Learning to Solve Related Linear Systems

Disha Hegde — *University of Southampton, United Kingdom*
Jon Cockayne — *University of Southampton, United Kingdom*

Abstract

Solving multiple parametrised related systems is an essential component of many numerical tasks, and learning from the already solved systems will make this process faster. In this work, we propose a novel probabilistic linear solver over the parameter space. This leverages information from the solved linear systems in a regression setting to provide an efficient posterior mean and covariance. We advocate using this as companion regression model for the preconditioned conjugate gradient method, and discuss the favourable properties of the posterior mean and covariance as the initial guess and preconditioner. We also provide several design choices for this companion solver. Numerical experiments showcase the benefits of using our novel solver in a hyperparameter optimisation problem.

Proceedings of the 1st International Conference on Probabilistic Numerics (ProbNum) 2025, Sophia Antipolis, France. PMLR Volume 271.

1 Introduction

We consider the problem of solving multiple related linear systems. Suppose that these linear systems vary according to some governing parameter $\theta \in \Theta$. Then the set of problems we are interested in is of the form

$$A_\theta x_\theta^\star = b_\theta \tag{1}$$

where $A_\theta \in \mathbb{R}^{d\times d}$ is assumed to be symmetric positive definite for each $\theta \in \Theta$, $b_\theta \in \mathbb{R}^d$ is a right-hand-side vector that is given, and the objective is to recover $x_\theta^\star \in \mathbb{R}^d$. We will suppose that there is some externally supplied sequence of parameters (θ_i), $i = 1, 2, \ldots$ and index the systems according to $A_i \equiv A_{\theta_i}$ (similarly $x_i^\star$, b_i). Our task is then, given some information about systems seen for $i = 1, \ldots, n-1$, to accelerate convergence of an iterative solver for the next system generated by θ_n, in an online fashion.

In this work we argue that this problem can be structured as *regression problem* and use the tools of Gaussian process regression to solve it.

1.1 Related Work

In this work we will focus on iterative linear solvers to solve these related linear systems. Given the prevalence of related linear systems it is natural that there are many existing approaches that seek to solve this problem. These methods can roughly be split into the following three categories:

Initial Guess Various attempts have been made to find improved initial guesses for the iterative solver in this setting. Warm restarting—using the solution of the previous linear system as the initial guess—is a commonly used approach. As an extension of this, extrapolating from previous solutions has also been explored, e.g., polynomial extrapolation in Shterev (2015); Austin et al. (2021), using Taylor expansion in Clemens et al. (2003), using weighted averages in Ye et al. (2020). Projection methods are also used (e.g. Fischer, 1998; Markovinović and Jansen, 2006) to estimate an efficient initial guess. Though these attempts accelerate the linear solver to some extent, using a suitable preconditioner can provide a far greater acceleration.

Preconditioning One efficient way of increasing the efficiency of the linear solver is by preconditioning. A commonly used strategy to obtain a good preconditioner for subsequent problems is to start with a general preconditioner and adapt it in some way based on the solution of previous systems (e.g. Bergamaschi, 2020; Li, 2015; Carr et al., 2021). Machine learning based approaches like graph neural networks are commonly used to for this purpose (Häusner et al., 2024; Li et al., 2023).

Using covariances related to the problem as a preconditioner was also considered in Calvetti and Somersalo (2005) where a preconditioner for an ill-posed inverse problem is constructed as the covariance matrix of sampled solutions, interpreted as a random variable. The posterior covariance from a probabilistic linear solver with specific inverse prior has been used to accelerate generalized linear models in Tatzel et al. (2025).

Recycling Subspaces Another way of accelerating (Krylov-based) linear solvers is to *recycle* the Krylov subspaces explored for previous linear systems. The idea is to improve the search space for the current system by restricting it or enhancing it, leading to faster convergence. This is done by deflation (Saad et al., 2000; de Roos and Hennig, 2017; Daas et al., 2021), augmentation (Erhel and Guyomarc'h, 2000; Carlberg et al., 2016), or a combination of both.

1.2 Proposed Approach

We propose to extend the framework of *probabilistic linear solvers* (Hennig, 2015; Cockayne et al., 2019) to

build a solver for related linear systems. Existing probabilistic linear solvers (Section 2.2) typically function in a Bayesian framework by conditioning a prior distribution on observations of Eq. (1) (e.g. by left-multiplying by some matrix of *search directions* $(S^m)^\top \in \mathbb{R}^{m\times d}$). We propose to extend this approach by extending the prior over the parameter space Θ, so that, after conditioning, we can use the posterior to extrapolate to other problems (Section 3). We refer to this idea as using the probabilistic solver as a *companion regression model* for a classical iterative method. Specifically, we will use the posterior predictive mean as an initial guess for subsequent systems, and the posterior predictive covariance as the preconditioner for an iterative linear solver.

Compared to the methods described in Section 1.1, our approach is similar to using an extrapolation method to construct an initial guess, and one of the learning-based methods to construct a the preconditioner. There is also a peripheral relation to subspace recycling, as we will establish that, to ensure convergence, the initial guess and preconditioner must relate to each other in a particular subspace (see Theorem 7).

1.3 Structure of the Paper

The paper proceeds as follows. In Section 2 we discuss the background on iterative solvers (Section 2.1), probabilistic linear solvers (Section 2.2) and vector-valued Gaussian processes (GPs) (Section 2.3). In Section 3 we use these tools to extend probabilistic linear solvers across a parameter space, and discuss how to utilise the posterior to accelerate subsequent solves. We provide some theoretical results in Section 4 and discuss efficient computation strategies in Section 5. In Section 6, we perform a simulation study (Section 6.1) to demonstrate the advantages of our method, followed by an application on a GP hyperparameter optimisation problem in Section 6.2. All the proofs are contained in Section A. Some practical implementation details and complexity analysis are discussed in Section B.1 and Section B.2. Finally, some additional numerical results are presented in Section C.

2 Background

In this section we will present the required background for the paper. We will primarily focus on solution of single linear systems, and will suppress dependence of this system on the parameter θ to simplify the exposition.

2.1 Iterative Solvers

Iterative linear solvers construct a sequence of iterates (x^j) which are such that $x^j \to x^\star$ as $j \to \infty$. Well-known examples of such solvers are *stationary iterative methods* such as Richardson's method; in such solvers the iterate x^{j+1} is constructed by applying an affine map to the previous iterate x^j. However such solvers are rarely used in contemporary applications, apart from occasionally to provide preconditioners (see e.g. Chen (2005, Section 5.6)); this is due to the fact that the reduction in error $x^j - x^\star$ is typically rather slow, and that convergence is only guaranteed in the limit as $j \to \infty$.

More sophisticated solvers such as Krylov methods (see e.g. Golub and Loan, 2013, Section 11.3 and Section 11.4) are more widely used in practice as a result of their faster convergence. This faster convergence is obtained because the map applied to x^j to obtain x^{j+1} is typically *nonlinear* rather than affine. As a result convergence is both faster and, for many methods, guaranteed after a finite number of iterations[1]. In this work, we specifically focus on solving symmetric positive definite matrices, and thus restrict ourselves to the conjugate gradient (CG) method (Golub and Loan, 2013, Section 11.3).

The convergence of these methods typically depends on two factors:

1. How far the initial iterate is from the true solution, i.e. the initial error $x^0 - x^\star$.
2. Some measure of the problem's conditioning, often measured as a function of the condition number of the matrix A_θ, or some other matrix related to it.

It is natural to seek to reduce these two quantities, to try and realise an acceleration of the problem. Improving the problem's conditioning is commonly achieved by building *preconditioners* (Golub and Loan, 2013, Section 11.5). For a static problem (i.e. for a single fixed θ) one is reliant on an understanding of the problem's structure or on generic methods to either improve the initial guess or construct preconditioners. However, in the present setting where we have access to multiple related linear systems, one can try *learn* good initial guesses and/or good preconditioners based on observations of Eq. (1) for different values of θ. In this work we will accomplish using a statistical approach based on GP regression (Rasmussen and Williams, 2006).

2.2 Probabilistic Linear Solvers

Recent work has recast some iterative methods as probabilistic numerical methods through probabilistic linear solvers. Here we focus on the solution-based Bayesian approach to probabilistic linear solvers as described in Cockayne et al. (2019). This can be viewed as an iterative method in which the iterates are Bayesian posterior distributions (μ^j), $j = 1, \ldots, d$ over $\mathbb{R}^d$; specifically, these are some prior distribution μ_0 conditioned sequentially on observations of the form

$$(s^j)^\top A x = (s^j)^\top b. \tag{2}$$

The vectors $s^j \in \mathbb{R}^d$ are generally called *search directions*. These are assumed to be linearly independent and are often also constructed sequentially, by examining the distribution μ^{j-1} to build s^j. We also introduce the search direction matrices $S^j = [s^1, \ldots, s^j] \in \mathbb{R}^{d\times j}$.

[1] These statements are true only in exact precision; in general convergence in finite precision is often slower

For reasons of conjugacy μ^0 is taken to be Gaussian; $\mu^0 \sim \mathcal{N}(x^0, \Sigma^0)$. Using the closed-form expression for a Gaussian posterior under noise-free linear observations such as in Eq. (2), one can then show that for all $j = 1, \ldots, d$ we have the following expressions for the iterates μ^j:

$$\mu^j = \mathcal{N}(x^j, \Sigma^j) \tag{3a}$$
$$x^j = x^0 + \Sigma^0 A^\top S^j (\Lambda^j)^{-1} (S^j)^\top (b - A x_0) \tag{3b}$$
$$\Sigma^j = \Sigma^0 - \Sigma^0 A^\top S^j (\Lambda^j)^{-1} (S^j)^\top A \Sigma^0 \tag{3c}$$
$$\Lambda^j = (S^j)^\top A \Sigma^0 A^\top S^j. \tag{3d}$$

BayesCG A prominent choice of search directions in this framework results in the Bayesian conjugate gradient method (BayesCG; Cockayne et al. (2019)). Here the search directions are constructed sequentially via the Lanczos process (Golub and Loan, 2013, Section 10.3), with each successive direction being an orthonormalised version of the previous residual, precisely given by:

$$\tilde{s}^j = r^{j-1} - (r^{j-1})^\top A \Sigma^0 A^\top s^{j-1} \cdot s^{j-1} \tag{4}$$

where $\tilde{s}^1 = r^0$, $s^j = \tilde{s}^j / \|s^j\|_{A\Sigma^0 A^\top}$, and $r^j = b - Ax^j$. One can show that the s^j are orthonormal in the $A\Sigma^0 A^\top$ inner product (Cockayne et al., 2019, Proposition 7).

These directions obtain information about the unknown solution in directions where the residual has largest magnitude, focussing computation on the largest errors in the present estimate. Two important mathematical consequences of this are that (i) one can show that the error $\|x^j - x^\star\|$ decays at a geometric rate in j, and (ii) the orthogonality properties of the search directions are such that the matrix Λ^j is diagonal, which eliminates an $\mathcal{O}(j^3)$ implementation cost.

Prior Choice While BayesCG is defined for generic prior covariance Σ^0, the choice $\Sigma^0 = A^{-1}$ has particular significance, as it results in iterates that coincide with the iterates of CG (see Cockayne et al., 2019, Proposition 4). While in general this is an impractical choice of prior, as calculation of the posterior covariance requires that A^{-1} be realised, defeating the point of an iterative solution to the system, in certain applications (Wenger et al. (2022b, 2024); Tatzel et al. (2025); Pförtner et al. (2025); Hegde et al. (2025)) A^{-1} cancels in downstream calculations, making this a more practical choice. A proxy for A^{-1} considered in Cockayne et al. (2019) is to use a preconditioner for A. Other works such as Reid et al. (2022); Vyas et al. (2025) seek to use the A^{-1} prior implicitly by performing a small number of "post-iterations" of CG to compute a low-rank approximation to the posterior.

2.3 Vector-valued Gaussian processes

The central focus of this work is on extending probabilistic linear solvers, which thus far have only been applied to single linear systems, to the setting of multiple related linear systems. The main tool we will use to accomplish this are vector-valued GPs. Essentially, we translate from a Gaussian distribution over the solution x of a single linear system to a GP over the solution x_θ as a function of θ. We first recap the scalar-valued case, before discussing vector-valued GPs.

2.3.1 Scalar-Valued GP regression

Definition 1 (Gaussian process). *(Álvarez et al., 2012, Definition 2.1) A random function $x : \Theta \to \mathbb{R}$ is said to be a* Gaussian process *if, for any finite-dimensional subset $T = \{\theta_1, \ldots, \theta_{n_T}\} \subset \Theta$, it holds that $x(T) := [x(\theta_1), \ldots, x(\theta_{n_T})]$ follows a multivariate Gaussian distribution.*

GPs are typically parameterised by their mean function $m(\theta) = \mathbb{E}(x(\theta))$ and their (positive-definite) covariance kernel $k(\theta, \theta') = \mathrm{Cov}(x(\theta), x(\theta'))$; we will use the notation $x \sim \mathcal{GP}(m, k)$. The primary use case of GPs is to perform regression. Specifically, supposing we are given data $\mathcal{D} := \{(\theta_i, y_i)\}_{i=1}^n$ where $y_i = x(\theta_i) + \zeta_i$, $\zeta_i \overset{\text{IID}}{\sim} \mathcal{N}(0, \sigma^2)$, then the posterior distribution $x \mid \mathcal{D}$ possesses a closed-form given by

$$x \mid \mathcal{D} \sim \mathcal{N}(\bar{m}, \bar{k})$$
$$\bar{m}(\theta) = m(\theta) + k(\theta, T)[k(T,T) + \sigma^2 I]^{-1}(y - m(T))$$
$$\bar{k}(\theta, \theta') = k(\theta, \theta') - k(\theta, T)[k(T,T) + \sigma^2 I]^{-1} k(T, \theta')$$

where $T = \{\theta_1, \ldots, \theta_n\}$.

2.3.2 Vector-Valued Case

The most straightforward way to extend the above to a vector-valued setting is to think of augmenting the domain of a scalar-valued process with an integer-valued index, representing the index of the (vector-valued) output function. More formally we construct a Gaussian process on the domain $\Theta \times [1, d]$, where $[1, d]$ is the integers from 1 to d, and abbreviate $x_i(\theta) \equiv x(\theta, i)$. We represent the mean as $m_i(\theta) = \mathbb{E}(x_i(\theta))$ and the (positive definite) kernel as $k_{ij}(\theta, \theta') = \mathrm{Cov}(x_i(\theta), x_j(\theta'))$. The marginal distribution for each θ is therefore multivariate normal, and given by

$$x(\theta) = \mathcal{N}(m(\theta), C(\theta, \theta))$$

where $m(\theta) \in \mathbb{R}^d$ has i^{th} component $m_i(\theta)$ while $C(\theta, \theta) \in \mathbb{R}^{d \times d}$ has (i, j)-component $k_{ij}(\theta, \theta)$, with positive definiteness assured by positive definiteness of the covariance kernel.

Under observations of the form $y_i = x(\theta_i) + \zeta_i$, GP regression can then be performed by "stacking" the covariance matrices $C(\theta, \theta')$ in appropriate blocks, and applying a formula analogous to that given above. A more explicit description of this can be found in many works, e.g. Álvarez et al. (2012). In this work, however, as we aim to model the solution vectors x_θ over the parameter space, we require more general observations of the vector-valued function $x(\theta)$, as described in the next section.

3 Learning to Solve Related Systems

We will now extend the methodology in Section 2.2 to multiple related systems. To simplify notation we will leave iteration numbers implicit, so that $S_i^j = S_i$ etc. As described in Section 2.2, we will suppose we are given information by projecting the linear systems against a sequence of projection matrices $S_i \in \mathbb{R}^{d\times m_i}$ with linearly independent columns. i.e. we have

$$S_i^\top A_i x_i^\star = S_i^\top b_i =: y_i. \tag{5}$$

To use this information from previously solved systems $i = 1, 2, \dots n$ and the current nth system, we augment the dataset with the search direction matrices, i.e. $\mathcal{D}_n := \{(\theta_i, S_i, y_i)\}_{i=1}^n$. The posterior distribution for this choice of data is described in the next section.

3.1 Full Posterior

We use a multi-output Gaussian process prior as described in Section 2.3. Specifically we take $x_\theta \sim \mu_0$, where $\mu_0 = \mathcal{N}(\bar{x}_0, C_0)$, $\bar{x}_0 = \bar{x}_0(\theta)$ and $C_0 : \Theta \times \Theta \to \mathbb{R}^{d\times d}$ is a positive-definite covariance function. The posterior distribution is given in the next proposition.

Proposition 2. *We have that $x_\theta \mid \mathcal{D}_n \sim \mathcal{N}(\bar{x}_n, C_n)$, where*

$$\bar{x}_n(\theta) = x_0(\theta) + K_n(\theta) G_n^{-1} z_n$$
$$C_n = C_n(\theta,\theta) = C_0(\theta,\theta) - K_n(\theta) G_n^{-1} K_n^\top(\theta)$$

and the matrices $K_n(\theta), G_n$ are each block matrices with dimensions $d \times M_n$ and $M_n \times M_n$ respectively, where $M_n = \sum_{i=1}^n m_i$, while $z_n \in \mathbb{R}^{M_n}$. Specifically we have

$$K_n(\theta) = \left[C_0(\theta,\theta_j) A_j^\top S_j\right]_{1\le j\le n}$$
$$G_n = \left[S_i^\top A_i C_0(\theta_i,\theta_j) A_j^\top S_j\right]_{1\le i,j\le n}$$
$$z_n = \left[y_i - S_i^\top A_i x_0(\theta_i)\right]_{1\le i\le n}$$

Finally, the cross covariance $\mathrm{Cov}(x_\theta, x'_\theta) = C_n(\theta,\theta')$, where

$$C_n(\theta,\theta') = C_0(\theta,\theta') - K_n(\theta) G_n^{-1} K_n^\top(\theta').$$

The next result guarantees invertibility of G_n provided the S_i have linearly independent columns.

Lemma 3. *Assume A_i is invertible and S_i has linearly independent columns for $i = 1, \dots, n$. Then G_n is invertible.*

Although we focus on symmetric positive definite matrices in this work, the posterior given in Proposition 2 holds more generally and, in particular, is well defined if A_i is nonsingular for $i = 1, \dots, n$. It could therefore be treated as a probabilistic linear solver in its own right, though this is not the focus of this paper.

Having defined the posterior distribution we now turn to how it can be used in practice. As mentioned in Section 1, we propose to use the posterior as a companion regression model, and use it to construct both initial guesses and preconditioners that can be passed to a classical iterative linear solver. In the next section we will present some theoretical results that justify this approach.

4 Theoretical Results

We now present some theoretical properties of the posterior covariance of the companion model. Our first result shows that the posterior predictive covariance has full rank (and is thus positive definite), when evaluated at locations not contained in the training set.

Theorem 4. *Assume the covariance C_0 is positive-definite. Then $C_{n-1}(\theta_n, \theta_n)$ is full rank for all $\theta \notin T$.*

The next result shows that when evaluated at the last point contained in the training set, the posterior covariance is rank-deficient. Note that this holds more generally for any point in the training set (i.e. $C_n(\theta_i, \theta_i)$ is rank deficient for any $i = 1, \dots, n$). However only the last point is relevant for subsequent developments.

Theorem 5. *The rank of $C_n(\theta_n, \theta_n)$ is $d - m_n$. Its null space is spanned by the columns of $A_n^\top S_n$.*

The above theorem explicitly provides the null space of the posterior covariance. This motivates our next result, which states that the posterior mean from the companion regression method is equal to the actual solution of the linear system, in the null space of the posterior covariance.

Theorem 6. *In the null space of $C_n(\theta_n, \theta_n)$ it holds that $\bar{x}_n(\theta_n) = x_n^\star$, i.e.*

$$S_n^\top A_n x_n(\theta_n) = S_n^\top A_n x_n^\star.$$

This result provides us the motivation to use the combination of the posterior predictive mean and covariance as initial guess and the preconditioner for preconditioned CG (pCG) (see Algorithm 1). As the posterior mean is already correct in the null space of the posterior covariance, using the posterior covariance as the preconditioner allows the fast exploration of the unexplored directions where the solution is yet to be identified. In fact, the next theorem shows that this combination is essential to use the posterior covariance as preconditioner for pCG.

While pCG has been shown to be well-defined for positive semi-definite matrices (Kaasschieter, 1988; Hayami, 2020), and for projection-type rank-deficient preconditioners (Frank and Vuik, 2001), using a positive semi-definite preconditioner that does not commute with A has not been explored in the literature, to the best of our knowledge. In the following result we prove that when a singular preconditioner is used in pCG, the algorithm still converges to the truth provided that the initial guess satisfies a specific condition.

Theorem 7. *Given a symmetric positive semi-definite preconditioner P, let U_1 denote a matrix whose columns form an orthonormal basis of the range of P and U_2 a matrix whose columns form an orthonormal basis of its null space. Consider solving the linear system $Ax^\star = b$*

using pCG from Algorithm 1. We have that if $U_2^\top x_0 = U_2^\top x^\star$*, then pCG with singular preconditioner* P *has iterates*

$$x_k = U_1 \tilde{x}_k + U_2 U_2^\top x^\star$$

where $\tilde{x}_k$ *are iterates from applying pCG to a modified system* $U_1^\top A U_1 x^\star = \bar{b}$*, (where* $\bar{b} = U_1^\top b - U_1^\top A U_2 U_2^\top x_0$*), with positive-definite preconditioner* $U_1^\top P U_1$*.*

The theorem stated above has some relation to *projected CG* (Zheng et al., 2020), which considers using singular preconditioners within nonlinear CG methods for convex optimisation to constraints that ensure the iterate remains in the feasible set. Our focus on the linear setting allows us to establish stronger results than can be applied in the more general nonlinear CG setting.

As Theorem 6 guarantees that our posterior mean satisfies this condition, we now have the liberty to use the posterior from the companion regression model to accelerate the conjugate gradient solver. In the next section we discuss some practical details.

5 Modelling Choices

We now consider the critical problem of choosing the prior and search-directions. Together these completely specify the companion regression model for the iterative solver.

5.1 Choices of Prior

We mostly ignore the task of choosing the prior mean. While this will clearly have a strong influence on the effectiveness of the initial guess provided by the companion regression model, it is generally problem-specific. Instead, we focus on the prior covariance, for which several natural choices present themselves.

Tensor Product Prior The tensor product prior is perhaps the most common choice for multi-output GP regression (e.g. Álvarez et al., 2012, Section 4). With this prior we take $C_0(\theta, \theta') = k(\theta, \theta')\Sigma$, where Σ is a fixed symmetric positive-definite $d \times d$ matrix, while $k(\theta, \theta')$ is a scalar-valued positive-definite kernel function. The (i, j) block of G_n can therefore be simplified to $G_{n,(i,j)} = k(\theta_i, \theta_j) \times S_i^\top A_i \Sigma A_i^\top S_j$, so that G_n can be factorised as

$$G_n = diag\left[S_i^\top A_i\right]_{1 \le i \le n} \left[\Sigma \otimes k(T, T)\right] diag\left[A_i^\top S_i\right]_{1 \le i \le n}$$

where $k(T, T) \in \mathbb{R}^{n \times n}$ is the kernel function on $T = (\theta_1, \theta_2, \dots \theta_n)$.

Unlike in standard multi-output regression problems, there is no particular structure that can be exploited in G_n (i.e. G_n does not have the form of a Kronecker product, whose inverse can be expressed as the product of the two matrix inverses). This is because the information used is heterogeneous apart from in the special case $S_i^\top A_i$ is independent of i, e.g. if $S_i = A_i^{-1}$.

A Natural, Nonseparable Prior Covariance Motivated by analogy with the literature on probabilistic linear solvers, we propose an (impractical) *nonseparable* prior covariance. When A_θ is symmetric positive definite for all θ, the prior $C_0(\theta, \theta) = A_\theta^{-1}$ has a special status as discussed in Section 2.2. Thus, taking L_θ to be a factorisation of A_θ such that $A_\theta = L_\theta L_\theta^\top$, this motivates the cross covariance $C_0(\theta, \theta') = L_\theta^{-1} k(\theta, \theta') L_{\theta'}^{-T}$. Thus, we consider nonseparable priors of the form

$$C_0(\theta, \theta) = k(\theta, \theta) A_\theta^{-1} \qquad C_0(\theta, \theta') = L_\theta^{-1} k(\theta, \theta') L_{\theta'}^{-1}$$

where $k(\theta, \theta')$ is again a scalar-valued positive-definite kernel function.

Under this choice of prior note that we can simplify all but one of the matrices involved in computation of the posterior to remove explicit dependence on A_i^{-1}. For example, we have that

$$G_n = \begin{pmatrix} S_1^\top A_1 S_1 & S_1^\top L_1 L_2^\top S_2 & \dots & S_1^\top L_1 L_n^\top S_n \\ S_2^\top L_2 L_1^\top S_1 & S_2^\top A_2 S_2 & \dots & S_2^\top L_2 L_n^\top S_n \\ \vdots & \vdots & \ddots & \vdots \\ S_n^\top L_n L_1^\top S_1 & S_n^\top L_n L_2^\top S_2 & \cdots & S_n^\top A_n S_n \end{pmatrix}.$$

However, note that the above still depends on the factors L_i, computation of which is still associated with a cubic cost, rendering this impractical. Moreover the cancellation does not mitigate dependence of the posterior covariance $C_n(\theta, \theta)$ on A_θ^{-1}, since it still has the form of a downdate of A_θ^{-1}. However we note that in several recent works (Wenger et al., 2022b) it has been shown that one can use posteriors of this form in downstream applications in such a way that A_θ^{-1} cancels. We do not exploit this fact in this paper, but observe this only to highlight that this prior choice is also of practical, not just theoretical interest.

Note that this choice of prior is still not completely general as it assumes that smoothness as a function of θ is identical for all components of the solution vector. However, nothing in the methodology presented prevents using general prior covariances.

5.2 Choices of Search Directions

Thus far we have made no assumptions on $S_1, \dots, S_n$ apart from that they are linearly independent. We now consider some natural choices.

Observation of the Solution Assuming we are willing to compute $x_1^\star, \dots, x_n^\star$, for the training set, it makes sense to consider the choice $S_i = A_i^{-1}$, corresponding to direct observation of the solution for each θ_i. This simplifies the posterior mean and covariance to be:

$$\bar{x}_n(\theta) = x_0(\theta) + C_0(\theta, T) C_0(T, T)^{-1} X_n^\star$$
$$C_n(\theta, \theta') = C_0(\theta, \theta') - C_0(\theta, T) C_0(T, T)^{-1} C_0(\theta', T)^\top$$

where $C_0(\theta, T) \in \mathbb{R}^{d \times dn}$ is a block matrix with elements $C(\theta, \theta_i)$, $j = 1, 2, \dots n$, $C_0(T, T) \in \mathbb{R}^{dn \times dn}$ is a black matrix with elements $C(\theta_i, \theta_j)$, $i, j = 1, 2, \dots n$; and $X_n^\star \in \mathbb{R}^{dn}$ is the stacked form of solutions of linear systems $x_i^\star, i = 1, 2, \dots n$.

This is equivalent to performing a multi-output Gaussian process regression with the parameter θ as the input variable and the solution $x_\theta^\star$ as the output variable, with C_0 as the prior kernel.

However, this choice compels us to update the companion regression model after the linear system has been solved, utilising no information about the current linear system being solved to condition on, while computing its initial guess and the preconditioner. We now propose choices of search directions that can more actively use the information from the linear system being solved.

Subset of Data Related to the above, we could also take S_i to be given by observing a subset of coordinates of b_i. Let $\mathbb{I}_n \subset 1\colon d$; we then take $S_i = I_{:,\mathbb{I}_{m_i}}$, i.e. the identity matrix with a subset of columns defined by $\mathbb{I}_{m_i}$. Attention then turns to how we should select $\mathbb{I}_{m_i}$. We take an approach similar to the one in Cockayne and Duncan (2021). We select the coordinates of b_i by minimising a heuristic based on fill distance between the new selected co-ordinates and the old co-ordinates in an augmented space of co-ordinates and the parameters of the linear system.

Krylov-Based Directions We can view the predictive distribution of μ_n at θ_{n+1} as providing a prior distribution for a probabilistic linear solver at that location. For convenience let $\mu_{n,n+1}$ denote this predictive distribution. We then have

$$\mu_{n,n+1} \sim \mathcal{N}(\bar{x}_{n,n+1}, C_{n,n+1}),$$

where $\bar{x}_{n,n+1} = \bar{x}_n(\theta_{n+1})$ and $C_{n,n+1} = C_n(\theta_{n+1}, \theta_{n+1})$.

An appealing choice of directions would then be the "adaptive" BayesCG search directions. These are obtained by running BayesCG with prior given by $\mu_{n,n+1}$, so that the prior mean / initial iterate is $\bar{x}_{n,n+1}$ and the prior covariance is $C_{n,n+1}$. Using Eq. (4), this leads to search directions of the form

$$\begin{aligned} s_i = r_{i-1} &- r_{i-1} A_{n+1} C_0(\theta_{n+1}, \theta_{n+1}) A_{n+1}^\top s_{i-1} \cdot s_{i-1} \\ &- r_{i-1} A_{n+1} K_n(\theta_{n+1}) G_n^{-1} K_n^\top(\theta_{n+1}) A_{n+1}^\top s_{i-1} \cdot s_{i-1}. \end{aligned}$$

5.3 Practical Implementation

There are few more nuances to consider while implementing this practically. We present the conjugate gradient (PCG) and algorithms for updating the companion regression model (CompCG) algorithm in Algorithm 1 and Algorithm 2. Optimising hyperparameters for the regression model forms an important part our algorithm. While we do not identify a single best method to do this, we discuss our approach as well as efficient implementation employing Cholesky factor updating in Section B.1. Limiting M_n—the size of G_n—is important to keep our method computationally feasible, and we present a few ways to ensure this by limiting the size of the search direction matrices, limiting the frequency of updates and truncation of information in Section B.1.1. We also analyse the computational complexity of this companion regression model in Section B.2.

6 Experiments

We now turn to empirical evaluation of the novel solver. In Section 6.1 we perform a simulation study in an idealised setting, while in Section 6.2 we consider application to a more realistic GP hyperparameter optimisation problem. A python implementation of this solver is available on `GitHub`[2].

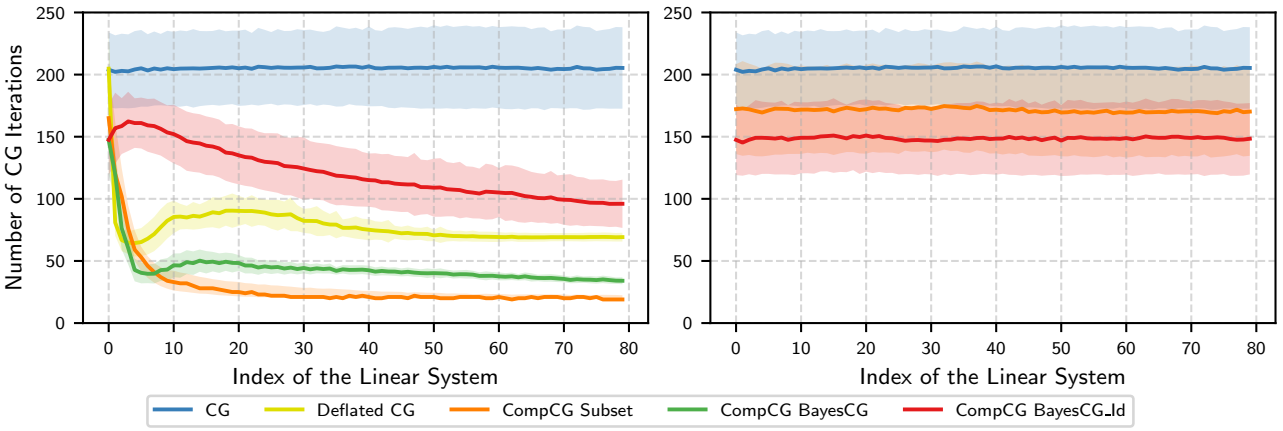
6.1 Simulation Study

Experimental Setup For this simulation study, we aim to evaluate performance in the correctly specified setting. To simulate an optimisation problem let $\Theta = \mathbb{R}^{d'}$, $d' = 200$. We take $\theta_i = \theta_{i-1} + \frac{0.05\epsilon_i}{i}$, where $\epsilon_i \in \mathbb{R}^{d'}$ with j^{th} component $\epsilon_{ij} \sim \mathcal{U}(0,1)$ ensuring that the parameters become closer together for larger i. We then sample $\{x_{\theta_i}\}_{i=1}^n$ jointly from the prior. The prior mean was set to zero, and we used the tensor product prior covariance as described in Section 5.1, with $\Sigma = I_d$ and k as Matèrn $\frac{3}{2}$ kernel. The matrix A_θ of dimension $d = 500$ is defined as $A_\theta = U_1 \Lambda_\theta U_1^\top$, where U_1 is an orthogonal matrix drawn from the Haar measure (fixed for all θ), and $\Lambda_\theta = \text{diag}([\theta; U])$ where $U \in \mathbb{R}^{d-d'}$ with $U_j \sim \mathcal{U}(0.8, 100)$ (again fixed for all θ). We then compute the right-hand side from $b_\theta = A_\theta x_\theta$. To explore the complete potential of our method, we do not employ any cautious updating strategies (Section B.1.1) in this section. We run the CG iterations for all the methods up to the default relative tolerance of 10^{-5}.

We first evaluate the performance of different choices of priors and search directions. Fig. 1a shows the number of CG evaluations required for each linear system for each of the methods, averaged over 100 runs, with 1 standard deviation as the shaded region. Subset and BayesCG refer to the search directions described in Section 5.2, while BayesCG_Id refers to the BayesCG search directions computed using zero prior mean and identity prior covariance, as a less computationally expensive varient of BayesCG. We also plot an additional baseline of deflated CG, a recycled deflation-based preconditioner obtained using m Ritz vectors (as described in Gaul (2014), implemented from the Python package `krypy`) as a method that adapts according to the previously solved systems.

We see that all three companion CG (CompCG) variants take a fewer CG iterations to reach the required error tolerance. As the BayesCG search directions are more informative, CompCG BayesCG converges faster than BayesCG_Id; this is because the former uses directions that are adapted to the predictive covariance, i.e. that are aware of what directions have already been explored by the regression model, while the latter are ignorant of this. The subset search directions outperform the BayesCG search directions, however, representing a lower cost but computationally competitive alternative. We also see that the deflation based preconditioner marginally outperforms CompCG for initial linear systems, but as the model is trained more, CompCG (both

[2] https://github.com/hegdedisha/learning_related_systems



(a) Number of Conjugate Gradient Iterations for different choices of search directions, compared to CG and deflated CG.

(b) Number of Conjugate Gradient Iterations for different choices of search directions without learning, compared to CG.

Figure 1: Number of Conjugate Gradient Iterations for different choices of search directions, compared to CG.

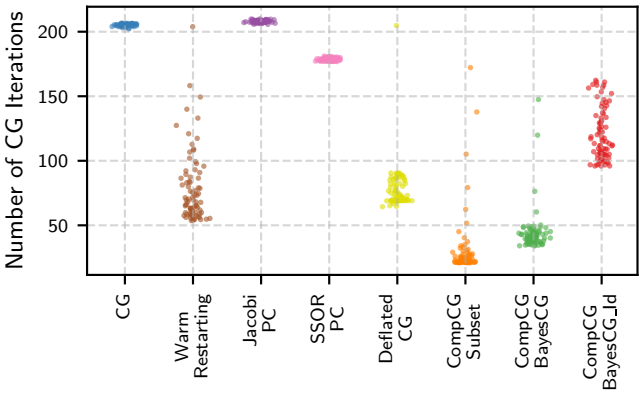


Figure 2: Comparison of number of conjugate gradient iterations for different preconditioners.

subset and BayesCG variants) overtakes deflation preconditioner.

We also examine the effect of the global prior model, as opposed to the preconditioning effect by solely performing pre-iterations with search directions to determine an initial guess and preconditioner. Fig. 1b contrasts Fig. 1a by ignoring the regression model; that is, it assumes that the x_θ are independent across θ. Note that CompCG BayesCG is not displayed here, as the predictive distribution in this setting is simply the prior. We can see that this performs significantly worse than CompCG in Fig. 1a, suggesting that the regression model provides value over-and-above the preconditioning effect from the pre-iterations, i.e. some *global* preconditioning effect has been learned from the data *in addition to* the preconditioning effect from pre-iterations. We also note that in this case BayesCG_Id search directions outperform the Subset search directions. This is due to the fact that the Subset search directions borrows its strength from the learned information.

In Fig. 2, we compare the performance of the CompCG method (for different search directions) with linear solvers that are routinely used in practice:

1. CG,
2. CG with warm-restarting,
3. Preconditioned CG with:
   (a) Jacobi preconditioner,
   (b) SSOR preconditioner with $\omega = 1$, the Gauss-Seidel preconditioner.
4. Deflated CG,

with each point indicating the total number of CG iterations across all linear systems. For further details on preconditioners used see Golub and Loan (2013, Section 11.5.3). We notice that all three variants of CompCG outperform both warm restarting and the Jacobi preconditioner, while CompCG Subset and BayesCG also outperform the SSOR preconditioner and deflated CG.

6.2 GP Hyperparameter Optimisation Problem

We now consider a more applied setting in which related linear systems arise. We consider a GP regression problem on the ERA5 global 2m temperature dataset (Hersbach et al., 2023), for a single timestamp on 1st January 2024. To perform this regression problem one must optimise the hyperparameters of the GP. The standard approach to doing so is by minimising the log-marginal likelihood given by,

$$\log p(y|T,\phi) = -\frac{1}{2}y^\top k_\phi(T,T)^{-1}y - \frac{1}{2}\log\det k_\phi(T,T) - \frac{n}{2}\log(2\pi),$$

results in a sequence of linear systems of the form $k_\phi(T,T)z = y$ that must be solved (see Rasmussen and Williams (2006, Chapter 5)). We use a Matèrn 3/2 kernel function, setting the kernel hyperparameters to be the length-scale, amplitude and noise variance (collectively denoted by ϕ). These define the parameter space Θ for the sequence of regression problems. We minimise the negative log-likelihood using the `scipy` (Virtanen et al., 2020) implementation of L-BFGS (as described in Byrd et al. (1995); Zhu et al. (1997)). The log determinant is approximated using approximation methods from Wenger et al. (2022a), to avoid the cubic cost this would otherwise represent.

d	162	648	800	1035	1352	1800	2592	4050	7200	10368	16200
Time for solving the linear systems (sec)											
CG	0.25	0.85	0.99	1.55	1.32	3.13	7.09	73.31	193.94	347.00	1671.25
CompCG Subset	**0.21**	**0.43**	0.51	**0.79**	1.89	1.47	3.09	14.19	**48.09**	131.81	**177.77**
CompCG BayesCG	0.24	0.53	**0.48**	0.79	**1.20**	**1.42**	**2.51**	**13.22**	59.34	**95.54**	272.27
Average time for solving linear systems per optimiser iteration (sec)											
CG	0.0031	0.0112	0.0118	0.0185	0.0220	0.0373	0.0844	0.8728	2.8521	6.6730	19.8959
CompCG Subset	0.0035	0.0083	0.0106	**0.0116**	0.0224	0.0263	0.0514	**0.2729**	**0.7514**	1.6476	4.0402
CompCG BayesCG	**0.0029**	**0.0069**	**0.0086**	0.0142	**0.0215**	**0.0253**	**0.0419**	0.2754	0.8726	**1.4929**	**3.7815**
Wall-time (sec)											
CG	0.82	4.78	4.99	7.50	**8.75**	24.38	51.84	407.08	1475.93	4554.74	11241.33
CompCG Subset	**0.62**	**2.80**	**3.41**	6.46	10.86	**13.79**	59.34	404.10	**1202.75**	**2963.18**	8440.28
CompCG BayesCG	0.81	2.90	3.84	**6.09**	14.50	23.33	**49.27**	**297.70**	1955.75	3135.40	**7469.10**

Table 1: Results of NLL optimisation for the regression problem in Section 6.2 for varying d.

This dataset has a spatial resolution of approximately 31km, which results in a grid of approximately 1 million points. We consider subsets of this grid with varying resolution, leading to regression training sets of differing size. Unlike Section 6.1, we build a more conservative companion regression model here using measures explained in Section B.1.1. We update the model if the number of CG iterations is high, and reset the model if we see a significant jump in the NLL (indicated by very high number of CG iterations) indicating that a new computational regime has been encountered. We also restrict the size of the training dataset to 4, to ensure fast inversion of G_n. Table 1 shows that time required for the optimiser to converge for each of CG, CompCG Subset and CompCG BayesCG, for varying sizes d of the linear systems to be solved (we do not include CompCG BayesCG_Id here as it shows poor performance for the experiments in Section C.1). A more detailed version of this table is presented in the appendix as Table 2. All the computations were performed on University of Southampton's Iridris5 cluster on a single node with 40 CPU cores.

Both versions of CompCG outperform CG in terms of total time required for solving the linear systems and average time for solving linear systems per optimiser iteration for the all the cases, with the difference generally increasing as the d increases. However, this dominance is not clearly reflected in wall-times. We speculate this is due to a combination of the optimiser taking a suboptimal path through the parameter space before reaching the minimum NLL and the randomness in the calculation of the determinant. This is a pathology of the integration of CompCG with the optimiser that we will aim to study in future work, and could perhaps be addressed by better log determinant approximation routines and more careful optimiser choice.

7 Conclusion

We introduced a companion regression model for the conjugate gradient linear solver that provides an efficient initial guess and preconditioner pair for fast solution of related linear systems. We analysed the properties of this choice theoretically, and we highlighted through our simulations that this can drastically decrease the number of iterations of the linear solver and improve the computation time in problems that employ linear solvers in a loop, such as hyperparameter optimisation problems.

The primary limitation of this companion regression model is the additional time taken for the computation of the regression model. Even though the efficiency of the initial guess and the preconditioner obtained reduces the number of conjugate gradient iterations required for the traditional solver, this reduction in time does not always make up for the additional time required to maintain the regression model. It should be noted that the regression model here is implemented Python; more efficient low-level implementation would likely address this in part.

Along with the more efficient implementation, there are several interesting opportunities for future work. Although we empirically see that conditioning on previously solved system to obtain the preconditioner improves the performance (Figs. 1a and 1b), our theory in Section 4 about recycling subspaces does not explicitly provide us the motivation to use the learned data. We plan to explore this phenomenon more theoretically in our future work. We do not delve into tuning the hyperparameter for the regression model here and it would be interesting to explore more sophisticated hyperparameter optimisation techniques. We use a general separable covariance function for the regression model here, but it would be useful to exploit a problem-specific covariance function which would make the regression model more efficient. Another interesting approach would be to utilise thinned search directions to update the regression model, as Fig. 4 indicates some search directions might be more informative than others. Finally, we focus on symmetric positive definite matrices and the CG solver in the work. But as mentioned before, our regression model for the companion solver only requires invertible linear systems, and thus we plan to extend this companion solver framework to more general solvers in the future.

Acknowledgements

JC was supported by EPSRC grant EP/FY001028/F1.

References

A. P. Austin, N. Chalmers, and T. Warburton. Initial guesses for sequences of linear systems in a GPU-accelerated incompressible flow solver. *SIAM Journal on Scientific Computing*, 43(4):C259–C289, 2021. doi: 10.1137/20M1368677.

L. Bergamaschi. A survey of low-rank updates of preconditioners for sequences of symmetric linear systems. *Algorithms*, 13(4):100, 2020. doi: 10.3390/a13040100.

D. S. Bernstein. *Matrix mathematics: theory, facts, and formulas*. Princeton University Press, Princeton, N.J, 2nd ed edition, 2009. ISBN 978-0-691-13287-7 978-0-691-14039-1.

R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995. doi: 10.1137/0916069.

D. Calvetti and E. Somersalo. Priorconditioners for linear systems. *Inverse problems*, 21(4):1397, 2005. doi: 10.1088/0266-5611/21/4/014.

K. Carlberg, V. Forstall, and R. Tuminaro. Krylov-subspace recycling via the POD-augmented conjugate-gradient method. *SIAM Journal on Matrix Analysis and Applications*, 37(3):1304–1336, Jan. 2016. ISSN 0895-4798. doi: 10.1137/16M1057693.

A. Carr, E. de Sturler, and S. Gugercin. Preconditioning parametrized linear systems. *SIAM Journal on Scientific Computing*, 43(3):A2242–A2267, Jan. 2021. ISSN 1064-8275. doi: 10.1137/20M1331123.

K. Chen. *Matrix Preconditioning Techniques and Applications*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge, 2005. ISBN 978-0-521-83828-3.

M. Clemens, M. Wilke, and T. Weiland. Extrapolation strategies in numerical schemes for transient magnetic field simulations. *IEEE Transactions on Magnetics*, 39(3):1171–1174, May 2003. ISSN 1941-0069. doi: 10.1109/TMAG.2003.810523.

J. Cockayne and A. Duncan. Probabilistic gradients for fast calibration of differential equation models. *SIAM/ASA Journal on Uncertainty Quantification*, 9(4):1643–1672, Jan. 2021. ISSN 2166-2525. doi: 10.1137/20m1364424.

J. Cockayne, C. J. Oates, I. C. Ipsen, and M. Girolami. A Bayesian conjugate gradient method (with discussion). *Bayesian Analysis*, 14(3), Sept. 2019. ISSN 1936-0975. doi: 10.1214/19-BA1145.

H. A. Daas, L. Grigori, P. Hénon, and P. Ricoux. Recycling krylov subspaces and truncating deflation subspaces for solving sequence of linear systems. *ACM Transactions on Mathematical Software*, 47(2):1–30, June 2021. ISSN 0098-3500, 1557-7295. doi: 10.1145/3439746.

F. de Roos and P. Hennig. Krylov subspace recycling for fast iterative least-squares in machine learning. 2017. doi: 10.48550/arXiv.1706.00241.

J. Erhel and F. Guyomarc'h. An augmented conjugate gradient method for solving consecutive symmetric positive definite linear systems. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1279–1299, Jan. 2000. ISSN 0895-4798. doi: 10.1137/S0895479897330194.

P. F. Fischer. Projection techniques for iterative solution of Ax = b with successive right-hand sides. *Computer Methods in Applied Mechanics and Engineering*, 163(1):193–204, Sept. 1998. ISSN 0045-7825. doi: 10.1016/S0045-7825(98)00012-7.

J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23(2), Jan. 2001. ISSN 1064-8275. doi: 10.1137/S1064827500373231.

J. Gallier. The Schur complement and symmetric positive semidefinite (and definite) matrices. *Penn Engineering*, pages 1–12, 2010.

A. Gaul. *Recycling Krylov subspace methods for sequences of linear systems – analysis and applications*. PhD thesis, July 2014.

G. H. Golub and C. F. V. Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, 4 edition, 2013. ISBN 978-1-4214-0794-4 978-1-4214-0859-0.

K. Hayami. Convergence of the conjugate gradient method on singular systems. 2020. doi: 10.48550/arXiv.1809.00793.

D. Hegde, M. Adil, and J. Cockayne. Calibrated computation-aware Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 2098–2106. PMLR, 2025.

P. Hennig. Probabilistic interpretation of linear solvers. *SIAM Journal on Optimization*, 25(1):234–260, Jan. 2015. ISSN 1052-6234. doi: 10.1137/140955501.

H. Hersbach, B. Bell, P. Berrisford, G. Biavati, A. Horányi, J. Muñoz Sabater, J. Nicolas, C. Peubey, R. Radu, I. Rozum, D. Schepers, A. Simmons, C. Soci, D. Dee, and J.-N. Thépaut. ERA5 hourly data on single levels from 1940 to present, 2023. Accessed on 24-09-2024. Licence permits free use for any lawful purpose; see https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels for more details.

P. Häusner, O. Öktem, and J. Sjölund. Neural incomplete factorization: learning preconditioners for the conjugate gradient method. Feb. 2024. doi: 10.48550/arXiv.2305.16368.

E. F. Kaasschieter. Preconditioned conjugate gradients for solving singular systems. *Journal of Computational and Applied Mathematics*, 24(1):265–275, Nov. 1988. ISSN 0377-0427. doi: doi/10.1016/0377-0427(88)90358-5.

M. Li. *Recycling Preconditioners for Sequences of Linear Systems and Matrix Reordering*. PhD thesis, Virginia Tech, 2015.

Y. Li, P. Y. Chen, T. Du, and W. Matusik. Learning preconditioners for conjugate gradient PDE solvers. In *International Conference on Machine Learning*, pages 19425–19439. PMLR, 2023.

R. Markovinović and J. D. Jansen. Accelerating iterative solution methods using reduced-order models as solution predictors. *International Journal for Numerical Methods in Engineering*, 68(5):525–541, 2006. ISSN 1097-0207. doi: 10.1002/nme.1721.

M. Pförtner, J. Wenger, J. Cockayne, and P. Hennig. Computation-aware Kalman filtering and smoothing. In *International Conference on Artificial Intelligence and Statistics*, pages 2071–2079. PMLR, 2025.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*, volume 2. MIT press Cambridge, MA, 2006. ISBN 026218253X.

T. W. Reid, I. C. F. Ipsen, J. Cockayne, and C. J. Oates. BayesCG as an uncertainty aware version of CG. Oct. 2022. doi: 10.48550/arXiv.2008.03225.

Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing*, 21 (5):1909–1926, Jan. 2000. ISSN 1064-8275. doi: 10.1137/S1064829598339761.

K. S. Shterev. Iterative process acceleration of calculation of unsteady, viscous, compressible, and heat-conductive gas flows. *International Journal for Numerical Methods in Fluids*, 77(2):108–122, 2015. ISSN 1097-0363. doi: 10.1002/fld.3979.

L. Tatzel, J. Wenger, F. Schneider, and P. Hennig. Accelerating non-conjugate gaussian processes by trading off computation for uncertainty. *Transactions on Machine Learning Research*, 2025.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

N. Vyas, D. Hegde, and J. Cockayne. Randomised postiterations for calibrated BayesCG. In *First International Conference on Probabilistic Numerics*, 2025.

J. Wenger, G. Pleiss, P. Hennig, J. Cunningham, and J. Gardner. Preconditioning for scalable Gaussian process hyperparameter optimization. In *International Conference on Machine Learning*, pages 23751–23780. PMLR, 2022a.

J. Wenger, G. Pleiss, M. Pförtner, P. Hennig, and J. P. Cunningham. Posterior and computational uncertainty in Gaussian processes. *Advances in Neural Information Processing Systems*, 35:10876–10890, Dec. 2022b.

J. Wenger, K. Wu, P. Hennig, J. R. Gardner, G. Pleiss, and J. P. Cunningham. Computation-aware Gaussian processes: Model selection and linear-time inference. In *Advances in Neural Information Processing Systems*, 2024.

S. Ye, Y. Lin, L. Xu, and J. Wu. Improving initial guess for the iterative solution of linear equation systems in incompressible flow. *Mathematics*, 8(1):119, Jan. 2020. ISSN 2227-7390. doi: 10.3390/math8010119.

J. Zhang, G. Zhu, R. W. Heath Jr., and K. Huang. Grassmannian learning: Embedding geometry awareness in shallow and deep learning. Aug. 2018. doi: 10.48550/arXiv.1808.02229.

L. Zheng, L. Yang, and Y. Liang. A conjugate gradient projection method for solving equations with convex constraints. *Journal of Computational and Applied Mathematics*, 375:112781, Sept. 2020. ISSN 0377-0427. doi: 10.1016/j.cam.2020.112781.

C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997. doi: 10.1145/279232.279236.

M. A. Álvarez, L. Rosasco, and N. D. Lawrence. Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning*, 4(3):195–266, June 2012. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000036.

A Proofs

Proof of Lemma 3. We define $T = \{\theta_1, \ldots, \theta_n\} \subset \Theta$. Let

$$\mathcal{A}_n^\top \mathcal{S}_n = \begin{pmatrix} A_1^\top S_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_n^\top S_n \end{pmatrix}$$

and note that G_n can be expressed as:

$$G_n = (\mathcal{A}_n^\top \mathcal{S}_n)^\top C_0(T,T) \mathcal{A}_n^\top \mathcal{S}_n.$$

It follows from Golub and Loan (2013, Theorem 4.2.1) that G_n is positive definite (and therefore invertible) if $\mathcal{A}_n^\top \mathcal{S}_n$ has full column rank. Clearly,

$$\operatorname{rank}(\mathcal{A}_n^\top \mathcal{S}_n) = \sum_{i=1}^n \operatorname{rank}(A_i^\top S_i) = \sum_{i=1}^n \operatorname{rank}(S_i^\top A_i) = \sum_{i=1}^n \operatorname{rank}(S_i^\top) = \sum_{i=1}^n m_i = M_n \tag{6}$$

which follows from the facts that A_i is invertible and S_i have full column rank by assumption. Therefore $\mathcal{A}_n^\top \mathcal{S}_n$ has full column rank, completing the proof. □

Proof of Theorem 4. We know that

$$C_{n-1}(\theta,\theta) = C_0(\theta,\theta) - K_{n-1}(\theta) G_{n-1}^\top K_{n-1}^\top(\theta).$$

Consider the matrix M_1

$$M_1 = \begin{bmatrix} G_{n-1} & K_{n-1}^\top(\theta) \\ K_{n-1}(\theta) & C_0(\theta,\theta) \end{bmatrix}$$

For a finite set $T \subset \Theta$, $T = \{\theta_1, \ldots, \theta_{n_T}\}$, we define

$$C_0(T,\theta) = \begin{bmatrix} C_0(\theta_1,\theta) & \cdots & C_0(\theta_{n_T},\theta) \end{bmatrix}$$

and similarly

$$C_0(T,T) = \begin{bmatrix} C_0(\theta_1,\theta_1) & \cdots & C_0(\theta_1,\theta_{n_T}) \\ \vdots & \ldots & \vdots \\ C_0(\theta_{n_T},\theta_1) & \cdots & C_0(\theta_{n_T},\theta_{n_T}) \end{bmatrix}$$

so that $C_0(T,\theta) \in \mathbb{R}^{(dn_T)\times d}$ and $C_0(T,T) \in \mathbb{R}^{(dn_T)\times(dn_T)}$. Next, let

$$\mathcal{A}_{n-1} = \begin{bmatrix} A_1 & \cdots & 0 \\ \vdots & \ldots & \vdots \\ 0 & \cdots & A_{n-1} \end{bmatrix}$$
$$\mathcal{S}_{n-1} = \begin{bmatrix} S_1 & \cdots & 0 \\ \vdots & \ldots & \vdots \\ 0 & \cdots & S_{n-1} \end{bmatrix}.$$

Also define $T_{n-1} = \{\theta_1, \ldots, \theta_{n_T-1}\}$, and $\theta = \theta_{n_T}$. Thus we have that

$$K_{n-1}(\theta) = \mathcal{S}_{n-1}^\top \mathcal{A}_{n-1} C_0(T_{n-1},\theta)$$
$$G_{n-1} = \mathcal{S}_{n-1}^\top \mathcal{A}_{n-1} C_0(T_{n-1},T_{n-1}) \mathcal{A}_{n-1}^\top \mathcal{S}_{n-1}.$$

Next consider the Schur complement $M_1/C_0(\theta,\theta)$, defined as

$$\begin{aligned} M_1/C_0(\theta,\theta) &= G_{n-1} - K_{n-1}^\top(\theta) C_0(\theta,\theta)^{-1} K_{n-1}(\theta) \\ &= \mathcal{S}_{n-1}^\top \mathcal{A}_{n-1} C_0(T_{n-1},\Theta) \mathcal{A}_{n-1}^\top \mathcal{S}_{n-1} - \mathcal{S}_{n-1}^\top \mathcal{A}_{n-1} C_0(T_{n-1},\theta) C_0(\theta,\theta)^{-1} C_0(\theta,T_{n-1}) \mathcal{A}_{n-1}^\top \mathcal{S}_{n-1} \\ &= M_1^1 M_1^2 (M_1^1)^\top \end{aligned}$$

where

$$M_1^1 := \mathcal{S}_{n-1}^\top \mathcal{A}_{n-1}$$
$$M_1^2 := C_0(T_{n-1},T_{n-1}) - C_0(T_{n-1},\theta) C_0(\theta,\theta)^{-1} C_0(\theta,T_{n-1})$$

Next we note that M_1^2 is again a Schur complement, this time of the matrix M_2 given by

$$M_2 = \begin{bmatrix} C_0(T_{n-1}, T_{n-1}) & C_0(T_{n-1}, \theta_n) \\ C_0(\theta, T_{n-1}) & C_0(\theta, \theta) \end{bmatrix}$$

so that $M_1^2 = M_2/C_0(\theta, \theta)$. By assumption that C_0 is positive definite, both $C_0(\theta, \theta)$ and M_2 are positive definite. Thus, from Gallier (2010, Proposition 2.1), the Schur complement M_1^2 is positive-definite. We have seen in Eq. (6) that M_1^1 has full row-rank. Therefore, from Golub and Loan (2013, Theorem 4.2.1), $M_1/C_0(\theta, \theta)$ is positive-definite, and thus non-singular.

Since G_{n-1}, $C_0(\theta_n, \theta_n)$ and $M_1/C_0(\theta_n, \theta_n)$ are non-singular, from the matrix inversion lemma (Bernstein, 2009, Corollary 2.8.8.), we have that $C_{n-1}(\theta_n, \theta_n) = M_1/G_{n-1}$ is also non-singular. Thus $\text{rank}(C_{n-1}(\theta_n, \theta_n)) = d$.

□

Lemma 8. *Given the posterior covariance $C_{n-1}(\theta, \theta)A_n^T S_n$ computed without including the evaluated location θ, posterior computed including the evaluating location can be computed iteratively as*

$$C_n(\theta, \theta) = C_{n-1}(\theta, \theta) - C_{n-1}(\theta, \theta)A_n^T S_n \left(S_n^T A_n C_{n-1}(\theta, \theta) A_n^T S_n\right)^{-1} S_n^T A_n C_{n-1}(\theta, \theta) \tag{7}$$

Proof. We can see that

$$\begin{aligned} C_n(\theta, \theta) &= C_0(\theta, \theta) \\ &- \begin{bmatrix} K_{n-1}(\theta) & C_0(T_{n-1}, \theta_n) \end{bmatrix} \begin{bmatrix} G_{n-1} & K_{n-1}(\theta_n)^\top A_n^\top S_n \\ (K_{n-1}(\theta_n)^\top A_n^\top S_n)^\top & S_n^\top A_n C_0(\theta_n, \theta_n) A_n^\top S_n \end{bmatrix}^{-1} \begin{bmatrix} K_{n-1}(\theta)^\top \\ C_0(T_{n-1}, \theta_n) \end{bmatrix} \end{aligned}$$

The Schur complement of the matrix of the matrix to be inverted simplifies to $S_n^\top A_n C_{n-1}(\theta_n, \theta_n) A_n^\top S_n$. As seen in proof of Lemma 3, using Theorem 4, this Schur complement is invertible. Thus the matrix inversion lemma (Bernstein, 2009, Corollary 2.8.8.) can be used here. Using this block matrix inversion result to simplify the above expression gives us Lemma 8. □

Proof of Theorem 5. From Lemma 8, we have that

$$C_n(\theta_n, \theta_n) = C_{n-1}(\theta_n, \theta_n) - C_{n-1}(\theta_n, \theta_n)A_n^\top S_n \left(S_n^\top A_n C_{n-1}(\theta_n, \theta_n) A_n^\top S_n\right)^{-1} S_n^\top A_n C_{n-1}(\theta_n, \theta_n)$$

Consider

$$M_3 = \begin{bmatrix} S_n^\top A_n C_{n-1}(\theta_n, \theta_n) A_n^\top S_n & S_n^\top A_n C_{n-1}(\theta_n, \theta_n) \\ C_{n-1}(\theta_n, \theta_n) A_n^\top S_n & C_{n-1}(\theta_n, \theta_n) \end{bmatrix}$$

Clearly $C_n(\theta_n, \theta_n) = M_3/S_n^\top A_n C_{n-1}(\theta_n, \theta_n) A_n^\top S_n$. From Guttmans rank additivity theorem (Bernstein, 2009, Fact 6.5.6), we have that

$$\text{rank}(M_3) = \text{rank}(S_n^\top A_n C_{n-1}(\theta_n, \theta_n) A_n^\top S_n) + \text{rank}(C_n(\theta_n, \theta_n)). \tag{8}$$

We now calculate $\text{rank}(S_n^\top A_n C_{n-1}(\theta_n, \theta_n) A_n^\top S_n)$: since from Theorem 4, we have that $C_{n-1}(\theta_n, \theta_n)$ has full rank and is thus invertible. Since A_n is also invertible, from Bernstein (2009, Proposition 2.6.3) we have that

$$\text{rank}(S_n^\top A_n C_{n-1}(\theta_n, \theta_n) A_n^\top S_n) = \text{rank}(S_n) = m_n.$$

Next we calculate $\text{rank}(M_3)$. Clearly M_3 can be decomposed as follows

$$M_3 = \underbrace{\begin{bmatrix} S_n^\top A \\ I_d \end{bmatrix}}_{M_3^1} \underbrace{\begin{bmatrix} C_{n-1}(\theta, \theta) \end{bmatrix}}_{M_3^2} \underbrace{\begin{bmatrix} A_n^\top S_n & I_d \end{bmatrix}}_{(M_3^1)^\top}.$$

Here M_3^1 has full column rank and M_3^2 has full rank, thus $\text{rank}(M_3^1 M_3^2) = \text{rank}(M_3^2) = d$ (Bernstein, 2009, Proposition 2.6.3 and 2.6.1). Similarly, $(M_3^1)^\top$ has full row rank, and thus $\text{rank}(M_3) = \text{rank}((M_3^1 M_3^2) M_3^{1\top}) = \text{rank}(M_3^1 M_3^2) = d$. Plugging these numbers into Eq. (8) and rearranging we obtain

$$\text{rank}(C_n(\theta_n, \theta_n)) = d - m_n$$

as required.

For the null space we simply apply $A_n^\top S_n$ on the right to show this directly:

$$\begin{aligned} C_n(\theta_n, \theta_n) &= C_0(\theta_n, \theta_n) - K_n(\theta_n) G_n^{-1} K_n^\top(\theta_n) \\ C_n(\theta_n, \theta_n) A_n^\top S_n &= C_0(\theta_n, \theta_n) A_n^\top S_n - K_n(\theta_n) G_n^{-1} K_n^\top(\theta_n) A_n^\top S_n \end{aligned}$$

Now examining the rightmost matrix we see that:

$$G_n^{-1}K_n^\top(\theta_n)A_n^\top S_n = \begin{bmatrix} G_{n-1} & K_n(\theta_{n-1})^\top A_n^\top S_n \\ S_n^\top A_n K_n(\theta_{n-1}) & S_n^\top A_n C_0(\theta_n,\theta_n)A_n^\top S_n \end{bmatrix}^{-1} \begin{bmatrix} K_n(\theta_{n-1})^\top A_n^\top S_n \\ S_n^\top A_n C_0(\theta_n,\theta_n)A_n^\top S_n \end{bmatrix} = \begin{bmatrix} 0 \\ I_{m_n} \end{bmatrix} \tag{9}$$

since the matrix to which the inverse applied is a column of G_n. Therefore

$$\begin{aligned} C_n(\theta_n,\theta_n)A_n^\top S_n &= C_0(\theta_n,\theta_n)A_n^\top S_n - \begin{bmatrix} K_n(\theta_{n-1}) & C_0(\theta_n,\theta_n)A_n^\top S_n \end{bmatrix} \begin{bmatrix} 0 \\ I_{m_n} \end{bmatrix} \\ &= C_0(\theta_n,\theta_n)A_n^\top S_n - C_0(\theta_n,\theta_n)A_n^\top S_n \\ &= 0. \end{aligned}$$

Thus $\text{span}(A_n^\top S_n) \in \text{null}(C_n(\theta_n,\theta_n))$. Since $\text{rank}(C_n(\theta_n,\theta_n)) = d - m_n$, $\text{nullity}(C_n(\theta_n,\theta_n)) = m_n$. Since $\text{rank}(A_n^\top S_n) = m_n$, it follows that $\text{span}(A_n^\top S_n)$ is the whole null space of $C_n(\theta_n,\theta_n)$, completing the proof.

□

Proof of Theorem 6. Again we can show this directly by applying $S_n^\top A_n$ to $\bar{x}_n(\theta_n)$. We have

$$\begin{aligned} S_n^\top A_n \bar{x}_n(\theta_n) &= S_n^\top A_n x_0(\theta_n) + S_n^\top A_n K_n(\theta_n) G_n^{-1} z_n \\ &= S_n^\top A_n x_0(\theta) + \begin{bmatrix} 0 & I_{m_n} \end{bmatrix} \begin{bmatrix} z_{n-1} \\ y_n - S_n^\top A_n x_0(\theta_n) \end{bmatrix} \end{aligned}$$

from Eq. (9). Therefore

$$\begin{aligned} S_n^\top A_n \bar{x}_n(\theta_n) &= S_n^\top A_n x_0(\theta) + y_n - S_n^\top A_n x_0(\theta_n) \\ &= S_n^\top A_n x^\star(\theta_n) \end{aligned}$$

as required. □

Proof of Theorem 7. The PCG algorithm for positive-definite preconditioner is defined as in Algorithm 1. We prove here that $x_k \to x^*$ in this algorithm even when the preconditioner is positive-semi definite, given a specific structure of the initial guess x_0 as stated in the theorem.

First note that since P is symmetric, it is diagonalisable, i.e. $P = U\Lambda U^\top$, where U is a unitary matrix and Λ is a diagonal matrix with diagonal entries as the eigenvalues of P. As P is a positive semi-definite matrix, we can write Λ as

$$\Lambda = \begin{pmatrix} \Lambda_1 & 0 \\ 0 & 0 \end{pmatrix}$$

where Λ_1 is a diagonal matrix with positive entries, i.e.,

$$\Lambda_1 = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_r \end{pmatrix}, \lambda_1 \geq \lambda_2 \ldots \lambda_r \geq 0, r = \text{rank}(P) = \dim(\mathcal{R}(P)).$$

Thus, Λ_1 is a symmetric positive definite matrix. We now define partitions U_1 and U_2 of matrix U, i.e.,

$$U = \begin{bmatrix} U_1 & U_2 \end{bmatrix}$$

such that the columns of U_1 and U_2 are orthonormal bases of $\mathcal{R}(P)$ and $\mathcal{R}(P)^\perp$ respectively. We now re-express the linear system in terms of U:

$$\begin{aligned} \tilde{x} &= U^\top x \\ \tilde{b} &= U^\top b \\ \tilde{A} &= U^\top A U \end{aligned}$$

so that clearly $\tilde{A}\tilde{x} = \tilde{b}$. We now express the quantities used the Algorithm 1 in terms of the projected system. Considering the initial steps, the initial residual is given by

$$\tilde{r}_0 = U^\top r_0 = \begin{bmatrix} \tilde{r}_0^{(1)} \\ \tilde{r}_0^{(2)} \end{bmatrix}$$

while

$$\tilde{z}_0 = U^\top z_0 = \begin{bmatrix} \Lambda_1 \tilde{r}_0^{(1)} \\ 0 \end{bmatrix}$$

and finally

$$\tilde{p}_0 = U^\top p_0 = \begin{bmatrix} \Lambda_1 \tilde{r}_0^{(1)} \\ 0 \end{bmatrix}$$

Thus we see that in the basis defined by U, the first search direction modifies $\tilde{x}_0^{(1)}$ by taking a step in the direction $\Lambda_1 \tilde{r}_0^{(1)}$, but makes no change to $\tilde{x}_0^{(2)} = \tilde{x}^{\star(2)}$.

Continuing to the iterates, proceeding inductively,

$$\tilde{\alpha}_k = \frac{\tilde{r}_k^\top \tilde{z}_k}{\tilde{r}_{k-1}^\top \tilde{z}_{k-1}} = \frac{(\tilde{r}_k^{(1)})^\top \tilde{z}_k^{(1)}}{(\tilde{r}_{k-1}^{(1)})^\top \tilde{z}_{k-1}^{(1)}}$$

Note that both the numerator and denominator simplify as $\tilde{z}_k^{(2)} = 0$. Similarly,

$$\tilde{x}_{k+1} = U^\top x_{k+1} = \begin{bmatrix} \tilde{x}_k^{(1)} + \tilde{\alpha}_k \tilde{p_k}^{(1)} \\ \tilde{x}_k^{(2)} \end{bmatrix}$$

i.e. the second component of $\tilde{x}_{k+1}$ is unchanged, and

$$\tilde{r}_{k+1} = U^\top r_{k+1} = \begin{bmatrix} \tilde{r}_k^{(1)} - \tilde{\alpha}_k \tilde{A}^{(1)} \tilde{p}_k^{(1)} \\ \tilde{\alpha}_k U_2^\top A U_1 \tilde{p}_k^{(1)} \end{bmatrix}$$

where the last line is due to $\tilde{p}_k^{(2)} = 0$. Next applying the preconditioner we have

$$\tilde{z}_{k+1} = U^\top P r_{k+1} = \begin{bmatrix} \Lambda_1 r_{k+1}^{(1)} \\ 0 \end{bmatrix}.$$

Considering β_k we have

$$\tilde{\beta}_k = \frac{\tilde{r}_{k+1}^\top \tilde{z}_{k+1}}{\tilde{z}_k^\top \tilde{z}_k} = \frac{(r_{k+1}^{(1)})^\top z_{k+1}^{(1)}}{(r_k^{(1)})^\top z_k^{(1)}}$$

and lastly the next search direction is given by

$$\tilde{p}_{k+1} = U^\top p_{k+1} = \begin{bmatrix} \tilde{z}_{k+1}^{(1)} + \tilde{\beta}_k \tilde{p}_k^{(1)} \\ 0 \end{bmatrix}$$

again maintaining the structure of the second component being zero. The norm of the error is given by

$$\|\tilde{r}_k\| = (\tilde{r}_k^\top \tilde{r}_k)^{(1/2)} = \|\tilde{r}_k^{(1)}\| + \|\tilde{r}_k^{(2)}\|$$

We can see a clear structure in these quantities in terms of $\tilde{x}_k^{(1)}$ and $\tilde{x}_k^{(2)}$, and they can be separated. Thus we evaluate these two components separately. Only considering $\tilde{x}_k^{(1)}$ component, we have the following algorithm:

x_0: initial guess
$\tilde{x}_0 = \begin{bmatrix} U_1^\top x_0 \\ U_2^\top x_0 \end{bmatrix} = \begin{bmatrix} \tilde{x}_0^{(1)} \\ \tilde{x}_0^{(2)} \end{bmatrix}$
$\tilde{r}_0^{(1)} = \tilde{b}^{(1)} - U_1^\top A U_1 U_1^\top x_0 - U_1^\top A U_2 U_2^\top x_0$
$\tilde{r}_0^{(1)} = \left(U_1^\top b - U_1^\top A U_2 U_2^\top x_0\right) - U_1^\top A U_1 U_1^\top x_0$
$\tilde{z}_0^{(1)} = \Lambda_1 \tilde{r}_0^{(1)}$
$\tilde{p}_0^{(1)} = \tilde{z}_0^{(1)}$
$k = 0$
while $\|r_k\| > \epsilon$ **do**
    $\tilde{\alpha}_k = \frac{\tilde{r}_k^{(1)\top} \tilde{z}_k^{(1)}}{\tilde{r}_{k-1}^{(1)\top} \tilde{z}_{k-1}^{(1)}}$
    $\tilde{x}_{k+1}^{(1)} = \tilde{x}_k^{(1)} + \tilde{\alpha}_k \tilde{p}_k^{(1)}$

$$
\begin{aligned}
\tilde{r}_{k+1}^{(1)} &= \tilde{r}_k^{(1)} - \tilde{\alpha}_k^{(1)} \tilde{A}\tilde{p}_k^{(1)} \\
\tilde{z}_{k+1}^{(1)} &= \Lambda_1 \tilde{r}_{k+1}^{(1)} \\
\tilde{\beta}_k^{(1)} &= \frac{\tilde{r}_{k+1}^{(1)\top} \tilde{z}_{k+1}^{(1)}}{\tilde{r}_k^{(1)\top} \tilde{z}_k^{(1)}} \\
\tilde{p}_{k+1}^{(1)} &= \tilde{z}_{k+1}^{(1)} + \tilde{\beta}_k \tilde{p}_k^{(1)}
\end{aligned}
$$

Note that $U_1^\top AU$ is positive definite because A is positive definite and U has full column rank (refer to Golub and Loan (2013, Theorem 4.2.1)). The above algorithm is equivalent to Algorithm 1 for solving the system $U_1^\top AU_1 x^\star = \bar{b}$, (where $\bar{b} = U_1^\top b - U_1^\top AU_2U_2^\top x_0$) with positive-definite preconditioner Λ_1. Thus,

$$
\begin{aligned}
\tilde{x}_k^1 &\to (U_1^\top AU_1)^{-1}\bar{b} \\
&= U_1^\top A^{-1}U_1U_1^\top c - U_1^\top A^{-1}U_2(U_2^\top A^{-1}U_2)^{-1}U_2^\top A^{-1}U_1U_1^\top c
\end{aligned}
$$

Where $c = b - AU_2U_2^\top x_0$. The $\tilde{x}_k^{(2)}$ component equal to $\tilde{x}_0^{(2)} = U_2^\top x_0$ throughout the algorithm. We can see that

$$
x_k = U\tilde{x}_k = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \tilde{x}_k^{(1)} \\ \tilde{x}_k^{(2)} \end{bmatrix} = U_1\tilde{x}_k^{(1)} + U_2\tilde{x}_k^{(2)}
$$

With this, we get the following convergence of x_k -

$$
\begin{aligned}
x_k &\to U_1U_1^\top A^{-1}U_1U_1^\top c - U_1U_1^\top A^{-1}U_2(U_2^\top A^{-1}U_2)^{-1}U_2^\top A^{-1}U_1U_1^\top c + U_2U_2^\top x_0 \\
&= A^{-1}c - A^{-1}U_2(U_2^\top A^{-1}U_2)^{-1}U_2^\top A^{-1}c + U_2U_2^\top x_0
\end{aligned}
$$

Now substituting the value of c,

$$
= A^{-1}b - A^{-1}U_2(U_2^\top A^{-1}U_2)^{-1}\left(U_2^\top A^{-1}b - U_2^\top x_0\right)
$$

Given the condition that the initial guess is correct in the null space of the preconditioner, i.e., $U_2^\top x_0 = U_2^\top A^{-1}b$, our preconditioned Conjugate Gradient algorithm converges to the solution of the linear system $Ax = b$.

□

B Implementation Details

B.1 Practical Implementation

An algorithmic description of the proposed CompCG method is given in Algorithm 2. There are several implementation details which are discussed below.

Cholesky Updates To calculate the posterior mean and covariance we must invert G_n, which would have cost $\mathcal{O}(M_n^3)$ if implemented naively. However, since we are in a sequential data setting we can exploit Cholesky update formulae to reduce this to a cost of $\mathcal{O}(m_n^3)$. Let L_{11} be the lower Cholesky factor of a symmetric positive definite matrix A_{11}. Then the Cholesky decomposition of the block symmetric positive definite matrix

$$
A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix}
$$

can be computed as

$$
L = \begin{pmatrix} L_{11} & 0 \\ L_{12} & L_{22} \end{pmatrix}
$$

where $L_{12} = (L_{11}^{-1}A_{12})^T$ and L_{22} is the Cholesky factor of $A_{12} - L_{12}L_{12}^T$. Thus, to add a new training data point, adding a new $m_n \times m_n$ block to G_{n-1} of size M_{n-1} would require a Cholesky factor for a matrix of size m_n (as compared to finding a Cholesky factor of size $M_n = \sum_1^n m_n$), a matrix-matrix multiplication of matrices of size $M_{n-1} \times m_n$, and a triangular solve of size M_{n-1} for the computation of the new G_n^{-1}.

Hyperparameter Optimisation As with any GP regression problem, performance of the CompCG method will be strongly affected by the choice of hyperparameters. In this work we do not explore this, and set the hyperparameters manually based on knowledge of the problem at hand. We explore the effect of this choice on our simulation study setting in Section C.1. A more principled approach would be to optimise the marginal likelihood on a small sample from the parameter space (e.g. sampled at random according to some distribution believed to cover significant values of θ, or using a small latin hypercube or sparse grid design). However we leave development of these ideas for future work.

B.1.1 Training Set Size

While the dominant cost of CompCG is $\mathcal{O}(m_n^3)$, if M_n becomes large this will nevertheless begin to dominate the cost of the computation. We therefore discuss several strategies for limiting the size of the training set, below.

Choosing the size of Search Direction matrix A natural way to reduce M_n is to limit the size of the search direction matrix, m_n. This represents a trade-off between more accurate posterior mean and covariance, and computational efficiency. We elect to use an ad-hoc choice of $m_i = \alpha d$ for some $0 < \alpha < 1$. For our experiments we set $\alpha = 0.2$, which seems to result in good performance.

Update Limitation We could also reduce the frequency of training set updates, rather than updating at every $i = 1, \ldots, n$. We have several suggestions here:

1. Update the training set when the number of CG iterations required to solve the linear system is high. A high number of CG iterations indicates that the posterior mean and covariance are not efficient as initial guess and preconditioner respectively, suggesting the model should be improved by incorporating more recent data.
2. Update the training set every j^{th} parameter visited, $j > 1$. This would work best if the parameters explored are near to each other, ensuring the training set does not have irrelevant points.
3. Update the training when the new parameter θ_k is sufficiently "far" from the parameters in the current training set T. This ensures that the training data is updated when the new linear system is different from the already explored parameter space. The decision of how far is sufficiently far in the parameter space is problem specific, however, and the cost of computing this distance could be non-negligible.
4. Optimally introduce search directions from S_n rather than selecting in an "all-or-nothing" fashion. For example, one could select $\tilde{m}_i \ll m_i$ directions that maximally differ from one another (e.g. in terms of a Grassmann distance (Zhang et al., 2018)), or using an SVD to optimally compress the basis (as seen in Pförtner et al. (2025)).

Ultimately, the best approach depends on the problem at hand. For our case of hyperparameter optimisation in Section 6.2, as the parameter space is presumed to change as the optimiser proceeds, leading to a varying local parameter space, we use the number of CG iterations to decide on updating the training set. This decision to update the model is taken after solving the linear system, by adding an additional checkpoint after line 12 in Algorithm 2.

Truncation In addition to reducing the number of training set updates, we could also periodically truncate the training set to retain only the most relevant information. Resetting the model to remove all the data is a natural choice in many applications, e.g. when an optimiser moves to explore a new portion of the parameter space, previous information may no longer be relevant. We exploit this in Section 6.2, where we reset the model when there is a big change in the computed NLL, indicating a change in the parameter space. One could also optimally compress the training set as a whole using similar strategies as discussed in point 4 under "Update limitation", above. Block Cholesky downdates can be also be used to efficiently remove information from G_n. While adding extra blocks to the Cholesky decomposition is rather straightforward, removing blocks is more extensive, and we leave this implementation for future work.

B.2 Complexity Analysis

We analyse the cost required to solve a new linear system of the size d using m_n search directions, when the trained regression model has size $M_{n-1} = \sum_{i=1}^{n-1} m_i$. The computations required for the companion regression model can be spilt into two: (i) updating the model, and (ii) calculating the posterior mean and covariance. We ignore the (possibly non-negligible) cost of computing the search directions as this varies based on the method used, and focus only on the cost of updating and evaluating the regression model, as this is overhead on top of the cost of CG.

(i) Updating the model This requires matrix multiplications costing $\mathcal{O}(m_n d^2)$ and computing the Cholesky factor of G_n. Since the factor of G_{n-1} is available this can be updated as described above, costing $\mathcal{O}(M_{n-1} m_n^2 + M_{n-1}^2 m_n + m_n^3)$.

(ii) Mean and covariance calculations These require two triangular solves costing $\mathcal{O}(M_n^2 d)$ (lines 7 and 8 in Algorithm 2), and matrix multiplications to calculate K_n and posteriors (in lines 5, 9 and 10 of Algorithm 2) costing $\mathcal{O}(M_n d^2 + M_n d)$ resulting in the total cost of $\mathcal{O}(M_n^2 d + M_n d + M_n d^2)$.

Summary In total, the companion regression model would cost $\mathcal{O}(m_n^3 + M_{n-1}^2 m_n + M_n^2 d + m_n^2 M_{n-1} + M_n d^2 + m_n d^2 + M_n d)$. Assuming we keep the size of the training set in check using truncation (i.e. keeping $M_n = \mathcal{O}(m_n)$), ignoring the lower order terms, this simplifies to $\mathcal{O}(m_n^3 + m_n^2 d + m_n d^2)$. In case of an ill-conditioned dense matrix, we argue that as long as the information y_i's are of small size, i.e., $m_n \ll d$ and thus $M_{n-1} \ll d$, this cost is justified as the reduction in the number of CG iterations makes up for the additional cost involved in maintaining the companion regression model.

Algorithm 1 The preconditioned conjugate gradient solver

procedure PCG(A, b, initial guess x_0, preconditioner P, error tolerance ϵ)
  $r_0 = b - Ax_0$
  $z_0 = Pr_0$
  $p_0 = z_0$
  $k = 0$
  while $\|r_k\| > \epsilon$ **do**
    $\alpha_k = \frac{r_k^\top z_k}{r_{k-1}^\top z_{k-1}}$
    $x_{k+1} = x_k + \alpha_k p_k$
    $r_{k+1} = r_k - \alpha_k A p_k$
    $z_{k+1} = P r_{k+1}$
    $\beta_k = \frac{r_{k+1}^\top z_{k+1}}{r_k^\top z_k}$
    $p_{k+1} = z_{k+1} + \beta_k p_k$
  return x_k

Algorithm 2 Conjugate gradient solver with companion regression model

procedure CompCG($\{A_n, b_n, \theta_n\}_{n=1}^l$, error tolerance: ϵ, prior covariance: C_0, type of search direction: SD, size of search direction matrix: m)
  for $n = 1, 2, \cdots l$ **do**
    S = SearchDirection(SD, m, A_n, b_n, θ_n)
    Update T, $\mathcal{A}^\top\mathcal{S}$, z with θ_n, A_n and S
    $K = C_0(\theta_n, T) \cdot \mathcal{A}^\top\mathcal{S}$
    G_{chol} = CholeskyUpdate(G_{chol}, A_n,S, θ_n) ▷ Block Cholesky update
    $\beta_1 = G_{chol}^{-1} K^\top$
    $\beta = G_{chol}^{-1} \beta_1$
    $\bar{x} = \beta^\top \cdot z$ ▷ Posterior predictive mean
    $C = C_0(\theta_n, \theta_n) - \beta^\top \cdot K^\top$ ▷ Posterior predictive covariance
    x_n = PCG(A_n, b_n, $x_0 = \bar{x}$, $P = C$, $\epsilon = \epsilon$)
    return x_n

C Additional Results

C.1 Simulation Study

Fig. 3 shows CG iterations for varying lengthscale as a test to how an incorrectly chosen lengthscale would affect the performance of CompCG. Note that the theoretical results leading to Theorem 7 do not depend on how this length-scale is chosen. However, the additional preconditioning effect observed between Figs. 1a and 1b is likely to be significantly affected by a poor hyperparameter choice. We observe that CompCG Subset and CompCG BayesCG are quite robust. However, CompCG BayesCG_Id runs into precision issues when the lengthscale is high (10), leading to poor performance. We note here that CompCG BayesCG_Id is the worst performing variant among the three CompCG variants, and this experiment reinforces our preference for CompCG Subset and CompCG BayesCG over CompCG BayesCG_Id.

The GP mean from our regression model could be a good initial guess, in principle, independent of the preconditioner. But Fig. 5 (of CG iterations using only initial guess computed from different search directions for the regression model) shows that just using the mean in this way does not significantly reduce the total number of iterations compared to CG. We hypothesise that this is because CG still ends up exploring the space in which the mean has already been identified because of the lack of deflation from the covariance preconditioner. Although this does not directly give us a motivation for extending the prior across the parameter space, there is still some evidence in our paper that the dependence structure imparts additional preconditioning benefit compared to just running some pre-iterations of BayesCG and using these to construct an initial guess and preconditioner, as seen in Fig. 1a and Fig. 1b. This work does not include a theoretical justification for this, but we plan to explore this phenomenon more in future work.

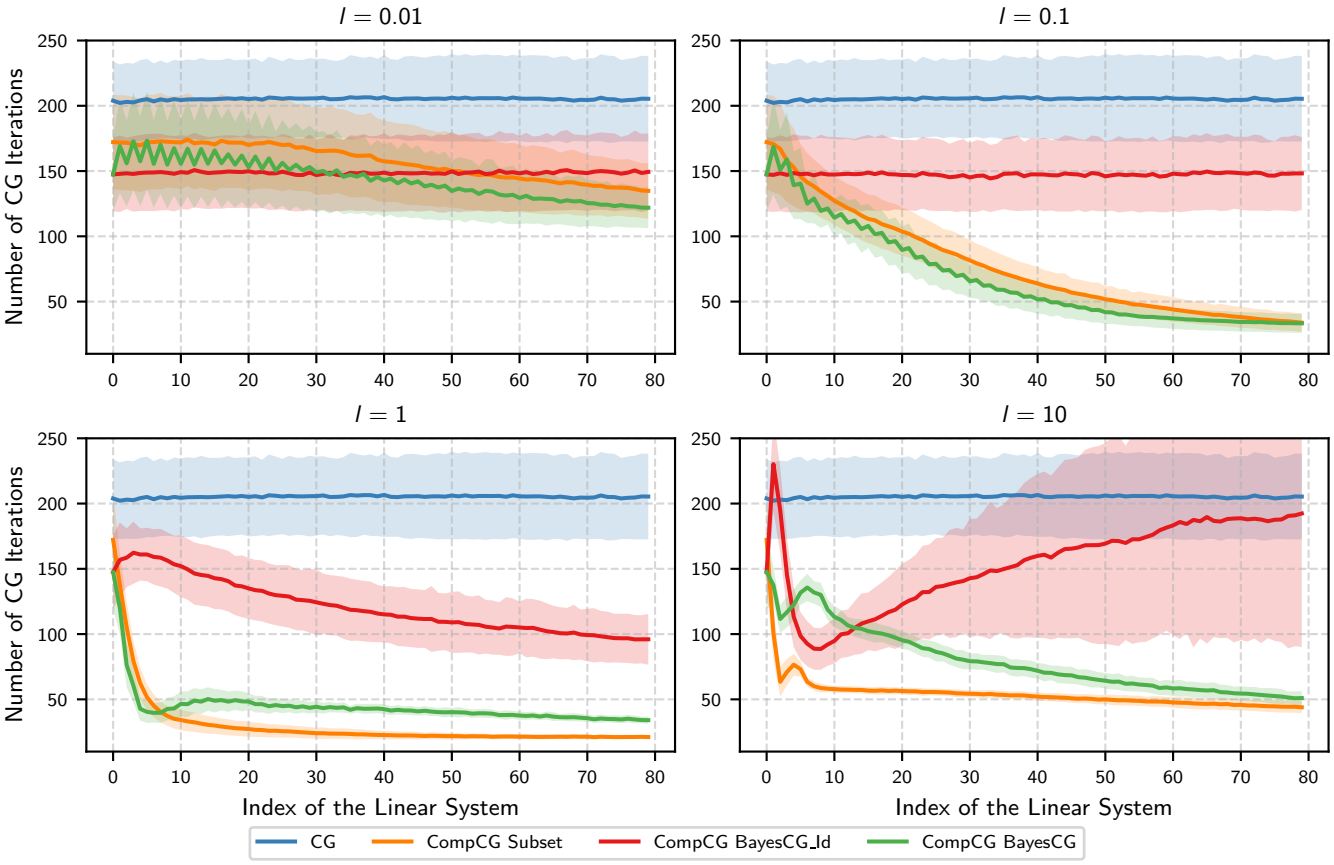


Figure 3: Number of CG iterations with differing lengthscale for the regression model

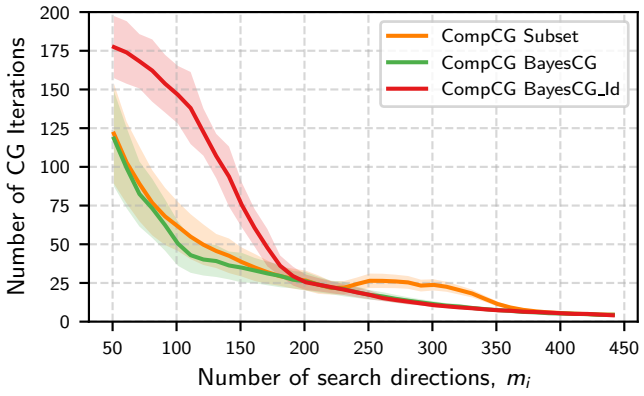


Figure 4: Number of CG iterations with differing number of search directions

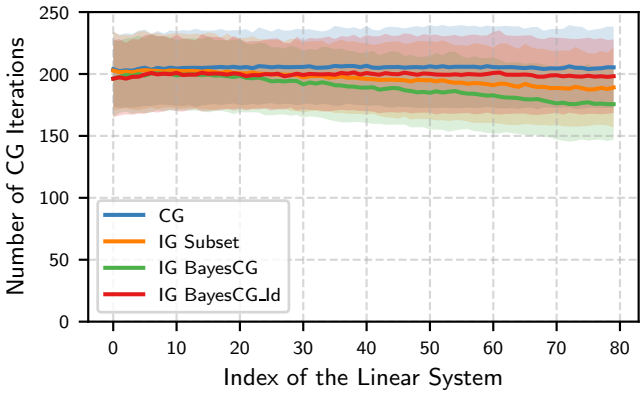


Figure 5: Number of CG iterations using only initial guess

d	162	648	800	1035	1352	1800	2592	4050	7200	10368	16200
Time for solving the linear systems (sec)											
CG	0.25	0.85	0.99	1.55	1.32	3.13	7.09	73.31	193.94	347.00	1671.25
CompCG Subset	**0.21**	**0.43**	0.51	**0.79**	1.89	1.47	3.09	14.19	**48.09**	131.81	**177.77**
CompCG BayesCG	0.24	0.53	**0.48**	0.79	**1.20**	**1.42**	**2.51**	**13.22**	59.34	**95.54**	272.27
Average time for solving linear systems per optimiser iteration (sec)											
CG	0.0031	0.0112	0.0118	0.0185	0.0220	0.0373	0.0844	0.8728	2.8521	6.6730	19.8959
CompCG Subset	0.0035	0.0083	0.0106	**0.0116**	0.0224	0.0263	0.0514	**0.2729**	**0.7514**	1.6476	4.0402
CompCG BayesCG	**0.0029**	**0.0069**	**0.0086**	0.0142	**0.0215**	**0.0253**	**0.0419**	0.2754	0.8726	**1.4929**	**3.7815**
Wall-time (sec)											
CG	0.82	4.78	4.99	7.50	**8.75**	24.38	51.84	407.08	1475.93	4554.74	11241.33
CompCG Subset	**0.62**	**2.80**	**3.41**	6.46	10.86	**13.79**	59.34	404.10	**1202.75**	**2963.18**	8440.28
CompCG BayesCG	0.81	2.90	3.84	**6.09**	14.50	23.33	**49.27**	**297.70**	1955.75	3135.40	**7469.10**
NLL											
CG	3264.2	7375.3	9164.8	11352.1	13477.0	16124.5	21192.6	29123.3	44313.8	59584.7	82928.9
CompCG Subset	3261.2	7348.2	9111.6	11343.4	13464.0	16126.6	21184.6	29119.2	44289.1	59555.7	82909.6
CompCG BayesCG	3267.1	7349.1	9122.4	11341.5	13480.7	16118.5	21165.3	29112.8	44296.4	59595.0	82958.4
Total no. of CG iterations											
CG	5156	9663	12086	13572	10298	16794	21776	24143	24686	22237	47268
CompCG Subset	2028	2193	1927	2307	3033	2472	2464	2811	3197	3785	3391
CompCG BayesCG	2381	2796	2157	2362	2384	2481	2537	2293	3379	3236	4154
No. of optimiser iterations											
CG	80	76	84	84	60	84	84	84	68	52	84
CompCG Subset	60	52	48	68	84	56	60	52	64	80	44
CompCG BayesCG	84	76	56	56	56	56	60	48	68	64	72

Table 2: Additional results of NLL optimisation for the regression problem in Section 6.2 for varying d

The amount of information used in training the regression model, determined by the number of search directions m (as seen in Eq. (5)), strongly influences the performance of the linear solver. Fig. 4 shows the number of CG iterations required to solve the 40^{th} linear system averaged over 20 runs, as a function of m_i. As expected, Fig. 4 shows that the number of CG iterations decreasing with increasing m_i, as this results in better model training.

C.2 GP Hyperparameter Optimisation Problem

We present additional results of the GP hyperparameter optimisation problem as described in Section 6.2 here.