The CLP: Constrained Linear Predictors

February 18, 2024

The Goal

Consider a second order random process X, such that at each value of $t \in \mathbb{R}$, we have a random variable X_t.

We may randomly sample this vector at n points, gaining a vector $\vec{T} = (t_1, t_2, t_3, \ldots)$ of times at which the samples were made, and $\vec{X} = (X_{t_i})$. Strictly speaking these are both random variables in and of themselves, up until the moment that we 'realise' them. We can index into these vectors using the the integer $0 \leq i < n$, and we assume without loss of generality that the samples are sorted in time, such that $t_i < t_{i+1} \forall i$.

In the case of the BLP, we wish to find a predictor, $\hat{X}_t$, which will predict the values of X_t on a set of 'prediction points', $t \in T$, subject to three further conditions:

- We are willing to present an *a priori* guess at the functional form of the predictor, in the form of a 'prior function' $g(t)$.
- The only thing we 'know' (or are willing to *ansatz*) about X_t is the second moment kernel (a generalisation of the covariance):
$$\langle (X_t - g(t))(X_s - g(s)) \rangle = k(t, s)$$
- Our predictor should be linear, such that:
$$\hat{X}_t = g(t) + \vec{a}_t \cdot \left(\vec{X} - \vec{G} \right)$$
Where $G_i = g(t_i)$

We again reiterate that $X_t, \vec{X}$ and $\hat{X}_t$ are - strictly speaking - random variables until we make them into real numbers at the moment we wish to actually make a prediction. $\vec{a}_t$ is a real n-tuple, which takes on different values at each value of t.

These are the ingredients of the standard BLP. The goal of this work is to extend this by adding one further condition:

- The Predictor should obey a number of constraints of the form $h(\{X_t\}) = 0$

We are therefore attempting to formulate the *Constrained Linear Predictor* (CLP)

1 Deriving the CLP

We define the CLP as the linear predictor which minimises the Mean Squared Error, averaged across all realisations of the random variable, computed at the set T of points at which we wish to make predictions, and which obeys our constraints.

Therefore, the CLP minimises the following Lagrangian:

$$\mathcal{L} = \sum_{t \in T} \langle (X_t - \hat{X}_t)^2 \rangle - \sum_j \lambda_j h_j(\{\hat{X}\}) \tag{1}$$

Here $h_j(\{\hat{X}_t\})$ is the j^{th} constraint on the *prediction points*[1], such that $h_j = 0$ when the constraint is met, and is non-zero otherwise, with the sum running over all such constraints. $\lambda_j \in \mathbb{R}$ are the associated Lagrange Multipliers. In the standard BLP we are able to treat the Lagrangian as separable in each element of T - minimising the MSE individually at each $t \in T$ is equivalent to performing a global minimisation: in the CLP this is not true, and we must consider the global case.

The issue at present is that we do not know what the behaviour of X_t is – we might have an initial guess (i.e. our prior, $g(t)$), but the entire purpose of this exercise is that we do not know X_t. However, by expanding out the brackets, we are able to write the Lagrangian in the following form:

$$\begin{aligned}
\mathcal{L} &= \left[\sum_{t \in T} \left\langle X_t'^2 \right\rangle - 2\vec{a}_t \cdot \left\langle X_t' \vec{X}' \right\rangle + \left\langle (\vec{a}_t \cdot \vec{X}')^2 \right\rangle \right] - \sum_j \lambda_j h_j(\{\hat{X}\}) \\
&= \left[\sum_{t \in T} \left\langle X_t'^2 \right\rangle - 2\vec{a}_t \cdot \vec{k}_t + \vec{a}_t \cdot (K\vec{a}_t) \right] - \sum_j \lambda_j h_j(\{\hat{X}\})
\end{aligned} \tag{2}$$

Where:

$$\begin{aligned}
X_t' &= X_t - g(t) \\
\vec{X}' &= \vec{X} - \vec{G} \\
\vec{k}_t &\in \mathbb{R}^n \text{ such that } \left[\vec{k}_t\right]_i = k(t, t_i) \\
K &\in \mathbb{R}^{n \times n} \text{ such that } K_{ij} = k(t_i, t_j)
\end{aligned} \tag{3}$$

Note that since the kernel is, by definition, symmetric in its arguments, $K^T = K$. Note that we have also taken the explicit step of writing our kernel as a relationship between the *transformed* data - i.e. X' - the imposition of different functions $g(t)$ might therefore warrant different kernels. This is true even if the transform is the (commonly used) constant 'mean scaling', $g(t) = \langle X_t \rangle \approx \frac{1}{n} \vec{X} \cdot \mathbb{1}$.

By performing this transform we have placed the incomputable terms - that of $\left\langle (X_t')^2 \right\rangle$ into a constant term. Since Lagrangians are invariant under constant scaling, it is possible to find an optimal value of $\vec{a}_t$ using only the remaining computable terms.

However - as we shall see - we are in the uncomfortable position of trying to impose conditions on the predicted values, $P_i = \hat{X}_{t_i} = g(t_i) + \vec{a}_{t_i} \cdot \vec{X}'$ whilst our object of interest is now the vector $\vec{a}_{t_i}$.

[1]For clarity and avoidance of symbol-collision with the other X-s, we will denote the prediction points as $P_i = \hat{X}_{t_i} = g(t_i) + \vec{a}_t \cdot \vec{X}'$

We therefore limit ourselves to the case of *linear constraints*, i.e., those which can be written in the following form:

$$\begin{aligned} h_j(\{P\}) &= c_j - \sum_k d_{jk} P_k \\ &= c_j - \sum_k d_{jk} \left(g(t_k) + \vec{a}_{t_k} \cdot \vec{X}' \right) \end{aligned} \tag{4}$$

We can then take the derivative of the Lagrangian with respect to $\vec{a}_{t_i}$, and find that:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \vec{a}_{t_i}} &= 2K\vec{a}_{t_i} - 2\vec{k}_i - \sum_j \lambda_j \frac{\partial h_j}{\partial \vec{a}_{t_i}} \\ &= 2K\vec{a}_{t_i} - 2\vec{k}_i + \left(\sum_j \lambda_j b_{ji} \right) \vec{X}' \\ &= 2K\vec{a}_{t_i} - 2\vec{k}_i + \eta_i \vec{X}' \end{aligned} \tag{5}$$

Hence, the optimal value of $\vec{a}_{t_i}$ is:

$$\begin{aligned} \vec{a}_{t_i} &= K^{-1} \left(\vec{k}_i - \frac{\eta_i}{2} \vec{X}' \right) \\ &= \vec{v}_i - \frac{\eta_i}{2} \vec{w} \end{aligned} \tag{6}$$

The optimal predicted value is:

$$\begin{aligned} P_i &= g(t_i) + \vec{a}_{t_i} \cdot \vec{X}' \\ &= g(t_i) + \vec{v}_i \cdot \vec{X}' - \frac{\eta_i}{2} \vec{w} \cdot \vec{X}' \\ &= g(t_i) + A_i - \frac{\eta_i}{2} B \end{aligned} \tag{7}$$

1.1 Exact Constraints

In the case where the constraints h_j are exact – i.e. the sets $\{c\}$ and $\{d\}$ are exactly determined, we may therefore analytically solve to find the set of Lagrange multipliers, then $\vec{\eta}$, and hence compute the predictor. We note that $\vec{\eta}$ can be written as:

$$\vec{\eta} = D^T \vec{\lambda} \tag{8}$$

Where $D_{ij} = d_{ij}$ is the constraint matrix, $\vec{\eta}_k = \eta_k$ is a vector on $\mathbb{R}^N$ and $\vec{\lambda}_k = \lambda_k$ is a vector on $\mathbb{R}^m$, where m is the number of constraints. The requirement that the constraints are met can be written as:

$$D\vec{p} = \vec{c} \tag{9}$$

Where $\vec{p}_i = P_i$ is another vector on $\mathbb{R}^n$ and $\vec{c}_i = c_i \in \mathbb{R}^m$. Writing $g(t_i) + A_i = q_i$, this is then:

$$D \left(\vec{q} - \frac{B}{2} D^T \vec{\lambda} \right) = \vec{c} \iff \vec{\lambda} = \frac{2}{B} \left(DD^T \right)^{-1} \left(D\vec{q} - \vec{c} \right) \tag{10}$$

Therefore:

$$\vec{p} = \left(\mathbb{1}_N - D^T(DD^T)^{-1}D\right)\vec{q} + D^T(DD^T)^{-1}\vec{c} \tag{11}$$

In the case where there is only a single constraint ($m = 1$), this simplifies such that $D \to \vec{d}^T$:

$$\vec{p} = \vec{q} + \frac{c - \vec{q}\cdot\vec{d}}{\vec{d}^2}\vec{d} \tag{12}$$

1.2 Inexact Constraints

In the case where the constraints are not exact, but serve to enforce bounds – i.e. monotonicity or positivity – there is a problem since the parameters of the constraint are not fixed. We may not care, for example, how much greater X_{i+1} is than X_i is, only that it *is* greater.

We could enforce this through slack variables and utilise the KKT conditions, however for our purposes it is better to *parameterize* the constraint.

Various parameterizations are possible, but perhaps the most comprehensible is to consider that the *prediction* points, P_i are a function of some other parameters $\vec{\theta} \in R^m$, such that:

$$\begin{aligned} P_i &= \mathcal{T}_i(\vec{\theta}) \\ h_j(\mathcal{T}_i(\vec{\theta})) &= 0 \;\forall\; i, j, \vec{\theta} \end{aligned} \tag{13}$$

For example, in the case of enforcing positivity, we might have that $P_i = e^{z_i}$, which is equivalent to asserting that $d_{ij} = \delta_{ij}$ and $c_i = e^{z_i}$. Rearranging Eq. (7), we are able to write η_i as a function of this Transform, and hence write $\vec{a}_{t_i}$ in the following form:

$$\vec{a}_{t_i} = \vec{v}_i + \frac{P_i(\vec{\theta}) - A_i - g(t_i)}{B}\vec{w} \tag{14}$$

This might seem somewhat tautological - we have written $\vec{a}_{t_i}$ in terms of the prediction values - but the entire purpose of $\vec{a}_{t_i}$ is to make predictions!

The usefulness of this comes evident when we insert Eq. (14) back into the Lagrangian – essentially performing a change of coordinates from $\mathcal{L}(\vec{a}, \vec{\theta})$ to $\mathcal{L}(\vec{\theta})$, since we have now ensured that $\vec{a}_t$ will always be at its optimal value for each value of $\vec{\theta}$.

$$\vec{k}_i \cdot \vec{a}_{t_i} = \vec{v}_i \cdot \vec{k}_i + \frac{P_i(\vec{\theta}) - A_i - g(t_i)}{B}\vec{w}\cdot\vec{k}_i \tag{15}$$

$$\begin{aligned} \vec{a}_{t_i}\cdot(K\vec{a}_{t_i}) &= \left(\vec{v}_i + \frac{P_i(\vec{\theta}) - A_i - g(t_i)}{B}\vec{w}\right)\cdot\left(\vec{k}_i + \frac{P_i(\vec{\theta}) - A_i - g(t_i)}{B}\vec{X}'\right) \\ &= \vec{v}_i\cdot\vec{k}_i + \left(\frac{P_i(\vec{\theta}) - A_i - g(t_i)}{B}\right)\left(\vec{w}\cdot\vec{k}_i + A_i\right) + \frac{\left(P_i(\vec{\theta}) - A_i - g(t_i)\right)^2}{B} \end{aligned} \tag{16}$$

Since $\vec{w} \cdot \vec{k}_i = (K^{-1}\vec{X}')\vec{k}_i = (K^{-1}\vec{k}_i)\vec{X}\prime = \vec{v}_i \cdot \vec{X}' = A_i$ due to the symmetry of K, and the constraints are all automatically satisfied thanks to our parameterisation, we find that the Lagrangian simplifies to:

$$\begin{aligned}
\mathcal{L}(\vec{\theta}) &= \sum_i \left(\left\langle (X_i')^2 \right\rangle - \vec{k}_i \cdot \vec{v}_i \right) + \frac{1}{B} \left(P_i(\theta) - A_i - g(t_i)\right)^2 \\
&= \text{const in } \vec{\theta} + \frac{1}{B} \sum_i \left(P_i(\theta) - A_i - g(t_i)\right)^2 \\
\mathcal{L}' &= \sum_i P_i \left(P_i(\theta) - 2(A_i + g(t_i))\right)
\end{aligned} \tag{17}$$

Where in the final line we took the opportunity to perform a rescaling (recalling that $B > 0$ is enforced by the positive definiteness of K) which leaves the optimum invariant. In some cases it is trivial to identify the optimal values of P_i - for example, in the case where $P_i = e^{\theta_i}$, the maximum is evidently:

$$P_i = \begin{cases} A_i + g(t_i) & \text{if this is } > 0 \\ 0 & \text{else} \end{cases} \tag{18}$$

In short, the CLP is equal to the BLP except when the condition is violated, at which point a hard cut is placed on it.

More complex conditions however, can lead to more complex behaviour - the monotonicity constraint, for example, exhibits the obvious behaviour that it again follows the BLP when it is monotonic, and is flat when the BLP has a negative gradient – but the *location* where the CLP becomes flat is non-trivial, with flatness necessarily occurring *before* the BLP changes direction: a tradeoff in following the BLP locally versus becoming too large too early without the ability to decrease due to the monotonic constraint.

In these cases a more complex search is required – where the behaviour of the constraint is evident *a priori* (such as the monotonic constraint), one can limit the space of the search. In the general case, however, a numerical optimisation is required.

The derivative of the Lagrangian with respect to the constraint parameters is:

$$\frac{\partial \mathcal{L}'}{\partial \theta_m} = 2 \sum_i \left(P_i - A_i - g(t_i)\right) \frac{\partial P_i}{\partial \theta_m} \tag{19}$$

This can be used to numerically optimise the values of $\vec{\theta}$

1.3 Inexact Constraints (Redux)

We note that we performed a fairly drastic change in approach between the exact constraints and the inexact constraints – is it possible to maintain the same approach for both?

We consider now that the parameters $\vec{c}$ of the constraints are functions of an (unconstrained) external parameter, $\vec{z} \in \mathbb{R}^m$ - letting $\vec{c} = \text{const}$ recovers the condition of the exact equalities. However, in any other case we must still find the values of $\vec{z}$ which optimise the global Lagrangian - and hence we need to rewrite our Lagrangian in terms of $\vec{c}$.

From Eq. (11), we can rewrite the predicted value-vector (recalling that $\vec{p}_i = P_i = \hat{X}_{t_i}$) as:

$$\begin{aligned} \vec{p} &= \vec{j} + R\vec{c}(\vec{z}) \\ R &= D^T(DD^T)^{-1} \\ \vec{j} = (\mathbb{1}_N - RD)\,\vec{q} \quad &\iff \quad j_i = g(t_i) + A_i + \sum_{j,k} R_{ij} D_{jk}(g(t_k) + A_k) \end{aligned} \tag{20}$$

We note that from a conceptual standpoint it is not a problem for the 'mixing' constraints D_{ij} to be the functions of $\vec{z}$, but this assumption allows us to precompute many of the otherwise troublesome entities. We can also rewrite $\vec{a}_{t_i}$ as:

$$\begin{aligned} \vec{a}_{t_i} &= \vec{v}_i - \frac{\eta_i}{2}\vec{w} \\ &= \vec{v}_i + \frac{[R\,(\vec{c} - D\vec{q})] \cdot \hat{e}_i}{B}\vec{w} \\ &= \vec{j}_i + \frac{(R\vec{c}) \cdot \hat{e}_i}{B}\vec{w} \end{aligned} \tag{21}$$

Where

$$\begin{aligned} R &= D^T(DD^T)^{-1} \\ \vec{j}_i &= \vec{v}_i - \frac{(RD\vec{q}) \cdot \hat{e}_i}{B}\vec{w} \end{aligned} \tag{22}$$

We therefore have:

$$\begin{aligned} \vec{k}_i \cdot \vec{a}_{t_i} &= \vec{v}_i \cdot \vec{k}_i + \frac{A_i}{B}\left((R\vec{c}) \cdot \hat{e}_i - (RD\vec{q}) \cdot \hat{e}_i\right) \\ &= \text{const in } \vec{c} + \frac{A_i}{B}(R\vec{c}) \cdot \hat{e}_i \\ \vec{a}_{t_i} \cdot (K\vec{a}_{t_i}) &= \left(\vec{j}_i + \frac{(R\vec{c}) \cdot \hat{e}_i}{B}\vec{w}\right) \cdot \left(K\vec{j}_i + \frac{(R\vec{c}) \cdot \hat{e}_i}{B}\vec{X}\right) \\ &= \text{const in } \vec{c} + 2\frac{(R\vec{c}) \cdot \hat{e}_i}{B}\vec{j}_i \cdot \vec{X} + \frac{1}{B}((R\vec{c}) \cdot \hat{e}_i)^2 \\ &= \text{const in } \vec{c} + 2\frac{(R\vec{c}) \cdot \hat{e}_i}{B}\left(A_i - (RD\vec{q}) \cdot \hat{e}_i\right) + \frac{1}{B}((R\vec{c}) \cdot \hat{e}_i)^2 \end{aligned} \tag{23}$$

Therefore:

$$\begin{aligned} \mathcal{L}' &= \sum_i \vec{a}_{t_i} \cdot K\vec{a}_{t_i} - 2\vec{k}_i \cdot \vec{a}_{t_i} \\ &= \text{const in } \vec{c} + \sum_i ((R\vec{c}) \cdot \hat{e}_i)^2 - 2(R\vec{c}) \cdot \hat{e}_i(RD\vec{q}) \cdot \hat{e}_i \\ &= \text{const in } \vec{c} + (R\vec{c})^2 - 2(R\vec{c}) \cdot (RD\vec{q}) \\ &= \text{const in } \vec{c} + (R\vec{c}(\vec{z}) - RD\vec{q})^2 \end{aligned} \tag{24}$$

The derivative with respect to the (unconstrained) vectors $\vec{z}$ is:

$$\begin{aligned}\frac{\partial \mathcal{L}'}{\partial z_m} &= (R\vec{c}(\vec{z}) - RD\vec{q}) \cdot R\frac{\partial \vec{c}}{\partial z_m} \\ &= (\vec{p}(\vec{z}) - \vec{q}) \cdot R\frac{\partial \vec{c}}{\partial z_m}\end{aligned} \tag{25}$$

Since $\vec{q}$ is the BLP prediction we can once again see that the derivative is zero if the BLP obeys the constraints ($\vec{c} - D\vec{q} = 0$), so the CLP will always revert to the BLP if this meets our constraints.

2 The CLUP

In the prior work, we assumed the the function $g(t)$ was 'handed down' to us to act as a prior function. However, this may induce biases in our predictor, meaning that:

$$\left\langle X_t - \hat{X}_t \right\rangle \neq 0 \tag{26}$$

We should ideally search for an *unbiased* predictor. The derivation of the Constrained Linear Unbiased Predictor (CLUP) should follow along similar lines to the standard BLUP, but we reproduce it in full for the sake of rigour.

We suppose that our random variable X_t can be written as:

$$X_t = m(t) + Y_t \tag{27}$$

Where $m : R \to R$ is the 'mean function' and Y_t is a zero-mean random variable. Therefore:

$$\langle X_t \rangle = m(t) \tag{28}$$

The main difference between $m(t)$ and $g(t)$ is that we assumed $g(t)$ was just a prior to 'help us along' without any intrinsic relation to X_t – here, however, we are asserting that $m(t)$ is a meaningful function – albeit an incomputable one, since we remain unwilling to assert any properties on $\langle X_t \rangle$. Because of this restriction, we cannot subtract away $m(t)$ from our data to formulate $\vec{X}' = \vec{Y}$ – we must keep everything in terms of our original, untransformed data.

We *can*, however, assert that $m(t)$ can be decomposed into a sum of basis functions, $\vec{\varphi}(t)$, where $\varphi_i(t) : \mathbb{R} \to \mathbb{R}$ is the i^{th} basis function. We therefore have:

$$\begin{aligned}m(t) &= \sum_{i=0}^{\omega} \phi_i(t)\beta_i \\ &= \vec{\beta} \cdot \vec{\varphi}(t)\end{aligned} \tag{29}$$

We again note that $\vec{\beta}$ is not a known value, however, we continue in the expectation that it will cancel out in future. We also note that without further information we must assume that $\omega \to \infty$, in practice we can limit the dimensionality by assuming that $m_\omega(t) \approx m(t)$ for finite ω. We can also formulate the matrix Φ:

$$\Phi \in \mathbb{R}^{\omega \times n} \text{ such that } \Phi_{ij} = \varphi_i(t_j) \tag{30}$$

Therefore:

$$\vec{X}_i = \sum_j^{\omega} \Phi_{ji}\beta_j \iff \vec{X} = \Phi^T\vec{\beta} + \vec{Y} \tag{31}$$

Our linear predictor takes the form:

$$\begin{aligned}\hat{X}_t &= \vec{a}_t \cdot \vec{X} \\ &= \vec{a}_t \cdot (\Phi^T\vec{\beta}) + \vec{a}_t \cdot \vec{Y} \\ &= \vec{\beta} \cdot (\Phi\vec{a}_t) + \vec{a}_t \cdot \vec{Y}\end{aligned} \tag{32}$$

We can also note that:

$$X_t - \hat{X}_t = (\Phi\vec{a}_t - \vec{\varphi}_t) \cdot \vec{\beta} + Y_t - \vec{a}_t \cdot \vec{Y} \tag{33}$$

Since $\langle Y \rangle = \vec{0}$ and $\langle Y_t \rangle = 0$ by definition, we note that the unbiased constraint is equal to:

$$\left\langle X_t - \hat{X}_t \right\rangle = 0 \iff (\Phi\vec{a}_t - \vec{\varphi}_t) \cdot \left\langle \vec{\beta} \right\rangle = 0 \tag{34}$$

Therefore if we write our unbiased constraint as $\mathcal{U}_t = \left\langle X_t - \hat{X}_t \right\rangle$, such that $\mathcal{U}_t = 0$ when the constraint is met:

$$\begin{aligned}\mu_t\mathcal{U}_t &= \left(\mu_t \left\langle \vec{\beta} \right\rangle\right) \cdot (\Phi\vec{a}_t - \vec{\varphi}_t) \\ &= \tilde{\mu}_t \cdot (\Phi\vec{a}_t - \vec{\varphi}_t)\end{aligned} \tag{35}$$

A suitable change of coordinates $\mu_t \to \tilde{\mu}_t$ therefore enables us to bypass the unknown $\left\langle \vec{\beta} \right\rangle$. Hence the Lagrangian of the system takes the form:

$$\begin{aligned}\mathcal{L}(\vec{a}, \vec{\lambda}, \{\tilde{\mu}\}) &= \sum_{t \in T} \left(\left\langle (X_t - \hat{X}_t)^2 \right\rangle + \tilde{\mu} \cdot (\Phi\vec{a}_t - \vec{\varphi}) \right) + \sum_j \lambda_j h_j(\{\hat{X}\}) \\ &= \sum_{t \in T} \left(\langle X_t^2 \rangle + \vec{a}_t \cdot (K\vec{a}_t) - 2\vec{k}_t \cdot \vec{a}_t + \tilde{\mu} \cdot (\Phi\vec{a}_t - \vec{\varphi}) \right) + \sum_j \lambda_j h_j(\{\hat{X}\})\end{aligned} \tag{36}$$

This is identical in form to Eq. (7), with the addition of some additional constraints - those labelled by $\tilde{\mu}$, which act to ensure that $\left\langle X_t - \hat{X}_t \right\rangle = 0$ for all $t \in T$, i.e. we are now including *unbiasedness* as a constraint.

We have also introduced $\vec{k}_t$ and K as the second moment matrices on X:

$$\begin{aligned} K &\in \mathbb{R}^{n \times n} \qquad & K_{ij} &= \left\langle X_{t_i} X_{t_j} \right\rangle \\ \vec{k}_t &\in \mathbb{R}^n \qquad & [\vec{k}_t]_i &= \langle X_t X_{t_i} \rangle \end{aligned} \tag{37}$$

We once again impose the linearity condition on our constraints, such that $h_j = c_j - \sum_k d_{jk}\hat{X}_{t_k}$:

$$\vec{h} = \vec{c} - D\vec{p} \tag{38}$$

Where $\vec{p}_i = \hat{X}_{t_i}$. The Lagrangian therefore simplifies to:

$$\mathcal{L}(\vec{a},\vec{\lambda},\{\tilde{\mu}\},\vec{c}) = \sum_{t\in T}\left(\left\langle X_t^2\right\rangle + \vec{a}_t\cdot(K\vec{a}_t) - 2\vec{k}_t\cdot\vec{a}_t + \tilde{\mu}\cdot(\Phi\vec{a}_t - \vec{\varphi}_t)\right) + \vec{\lambda}\cdot(\vec{c} - D\vec{p}) \tag{39}$$

The Lagrangian derivatives are:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\vec{a}_{t_i}} &= 2K\vec{a}_{t_i} - 2\vec{k}_i + \Phi^T\tilde{\mu} + \frac{\partial\vec{p}}{\partial\vec{a}_{t_i}}(D^T\lambda)\\ &= 2K\vec{a}_{t_i} - 2\vec{k}_i + \Phi^T\tilde{\mu} + \left(D^T\vec{\lambda}\cdot\hat{e}_i\right)\vec{X}\end{aligned} \tag{40}$$

$$\frac{\partial\mathcal{L}}{\partial\vec{\lambda}} = \vec{c} - D\vec{p} \tag{41}$$

$$\frac{\partial\mathcal{L}}{\partial\tilde{\mu}} = \Phi_t\vec{a}_t - \vec{\varphi}_t \tag{42}$$

Where $\hat{e}_i$ is the i^{th} basis vector, such that $\vec{\lambda}\cdot\hat{e}_i = \lambda_i$, the i^{th} Lagrange Multiplier [2]. The optimal value of $\vec{a}_t$ is therefore at:

$$\vec{a}_{t_i} = K^{-1}\vec{k}_i - 2K^{-1}\Phi^T\tilde{\mu}_i + \left(D^T\vec{\lambda}\cdot\hat{e}_i\right)K^{-1}\vec{X} \tag{43}$$

2.1 Applying Constraints

We now impose the unbiased constraint - substituting Eq. (43) into Eq. (42), which gives us:

$$\Phi K^{-1}\vec{k}_i - 2\Phi K^{-1}\Phi^T\tilde{\mu}_i + \left[(D^T\vec{\lambda})\cdot\hat{e}_i\right]\Phi K^{-1}\vec{X} = \vec{\varphi}_i \tag{44}$$

Here we have again switched notation to $\vec{\varphi}_i = \vec{\varphi}_{t_i}$ for convenience. Solving for $\tilde{\mu}_i$:

$$\tilde{\mu}_i = \frac{1}{2}\left(\Phi K^{-1}\Phi^T\right)^{-1}\left(\Phi K^{-1}\vec{k}_i + \left[(D^T\vec{\lambda})\cdot\hat{e}_i\right]\Phi K^{-1}\vec{X} - \vec{\varphi}_i\right) \tag{45}$$

Writing $M = \left(\Phi K^{-1}\Phi^T\right)$, we therefore have:

$$\begin{aligned}\vec{a}_{t_i} &= K^{-1}\Delta\vec{k}_i + C\vec{\varphi}_i + \left[(D^T\vec{\lambda})\cdot\hat{e}_i\right]K^{-1}\Delta\vec{X}\\ &\text{where } C = K^{-1}\Phi^T M^{-1}\\ &\text{and } \Delta = \mathbb{1} - \Phi^T M^{-1}\Phi K^{-1}\\ &\qquad\quad = (\mathbb{1} - C\Phi)^T\end{aligned} \tag{46}$$

We note that Δ has the following property:

$$K^{-1}\Delta = \Delta^T K^{-1} \tag{47}$$

[2]We note that we are having something of a collision with vectors associated with the i^{th} predictor (such as $\vec{a}_{t_i}$) and scalars associated with predictors or constraints which we have 'stacked' into vectors - such as λ_i and P_i. When indexing into labelled vectors we will attempt to make this clear - i.e. $[\vec{a}_{t_i}]_j$ is the j^{th} element of the vector associated with predictor i

We also note that:

$$\begin{aligned}\vec{a}_{t_i}^{\text{clup}} &= K^{-1}\Delta\vec{k}_i + C\vec{\varphi}_i + \left[(D^T\vec{\lambda})\cdot\hat{e}_i\right]K^{-1}\Delta\vec{X} \\ &= \vec{a}_{t_i}^{\text{blup}} + \left[(D^T\vec{\lambda})\cdot\hat{e}_i\right]K^{-1}\Delta\vec{X}\end{aligned} \tag{48}$$

I.e., we have found that (perhaps unsurprisingly) $\vec{a}_{t_i}^{\text{clup}}$ is equal to the BLUP case, plus a correction factor which ensures that the constraints are obeyed.

The prediction values are therefore:

$$\begin{aligned}\hat{X}_i = P_i &= \vec{a}_{t_i}\cdot\vec{X} \\ &= \vec{a}_{t_i}^{\text{blup}}\cdot\vec{X} + (D^T\vec{\lambda})\cdot\hat{e}_i\vec{X}\cdot K^{-1}\Delta\vec{X} \\ &= p_i^{\text{blup}} + (D^T\vec{\lambda})\cdot\hat{e}_i\beta\end{aligned} \tag{49}$$

Where:

$$\begin{aligned}p_i^{\text{blup}} &= \left(K^{-1}\Delta\vec{k}_i + C\vec{\varphi}_i\right)\cdot\vec{X} \\ \beta &= \vec{X}\cdot K^{-1}\Delta\vec{X}\end{aligned} \tag{50}$$

We can then form a vector of $\vec{p}_i = P_i$, such that:

$$\vec{p}^{\text{clup}} = \vec{p}^{\text{blup}} + \beta D^T\vec{\lambda} \tag{51}$$

Inserting this into the imposed constraints of Eq. (41), we find:

$$\begin{aligned}D\vec{p} &= \vec{c} \\ D\vec{p}^{\text{blup}} + \beta DD^T\vec{\lambda} &= \vec{c} \\ \vec{\lambda} &= \frac{1}{\beta}\left(DD^T\right)^{-1}\left(\vec{c} - D\vec{p}^{\text{blup}}\right)\end{aligned} \tag{52}$$

2.2 The CLUP

We can then insert this back into the definition of $\vec{a}_{t_i}$ to find:

$$\vec{a}_{t_i}^{\text{clup}} = \vec{a}_{t_i}^{\text{blup}} + \frac{\left(D^T\left(DD^T\right)^{-1}\left(\vec{c} - D\vec{p}^{\text{blup}}\right)\right)\cdot\hat{e}_i}{\beta}K^{-1}\Delta\vec{X} \tag{53}$$

Where:

$$\begin{aligned}C &= K^{-1}\Phi^T(\Phi K^{-1}\Phi^T)^{-1} \\ \Delta &= \mathbb{1} - \Phi^T(\Phi K^{-1}\Phi^T)^{-1}\Phi K^{-1} \\ \beta &= \vec{X}\cdot K^{-1}\Delta\vec{X} \\ \vec{a}_{t_i}^{\text{blp}} &= K^{-1}\vec{k}_i \\ \vec{a}_{t_i}^{\text{blup}} &= \Delta^T\vec{a}_{t_i}^{\text{blp}} + C\vec{\varphi}_i \\ p_i^{\text{blup}} &= \vec{a}_{t_i}^{\text{blup}}\cdot\vec{X}\end{aligned} \tag{54}$$

From this it is clear to see how the CLUP, the BLUP, the CLUP and the BLP all map onto each other. In the case where we have no constraints, then $l\vec{ambda} = 0$, and $\vec{a}_{t_i}^{\text{clup}} = \vec{a}_{t_i}^{\text{blup}}$, and in the case where we do not perform any simultaneous fitting, $C = 0$ and $\Delta = \mathbb{1}$, so $\vec{a}_{t_i}^{\text{blup}} = \vec{a}_{t_i}^{\text{blp}}$.

2.3 Optimising the CLUP

However, as before with the CLP, we have a problem if some of our constraints were inexact - that is, $\vec{c} = \vec{c}(\vec{z})$. We reiterate that this arises because, in our formalism, and inequality takes the following form:

$$\text{Constraint } i\text{: } a \geq b \iff a - b = \underbrace{e^z}_{c_i} \tag{55}$$

In the case where it is unknown *how much bigger* a is than b, we must find the value of c_i which minimises the Lagrangian. In short, we have formulated a way to write $\vec{a}_{t_i}$ as an exact-constraint-obeying function of some unknown parameters $\vec{z}$ – we must now optimise with respect to these parameters.

Therefore:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial z_k} &= \sum_i \frac{\partial \mathcal{L}}{\partial \vec{a}_{t_i}} \cdot \frac{\partial \vec{a}_{t_i}}{\partial z_k} \\ &= \sum_i \frac{\partial \mathcal{L}}{\partial \vec{a}_{t_i}} \cdot \left(\frac{\partial \vec{a}_{t_i}}{\partial \vec{c}} \frac{\partial \vec{c}}{\partial z_k} \right) \\ &= 2 \sum_i \left(K \vec{a}_{t_i} - \vec{k}_i \right) \cdot \left(\frac{\partial \vec{a}_{t_i}}{\partial \vec{c}} \frac{\partial \vec{c}}{\partial z_k} \right)\end{aligned} \tag{56}$$

In order to compute $\frac{\partial \vec{a}_{t_i}}{\partial \vec{c}}$ it is convenient to use outer products to write it in the following form:

$$\vec{a}_{t_i} = \vec{a}_{t_i}^{\text{blup}} + \left((K^{-1} \Delta \vec{X}) \otimes \hat{e}_i \right) \left(D^T (DD^T)^{-1} (\vec{c} - D\vec{\alpha}) \right) \tag{57}$$

Hence:

$$\frac{\partial \vec{a}_{t_i}}{\partial \vec{c}} = \left((K^{-1} \Delta \vec{X}) \otimes \hat{e}_i \right) D^T (DD^T)^{-1} \tag{58}$$

Recalling that $(\vec{u} \otimes \vec{w})^T = \vec{w} \otimes \vec{u}$, therefore we have:

$$\frac{\partial \mathcal{L}}{\partial z_k} = 2 \sum_i \left((K^{-1} \Delta \vec{X}) \otimes \hat{e}_i \right) \left(K \vec{a}_{t_i} - \vec{k}_i \right) \cdot D^T (DD^T)^{-1} \frac{\partial \vec{c}}{\partial z_k} \tag{59}$$

Exploiting the inner product once again:

$$\frac{\partial \mathcal{L}}{\partial z_k} = 2 \left(\sum_i \Delta^T \left[(\vec{a}_{t_i} - K^{-1} \vec{k}_i) \cdot \vec{\vec{X}} \right] \hat{e}_i \right) \cdot D^T (DD^T)^{-1} \frac{\partial \vec{c}}{\partial z_k} \tag{60}$$

Recall that:

$$\begin{aligned}\Delta^T \vec{a}_{t_i} &= (1 - C\Phi) \vec{a}_{t_i} \\ &= \vec{a}_{t_i} - C \vec{\varphi}_i\end{aligned} \tag{61}$$

This follows from the fact that $\vec{a}_{t_i}$ must obey the unbiasedness constraint. Expanding the definition of $\vec{a}_{t_i}$ and $\vec{a}_{t_i}^{\text{blup}}$:

$$\begin{aligned} \Delta^T \vec{a}_{t_i} &= \left(\vec{a}_{t_i}^{\text{blup}} + \frac{s_i}{\beta} K^{-1} \Delta \vec{X} \right) - C \vec{\varphi}_i \\ &= \left(\Delta^T K^{-1} \vec{k}_i + C \vec{\varphi}_i + \frac{s_i}{\beta} K^{-1} \Delta \vec{X} \right) - C \vec{\varphi}_i \\ &= \Delta^T K^{-1} \vec{k}_i + \frac{s_i}{\beta} K^{-1} \Delta \vec{X} \end{aligned} \tag{62}$$

Where:

$$\vec{s} = \left(D^T \left(D D^T \right)^{-1} \left(\vec{c} - D \vec{p}^{\text{blup}} \right) \right) \quad \Longleftrightarrow \quad s_i = \vec{s} \cdot \hat{e}_i \tag{63}$$

Therefore:

$$\begin{aligned} \vec{X} \cdot \left(\Delta^T \vec{a}_{t_i} - \Delta^T K^{-1} \vec{k}_i \right) &= \vec{X} \cdot \left(\Delta^T K^{-1} \vec{k}_i + \frac{s_i}{\beta} K^{-1} \Delta \vec{X} - \Delta^T K^{-1} \vec{k}_i \right) \\ &= \frac{s_i}{\beta} \vec{X} \cdot K^{-1} \Delta \vec{X} \\ &= s_i \end{aligned} \tag{64}$$

Therefore:

$$\frac{\partial \mathcal{L}}{\partial z_k} = 2 \left[(D D^T)^{-1} \left(\vec{c} - D \vec{p}^{\text{blup}} \right) \right] \cdot \frac{\partial \vec{c}}{\partial z_k} \tag{65}$$

In the case where $c_k = c_k(z_k)$, i.e. each constraint is associated with only one parameter, this simplifies to:

$$\frac{\partial \mathcal{L}}{\partial \vec{z}} = 2 \left[(D D^T)^{-1} \left(\vec{c} - D \vec{p}^{\text{blup}} \right) \right] \odot \vec{\delta}(\vec{c}) \tag{66}$$

Where $\odot$ is the elementwise Hadamard product and:

$$[\vec{\delta}(\vec{c})]_i = \frac{\partial c_i}{\partial z_i} \tag{67}$$

For example, in the case where $\vec{c} = e^{z_i}$, we find that $\vec{\delta} = \vec{c}$.

2.3.1 Sanity Check: Recovering the BLUP

We have already seen that the CLUP is a constrained version of the BLUP, so it should be trivially obvious that when the BLUP meets the constraints, then the CLUP coincides with the BLUP.

However, as a validation of our method, we note that the condition that the BLUP meets the constraints is exactly $D \vec{p}^{\text{blup}} = \vec{c}$, at which point Eq. (66) is trivially zero, since the left hand side of the dot product is the zero vector – hence the BLUP is an extremum of our Lagrangian in the case where it meets the constraints.

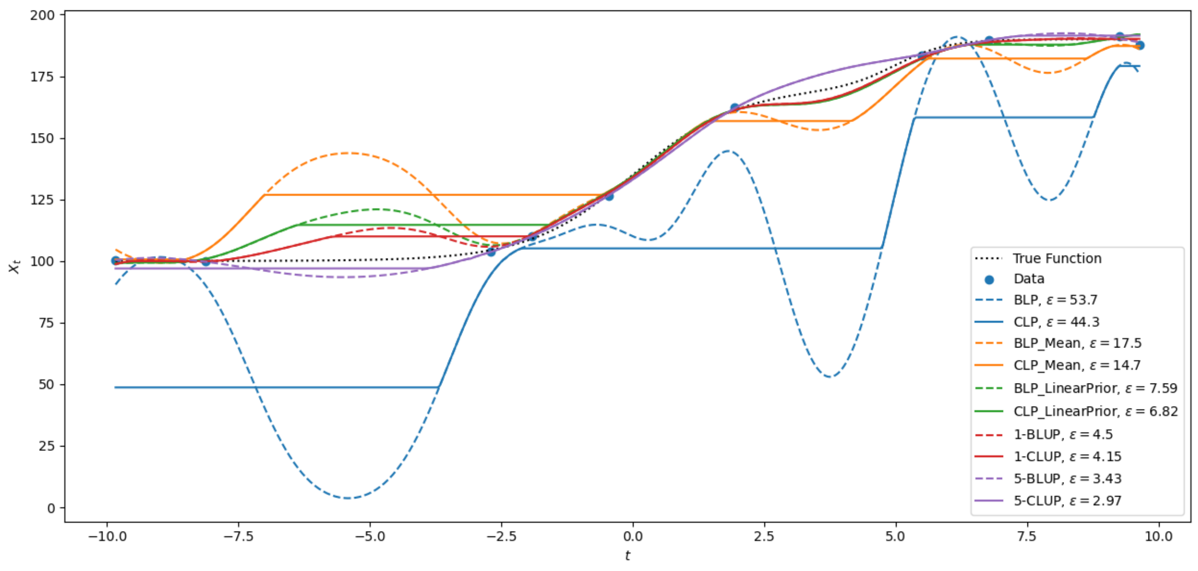

Figure 1: *A series of predictions on a dataset of $N = 10$ points drawn from the dotted black curve with Gaussian noise ($\sigma = 2$) added on. The models are described in detail in the text. The kernel has is a squared exponential with length-scale $\ell_0 = 1$ throughout. ϵ is the RMS deviation from the true, underlying function – this is distinct from the MSE, which is a function of the data.*

2.4 The CL(U)P in Action

In Figure 1, we show the relative performance of the constrained predictors when fitting a bi-sigmoid function with Gaussian noise generated. We impose the constraint that we know the underlying function is **monotonic**.

The monotonicity constraint means that we should always have $P_i \geq P_{i-1}$, assuming the prediction points are suitably time-ordered - our inexact constraint therefore takes the form:

$$\begin{aligned} \vec{c}(\vec{z}) \in \mathbb{R}^{n-1} \qquad & [\vec{c}(\vec{z})]_i = e^{z_i} \\ D \in \mathbb{R}^{(n-1)\times n} \qquad & D_{ij} = -\delta_{ij} + \delta_{(i+1)j} \end{aligned} \tag{68}$$

To our generated dataset, we fitted the following models:

- **BLP**: A standard BLP using the simple $\vec{X} \cdot K^{-1}\vec{k}$ predictor. We did not perform a mean-scaling on this predictor: $g(t) = 0$.
- **CLP**: As with the BLP, but with the addition of the monotonic constraint (see Eq. (68))
- **BLP_Mean/CLP_Mean**: As above, but the prior is set to $g(t) = \frac{1}{N}\mathbb{1} \cdot \vec{X}$, i.e. the sample mean.

- **BLP_LinearPrior/CLP_LinearPrior**: The prior is set to $g(t) = mt + c$, the straight line joining the two extreme datapoints in the sample. To avoid bootstrapping, these datapoints are then removed from the training dataset, so the fit is performed on $N - 2$ datapoints.
- n-**BLUP** the standard BLUP, with Φ expanded to n^{th} order.
- n-**CLUP**, as with the n-**BLUP**, but with the addition of the monotonic constraint.

We note that we deliberately offset and up-scaled the bi-sigmoid to highlight the difficulty faced by the BLP/CLP without the use of a prior function.

We see this reflected in Fig. 1: the BLP and CLP demonstrate extremely poor fits to the data; with the BLP oscillating down to 0 in the gaps between datapoints. The CLP tries to strike a balance between the BLP fit and remaining monotonic - the result is an underwhelming fit which goes nowhere near the data.

The addition of a prior $g(t)$ shows a significant improvement in the fit – the simple mean shift prevents the reversion down to 0, but i.e. at $t \approx -5$ results in a similar deviation upwards. The more complex LinearPrior models show a much better fit - the deviation at $t \approx -5$ is much smaller, and similar improvements are observed at $t \approx 3, +7$. This is noteworthy as the model has only 8 points to infer from, and yet shows a better fit than the BLP/CLP_Mean models which have more data to work from.

The BLUP/CLUP models again show an improvement - the 1-BLUP/1-CLUP models are directly comparable to the LinearPrior models since they both attempt to set a linear function, but the LinearPrior use only the extremum points, and must then remove those datapoints to prevent over-fitting: the 1-BLUP and 1-CLUP, however, simultaneously fit the linear fit and the predictor, which gives an improved fit.

Finally, the 5-BLUP/5-CLUP models fit a 5^{th} order polynomial, and demonstrate a significant improvement in the fit – the model is able to accurately anticipate that the line should be flat at $t \approx -5$.

We also note that, as measured by the true-RMS (ϵ in the figure), the inclusion of the monotonic constraint always improved the quality of the prediction – even the 5-BLUP which provided a good fit was bested by the 5-CLUP, by a non-trivial margin.

These conclusions are maintained even as we increase the number of datapoints (as seen in Fig. 2) – the margins become notably smaller, which is unsurprising as the predictor will tend towards becoming monotonic even without the inclusion of the constraints, the the CLP/CLUP generally outperform the BLP/BLP even when the amount of data becomes very large - as in Fig. 3

3 Optimising the Kernel

3.1 Leave-One-Out

We note that 'leaving a datapoint out' of the analysis is equivalent to the case where the error in the chosen datapoint is increased to an infinite degree of uncertainty. We can therefore write:

$$K_{ij}^{(m)}(\mu) = \begin{cases} K_{ii} + \mu^2 & \text{if } i = j = m \\ K_{ij} & \text{else} \end{cases} \iff K^{(m)} = K + \mu^2 R_m \tag{69}$$

The inverse of this matrix can be computed as:

$$(K_{ij}^{(m)})^{-1} = \left(\mathbb{1} + \mu^2 K^{-1} R_m\right)^{-1} K^{-1} \tag{70}$$

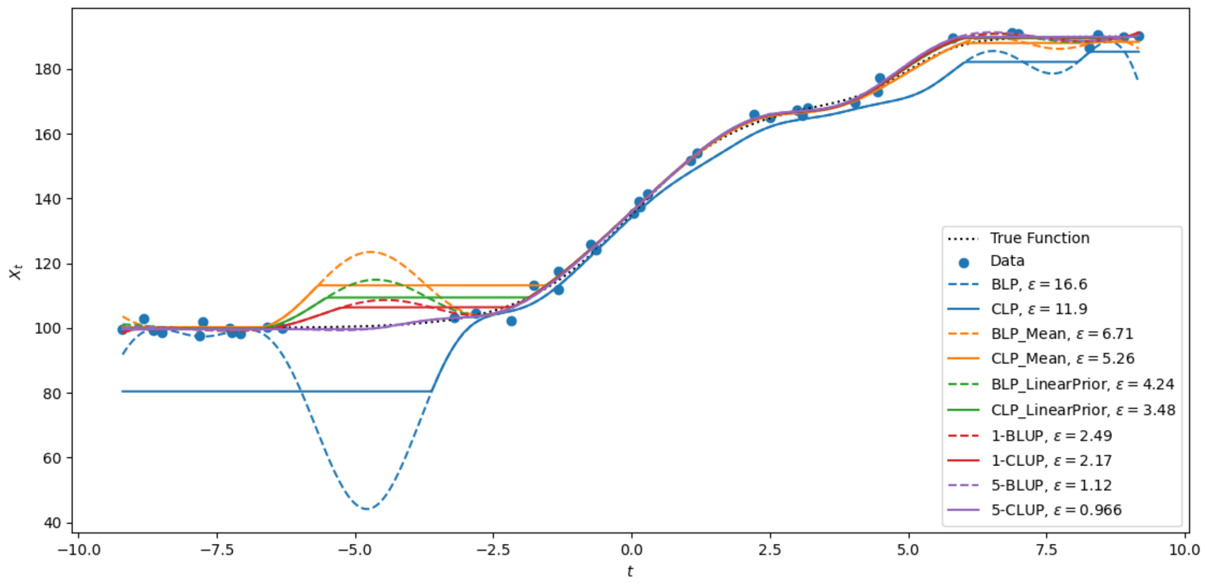

Figure 2: *As with 1, but with $N = 40$ datapoints ($N = 38$ for the 'Prior' models)*

The product $\mathbb{1} + K^{-1}R_m$ can be computed easily:

$$\begin{aligned}(\mathbb{1} + K^{-1}R_m)^{-1} &= \begin{pmatrix} 1 & 0 & \dots & 0 & \mu^2 K^{-1}_{0m} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \mu^2 K^{-1}_{1m} & 0 & \dots & 0 \\ \vdots \\ 0 & 0 & \dots & 0 & 1 + \mu^2 K^{-1}_{mm} & 0 & \dots & 0 \\ \vdots \\ 0 & 0 & \dots & 0 & \mu^2 K^{-1}_{nm} & 0 & \dots & 1 \end{pmatrix}^{-1} \\ &= \begin{pmatrix} 1 & 0 & \dots & 0 & -\frac{\mu^2 K^{-1}_{0m}}{1+\mu^2 K^{-1}_{mm}} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & -\frac{\mu^2 K^{-1}_{1m}}{1+\mu^2 K^{-1}_{mm}} & 0 & \dots & 0 \\ \vdots \\ 0 & 0 & \dots & 0 & \frac{1}{1+\mu^2 K^{-1}_{mm}} & 0 & \dots & 0 \\ \vdots \\ 0 & 0 & \dots & 0 & -\frac{\mu^2 K^{-1}_{nm}}{1+\mu^2 K^{-1}_{mm}} & 0 & \dots & 1 \end{pmatrix}\end{aligned} \tag{71}$$

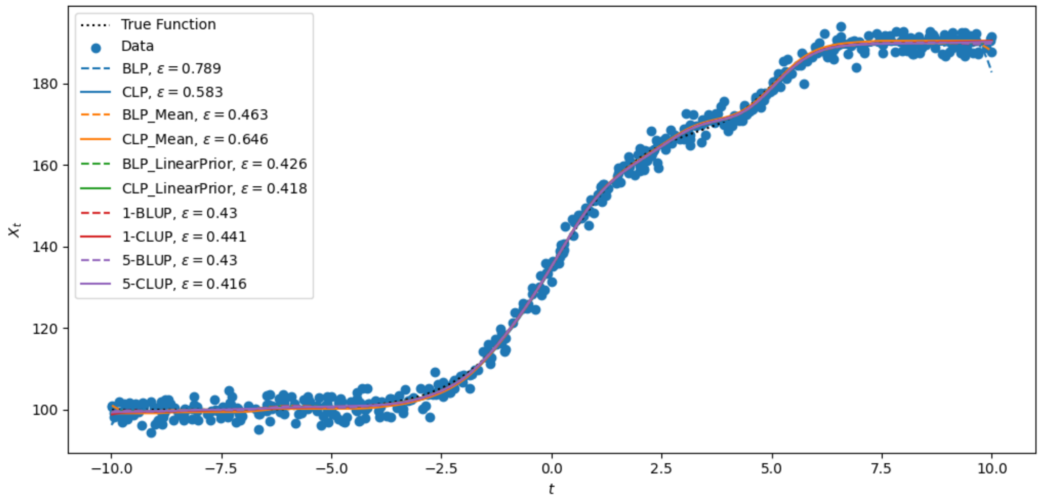

Figure 3: *As with 1, but with* $N = 520$ *datapoints (*$N = 518$ *for the 'Prior' models)*

In the limit that $\mu \to \infty$ (assuming that $K^{-1}_{mm} \neq 0$)

$$(\mathbb{1} + K^{-1}R_m)^{-1} = \begin{pmatrix} 1 & 0 & \dots & 0 & -\frac{K^{-1}_{0m}}{K^{-1}_{mm}} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & -\frac{K^{-1}_{1m}}{K^{-1}_{mm}} & 0 & \dots & 0 \\ \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots \\ 0 & 0 & \dots & 0 & -\frac{K^{-1}_{nm}}{K^{-1}_{mm}} & 0 & \dots & 1 \end{pmatrix} \tag{72}$$
$$= S(m)$$

Hence:

$$\begin{aligned} J(m)^{-1} &= \lim_{\mu\to\infty} \left(K^{(m)}(\mu)\right)^{-1} \\ &= S(m)K^{-1} \end{aligned} \tag{73}$$

Hence the modified $\mathrm{a_{blp}}$ is:

$$\vec{a}_t^{\,\mathrm{blp}}(m) = S(m)\vec{a}_t^{\,\mathrm{blp}} \tag{74}$$

And the validation predictor is:

$$\hat{X}(t,m) = \vec{a}_t^{\,\mathrm{blp}} \cdot S^T(m)\vec{X} \tag{75}$$

If we write

4 Initialisation

In the case where $\vec{w}$ must be optimised, we must first choose an initial starting point, $\vec{w}_0$, for the starting optimisation. The closer this is to the optimal point, the quicker the optimiser will converge.

In order to make this efficient, we rewrite the constraint vector $\vec{c}$ in the following fashion:

$$\vec{c}(\vec{w}) = \vec{\xi} + \psi(\vec{w}) \tag{76}$$

Here $\vec{\xi}$ contains both the equality constraints and any constant-offsets associated with the inequality constraints, such that we may then enforce the following conditions on $\vec{\psi}$:

$$\left[\vec{\psi}(\vec{w})\right]_i \begin{cases} = 0 & \text{if } i \text{ exact constraint} \\ \geq 0 & \text{else} \end{cases} \tag{77}$$

For example, if condition j is that $x_j \geq -4$, then $\xi_j = -4$ and $\psi_j = \exp(w_j)$. We also limit ourselves to the case where $\vec{w} = \psi^{-1}(\vec{c} - \vec{\xi})$ exists. Note that this is not a limitation on our general method, but rather a choice made for efficient initialisation.

The algorithm for determining the initialisation point is then:

1. Compute $\{\vec{a}^{\text{BLUP}}\}$ and hence $\hat{\vec{Z}}^{\text{blup}}$: the predictors using the normal BLUP algorithm
2. Let $\tilde{\vec{c}} = B\vec{p}_t = \vec{\xi} + \tilde{\vec{\varphi}}$
3. Project onto the constraint-meeting surface:
$$\varphi_j = \begin{cases} \tilde{\varphi}_j & \text{if } \tilde{\varphi}_j \geq 0 \\ 0 & \text{else} \end{cases}$$
4. Set $\vec{w}_0 = \psi^{-1}\left(\varphi_j\right)$

In practice, a small amount of numerical tolerance might be required (setting a $\varphi_j = 0$ when $\psi^{-1} = \ln(\varphi)$ is not numerically stable), so at step 3 we suggest setting $\varphi_j = \epsilon$, some very small numerical quantity.

4.1 Comments on Initialisation

This method of initialisation is a naïve projection from the BLUP onto the space of constraint-obeying functions. In some simple cases, this projection is in fact equal to the global maximum: the case of positive functions, for example - the naïve projection truncates the BLUP to be equal to 0 wherever the BLUP would become negative, which is exactly the global solution.

In some pathological cases, however, this projection might lead to a function extremely far away from both the global maximum and the BLUP: consider the case of a BSCLUP constrained to be monotonically increasing, but where the BLUP is monotonically *decreasing*. In this case, the projection of $\vec{w}_0$ would result in a flat line at the height of Z_0^{BLUP} – a rather significant deviation, and unlikely to be close to the optimum.

Experimentally, we find that this initialisation serves as a good initial *ansatz* as to the location of the optimum point for most real-world applications.

5 Prediction Errors

5.1 Why the BLUP Approach doesn't work

The approach in standard BLUP texts is to simply use that the prediction error is (approximately – some assumptions needed if I recall?) equal to the MSE evaluated at the optimum.

This, however, utilises the assumption that each of the prediction points is independent; an assumption that does not follow through with the BSCLUP. We have emphasised that the BSCLUP prediction is on the *entire* sequence/series of points - and hence any associated error must be computed on a global scale.

Simply put, it does not make sense to think about the error associated with just one point, when moving that point might have an impact on subsequent points (i.e., it is *impossible* to move a point upwards in a monotonic predictor-series if the subsequent point already has the same prediction value, as this would violate the constraint.)

This concern is not merely limited to the predictor-sequences, as predictor-series also violate the assumptions that allow the MSE to be used; a trivial example would be the error on a predictor-series constrained to be non-negative, but which is predicted to be equal to zero. It is evident that a symmetric error around $Z = 0$ would not be representative of the predictor error at that point.

We must therefore lend slightly more care and attention to our errors.

5.2 The MCMC Approach

Errors on sequences naturally lend themselves to an MCMC-style approach, as this provides a natural way to explore the intercorrelation between the sequence/series.

In an ideal scenario, we would simply vary the predictor values, $\{\hat{Z}_{\text{clup}}\}$, and use this to generate a score $\mathcal{L}$ which the MCMC engine could explore. This faces two major problems:

1. The score function $\mathcal{L}$ is expressed in terms of $\{\vec{a}\}$, but $\hat{Z}$ and $\vec{a}$ are related through a non-invertible dot-product.
2. With complex constraints, the majority of proposed variations to $\hat{Z}$ would be invalid, and hence the MCMC engine would not be able to produce a reliable chain.

We must therefore run the MCMC engine in $\vec{a}$-space; which has the unfortunate by-product of being much higher-dimensional, and therefore has a higher autocorrelation length. However, blindly proposing a new $\vec{a}$ falls afoul of point 2) raised above, namely that the majority of the time, the resulting predictions will not be valid.

I therefore propose 4 potential algorithms for generating a valid MCMC chain.

Algorithm 1: "Fuck You, Markov, You Don't Know Me"

This algorithm is simple: any proposed $\{\vec{a}\}$ which violate the constraints is given a score of $-\infty$, and the rest is left up to the MCMC engine to handle.

This *might* work in some of the inequality cases – it almost certainly won't work in exact constraints (i.e. the probability of the MCMC generating a curve with an integral equal to 1 (within machine precision) is vanishingly small).

I do not recommend this, but it is technically an option.

Algorithm 2: "Exactitude"

In this case, we treat the variation as happening on the space of $\{a_{\text{nqblup}}\}$ (nqBLUP = not-quite-best LUP, since we have varied it away from the optimum!). If the constraints were exact, then this is almost identical to simply varying the $\{\vec{a}\}$, you simply have to correct the predictor using the BSCLUP identity. If the constraints are inexact, then for each proposed $\{a_{\text{nqblup}}\}$ we compute the exact value of $\vec{c}$ which optimises the predictor; we then have a means of associating a variational score to a predictor which is away from the mean, but which is guaranteed to obey the correct behaviour.

This is probably the most theoretically justifiable algorithm; the variables within $\vec{w}$ were always a fiction and so 'optimising them away' to produce the 'optimised-variation' seems like the best approach.

The downside is that – aside from exact constraints and certain trivial cases – this is computationally very costly, and will take a vast amount of computing power to produce meaningful results.

Algorithm 3: "Dual Variation"

It is clear that the MCMC must vary $\vec{a}_{\text{nqblup}}$ in order to produce meaningful results - however, we might take objection to the optimisation of $\vec{w}$ which the "Exactitude" method - firstly on practical grounds, and secondly on the idea that we are explicitly varying *away* from the optimum – so why do we not also vary $\vec{w}$[3]?

In this case, we form a composite vector $\{\vec{a}_{\text{nqblup}}, \vec{w}\}$ such that for each variation we can construct a $\vec{c}$, and then through the BSCLUP identity a $\vec{a}_{\text{nqBSCLUP}}$ and hence a score.

The downside of this is that:

- We might argue the opposite way and say that unoptimised $\vec{w}$ values are meaningless
- This increases the number of dimensions (potentially up to twice as many), and so increases the autocorrelation time.

Algorithm 4: 'Eh, Close Enough'

This final algorithm works similarly to Algorithm 2, except that no direct optimisation is involved. After proposing a new $\{a_{\text{nq}\vec{\text{b}}\text{lup}}\}$, you then perform the Initialisation Projection:

1. Compute $\vec{\hat{\vec{Z}}}^{\text{nqblup}}$ using the normal BLUP algorithm
2. Let $\tilde{\vec{c}} = B\vec{\hat{\vec{Z}}}^{\text{nqblup}} = \vec{\xi} + \tilde{\varphi}(\vec{w})$

[3]I don't know if I believe this, but would be interested in some thoughts!

3. Project onto the constraint-meeting surface:

$$\varphi_j = \begin{cases} \tilde{\varphi}_j & \text{if } \tilde{\varphi}_j \geq 0 \\ 0 & \text{else} \end{cases}$$

4. Then set $\vec{c'} = \vec{\xi} + \vec{\varphi}$

5. Use $\{\vec{a}_{\text{nqblup}}\}$ and $\vec{c'}$ to construct a $\{\vec{a}_{\text{nqbsclup}}\}$

This guarantees that all proposed $\{\vec{Z}\}$ obey the constraints, however the projection performed is somewhat naive and may sometimes be far away from the optimum.

However, since we are varying $\vec{a}_{\text{nqblup}}$ freely, it can move very far away from the optimum, and so it is possible to generate arbitrary constraint-obeying $a_{\vec{\text{bsclup}}}$ (i.e., although the projection of $\vec{a}_{\text{blup}}$ is not guaranteed to be near the optimum, if we set $\vec{a}_{\text{nqblup}} = \vec{a}_{\text{bsclup}}$, the projection would trivially be equal to $\vec{a}_{\text{bsclup}}$, and therefore small variations from this position will also be projected into small variations from the optimum.)

This has the benefit of being able to explore arbitrary predictors (given enough time), without producing too many additional dimensions – the downside is that since the projections may make many $\vec{a}_{\text{nqblup}}$ produce the same $\vec{a}_{\text{bsclup}}$ (and hence the same score), the MCMC might think it has redundant dimensions, get confused, or otherwise have an excessively high autocorrelation time as it struggles to find which parameters are meaningful.