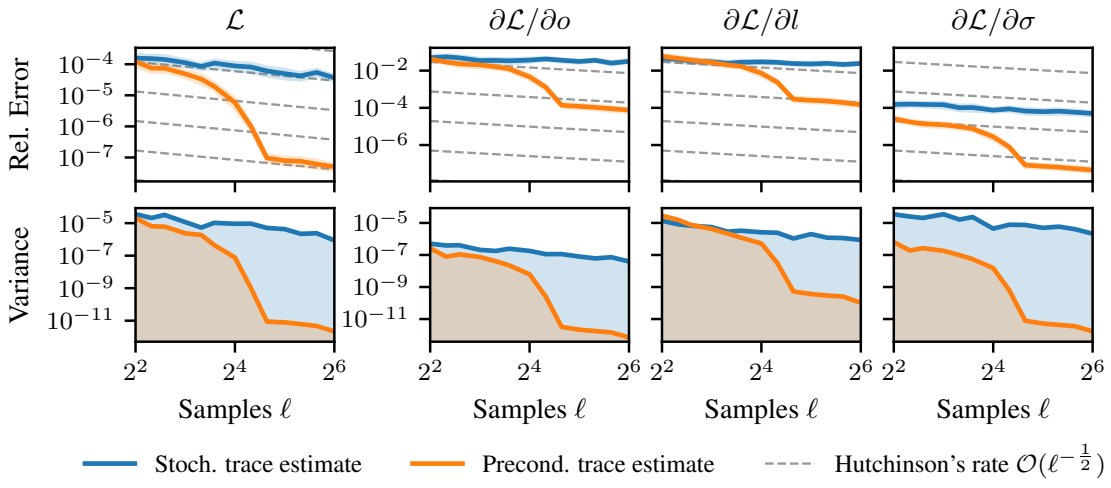
Optimizer Choice We used the L-BFGS optimizer in our experiments due to its favorable convergence properties. As an ablation experiment we compared to the Adam optimizer as sometimes used for its robustness to noise, when using stochastic approximations of the log-marginal likelihood [3, 113]. We find that with preconditioning optimization with L-BFGS significantly outperformed optimization with Adam, both in terms of training and test error, except for the "KEGGdir" dataset (cf. Table 4.2 and Table C.3). Additionally, L-BFGS converged faster across all experiments. This shows that variance reduction via preconditioning makes the use of second-order optimizers not only possible, but preferred for GP hyperparameter optimization when using stochastic approximations.



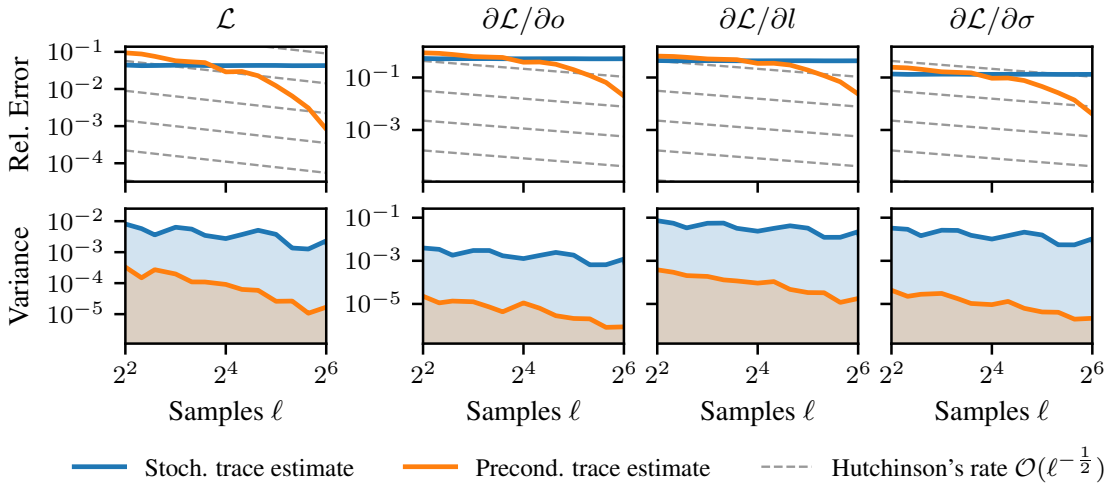
(a) Matérn(3/2) ($d = 1$).



(b) Matérn(3/2) ($d = 3$).



(c) RatQuad ($d = 1$).



(d) RatQuad ($d = 3$).

Figure C.1: *Bias and variance on synthetic datasets.* Relative error and variance of the stochastic estimators of the log-marginal likelihood and its derivative for increasing number of random vectors and preconditioner quality. Experiments were performed for different kernels on a synthetic dataset of size $n = 1{,}000$ with dimension $d \in \{1, 2, 3\}$. Plots show mean and 95% confidence intervals for the relative error computed over 25 repetitions.

Table C.1: *Bias and variance reduction for different kernels.* Bias and variance of the stochastic estimators for the log-marginal likelihood and its derivative(s) computed for synthetic data ($n = 1{,}000$, $\sigma^2 = 10^{-2}$) with 25 repetitions.

			$\mathcal{L}$		$\partial\mathcal{L}/\partial o$		$\partial\mathcal{L}/\partial l$		$\partial\mathcal{L}/\partial\sigma$	
Kernel	d	Prec.	Bias	Var.	Bias	Var.	Bias	Var.	Bias	Var.
Matérn(3/2)	1	0	2×10^{-1}	3×10^{-6}	2×10^{-1}	7×10^{-8}	7×10^{-1}	5×10^{-7}	5×10^{-1}	3×10^{-6}
		128	5×10^{-6}	6×10^{-12}	3×10^{-6}	2×10^{-12}	9×10^{-6}	2×10^{-11}	5×10^{-6}	6×10^{-12}
	2	0	1	4×10^{-3}	1	2×10^{-3}	3	1×10^{-2}	3	2×10^{-2}
		128	8×10^{-4}	3×10^{-7}	3×10^{-4}	4×10^{-8}	9×10^{-4}	3×10^{-7}	5×10^{-4}	1×10^{-7}
	3	0	1	1×10^{-3}	6×10^{-1}	3×10^{-5}	1	4×10^{-5}	2	1×10^{-3}
		128	5×10^{-3}	5×10^{-6}	2×10^{-3}	1×10^{-6}	2×10^{-2}	8×10^{-6}	9×10^{-3}	2×10^{-6}
RBF	1	0	1×10^{-3}	8×10^{-7}	1×10^{-4}	6×10^{-9}	8×10^{-4}	4×10^{-7}	2×10^{-3}	1×10^{-6}
		128	1×10^{-6}	5×10^{-13}	7×10^{-7}	2×10^{-13}	1×10^{-5}	6×10^{-11}	1×10^{-6}	4×10^{-13}
	2	0	5×10^{-1}	5×10^{-6}	4×10^{-1}	2×10^{-7}	4	2×10^{-5}	1	4×10^{-6}
		128	1×10^{-6}	9×10^{-13}	6×10^{-7}	2×10^{-13}	1×10^{-5}	1×10^{-10}	9×10^{-7}	6×10^{-13}
	3	0	1	1×10^{-5}	1	3×10^{-7}	9	3×10^{-5}	3	5×10^{-6}
		128	8×10^{-4}	2×10^{-7}	2×10^{-4}	2×10^{-8}	2×10^{-3}	3×10^{-6}	4×10^{-4}	6×10^{-8}
RatQuad	1	0	2×10^{-3}	9×10^{-7}	6×10^{-4}	3×10^{-8}	2×10^{-3}	7×10^{-7}	1×10^{-3}	1×10^{-6}
		128	2×10^{-6}	1×10^{-12}	9×10^{-7}	4×10^{-13}	1×10^{-5}	6×10^{-11}	1×10^{-6}	9×10^{-13}
	2	0	9×10^{-1}	7×10^{-6}	1	2×10^{-7}	6	5×10^{-6}	2	3×10^{-6}
		128	2×10^{-4}	1×10^{-8}	7×10^{-5}	2×10^{-9}	6×10^{-4}	1×10^{-7}	1×10^{-4}	4×10^{-9}
	3	0	1	2×10^{-2}	1	8×10^{-3}	7	1×10^{-1}	3	7×10^{-2}
		128	2×10^{-3}	2×10^{-6}	4×10^{-4}	1×10^{-7}	3×10^{-3}	4×10^{-6}	7×10^{-4}	3×10^{-7}

Table C.2: *Hyperparameter optimization on UCI datasets.* GP regression using a Matérn$(\frac{3}{2})$ kernel and pivoted Cholesky preconditioner of size 500 with $\ell = 50$ random samples. Hyperparameters were optimized with L-BFGS for at most 20 steps using early stopping via a validation set. All results, but "3DRoad", are averaged over 10 runs.

					Model evals.		Runtime (s)		Speedup		$-\mathcal{L}_{\text{train}} \downarrow$		$-\mathcal{L}_{\text{test}} \downarrow$		RMSE $\downarrow$	
					mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
Dataset	n	d	Prec. Qual.	Opt. Steps												
Elevators	12 449	18	0	19	42.2	7.5	53.0	7.7	1.0	0.0	0.465	0.003	0.402	0.010	0.348	0.005
			500	19	36.4	1.2	39.2	0.8	1.4	0.2	0.438	0.003	0.402	0.010	0.348	0.005
Bike	13 034	17	0	19	32.3	1.3	30.6	1.1	1.0	0.0	−0.998	0.011	−0.993	0.014	0.045	0.004
			500	19	31.4	2.1	37.1	1.5	0.8	0.0	−0.999	0.019	−0.988	0.018	0.045	0.003
Kin40k	30 000	8	0	8	19.0	4.4	186.5	68.0	1.0	0.0	−0.334	0.002	−0.314	0.002	0.093	0.001
			500	6	15.0	0.4	44.6	1.3	2.7	0.1	−0.433	0.006	−0.314	0.004	0.095	0.001
Protein	34 297	9	0	15	124.2	0.4	892.6	19.1	1.0	0.0	0.996	0.003	0.887	0.008	0.572	0.007
			500	7	17.8	0.6	42.5	5.0	4.2	0.4	0.927	0.004	0.884	0.005	0.558	0.008
KEGGdir	36 620	20	0	19	55.8	11.7	1450.3	253.4	1.0	0.0	−0.950	0.009	−0.946	0.034	0.086	0.004
			500	19	42.9	0.7	173.7	2.9	8.4	1.5	−1.004	0.009	−0.949	0.031	0.086	0.004
3DRoad	326 155	3	0	9	68.0		82 200.0		1.0		0.773		1.436		0.298	
			500	9	19.0		7306.0		11.3		0.128		1.169		0.127	

Table C.3: *Hyperparameter optimization using Adam.* GP regression using a Matérn$(\frac{3}{2})$ kernel and pivoted Cholesky preconditioner of size 500 with $\ell = 50$ random samples. Hyperparameters were optimized with Adam for at most 20 steps using early stopping via a validation set.

Dataset	n	d	Prec. Quality	$-\mathcal{L}_{\text{train}} \downarrow$	$-\mathcal{L}_{\text{test}} \downarrow$	RMSE $\downarrow$	Runtime (s)
Elevators	12 449	18	500	0.4803	0.4593	0.3684	109.0000
Bike	13 034	17	500	0.2265	0.3473	0.2300	64.0000
Kin40k	30 000	8	500	0.4392	−0.1200	0.0982	159.0000
Protein	34 297	9	500	0.9438	0.9319	0.5681	92.0000
KEGGdir	36 620	20	500	-1.0070	−1.0390	0.0810	239.0000

Bibliography

[1] J. Wenger and P. Hennig. "Probabilistic Linear Solvers for Machine Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020 (cit. on pp. iii, vi, 21, 23, 28, 48, 49, 87).

[2] J. Wenger, G. Pleiss, M. Pförtner, P. Hennig, and J. P. Cunningham. "Posterior and Computational Uncertainty in Gaussian Processes". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022 (cit. on pp. iii, vi, 22, 23).

[3] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson. "GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2018) (cit. on pp. iii, vi, 21, 23, 42, 47, 48, 53, 55, 58–61, 68–70, 72, 74, 127, 135).

[4] J. Wenger, G. Pleiss, P. Hennig, J. P. Cunningham, and J. R. Gardner. "Preconditioning for Scalable Gaussian Process Hyperparameter Optimization". In: *International Conference on Machine Learning (ICML)*. 2022 (cit. on pp. iii, vi, 23, 42, 47, 48, 53, 74).

[5] C. F. Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. Vol. 7. FA Perthes, 1877 (cit. on p. 2).

[6] D. Teets and K. Whitehead. "The discovery of Ceres: How Gauss became famous". In: *Mathematics Magazine* 72.2 (1999), pp. 83–93 (cit. on p. 2).

[7] W. K. Bühler. *Gauss: A biographical study*. Springer, 2012 (cit. on p. 2).

[8] S. M. Stigler. "Gauss and the Invention of Least Squares". In: *The Annals of Statistics* 9.3 (1981), pp. 465–474. DOI: 10.1214/aos/1176345451 (cit. on p. 2).

[9] S. Legg and M. Hutter. "A collection of definitions of intelligence". In: *Frontiers in Artificial Intelligence and Applications* 17 (2007) (cit. on p. 2).

[10] S. Legg and M. Hutter. "Universal intelligence: A definition of machine intelligence". In: *Minds and machines* 17 (2007), pp. 391–444 (cit. on p. 2).

[11] T. M. Mitchell. *Machine learning*. Vol. 1. McGraw Hill New York, 1997 (cit. on p. 2).

[12] A. L. Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of Research and Development* 44 (1959), pp. 206–226. DOI: 10.1147/rd.441.0206 (cit. on p. 2).

[13] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress". In: *IBM Journal of Research and Development* 11.6 (1967), pp. 601–617. DOI: 10.1147/rd.116.0601 (cit. on p. 2).

[14] P. Raghavan. *How AI Is Powering a More Helpful Google*. 2020. URL: https://blog.google/products/search/search-on/ (cit. on p. 3).

[15] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, et al. "Applied machine learning at facebook: A datacenter infrastructure perspective". In: *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2018, pp. 620–629 (cit. on p. 3).

[16] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 10684–10695 (cit. on p. 3).

[17] OpenAI. *GPT-4 Technical Report*. 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 (cit. on p. 3).

[18] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. *Evaluating Large Language Models Trained on Code*. 2021. DOI: 10.48550/arXiv.2107.03374. arXiv: 2107.03374 (cit. on p. 3).

[19] D. Tabernik, S. Šela, J. Skvarč, and D. Skočaj. "Segmentation-based deep-learning approach for surface-defect detection". In: *Journal of Intelligent Manufacturing* 31.3 (2020), pp. 759–776. DOI: 10.1007/s10845-019-01476-x (cit. on p. 3).

[20] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline. "Machine learning for medical imaging". In: *Radiographics* 37.2 (2017), pp. 505–515 (cit. on p. 3).

[21] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová. "Machine learning and the physical sciences". In: *Reviews of Modern Physics* 91.4 (2019) (cit. on pp. 3, 37).

[22] K. Kashinath, M. Mustafa, A. Albert, J.-L. Wu, C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, R. Wang, A. Chattopadhyay, A. Singh, A. Manepalli, D. Chirila, R. Yu, R. Walters, B. White, H. Xiao, H. A. Tchelepi, P. Marcus, A. Anandkumar, P. Hassanzadeh, and n. Prabhat. "Physics-informed machine learning: case studies for weather and climate modelling". In: vol. 379. 2194. 2021, p. 20200093. DOI: 10.1098/rsta.2020.0093 (cit. on p. 3).

[23] M.-F. Ho, S. Bird, and R. Garnett. "Damped Lyman-α absorbers from Sloan digital sky survey DR16Q with Gaussian processes". In: *Monthly Notices of the Royal Astronomical Society* 507.1 (2021), pp. 704–719. DOI: 10.1093/mnras/stab2169 (cit. on p. 3).

[24] A. C. Miller, L. Anderson, B. Leistedt, J. P. Cunningham, D. W. Hogg, and D. M. Blei. "Mapping interstellar dust with Gaussian processes". In: *The Annals of Applied Statistics* 16.4 (2022), pp. 2672–2692 (cit. on p. 3).

[25] G. Bertone, M. P. Deisenroth, J. S. Kim, S. Liem, R. R. de Austri, and M. Welling. *Accelerating the BSM interpretation of LHC data with machine learning*. 2016. DOI: 10.48550/arXiv.1611.02704. arXiv: 1611.02704 (cit. on p. 3).

[26] P. T. Komiske, E. M. Metodiev, B. Nachman, and M. D. Schwartz. "Pileup Mitigation with Machine Learning (PUMML)". In: *Journal of High Energy Physics* 2017.12 (2017), pp. 1–20 (cit. on pp. 3, 4).

[27] M. Frate, K. Cranmer, S. Kalia, A. Vandenberg-Rodes, and D. Whiteson. *Modeling Smooth Backgrounds and Generic Localized Signals with Gaussian Processes*. 2017. DOI: 10.48550/arXiv.1709.05681. arXiv: 1709.05681 (cit. on p. 3).

[28] M. M. Churchland, J. P. Cunningham, M. T. Kaufman, J. D. Foster, P. Nuyujukian, S. I. Ryu, and K. V. Shenoy. "Neural Population Dynamics during Reaching". In: *Nature* 487.7405 (July 2012), pp. 51–56. ISSN: 1476-4687. DOI: 10.1038/nature11129 (cit. on pp. 3, 4).

[29] J. P. Cunningham and B. M. Yu. "Dimensionality Reduction for Large-Scale Neural Recordings". In: *Nature Neuroscience* 17.11 (2014), pp. 1500–1509. ISSN: 1097-6256, 1546-1726. DOI: 10.1038/nn.3776 (cit. on p. 3).

[30] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. "Highly Accurate Protein Structure Prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2 (cit. on p. 3).

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 25. 2012 (cit. on p. 4).

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. DOI: 10.48550/arXiv.1706.03762 (cit. on p. 4).

[33] N. Jaquier, V. Borovitskiy, A. Smolensky, A. Terenin, T. Asfour, and L. Rozo. "Geometry-Aware Bayesian Optimization in Robotics Using Riemannian Matérn Kernels". In: *Conference on Robot Learning (CoRL)*. 2022 (cit. on p. 4).

[34] A. C. Walls, Y.-J. Park, M. A. Tortorici, A. Wall, A. T. McGuire, and D. Veesler. "Structure, Function, and Antigenicity of the SARS-CoV-2 Spike Glycoprotein". In: *Cell* 181.2 (2020), 281–292.e6. ISSN: 0092-8674, 1097-4172. DOI: 10.1016/j.cell.2020.02.058 (cit. on p. 4).

[35] S. D. Axen, A. Gessner, C. Sommer, N. Weitzel, and Á. Tejero-Cantero. "Spatiotemporal Modeling of European Paleoclimate Using Doubly Sparse Gaussian Processes". In: *Workshop on Gaussian Processes, Spatiotemporal Modeling, and Decision-making Systems (NeurIPS)*. 2022. DOI: 10.48550/arXiv.2211.08160 (cit. on p. 4).

[36] A. Rajkomar, M. Hardt, M. D. Howell, G. Corrado, and M. H. Chin. "Ensuring Fairness in Machine Learning to Advance Health Equity". In: *Annals of Internal Medicine* 169.12 (2018), pp. 866–872. DOI: 10.7326/M18-1990 (cit. on p. 7).

[37] B. Hutchinson, V. Prabhakaran, E. Denton, K. Webster, Y. Zhong, and S. Denuyl. *Social Biases in NLP Models as Barriers for Persons with Disabilities*. 2020. DOI: 10.48550/arXiv.2005.00813. arXiv: 2005.00813 (cit. on p. 7).

[38] J. Buolamwini and T. Gebru. "Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification". In: *Journal of Machine Learning Research* (2018) (cit. on p. 7).

[39] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Association for Computing Machinery, 2021, pp. 610–623. DOI: 10.1145/3442188.3445922 (cit. on p. 7).

[40] M. J. Dupré and F. J. Tipler. "New axioms for rigorous Bayesian probability". In: *Bayesian Analysis* 4.3 (2009), pp. 599–606. DOI: 10.1214/09-BA422 (cit. on p. 9).

[41] A. Wald. "Statistical Decision Functions". In: *The Annals of Mathematical Statistics* 20.2 (1949), pp. 165–205. DOI: 10.1214/aoms/1177730030 (cit. on p. 9).

[42] A. Zellner. "Optimal Information Processing and Bayes's Theorem". In: *The American Statistician* 42.4 (1988), pp. 278–280. DOI: 10.2307/2685143 (cit. on p. 11).

[43] M. E. Khan and H. Rue. *The Bayesian Learning Rule*. 2022. DOI: 10.48550/arXiv.2107.04562 (cit. on p. 11).

[44] M. Kanagawa, P. Hennig, D. Sejdinovic, and B. K. Sriperumbudur. *Gaussian processes and kernel methods: A review on connections and equivalences*. 2018. arXiv: 1807.02582 (cit. on pp. 12, 43, 52, 103, 104, 107).

[45] P. Hennig. *Probabilistic machine learning*. Lecture Course. University of Tübingen, 2020 (cit. on p. 15).

[46] M. Pförtner, I. Steinwart, P. Hennig, and J. Wenger. *Physics-Informed Gaussian Process Regression Generalizes Linear PDE Solvers*. 2022. arXiv: 2212.12474 (cit. on pp. 15, 16, 23, 75).

[47] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006 (cit. on pp. 15, 26, 35, 42, 50, 58, 102).

[48] H. Wendland. *Scattered data approximation*. Vol. 17. Cambridge University Press, 2004 (cit. on p. 15).

[49] M. van der Wilk, C. E. Rasmussen, and J. Hensman. "Convolutional Gaussian Processes". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. DOI: 10.48550/arXiv.1709.01894 (cit. on p. 17).

[50] B. Adlam, J. Lee, S. Padhy, Z. Nado, and J. Snoek. *Kernel Regression with Infinite-Width Neural Networks on Millions of Examples*. 2023. DOI: 10.48550/arXiv.2303.05420 (cit. on p. 17).

[51] P. Hennig, M. A. Osborne, and M. Girolami. "Probabilistic numerics and uncertainty in computations". In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 471.2179 (2015) (cit. on pp. 19, 27, 32, 42, 48).

[52] J. Cockayne, C. Oates, T. J. Sullivan, and M. Girolami. "Bayesian probabilistic numerical methods". In: *SIAM Review* 61.4 (2019), pp. 756–789 (cit. on pp. 19, 27, 42, 48).

[53] C. Oates and T. J. Sullivan. "A modern retrospective on probabilistic numerics". In: *Statistics and Computing* (2019) (cit. on pp. 19, 27, 32, 42, 48).

[54] P. Hennig, M. A. Osborne, and H. P. Kersting. *Probabilistic Numerics: Computation as Machine Learning*. Cambridge University Press, 2022. ISBN: 9781316681411. DOI: 10.1017/9781316681411 (cit. on pp. 19, 42, 48).

[55] J. Wenger, N. Krämer, M. Pförtner, J. Schmidt, N. Bosch, N. Effenberger, J. Zenn, T. K. Alexandra Gessner, F.-X. Briol, M. Mahsereci, and P. Hennig. *ProbNum: Probabilistic Numerics in Python*. 2021. arXiv: 2112.02100 (cit. on pp. 21, 23, 53, 73).

[56] J. Wenger, H. Kjellström, and R. Triebel. "Non-Parametric Calibration for Classification". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2020 (cit. on p. 23).

[57] Y. Saad. *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992 (cit. on pp. 26, 32).

[58] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997 (cit. on pp. 26, 32, 61, 112).

[59] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 2013 (cit. on pp. 26, 31, 32, 58).

[60] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006 (cit. on pp. 26, 28).

[61] T. Hofmann, B. Schölkopf, and A. J. Smola. "Kernel methods in machine learning". In: *The Annals of Statistics* (2008), pp. 1171–1220 (cit. on p. 26).

[62] R. E. Kalman. "A new approach to linear filtering and prediction problems". In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45 (cit. on p. 26).

[63] F. R. K. Chung. *Spectral graph theory*. American Mathematical Society, 1997 (cit. on p. 26).

[64] C. A. J. Fletcher. *Computational Galerkin methods*. Springer, 1984, pp. 72–85 (cit. on pp. 26, 37).

[65] J. Nocedal and S. Wright. *Numerical optimization*. Springer, 2006 (cit. on pp. 26, 31, 32, 34, 58, 79, 81, 95).

[66] P. Hennig. "Probabilistic Interpretation of Linear Solvers". In: *SIAM Journal on Optimization* 25.1 (2015), pp. 234–260 (cit. on pp. 27, 28, 32, 48, 49, 81).

[67] S. Bartels, J. Cockayne, I. C. Ipsen, and P. Hennig. "Probabilistic linear solvers: A unifying view". In: *Statistics and Computing* 29.6 (2019), pp. 1249–1263 (cit. on pp. 27, 28, 32, 48, 49).

[68] P. Lévy. *Calcul des probabilités*. J. Gabay, 1925 (cit. on p. 28).

[69] C. F. Van Loan. "The ubiquitous Kronecker product". In: *Journal of Computational and Applied Mathematics* 123.1-2 (2000), pp. 85–100 (cit. on p. 28).

[70] P. Hennig and M. Kiefel. "Quasi-Newton method: A new direction". In: *Journal of Machine Learning Research* 14 (2013), pp. 843–865 (cit. on pp. 28, 32, 82).

[71] M. R. Hestenes and E. Stiefel. "Methods of conjugate gradients for solving linear systems". In: *Journal of Research of the National Bureau of Standards* 49 (1952) (cit. on pp. 30, 32, 47, 48, 58, 61, 92).

[72] M. H. Gutknecht and Z. Strakos. "Accuracy of two three-term and three two-term recurrences for Krylov space solvers". In: *SIAM Journal on Matrix Analysis and Applications* 22.1 (2000), pp. 213–229 (cit. on p. 30).

[73] J. Cockayne, C. Oates, I. C. Ipsen, and M. Girolami. "A Bayesian Conjugate Gradient Method". In: *Bayesian Analysis* 14.3 (2019), pp. 937–1012 (cit. on pp. 30, 32, 34, 48, 49).

[74] M. Seeger. *Low Rank Updates for the Cholesky Decomposition*. Tech. rep. University of California at Berkeley, 2008 (cit. on pp. 30, 79).

[75] C. C. Paige. "Computational variants of the Lanczos method for the eigenproblem". In: *IMA Journal of Applied Mathematics* 10.3 (1972), pp. 373–381 (cit. on p. 31).

[76] H. D. Simon. "Analysis of the symmetric Lanczos algorithm with reorthogonalization methods". In: *Linear Algebra and its Applications* 61 (1984), pp. 101–131 (cit. on p. 31).

[77] C. Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Government Press Office Los Angeles, CA, 1950 (cit. on pp. 31, 58, 61, 113).

[78] P. Drineas and M. W. Mahoney. "RandNLA: randomized numerical linear algebra". In: *Communications of the ACM* 59.6 (2016), pp. 80–90 (cit. on p. 32).

[79] A. Gittens and M. W. Mahoney. "Revisiting the Nyström Method for Improved Large-Scale Machine Learning". In: *Journal of Machine Learning Research* 17.1 (2016), pp. 3977–4041 (cit. on p. 32).

[80] S. Bartels and P. Hennig. "Probabilistic Approximate Least-Squares". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2016 (cit. on p. 32).

[81] J. E. Dennis Jr and J. J. Moré. "Quasi-Newton methods, motivation and theory". In: *SIAM Review* 19.1 (1977), pp. 46–89 (cit. on pp. 32, 82).

[82] T. Karvonen and S. Sarkkä. "Approximate state-space Gaussian processes via spectral transformation". In: *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*. 2016, pp. 1–6 (cit. on p. 34).

[83] A. Solin, J. Hensman, and R. E. Turner. "Infinite-Horizon Gaussian Processes". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018 (cit. on p. 34).

[84] H. Weyl. “Das asymptotische Verteilungsgesetz der Eigenwerte linearer partieller Differentialgleichungen (mit einer Anwendung auf die Theorie der Hohlraumstrahlung)”. In: *Mathematische Annalen* 71.4 (1912), pp. 441–479 (cit. on p. 35).

[85] U. D. of Transportation. *Airline On-Time Performance Data*. https://www.transtats.bts.gov/. Accessed: 2020-05-26. 2020 (cit. on p. 36).

[86] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. “The FEniCS project version 1.5”. In: *Archive of Numerical Software* 3.100 (2015) (cit. on p. 37).

[87] M. L. Parks, E. De Sturler, G. Mackey, D. D. Johnson, and S. Maiti. “Recycling Krylov subspaces for sequences of linear systems”. In: *SIAM Journal on Scientific Computing* 28.5 (2006), pp. 1651–1674 (cit. on p. 37).

[88] F. de Roos and P. Hennig. *Krylov Subspace Recycling for Fast Iterative Least-Squares in Machine Learning*. 2017. arXiv: 1706.00241 (cit. on p. 37).

[89] H. Zhu, C. K. Williams, R. Rohwer, and M. Morciniec. “Gaussian regression and optimal finite dimensional linear models”. In: *Neural Networks and Machine Learning*. 1997 (cit. on p. 42).

[90] G. F. Trecate, C. K. Williams, and M. Opper. “Finite-dimensional approximation of Gaussian processes”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 1999 (cit. on p. 42).

[91] A. Rahimi and B. Recht. “Random Features for Large-Scale Kernel Machines”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2007 (cit. on pp. 42, 48, 68, 133).

[92] A. Wilson and H. Nickisch. “Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP)”. In: *International Conference on Machine Learning (ICML)*. 2015 (cit. on pp. 42, 48, 60, 69).

[93] A. G. Wilson, C. Dann, and H. Nickisch. *Thoughts on Massively Scalable Gaussian Processes*. 2015. arXiv: 1511.01870 (cit. on pp. 42, 60).

[94] Z. Yang, A. Wilson, A. Smola, and L. Song. “A la carte–learning fast kernels”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2015 (cit. on p. 42).

[95] P. Izmailov, A. Novikov, and D. Kropotov. “Scalable Gaussian processes with billions of inducing inputs via tensor train decomposition”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2018 (cit. on p. 42).

[96] T. Evans and P. Nair. “Scalable Gaussian processes with grid-structured eigenfunctions (GP-GRIEF)”. In: *International Conference on Machine Learning (ICML)*. 2018 (cit. on p. 42).

[97] A. Zandieh, N. Nouri, A. Velingker, M. Kapralov, and I. Razenshteyn. “Scaling up kernel ridge regression via locality sensitive hashing”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2020 (cit. on p. 42).

[98] A. V. Vecchia. “Estimation and model identification for continuous spatial processes”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 50.2 (1988), pp. 297–312 (cit. on p. 42).

[99] A. Datta, S. Banerjee, A. O. Finley, and A. E. Gelfand. “Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets”. In: *Journal of the American Statistical Association* 111.514 (2016), pp. 800–812 (cit. on p. 42).

[100] M. Katzfuss and J. Guinness. “A General Framework for Vecchia Approximations of Gaussian Processes”. In: *Statistical Science* 36.1 (2021), pp. 124–141 (cit. on p. 42).

[101] F. Schäfer, M. Katzfuss, and H. Owhadi. “Sparse Cholesky Factorization by Kullback-Leibler Minimization”. In: *SIAM Journal on Scientific Computing* 43.3 (2021), A2019–A2046 (cit. on p. 42).

[102] A. J. Smola and P. Bartlett. “Sparse greedy Gaussian process regression”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2000 (cit. on pp. 42, 47).

[103] C. Williams and M. Seeger. “Using the Nyström method to speed up kernel machines”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2001 (cit. on pp. 42, 47).

[104] P. Drineas and M. W. Mahoney. “On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning”. In: *Journal of Machine Learning Research* 6 (2005), pp. 2153–2175 (cit. on pp. 42, 133).

[105] M. W. Seeger, C. K. Williams, and N. Lawrence. “Fast Forward Selection to Speed Up Sparse Gaussian Process Regression”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2003 (cit. on pp. 42, 47).

[106] E. Snelson and Z. Ghahramani. “Sparse Gaussian processes using pseudo-inputs”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2005 (cit. on p. 42).

[107] J. Quiñonero-Candela and C. E. Rasmussen. “A unifying view of sparse approximate Gaussian process regression”. In: *Journal of Machine Learning Research* 6 (2005), pp. 1939–1959 (cit. on pp. 42, 47, 96).

[108] M. Titsias. “Variational learning of inducing variables in sparse Gaussian processes”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2009 (cit. on pp. 42, 48, 60).

[109] J. Hensman, N. Fusi, and N. D. Lawrence. “Gaussian processes for Big data”. In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2013 (cit. on pp. 42, 47, 48, 53, 60).

[110] M. Gibbs. “Bayesian Gaussian processes for classification and regression”. In: *University of Cambridge, Cambridge* (1997) (cit. on pp. 42, 69).

[111] I. Murray. “Gaussian processes and fast matrix-vector multiplies”. In: *Numerical Mathematics in Machine Learning Workshop (ICML)*. 2009 (cit. on pp. 42, 47, 48, 58, 69).

[112] K. Cutajar, M. Osborne, J. Cunningham, and M. Filippone. “Preconditioning kernel matrices”. In: *International Conference on Machine Learning (ICML)*. 2016 (cit. on pp. 42, 47, 48, 58, 68, 69, 72).

[113] K. A. Wang, G. Pleiss, J. R. Gardner, S. Tyree, K. Q. Weinberger, and A. G. Wilson. "Exact Gaussian processes on a million data points". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019 (cit. on pp. 42, 47, 48, 55, 58, 60, 69, 72, 135).

[114] A. Artemev, D. R. Burt, and M. van der Wilk. "Tighter Bounds on the Log Marginal Likelihood of Gaussian Process Regression Using Conjugate Gradients". In: *International Conference on Machine Learning (ICML)*. 2021 (cit. on pp. 42, 47, 48, 58, 69).

[115] M. Bauer, M. van der Wilk, and C. E. Rasmussen. "Understanding probabilistic sparse Gaussian process approximations". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2016) (cit. on p. 42).

[116] J. H. Huggins, T. Campbell, M. Kasprzak, and T. Broderick. "Scalable Gaussian process inference with finite-data mean and variance guarantees". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2019 (cit. on pp. 42, 48).

[117] F. Schäfer, T. Sullivan, and H. Owhadi. "Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity". In: *Multiscale Modeling and Simulation* 19.2 (2021), pp. 688–730 (cit. on pp. 47, 48, 67).

[118] J. P. Cunningham, K. V. Shenoy, and M. Sahani. "Fast Gaussian process methods for point process intensity estimation". In: *International Conference on Machine Learning (ICML)*. 2008 (cit. on pp. 47, 48, 69).

[119] A. Potapczynski, L. Wu, D. Biderman, G. Pleiss, and J. P. Cunningham. "Bias-Free Scalable Gaussian Processes via Randomized Truncations". In: *International Conference on Machine Learning (ICML)*. 2021 (cit. on pp. 47, 58).

[120] B. W. Silverman. "Some aspects of the spline smoothing approach to non-parametric regression curve fitting". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 47.1 (1985), pp. 1–21 (cit. on p. 47).

[121] L. Csató and M. Opper. "Sparse on-line Gaussian processes". In: *Neural Computation* 14.3 (2002), pp. 641–668 (cit. on p. 47).

[122] V. Wild, M. Kanagawa, and D. Sejdinovic. *Connections and Equivalences between the Nyström Method and Sparse Variational Gaussian Processes*. 2021. arXiv: 2106.01121 (cit. on pp. 47, 96).

[123] D. R. Burt, C. E. Rasmussen, and M. Van Der Wilk. "Rates of convergence for sparse variational Gaussian process regression". In: *International Conference on Machine Learning (ICML)*. 2019 (cit. on p. 48).

[124] A. G. Journel and C. J. Huijbregts. *Mining geostatistics*. Academic Press London, 1976 (cit. on pp. 48, 107).

[125] J. T. Wilson, V. Borovitskiy, A. Terenin, P. Mostowsky, and M. Deisenroth. "Efficiently sampling functions from Gaussian process posteriors". In: *International Conference on Machine Learning (ICML)*. 2020 (cit. on pp. 48, 107).

[126] J. T. Wilson, V. Borovitskiy, A. Terenin, P. Mostowsky, and M. P. Deisenroth. "Pathwise Conditioning of Gaussian Processes". In: *Journal of Machine Learning Research* (2021) (cit. on pp. 48, 107).

[127] B. Charlier, J. Feydy, J. A. Glaunès, F.-D. Collin, and G. Durif. "Kernel Operations on the GPU, with Autodiff, without Memory Overflows". In: *Journal of Machine Learning Research* 22.74 (2021), pp. 1–6 (cit. on pp. 48, 53, 58).

[128] J. Cockayne, I. C. Ipsen, C. J. Oates, and T. W. Reid. "Probabilistic Iterative Methods for Linear Systems". In: *Journal of Machine Learning Research* (2021) (cit. on p. 48).

[129] T. W. Reid, I. C. F. Ipsen, J. Cockayne, and C. J. Oates. *BayesCG As An Uncertainty Aware Version of CG*. 2022. arXiv: 2008.03225 (cit. on p. 48).

[130] S. Kaczmarz. "Angenäherte Auflösung von Systemen linearer Gleichungen". In: *Bulletin International de l'Académie Polonaise des Sciences et des Lettres. Classe des Sciences Mathématiques et Naturelles. Série A, Sciences Mathématiques* (1937), pp. 355–357 (cit. on p. 49).

[131] T. Strohmer and R. Vershynin. "A randomized Kaczmarz algorithm with exponential convergence". In: *Journal of Fourier Analysis and Applications* 15.2 (2009), pp. 262–278 (cit. on pp. 49, 50).

[132] R. M. Gower and P. Richtárik. "Randomized iterative methods for linear systems". In: *SIAM Journal on Matrix Analysis and Applications* 36.4 (2015), pp. 1660–1690 (cit. on pp. 49, 50).

[133] R. M. Gower. *Sketch and project: Randomized iterative methods for linear systems and inverting matrices*. 2016. arXiv: 1612.06013 (cit. on p. 49).

[134] S. Bartels and P. Hennig. "Conjugate Gradients for Kernel Machines". In: *Journal of Machine Learning Research* 21.55 (2020), pp. 1–42 (cit. on p. 49).

[135] D. Dua and C. Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml (cit. on pp. 53, 70).

[136] G. Pleiss, J. Gardner, K. Weinberger, and A. G. Wilson. "Constant-time predictive distributions for Gaussian processes". In: *International Conference on Machine Learning (ICML)*. 2018 (cit. on pp. 53, 60).

[137] M. Anitescu, J. Chen, and L. Wang. "A matrix-free approach for solving the parametric Gaussian process maximum likelihood problem". In: *SIAM Journal on Scientific Computing* 34.1 (2012), A240–A262 (cit. on pp. 58, 69).

[138] S. Ubaru, J. Chen, and Y. Saad. "Fast estimation of $\mathrm{tr}(f(A))$ via stochastic Lanczos quadrature". In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (2017), pp. 1075–1099 (cit. on pp. 58, 59, 61, 64, 68, 69, 113, 114).

[139] K. Dong, D. Eriksson, H. Nickisch, D. Bindel, and A. G. Wilson. "Scalable log determinants for Gaussian process kernel learning". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2017) (cit. on pp. 58, 69).

[140] M. F. Hutchinson. "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines". In: *Communications in Statistics-Simulation and Computation* 18.3 (1989), pp. 1059–1076 (cit. on pp. 58, 60, 69).

[141] H. Avron and S. Toledo. “Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix”. In: *Journal of the ACM* 58.2 (2011), pp. 1–34 (cit. on pp. 58, 63, 68, 69).

[142] G. H. Golub and G. Meurant. *Matrices, moments and quadrature with applications*. Vol. 30. Princeton University Press, 2009 (cit. on pp. 60, 61, 69, 113).

[143] R. P. Adams, J. Pennington, M. J. Johnson, J. Smith, Y. Ovadia, B. Patton, and J. Saunderson. *Estimating the Spectral Density of Large Implicit Matrices*. 2018. arXiv: 1802.03451 (cit. on pp. 62, 69).

[144] R. A. Meyer, C. Musco, C. Musco, and D. P. Woodruff. “Hutch++: Optimal Stochastic Trace Estimation”. In: *Symposium on Simplicity in Algorithms (SOSA)*. Society for Industrial and Applied Mathematics. 2021, pp. 142–155 (cit. on pp. 62, 63, 68, 69, 115).

[145] F. Roosta-Khorasani and U. Ascher. “Improved bounds on sample size for implicit matrix trace estimators”. In: *Foundations of Computational Mathematics* 15.5 (2015), pp. 1187–1212 (cit. on pp. 63, 68, 69).

[146] D. Persson, A. Cortinovis, and D. Kressner. *Improved variants of the Hutch++ algorithm for trace estimation*. 2021. arXiv: 2109.10659 (cit. on pp. 63, 68, 69).

[147] S. Jiang, H. Pham, D. P. Woodruff, Qiuyi, and Zhang. “Optimal Sketching for Trace Estimation”. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2021) (cit. on pp. 63, 68, 69).

[148] D. S. Kershaw. “The incomplete Cholesky—conjugate gradient method for the iterative solution of systems of linear equations”. In: *Journal of Computational Physics* 26.1 (1978), pp. 43–65 (cit. on p. 68).

[149] M. Mutnỳ and A. Krause. “Efficient high dimensional Bayesian optimization with additivity and quadrature Fourier features”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018 (cit. on pp. 68, 130, 131).

[150] M. Skorski. “Modern Analysis of Hutchinson’s Trace Estimator”. In: *Annual Conference on Information Sciences and Systems (CISS)*. IEEE. 2021, pp. 1–5 (cit. on p. 68).

[151] Y. Saatçi. “Scalable inference for structured Gaussian process models”. PhD thesis. University of Cambridge, 2012 (cit. on p. 69).

[152] A. C. Faul, G. Goodsell, and M. J. D. Powell. “A Krylov subspace algorithm for multiquadric interpolation in many dimensions”. In: *IMA Journal of Numerical Analysis* 25.1 (2005), pp. 1–24 (cit. on p. 69).

[153] N. A. Gumerov and R. Duraiswami. “Fast radial basis function interpolation via preconditioned Krylov iteration”. In: *SIAM Journal on Scientific Computing* 29.5 (2007), pp. 1876–1899 (cit. on p. 69).

[154] M. L. Stein, J. Chen, and M. Anitescu. “Difference Filter Preconditioning for Large Covariance Matrices”. In: *SIAM Journal on Matrix Analysis and Applications* 33.1 (2012), pp. 52–72 (cit. on p. 69).

[155] J. Chen. "On the Use of Discrete Laplace Operator for Preconditioning Kernel Matrices". In: *SIAM Journal on Scientific Computing* 35.2 (2013), A577–A602 (cit. on p. 69).

[156] C. Bekas, E. Kokiopoulou, and Y. Saad. "An estimator for the diagonal of a matrix". In: *Applied Numerical Mathematics* 57.11-12 (2007), pp. 1214–1229 (cit. on p. 69).

[157] A. Cortinovis and D. Kressner. "On randomized trace estimates for indefinite matrices with an application to determinants". In: *Foundations of Computational Mathematics* (2021), pp. 1–29 (cit. on p. 69).

[158] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations (ICLR)* (2015) (cit. on p. 72).

[159] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003 (cit. on pp. 72, 113).

[160] A. Immer, M. Korzepa, and M. Bauer. "Improving predictions of Bayesian neural nets via local linearization". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2021. URL: http://arxiv.org/abs/2008.08400 (cit. on p. 74).

[161] L. Nazareth. "A Relationship between the BFGS and Conjugate Gradient Algorithms and its Implications for New Algorithms". In: *SIAM Journal on Numerical Analysis* 16.5 (1979), pp. 794–800 (cit. on p. 81).

[162] K. T. Conrad. *The minimal polynomial and some applications*. Tech. rep. University of Connecticut, 2008, p. 14 (cit. on p. 85).

[163] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. "A deflated version of the conjugate gradient algorithm". In: *SIAM Journal on Scientific Computing* 21.5 (2000), pp. 1909–1926 (cit. on pp. 94, 95).

[164] J. Frank and C. Vuik. "On the construction of deflation-based preconditioners". In: *SIAM Journal on Scientific Computing* 23.2 (2001), pp. 442–462 (cit. on p. 95).

[165] J.-P. Chiles and P. Delfiner. *Geostatistics: modeling spatial uncertainty*. Vol. 497. John Wiley & Sons, 2009 (cit. on p. 107).

[166] A. Doucet. *A note on efficient conditional simulation of Gaussian distributions*. Tech. rep. Departments of Computer Science and Statistics, University of British Columbia, 2010 (cit. on p. 107).

[167] M. Ledoux. *The concentration of measure phenomenon*. 89. American Mathematical Society, 2001 (cit. on p. 114).

[168] P.-M. Samson. "Concentration of measure inequalities for Markov chains and Φ-mixing processes". In: *The Annals of Probability* 28.1 (2000), pp. 416–461 (cit. on p. 114).

[169] S. P. Kasiviswanathan and M. Rudelson. *Restricted Isometry Property under High Correlations*. 2019. arXiv: 1904.05510 (cit. on p. 114).

[170] R. Adamczak. "A note on the Hanson-Wright inequality for random vectors with dependencies". In: *Electronic Communications in Probability* 20 (2015), pp. 1–13 (cit. on pp. 114, 115).

[171] F. Kittaneh. “On Lipschitz functions of normal operators”. In: *Proceedings of the American Mathematical Society* 94.3 (1985), pp. 416–418 (cit. on p. 117).

[172] H. Harbrecht, M. Peters, and R. Schneider. “On the low-rank approximation by the pivoted Cholesky decomposition”. In: *Applied Numerical Mathematics* 62.4 (2012), pp. 428–440 (cit. on p. 130).

[173] C. Eckart and G. Young. “The approximation of one matrix by another of lower rank”. In: *Psychometrika* 1.3 (1936), pp. 211–218 (cit. on p. 132).

[174] P. Drineas, R. Kannan, and M. W. Mahoney. “Fast Monte Carlo Algorithms for Matrices II: Computing a Low-Rank Approximation to a Matrix”. In: *SIAM Journal on Computing* 36 (2006), pp. 158–183 (cit. on p. 132).

[175] B. K. Sriperumbudur and Z. Szabó. “Optimal rates for random Fourier features”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015 (cit. on p. 133).

[176] D. S. Mitrinovic and P. M. Vasic. *Analytic inequalities*. Vol. 1. Springer, 1970 (cit. on p. 134).