Bayesian Parton Distribution Functions fit with Gaussian processes

Giacomo Petrillo
Department of Statistics, Computer Science, Applications (DISIA), University of Florence
giacomo.petrillo@unifi.it

November 30, 2023

Introduction

This report describes a work in progress (see Del Debbio, Giani, and Wilson 2022) to reimplement the parton distribution functions (PDF) fit of the Neural Network PDF (NNPDF) group using Gaussian processes. Our goals are:

- Mathematically legible assumptions, motivated by the difficulties in interpreting the neural network model.
- Interpretable uncertainty quantification that accounts at once for all sources of error, motivated by the doubts of the community on the final quoted fit error, which enters as an important component in many experimental measurements.
- A computationally faster fitting procedure.

These qualities together would produce both clearer results and, perhaps even more importantly, allow faster iteration and improvement of the model. Our strategy is trying to build a single coherent Bayesian inference. A natural technical choice for the distribution over the unknown PDF functions is Gaussian processes. Previous attempts at the analysis with Bayesian inference are Aggarwal et al. (2022), Gbedo and Mangin-Brinet (2017), and Mangin-Brinet and Gbedo (2017).

In what follows we briefly recap PDFs and Gaussian process regression for what concerns our needs, and then describe in turn progressively more sophisticated models.

Contents

1 PDFs

We work in the DGLAP evolution basis, so the 9 PDF functions $(0, 1] \to (-\infty, \infty)$ are

$$\Sigma \quad V \quad V_3 \quad V_8 \quad V_{15} \quad T_3 \quad T_8 \quad T_{15} \quad g. \tag{1}$$

We have excluded the bottom and top quarks because their contribution to the proton is negligible given their large mass, and the photon because yes. These are related to the flavor basis by

I don't know why the photon does not enter. Is it because it has low momentum?

$$\begin{aligned}
q_- &= q - \bar{q} & q_+ &= q + \bar{q} \\
V &= \sum_q q_- & \Sigma &= \sum_q q_+, \quad q \in \{d, u, s, c\} \\
V_3 &= u_- - d_- & T_3 &= u_+ - d_+ \\
V_8 &= u_- + d_- - 2s_- & T_8 &= u_+ + d_+ - 2s_+ \\
V_{15} &= u_- + d_- + s_- - 3c_- & T_{15} &= u_+ + d_+ + s_+ - 3c_+.
\end{aligned} \tag{2}$$

See eko.readthedocs.io/en/latest/theory/FlavorSpace.

The total momentum constraint is

$$\int_0^1 \mathrm{d}x\, x \left(\sum_q q(x) + \sum_q \bar{q}(x) + g(x) \right) = 1, \tag{3}$$

while the flavor constraints are

$$\int_0^1 \mathrm{d}x\, (q(x) - \bar{q}(x)) = \begin{cases} 2 & q = u \\ 1 & q = d \\ 0 & q \notin \{u, d\}, \end{cases} \tag{4}$$

and finally all functions must be zero for $x = 1$ due to the no single carrier constraint. Translated to the evolution basis, the constraints become

$$\begin{aligned}
\int_0^1 \mathrm{d}x\, x(\Sigma(x) + g(x)) &= 1 & \Sigma(1) = g(1) &= 0 \\
\int_0^1 \mathrm{d}x\, V(x) &= 3 & V(1) &= 0 \\
\int_0^1 \mathrm{d}x\, V_3(x) &= 1 & V_3(1) = T_3(1) &= 0 \\
\int_0^1 \mathrm{d}x\, V_8(x) &= 3 & V_8(1) = T_8(1) &= 0 \\
\int_0^1 \mathrm{d}x\, V_{15}(x) &= 3 & V_{15}(1) = T_{15}(1) &= 0.
\end{aligned} \tag{5}$$

Additionally, as "soft hypotheses," we expect Σ and g to follow a power law with negative exponent ≈ -1 as $x \to 0^+$, and all the functions to vary arbitrarily quickly as $x \to 0^+$. We also conjecture that the functions are nonnegative.

Is the proof of the posivity confirmed?

2 Data

Most of the DIS data has a linear relationship with PDFs. In the abstract, for each quantity to be measured y_i, calling f_j the PDFs, we have

$$y_i = \sum_j \int_0^1 \mathrm{d}x\, F_{ij}(x) f_j(x). \tag{6}$$

The operator coefficients $F_{ij}(x)$ are computed with a numerical approximation of DGLAP on a fixed grid of x values. Approximately, this grid is made up of 34 log-spaced values from 2×10^{-7} to 0.11, and 17 linearly spaced values from 0.11 to 1. Thus the expression we use in practice is

$$y_i = \sum_{jk} F_{ijk} f_j(x_k). \tag{7}$$

The F_{ijk} tensors are called "FK tables." The calculation of these matrices is not exact, so we have estimates $\hat{F}$ with their own uncertainty. The covariance matrix of $\hat{F}|F$ is in principle a huge six axes tensor, however it is extremely degenerate with effective rank about 10. So the error distribution can be conveniently expressed by combining some eigenvectors $F^{(l)}$ of the covariance tensor with Normal coefficients. Actually I suspect that the current degeneracy is an artifact of the computational method and is not substantial, but even if this was the case, I would expect anyway a quite degenerate spectrum.

Other data depends on simple nonlinear transformations of the kind

$$y_i = h\left(\sum_k F_{i1k} f_1(x_k), \sum_k F_{i2k} f_2(x_k), \dots\right) \tag{8}$$

while yet other data has a quadratic dependence:

$$y_i = \sum_{ijkl} F_{jkl} f_j(x_k) f_j(x_l). \tag{9}$$

The distributions of the data measurements $\hat{y}|y$ are mostly Normal with a dense covariance matrix, plus a small amount of Poisson data. The number of Normal datapoints is about 4500 total, of which 3000 linear. The relative errors are typically 2–10 %, with some exceptions down to 0.1 % and up to 50 %.

3 Gaussian process regression

For textbooks on the topic, see Gramacy (2020), Murphy (2023), Rasmussen and Williams (2006), Stein (1999), and Wendland (2004).

3.1 Definition of GP

We say a stochastic process f is a Gaussian process with mean function $m(x)$ and covariance function or kernel $k(x, x')$, and write

$$f \sim \mathcal{GP}(m, k), \tag{10}$$

where m and k have domain respectively $\mathcal{X}$ and $\mathcal{X} \times \mathcal{X}$ for some arbitrary index set $\mathcal{X}$, if for any finite collection of points $\mathbf{x} \in \mathcal{X}^n$, the vector $f(\mathbf{x}) = (f(x_1), \dots, f(x_n))$ is Normally distributed as

$$f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}^\top)), \tag{11}$$

where $m(\mathbf{x}) = (m(x_1), \dots, m(x_n))$ and

$$k(\mathbf{x}, \mathbf{x}^\top) = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}. \tag{12}$$

In particular this means that $E[f(x)] = m(x)$ and $\mathrm{Cov}[f(x), f(x')] = k(x, x')$.

A degenerate Normal density, i.e., a Normal distribution defined on a subspace, can be written with the pseudoinverse of the covariance matrix (see section A.1) as

$$\begin{aligned} &\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{\mathrm{pdet}(2\pi\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^\top \Sigma^- (\mathbf{y} - \boldsymbol{\mu})\right), \\ &\text{with } \mathbf{y} - \boldsymbol{\mu} = \Sigma\Sigma^+(\mathbf{y} - \boldsymbol{\mu}), \end{aligned} \tag{13}$$

where pdet is the pseudodeterminant, i.e., the product of the nonzero eigenvalues. Since pseudoinversion and rotation commute, this formula gives the density written with the projected quantities. To write the density in the full space, add a Dirac delta term $\delta((I - \Sigma\Sigma^+)(\mathbf{y} - \boldsymbol{\mu}))$ for the subspace constraint.

3.2 Inference

We use a Gaussian process as prior distribution over an unknown function for Bayesian inference. Following the notation of Rasmussen and Williams (2006), let $\mathbf{x}$ be the indices of datapoints, where we know the function values $f(\mathbf{x}) = \mathbf{y}$. Let $\mathbf{x}^*$ be the "test points," the indices where we want to know the value of the function. For example, often $\mathbf{x}^*$ is a finely spaced grid used to draw the function.

Consider the prior covariance matrix of $f((\mathbf{x}, \mathbf{x}^*))$, divided in blocks in the following way:

$$k((\mathbf{x}, \mathbf{x}^*), (\mathbf{x}, \mathbf{x}^*)^\top) = \begin{pmatrix} k(\mathbf{x}, \mathbf{x}^\top) & k(\mathbf{x}, \mathbf{x}^{*\top}) \\ k(\mathbf{x}^*, \mathbf{x}^\top) & k(\mathbf{x}^*, \mathbf{x}^{*\top}) \end{pmatrix} = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xx^*} \\ \Sigma_{x^*x} & \Sigma_{x^*x^*} \end{pmatrix}. \tag{14}$$

In terms of these blocks, the posterior distribution of the function values at the test points has this simple form:

$$\begin{aligned} f(\mathbf{x}^*) \mid f(\mathbf{x}) = \mathbf{y} \sim \mathcal{N}(&m(\mathbf{x}^*) + \Sigma_{x^*x}\Sigma_{xx}^-(\mathbf{y} - m(\mathbf{x})), \\ &\Sigma_{x^*x^*} - \Sigma_{x^*x}\Sigma_{xx}^-\Sigma_{xx^*}), \end{aligned} \tag{15}$$

where Σ_{xx}^- is any generalized inverse of Σ_{xx}, in particular, Σ_{xx}^+ is a valid choice (see section A.1). For the proof, see Schott (2017, ex. 7.4, p. 295).

3.3 Properties of the Normal distribution

The reason we need to evaluate the kernel only on the finite set of points we are dealing with, despite considering a distribution over an infinite dimensional space, is because of the property of multivariate Normal distributions that the marginal distributions are still Normal. In general the useful properties of the Normal distribution are:

Conditioning If

$$\begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}_y\\ \boldsymbol{\mu}_z\end{pmatrix}, \begin{pmatrix}\Sigma_{yy} & \Sigma_{yz}\\ \Sigma_{zy} & \Sigma_{zz}\end{pmatrix}\right), \tag{16}$$

then $\mathbf{z} \mid \mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_z + \Sigma_{zy}\Sigma_{yy}^{-}(\mathbf{y}-\boldsymbol{\mu}_y), \Sigma_{zz} - \Sigma_{zy}\Sigma_{yy}^{-}\Sigma_{yz})$. This also implies $\mathrm{Cov}[\mathbf{a}, \mathbf{b} \mid \mathbf{c}] = \Sigma_{ab} - \Sigma_{ac}\Sigma_{cc}^{-}\Sigma_{cb}$.

Linearity If $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and $\mathbf{z} = A\mathbf{y}$, then $\mathbf{z} \sim \mathcal{N}(A\boldsymbol{\mu}, A\Sigma A^{\top})$.

Marginalization As a particular case of linearity, if $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, then

$$\begin{pmatrix}y_{i_1}\\ \vdots\\ y_{i_n}\end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix}\mu_{i_1}\\ \vdots\\ \mu_{i_n}\end{pmatrix}, \begin{pmatrix}\Sigma_{i_1,i_1} & \cdots & \Sigma_{i_1,i_n}\\ \vdots & \ddots & \vdots\\ \Sigma_{i_n,i_1} & \cdots & \Sigma_{i_n,i_n}\end{pmatrix}\right). \tag{17}$$

Linearity is valid also for linear operators (Fourier, Mellin, etc.) on the Gaussian process. In particular $\mathrm{Cov}[f'(x), f(x')] = \partial_x k(x, x')$. We will make use of this to implement the integral constraints.

For a textbook reference on the multivariate Normal, see Tong (1990).

3.4 Hyperparameters

Usually the kernel and the mean function depend on some parameters θ, the "hyperparameters." A maximum likelihood estimate of θ can be obtained by maximizing the likelihood of $f(\mathbf{x})$:

$$\begin{aligned}\hat{\theta}_{\mathrm{ML}} &= \arg\max_{\theta} \mathcal{N}(\mathbf{y}; m(\mathbf{x};\theta), \Sigma_{xx}(\theta)) = \\ &= \arg\min_{\theta} \left(\log \mathrm{pdet}\, \Sigma_{xx}(\theta) + (\mathbf{y} - m(\mathbf{x};\theta))^{\top}\Sigma_{xx}^{+}(\theta)(\mathbf{y} - m(\mathbf{x};\theta))\right),\end{aligned} \tag{18}$$

where care must be taken that the rank of $\Sigma_{xx}(\theta)$ is fixed, and $\mathbf{y} - m(\mathbf{x};\theta)$ stays in the range of $\Sigma_{xx}(\theta)$ for any value of θ. If the latter condition does not hold, the expression as written projects $\mathbf{y} - m(\mathbf{x};\theta)$ on the support of the distribution.

Once we have $\hat{\theta}_{\mathrm{ML}}$, we can plug its value into the GP and use Equation 15. However we are more interested in a fully Bayesian model where θ has its own prior:

$$f \mid \theta \sim \mathcal{GP}(m(\cdot;\theta), k(\cdot,\cdot;\theta)), \qquad \theta \sim p_\theta. \tag{19}$$

In this case we get the posterior distributions

$$\begin{aligned} p(\theta \mid \mathbf{f}(\mathbf{x}) = \mathbf{y}) &\propto p(f(\mathbf{x}) = \mathbf{y} \mid \theta) p_\theta(\theta), \\ p(f(\mathbf{x}^*) = \mathbf{y}^*, \theta \mid f(\mathbf{x}) = \mathbf{y}) &= p(f(\mathbf{x}^*) = \mathbf{y}^* \mid f(\mathbf{x}) = \mathbf{y}, \theta) p(\theta \mid \mathbf{f}(\mathbf{x}) = \mathbf{y}),\end{aligned} \tag{20}$$

where the first line follows from Bayes' theorem and the second from the law of joint probability (a.k.a. the definition of conditional probability).

3.5 Kernels

The choice of kernel is very important, while typically the mean function is fixed to a constant. The kernel must define a positive definite operator, and it determines the smoothness of the function and the correlation length of the prior. The most known kernel is the "exponential quadratic"

$$k(x, x') = \exp\left(-\frac{1}{2}\|x - x'\|_2\right). \tag{21}$$

For other kernels, see Rasmussen and Williams (2006) or `https://gattocrucco.github.io/lsqfitgp/docs/kernelsref.html`. Typically kernels are constructed starting from basic known kernels using the following properties:

Linearity Let L be a linear operator acting on vectors indexed by $\mathcal{X}$. Then $L_x L_{x'} k(x, x')$ is a valid kernel.

Reindexing Since x is just an index from the point of view of the GP, it can be transformed arbitrarily, so $\tilde{k}(x, x') = k(T(x), T(x'))$ is a valid kernel.

Product Like the Hadamard product of two p.d. matrices is p.d., the product of two kernels is a kernel. To see this quickly, consider that if $E[f(x)] = E[g(x)] = 0$ and $f \perp\!\!\!\perp g$, then $\operatorname{Cov}[f(x)g(x), f(x')g(x')] = \operatorname{Cov}[f(x), f(x')] \operatorname{Cov}[g(x), g(x')]$.

To make the properties more legible, we expand them into a less minimal list of allowed operations on kernels/processes:

1. The sum of two kernels is a kernel.
2. The product of two kernels is a kernel.
3. You can add a nonnegative constant to a kernel.
4. You can transform the input space of the kernel.
5. You can change the variance of the kernel with $f(x)k(x, x')f(x')$.
6. A kernel raised to a nonnegative integer power is a kernel. Kernels which allow real exponents are called "infinitely divisible."
7. A power series with nonnegative coefficients which converges on the range of a kernel can be applied to it to get another kernel. Valid also for multiple kernels. Example: $\exp(k(x, x'))$.
8. $\partial_x \partial_{x'} k(x, x')$ is the kernel of the derivative of the process.
9. In general all the properties apply to multiple processes or to processes with values in $\mathbb{R}^d$ by adding an index axis that selects the process. Example: $f(x) \in \mathbb{R}^2$ can be represented with the extended index $\tilde{x} = (x, i)$, $i = 1, 2$, $\tilde{f}(\tilde{x}) = f_i(x)$. In these cases the kernel evaluated as $k((x, 1), (x', 2))$ is called the "cross kernel between f_1 and f_2."

3.6 Equivalence with linear regression

The inference with GPs can be written as a linear regression by diagonalizing the covariance operator. Let $\{\phi_i(x)\}$ be a complete orthonormal set that diagonalizes $k(x, x')$:

$$k(x, x') = \sum_i \lambda_i \phi_i(x) \phi_i(x'). \tag{22}$$

In this context this is known as Mercer's theorem. By linearity, if we take a collection of i.i.d. Normally distributed variables β_i, then the process

$$f(x) = \sum_i \sqrt{\lambda_i} \phi_i(x) \beta_i \tag{23}$$

is a Gaussian process with the given covariance function k and mean zero. We define

$$X_{ji} = \sqrt{\lambda_i} \phi_i(x_j), \tag{24}$$

such that the process evaluated on the data points becomes the linear regression equation

$$f(\mathbf{x}) = X\boldsymbol{\beta}, \tag{25}$$

where $\boldsymbol{\beta}$ has a prior mean zero and prior diagonal covariance matrix $\mathrm{Cov}[\boldsymbol{\beta}] = I$.

To make inference given $f(\mathbf{x}) = \mathbf{y}$, first we write the posterior distribution for $\boldsymbol{\beta}$:

$$\begin{aligned} \mathrm{Cov}[X\boldsymbol{\beta}] &= XX^\top, & E[\boldsymbol{\beta} \mid X\boldsymbol{\beta} = \mathbf{y}] &= X^\top (XX^\top)^+ \mathbf{y} = X^+ \mathbf{y}, \\ \mathrm{Cov}[\boldsymbol{\beta}, X\boldsymbol{\beta}] &= X^\top, & \mathrm{Cov}[\boldsymbol{\beta} \mid X\boldsymbol{\beta} = \mathbf{y}] &= I - X^\top (XX^\top)^+ X = I - X^+ X. \end{aligned} \tag{26}$$

This result can be obtained either with the conditioning formulas or directly on the Normal density by regularizing the Dirac delta, completing the squares and using one of the matrix identities from section A.2. Then we get the posterior on $f(\mathbf{x}^*)$ by plugging the one for $\boldsymbol{\beta}$ into Equation 23:

$$\begin{aligned} X^*_{ji} &= \phi_i(x^*_j), & f(\mathbf{x}^*) &= X^* \boldsymbol{\beta}, \\ E[f(\mathbf{x}^*) \mid f(\mathbf{x}) = \mathbf{y}] &= X^* E[\boldsymbol{\beta} \mid f(\mathbf{x}) = \mathbf{y}] = \\ &= X^* X^\top (XX^\top)^+ \mathbf{y}, \\ \mathrm{Cov}[f(\mathbf{x}^*) \mid f(\mathbf{x}) = \mathbf{y}] &= X^* \,\mathrm{Cov}[\boldsymbol{\beta} \mid f(\mathbf{x}) = \mathbf{y}] X^{*\top} = \\ &= X^* X^{*\top} - X^* X^\top (XX^\top)^+ XX^{*\top}. \end{aligned} \tag{27}$$

Since

$$\begin{aligned} \Sigma_{x^*x^*} &= \mathrm{Cov}[X^*\boldsymbol{\beta}] = X^* X^{*\top}, \\ \Sigma_{x^*x} &= \mathrm{Cov}[X^*\boldsymbol{\beta}, X\boldsymbol{\beta}] = X^* X^\top, \\ \Sigma_{xx} &= \mathrm{Cov}[X\boldsymbol{\beta}] = XX^\top, \end{aligned} \tag{28}$$

Equation 27 leads again to Equation 15.

3.7 Error propagation

Consider the linear regression problem written with a separate error term

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad \operatorname{Cov}[\boldsymbol{\beta}] = \Lambda, \quad \operatorname{Cov}[\boldsymbol{\varepsilon}] = V, \quad \operatorname{Cov}[\boldsymbol{\beta}, \boldsymbol{\varepsilon}] = E[\boldsymbol{\beta}] = E[\boldsymbol{\varepsilon}] = 0, \tag{29}$$

which can be brought back to the form in section 3.6 with the substitutions $\boldsymbol{\beta}' = (\boldsymbol{\beta}, \boldsymbol{\varepsilon})$ and $X' = (X, I)$. Normal distributions are assumed throughout. The posterior on $\boldsymbol{\beta}$ given $\mathbf{y} = \bar{\mathbf{y}}$ is

$$\begin{aligned} E[\boldsymbol{\beta} \mid \bar{\mathbf{y}}] &= \operatorname{Cov}[\boldsymbol{\beta}, \mathbf{y}] \operatorname{Cov}[\mathbf{y}]^+ \bar{\mathbf{y}} = \\ &= \Lambda X^\top (X\Lambda X^\top + V)^+ \bar{\mathbf{y}} = \\ &= (X^\top V^{-1} X + \Lambda^{-1})^{-1} X^\top V^{-1} \bar{\mathbf{y}}, \end{aligned} \qquad \begin{aligned} \operatorname{Cov}[\boldsymbol{\beta} \mid \bar{\mathbf{y}}] &= \operatorname{Cov}[\boldsymbol{\beta}] - \operatorname{Cov}[\boldsymbol{\beta}, \mathbf{y}] \operatorname{Cov}[\mathbf{y}]^+ \operatorname{Cov}[\mathbf{y}, \boldsymbol{\beta}] = \\ &= \Lambda - \Lambda X^\top (X\Lambda X^\top + V)^+ X\Lambda = \\ &= (X^\top V^{-1} X + \Lambda^{-1})^{-1}, \end{aligned} \tag{30}$$

where the form in the last line is obtained with the identities in section A.2, under the assumption of invertible matrices, and is the most common way the solution is written since it is convenient to compute when V is diagonal and $\boldsymbol{\beta}$ is shorter than $\mathbf{y}$, i.e., when the errors are uncorrelated and there are less parameters than datapoints.

A nice property of this posterior is that it can be interpreted as the *prior* distribution of a linear transformation $\tilde{\boldsymbol{\beta}}$ of $\mathbf{y}$ and $\boldsymbol{\beta}$:

$$\begin{aligned} \tilde{\boldsymbol{\beta}}_{\bar{\mathbf{y}}} &= \Lambda X^\top (X\Lambda X^\top + V)^+ (\bar{\mathbf{y}} - \mathbf{y}) + \boldsymbol{\beta}, \\ E[\tilde{\boldsymbol{\beta}}_{\bar{\mathbf{y}}}] &= E[\boldsymbol{\beta} \mid \bar{\mathbf{y}}], \qquad \operatorname{Cov}[\tilde{\boldsymbol{\beta}}_{\bar{\mathbf{y}}}] = \operatorname{Cov}[\boldsymbol{\beta} \mid \bar{\mathbf{y}}], \end{aligned} \tag{31}$$

where the equalities of the moments can be checked with direct calculation, which also shows that

$$\operatorname{Cov}[\tilde{\boldsymbol{\beta}}, \boldsymbol{\beta}] = \operatorname{Cov}[\boldsymbol{\beta} \mid \bar{\mathbf{y}}], \quad \operatorname{Cov}[\tilde{\boldsymbol{\beta}}, \boldsymbol{\varepsilon}] = -\operatorname{Cov}[\boldsymbol{\beta} \mid \bar{\mathbf{y}}] X^\top, \quad \operatorname{Cov}[\tilde{\boldsymbol{\beta}}, \mathbf{y}] = 0, \tag{32}$$

where for $\operatorname{Cov}[\tilde{\boldsymbol{\beta}}, \boldsymbol{\varepsilon}]$ we use the fact that $\ker(X\Lambda X^\top + V) \subseteq \ker(\Lambda X^\top)$. The matrix that enters in the definition of $\tilde{\boldsymbol{\beta}}$ is the one that appears in $E[\boldsymbol{\beta} \mid \bar{\mathbf{y}}]$.

Due to linearity, the covariance matrix of $\tilde{\boldsymbol{\beta}}$ can be decomposed as a sum which separates the contribution of the two independent variables $\boldsymbol{\varepsilon}$ and $\boldsymbol{\beta}$:

$$\begin{aligned} \frac{\partial \tilde{\boldsymbol{\beta}}}{\partial \boldsymbol{\varepsilon}^\top} &= -\Lambda X^\top (X\Lambda X^\top + V)^+ = \\ &= -(X^\top V^{-1} X + \Lambda^{-1})^{-1} X^\top V^{-1}, \end{aligned} \qquad \begin{aligned} \frac{\partial \tilde{\boldsymbol{\beta}}}{\partial \boldsymbol{\beta}^\top} &= I - \Lambda X^\top (X\Lambda X^\top + V)^+ X = \\ &= (X^\top V^{-1} X + \Lambda^{-1})^{-1} \Lambda^{-1}, \end{aligned} \tag{33}$$

$$\begin{aligned} \operatorname{Cov}[\tilde{\boldsymbol{\beta}}] &= \frac{\partial \tilde{\boldsymbol{\beta}}}{\partial \boldsymbol{\varepsilon}^\top} \operatorname{Cov}[\boldsymbol{\varepsilon}] \frac{\partial \tilde{\boldsymbol{\beta}}}{\partial \boldsymbol{\varepsilon}}^\top + \frac{\partial \tilde{\boldsymbol{\beta}}}{\partial \boldsymbol{\beta}^\top} \operatorname{Cov}[\boldsymbol{\beta}] \frac{\partial \tilde{\boldsymbol{\beta}}}{\partial \boldsymbol{\beta}}^\top = \\ &= (X^\top V^{-1} X + \Lambda^{-1})^{-1} (\underbrace{X^\top V^{-1} X}_{\text{from } \boldsymbol{\varepsilon}} + \underbrace{\Lambda^{-1}}_{\text{from } \boldsymbol{\beta}}) (X^\top V^{-1} X + \Lambda^{-1})^{-1}. \end{aligned} \tag{34}$$

In this sense, the posterior covariance matrix of $\boldsymbol{\beta}$ can be split in two components attributed to different sources of uncertainty: in the usual application, the prior on

the parameters and the error on the data. In general, if the regression is split in many independent components, the posterior covariance matrix of one of the vectors can be split analogously:

$$\begin{aligned}
\mathbf{y} &= X\boldsymbol{\beta} + \sum_i \boldsymbol{\varepsilon}_i, \qquad \operatorname{Cov}[\boldsymbol{\varepsilon}_i, \boldsymbol{\varepsilon}_j] = \delta_{ij} V_i, \qquad V = \sum_i V_i, \\
\tilde{\boldsymbol{\beta}} &= \operatorname{Cov}[\boldsymbol{\beta}, \mathbf{y}] \operatorname{Cov}[\mathbf{y}]^{+} (\bar{\mathbf{y}} - \mathbf{y}) + \boldsymbol{\beta}, \\
\operatorname{Cov}[\tilde{\boldsymbol{\beta}}] &= (X^\top V^{-1} X + \Lambda^{-1})^{-1} \Lambda^{-1} (X^\top V^{-1} X + \Lambda^{-1})^{-1} + && \text{from } \boldsymbol{\beta}, \\
&\quad + \sum_i \Lambda X^\top (X\Lambda X^\top + V)^{+} V_i (X\Lambda X^\top + V)^{+} X\Lambda && \text{from } \boldsymbol{\varepsilon}_i. \qquad (35)
\end{aligned}$$

Note that a change in the covariance matrix of one of the source variables also affects all the components corresponding to the other variables. The sense in which this construction splits the uncertainty is not as a sum of unrelated covariance matrices, but as terms proportional to how much the posterior mean would change if the prior means were changed within their variance. It answers the question: "If other people had a different but similar amount of information on one of these independent variables, how much would we expect their result to differ from ours?"

I need a more formal way to say this.

With Gaussian processes, the equivalent formulation of the posterior as distribution of a linear function is known as Matheron's rule, and reads

Citation for Matheron (see Terenin's paper)

$$\begin{aligned}
\tilde{f}_{\mathbf{y}}(\mathbf{x}^*) &= \Sigma_{x^* x} \Sigma_{xx}^{+} (\mathbf{y} - f(\mathbf{x})) + f(\mathbf{x}^*), \\
E[\tilde{f}_{\mathbf{y}}(\mathbf{x}^*)] &= E[f(\mathbf{x}^*) \mid f(\mathbf{x}) = \mathbf{y}], \\
\operatorname{Cov}[\tilde{f}_{\mathbf{y}}(\mathbf{x}^*)] &= \operatorname{Cov}[f(\mathbf{x}^*) \mid f(\mathbf{x}) = \mathbf{y}] = \operatorname{Cov}[\tilde{f}_{\mathbf{y}}(\mathbf{x}^*), f(\mathbf{x}^*)], \\
\operatorname{Cov}[\tilde{f}_{\mathbf{y}}(\mathbf{x}^*), f(\mathbf{x})] &= 0. \qquad (36)
\end{aligned}$$

Although in this section we have used zero mean processes for notational convenience, the formulae for $\tilde{\boldsymbol{\beta}}$ and $\tilde{f}$ work as written with non-zero means since the variables only appear as differences.

3.8 Concatenation

Split the data points in two vectors as $f(\mathbf{x}) = (f(\mathbf{x}_1), f(\mathbf{x}_2)) = (\mathbf{y}_1, \mathbf{y}_2)$. Accordingly, divide the prior covariance matrices in blocks as

$$\Sigma_{xx} = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}, \qquad \Sigma_{xx^*} = \begin{pmatrix} \Sigma_{1*} \\ \Sigma_{2*} \end{pmatrix}. \qquad (37)$$

The prediction equations (Equation 15), with prior mean zero, become

$$\begin{aligned}
E[f(\mathbf{x}^*) \mid \mathbf{y}_1, \mathbf{y}_2] &= \begin{pmatrix}\Sigma_{*1} & \Sigma_{*2}\end{pmatrix} \begin{pmatrix}\Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22}\end{pmatrix}^{-} \begin{pmatrix}\mathbf{y}_1 \\ \mathbf{y}_2\end{pmatrix} = \\
&= \Sigma_{*1}\Sigma_{11}^{-}\mathbf{y}_1 + (\Sigma_{*2} - \Sigma_{*1}\Sigma_{11}^{-}\Sigma_{12})\Sigma_{22|1}^{-}(\mathbf{y}_2 - \Sigma_{21}\Sigma_{11}^{-}\mathbf{y}_1), \\
\operatorname{Cov}[f(\mathbf{x}^*) \mid \mathbf{y}_1, \mathbf{y}_2] &= \Sigma_{x^*x^*} - \begin{pmatrix}\Sigma_{*1} & \Sigma_{*2}\end{pmatrix} \begin{pmatrix}\Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22}\end{pmatrix}^{-} \begin{pmatrix}\Sigma_{1*} \\ \Sigma_{2*}\end{pmatrix} = \\
&= \Sigma_{x^*x^*} - \Sigma_{*1}\Sigma_{11}^{-}\Sigma_{1*} + \\
&\quad - (\Sigma_{*2} - \Sigma_{*1}\Sigma_{11}^{-}\Sigma_{12})\Sigma_{22|1}^{-}(\Sigma_{2*} - \Sigma_{21}\Sigma_{11}^{-}\Sigma_{1*}),
\end{aligned} \tag{38}$$

where we have used Equation 118 to block invert Σ_{xx}, abbreviating the Schur complement as $\Sigma_{22|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-}\Sigma_{12}$. The same result can be obtained by conditioning on one vector at a time:

$$\begin{aligned}
E[(f(\mathbf{x}^*), f(\mathbf{x}_2)) \mid \mathbf{y}_1] &= \begin{pmatrix}\Sigma_{*1} \\ \Sigma_{21}\end{pmatrix} \Sigma_{11}^{+}\mathbf{y}_1, \\
\operatorname{Cov}[(f(\mathbf{x}^*), f(\mathbf{x}_2)) \mid \mathbf{y}_1] &= \begin{pmatrix}\Sigma_{x^*x^*} & \Sigma_{*2} \\ \Sigma_{2*} & \Sigma_{22}\end{pmatrix} - \begin{pmatrix}\Sigma_{*1} \\ \Sigma_{21}\end{pmatrix} \Sigma_{11}^{+} \begin{pmatrix}\Sigma_{1*} & \Sigma_{12}\end{pmatrix}, \\
E[f(\mathbf{x}^*) \mid \mathbf{y}_1, \mathbf{y}_2] &= E[f(\mathbf{x}^*) \mid \mathbf{y}_1] + \operatorname{Cov}[f(\mathbf{x}^*), f(\mathbf{x}_2) \mid \mathbf{y}_1] \cdot \\
&\quad \cdot \operatorname{Cov}[f(\mathbf{x}_2) \mid \mathbf{y}_1]^{+}(\mathbf{y}_2 - E[f(\mathbf{x}_2) \mid \mathbf{y}_1]), \\
\operatorname{Cov}[f(\mathbf{x}^*) \mid \mathbf{y}_1, \mathbf{y}_2] &= \operatorname{Cov}[f(\mathbf{x}^*) \mid \mathbf{y}_1] - \operatorname{Cov}[f(\mathbf{x}^*), f(\mathbf{x}_2) \mid \mathbf{y}_1] \cdot \\
&\quad \cdot \operatorname{Cov}[f(\mathbf{x}_2) \mid \mathbf{y}_1]^{+} \operatorname{Cov}[f(\mathbf{x}_2), f(\mathbf{x}^*) \mid \mathbf{y}_1].
\end{aligned} \tag{39}$$

Matheron's rule concatenates as

$$\tilde{f}_{*|12} = \tilde{f}_{*|1} + \Sigma_{*2|1}\Sigma_{22|1}^{+}(\mathbf{y}_2 - \tilde{f}_{2|1}), \tag{40}$$

where we have abbreviated the notation in obvious ways. If $f(\mathbf{x}_2) \perp\!\!\!\perp f(\mathbf{x}_1)$, i.e., $\Sigma_{12} = 0$, this reduces to

$$\tilde{f}_{*|12} = \tilde{f}_{*|1} + \Sigma_{*2}\Sigma_{22}^{+}(\mathbf{y}_2 - f_2). \tag{41}$$

This gives a convenient practical recipe to bookkeep error propagation as information is added. Start with the prior covariance matrix Σ, divided in blocks, of all initial sets of variables. Upon considering another variable (either for conditioning or prediction), extend Σ with new blocks as appropriate. Represent conditioning with Matheron's function as a collection of matrices that give the linear coefficients w.r.t. each input variable. An unconditioned variable falls under this case as the identity matrix. To compute a new Matheron variable, conditioned on variables which are themselves either independent or conditioned on the same set, first compute the covariance matrices required for Equation 40 using error propagation, then determine the matrix coefficients of the new variable.

3.9 Extension

Let $\mathbf{y} = Af(\mathbf{x}) + \boldsymbol{\varepsilon}$. We want $f(\mathbf{x}^*) \mid \mathbf{y}$. Since $\mathbf{y}$ depends only on $f(\mathbf{x})$, intuition suggests it is possible to first obtain $f(\mathbf{x}) \mid \mathbf{y}$ and then use it to compute the posterior on $f(\mathbf{x}^*)$:

the information must "pass through" $f(\mathbf{x})$. In other words, we want to "extend" the posterior on $f(\mathbf{x})$ to $f(\mathbf{x}^*)$. We first compute the posterior directly, and then try to write it in terms of $f(\mathbf{x}) \mid \mathbf{y}$:

Try to use Σ^- in place of Σ^+ in this section

$$\begin{aligned}
\operatorname{Cov}[(f(\mathbf{x}), f(\mathbf{x}^*), \boldsymbol{\varepsilon})] &= \begin{pmatrix} \Sigma_{xx} & \Sigma_{xx^*} & \\ \Sigma_{x^*x} & \Sigma_{x^*x^*} & \\ & & V \end{pmatrix}, \\
E[f(\mathbf{x}) \mid \mathbf{y}] &= \Sigma_{xx} A^\top (A\Sigma_{xx}A^\top + V)^+ \mathbf{y}, \\
E[f(\mathbf{x}^*) \mid \mathbf{y}] &= \Sigma_{x^*x} A^\top (A\Sigma_{xx}A^\top + V)^+ \mathbf{y} = \\
&= \Sigma_{x^*x} (\Sigma_{xx}^+ \Sigma_{xx}) A^\top (A\Sigma_{xx}A^\top + V)^+ \mathbf{y} = \\
&= \Sigma_{x^*x} \Sigma_{xx}^+ E[f(\mathbf{x}) \mid \mathbf{y}], \qquad (42)
\end{aligned}$$

where we can plug $\Sigma_{xx}^+ \Sigma_{xx}$ into the expression, even though it is not the identity, because the null space of Σ_{xx} is within the one of Σ_{x^*x},

$$\begin{aligned}
\operatorname{Cov}[f(\mathbf{x}) \mid \mathbf{y}] &= \Sigma_{xx} - \Sigma_{xx} A^\top (A\Sigma_{xx}A^\top + V)^+ A\Sigma_{xx}, \\
\operatorname{Cov}[f(\mathbf{x}^*) \mid \mathbf{y}] &= \Sigma_{x^*x^*} - \Sigma_{x^*x} A^\top (A\Sigma_{xx}A^\top + V)^+ A\Sigma_{xx^*} = \\
&= \Sigma_{x^*x^*} - \Sigma_{x^*x} (\Sigma_{xx}^+ \Sigma_{xx}) A^\top (A\Sigma_{xx}A^\top + V)^+ A (\Sigma_{xx}\Sigma_{xx}^+)\Sigma_{xx^*} = \\
&= \Sigma_{x^*x^*} - \Sigma_{x^*x} \Sigma_{xx}^+ (\Sigma_{xx} - \operatorname{Cov}[f(\mathbf{x}) \mid \mathbf{y}]) \Sigma_{xx}^+ \Sigma_{xx^*} = \\
&= \operatorname{Cov}[f(\mathbf{x}^*) \mid f(\mathbf{x})] + \Sigma_{x^*x} \Sigma_{xx}^+ \operatorname{Cov}[f(\mathbf{x}) \mid \mathbf{y}] \Sigma_{xx}^+ \Sigma_{xx^*}. \qquad (43)
\end{aligned}$$

Analogously, given the Matheron variable for $f(\mathbf{x}) \mid \mathbf{y}$, the Matheron rule for $f(\mathbf{x}^*) \mid \mathbf{y}$ is

$$\tilde{f}_{\mathbf{y}}(\mathbf{x}^*) = \Sigma_{x^*x} \Sigma_{xx}^+ (\tilde{f}_{\mathbf{y}}(\mathbf{x}) - f(\mathbf{x})) + f(\mathbf{x}^*), \qquad (44)$$

which can be checked to be equal to the one obtained directly starting from $\mathbf{y}$.

I should devise a general explanation of Matheron rules based on Gaussian Bayesian networks.

3.10 Equivalence with unbiased estimation

In the frequentist interpretation of linear regression (with fixed covariances), the prior distribution on $\boldsymbol{\beta}$ becomes an independent unbiased estimate of $\boldsymbol{\beta}$ with known covariance matrix, while the distribution of $\boldsymbol{\varepsilon}$ coincides with the distribution of $\mathbf{y}$ since $\boldsymbol{\beta}$ is considered a fixed unknown quantity. Normality of the distributions is not assumed. Then the same formula that gives the posterior mean also defines the unbiased minimum variance estimator:

do the version with pseudoinverses ⇒ If Λ is invertible and $(I - VV^+)X = 0$, then the minimum of the quadratic form is the same, but with V^+ instead of V^{-1}.

$$
\begin{aligned}
&E[\mathbf{y}] = X\boldsymbol{\beta}, \qquad E[\boldsymbol{\beta}_P] = \boldsymbol{\beta}, \qquad \mathrm{Cov}[(\mathbf{y}, \boldsymbol{\beta}_P)] = \begin{pmatrix} V & \\ & \Lambda \end{pmatrix}, \qquad \boldsymbol{\beta}_P = \mathbf{0}, \\
&\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \begin{pmatrix} \mathbf{y} - X\boldsymbol{\beta} \\ \boldsymbol{\beta}_P - \boldsymbol{\beta} \end{pmatrix}^\top \begin{pmatrix} V & \\ & \Lambda \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{y} - X\boldsymbol{\beta} \\ \boldsymbol{\beta}_P - \boldsymbol{\beta} \end{pmatrix} = \\
&\quad = \left(\begin{pmatrix} X^\top & I \end{pmatrix} \begin{pmatrix} V & \\ & \Lambda \end{pmatrix}^{-1} \begin{pmatrix} X \\ I \end{pmatrix} \right)^{-1} \begin{pmatrix} X^\top & I \end{pmatrix} \begin{pmatrix} V & \\ & \Lambda \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} = \\
&\quad = (X^\top V^{-1} X + \Lambda^{-1})^{-1} X^\top V^{-1} \mathbf{y}, \\
&E[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta}, \qquad \mathrm{Cov}[\hat{\boldsymbol{\beta}}] = (X^\top V^{-1} X + \Lambda^{-1})^{-1} \leq \\
&\qquad\qquad \leq \mathrm{Cov}[M\mathbf{y} + N\boldsymbol{\beta}_P], \, \forall M, N \, : \, E[M\mathbf{y} + N\boldsymbol{\beta}_P] = \boldsymbol{\beta}.
\end{aligned} \tag{45}
$$

The same estimator can also be obtained as ridge regression with regularization Λ, however in this case the estimator is not unbiased because it is not produced by an estimate $\boldsymbol{\beta}_P$, Λ is added by fiat.

While Bayesian linear regression is equivalent to Bayesian Gaussian process regression, frequentist linear regression is subtly different from frequentist Gaussian process prediction, called Kriging. In the latter, the whole Gaussian process is a random variable, so there are no fixed unknown parameters to be estimated. The "Kriging equation" is

$$
\hat{f}(\mathbf{x}^*) = \Sigma_{x^* x} \Sigma_{xx}^{-1} f(\mathbf{x}), \tag{46}
$$

which corresponds to the mean of the conditional distribution in Equation 15. $\hat{f}(\mathbf{x}^*)$ is an unbiased minimum variance predictor. A predictor differs from an estimator in that its bias and covariance matrix are defined w.r.t. a random variable instead of a fixed quantity:

$$
\begin{aligned}
&E[\hat{f}(\mathbf{x}^*) - f(\mathbf{x}^*)] = 0, \\
&\mathrm{Cov}[\hat{f}(\mathbf{x}^*) - f(\mathbf{x}^*)] = \Sigma_{x^* x^*} - \Sigma_{x^* x} \Sigma_{xx}^{-1} \Sigma_{x x^*} \leq \\
&\qquad\qquad \leq \mathrm{Cov}[Mf(\mathbf{x}) - f(\mathbf{x}^*)], \, \forall M \, : \, E[Mf(\mathbf{x}) - f(\mathbf{x}^*)] = 0.
\end{aligned} \tag{47}
$$

4 Implementation of GP regression

Let n be the number of data points and m the number of test points. In evaluating numerically Equation 15, the most expensive inevitable computational step is inverting Σ_{xx}, which has computational cost $O(n^3)$ in general. The matrix multiplications by $\Sigma_{x^* x}$ take $O(mn^2)$, which could be comparable if m is as large as n, however in practice matrix multiplications are faster, we typically have freedom in choosing m, and above all the inversion step has to be repeated many times during the inference over the hyperparameters.

Here the first law of numerical linear algebra applies:

DO NOT INVERT MATRICES.

The second applies too:

IF YOU THINK YOU HAVE FOUND A CASE WHERE INVERTING A MATRIX MAKES SENSE, GO TO SLEEP. ON WAKING UP YOU WILL MORE LIKELY THAN NOT REALIZE THE MATRIX WAS NOT TO BE INVERTED.

In a few sections I will invert a matrix. This, however, only implies that you should doubt the chances of success of this work, not that you are allowed to invert matrices.

4.1 Decompositions

To compute an expression where a matrix inverse appear, decompose it as a product of special matrices. The following table summarizes the decompositions relevant to our task, sorted by computational cost:

Name	Assumptions	Product	Cost (matmul = 1)
Cholesky	Positive definite	$LL^\top$	1
LDLT	Symmetric	$LDL^\top$	5
QR	–	OL^T	20
Diagonalization	Symmetric	ODO^T	30
SVD	–	$O_1DO_2^T$	80

Symbol	L	D	O
Matrix property	Lower triangular	Diagonal	Orthogonal

The computational cost is proportional to n^3. I've measured the factors reported in the table timing the operations on my laptop with 1000×1000 matrices.

These decompositions can be found in any numerical linear algebra library. They produce matrices that allow to easily solve linear systems. Triangular matrices can be solved with line by line reduction, orthogonal ones with transposition, diagonal ones with division.

The Cholesky decomposition is the conventional default since it's the fastest one. Use it in this way:

$$\begin{aligned} \Sigma_{xx} &= LL^\top, \\ \Sigma_{x^*x}\Sigma_{xx}^{-1}\Sigma_{xx^*} &= \Sigma_{x^*x}L^{-\top}L^{-1}\Sigma_{xx^*} = \\ &= (L^{-1}\Sigma_{xx^*})^\top(L^{-1}\Sigma_{xx^*}). \end{aligned} \tag{48}$$

Use a library routine to compute $L^{-1}M$, in Python I use `scipy.linalg.solve_triangular`. Using other decompositions or calculating the likelihood is analogous.

Here is a discussion of each decomposition:

Cholesky It's the fastest generic decomposition. Depending on the implementation, it can be faster than matrix multiplication. However it requires the matrix to be strictly positive definite. The covariance matrices of GPs tend to be very ill-conditioned. The condition number is the ratio in absolute values of the maximum to the minimum eigenvalue; "ill-conditioned" means having a large condition number: if it is comparable to the inverse of the floating point ε (the

smallest number that can be added to 1 without disappearing, 1×10^{-16} for 64 bit floats), then the matrix numerically counts as degenerate. In practice Cholesky will refuse to decompose many GP matrices because they are not strictly positive definite to numerical accuracy.

The solution is to add a multiple of the identity to the matrix: $M \mapsto M + uI$, with a small $u > 0$. This is equivalent to adding u to each eigenvalue. Assuming that $u \ll \lambda_{\max}$ and $\lambda_{\min} \ll u$, the condition number goes from $\lambda_{\max}/\lambda_{\min}$ to $\lambda_{\max}/u$. An empirical rule to determine u is

$$u = n\varepsilon\hat{\lambda}_{\max}, \qquad \hat{\lambda}_{\max} = \max_i \sum_j |M_{ij}|. \tag{49}$$

Any estimate $\hat{\lambda}_{\max}$ of the maximum eigenvalue will do, here I use the upper bound given by Gershgorin's theorem. It is probably a good idea to use some multiple of this formula, say 16.

The value of u has a large effect on the determinant, so the same u should be used to decompose matrices for evaluating Normal densities to be compared, which happens with hyperparameter optimization.

Pivoted Cholesky (h/t Ilhan Polat, Lucas 2004) 2x slower variation of Cholesky that does not require strict positive definiteness but only positive semidefiniteness. Indefinite or negative definite matrices do not trigger an error, but the result won't make sense. It is implemented in LAPACK as *PSTRF, accessible in scipy through `scipy.linalg.lapack.dpstrf`.

LDLT 5x slower variation of Cholesky that does not require definiteness. D is not diagonal but 2×2 block diagonal. I do not expect this to be useful for GP regression.

Diagonalization For symmetric matrices diagonalization is 30× slower than Cholesky (on my computer). It does not require positive definiteness, however the computed spectrum will be chock-full of small negative numbers that somehow need to be removed. There are two alternatives: 1) promote all small or negative eigenvalues to a minimum value, a default choice for the threshold can be the u of Equation 49, 2) remove the small/negative eigenvalues, making the matrix low-rank and so evaluating a pseudoinverse instead of an inverse.

Even though it is way slower than Cholesky, it is useful in one special case. Models normally include a Normal independent error term in addition to the GP, so the complete covariance matrix is $V = \Sigma_{xx} + \sigma^2 I$, with σ the standard deviation of the error. If σ is unknown, it is a free hyperparameter, and V in general would have to be decomposed again for each value of σ that occurs in any iterative inference algorithm. However, by diagonalizing Σ_{xx}, we have

$$\begin{aligned} \Sigma_{xx} &= ODO^\top, \\ V &= \Sigma_{xx} + \sigma^2 I = ODO^\top + \sigma^2 I = O(D + \sigma^2 I)O^\top, \end{aligned} \tag{50}$$

so the matrix is already decomposed even if σ changes. If agnostic inference on σ would require at least 30 likelihood evaluations, this is a win. If there are other free hyperparameters that do not take advantage of diagonalization, then it's probably not worth it.

Another usage of diagonalization is debugging. On small problems by default I use diagonalization such that if the matrix turns out not positive definite I can look at the eigenvalues. Sometimes in more contrived schemes, if the code is not working, I try diagonalization instead of Cholesky and it goes smooth. Or the contrary. I admit ignorance.

QR Works on rectangular matrices. Since the output must be rectangular, one of the two matrices must be truncated to rectangular, either the orthogonal or the triangular one (which can be chosen). Rectangular matrices occur in GP regression when considering linear transformations (like the FK tables). However I've still not found a use for QR, I've always been forced for some reason to use SVD which is 4× slower.

SVD Works on rectangular matrices too. It is powerful and accurate because it produces orthogonal matrices, but it's the slowest one. Having a decomposition in terms of orthogonal matrices may be necessary when dealing with pseudoinverses. If you don't need the full larger orthogonal matrix, you should compute a "thin" SVD.

4.2 Pseudoinverses

If there are constraints on the process, the kernel operator may be degenerate, and likewise the covariance matrices it generates. Degeneracy also arises with finite transformations if it the output space is larger: $\mathbf{z} = A\mathbf{y}$ with A of size $m \times n$ and $m > n$.

In another sense, degeneracy occurs numerically if the spectrum of Σ_{xx} has many small eigenvalues, which typically happens with smooth processes since groups of close points are highly correlated. In this case one has to decide if to "inflate" the small eigenvalues or to remove them and work with a degenerate distribution. The standard technique is inflation because it can be done without diagonalization, as described for the Cholesky decomposition in section 4.1.

Another advantage of inflation is that it is a differentiable (almost everywhere if implemented with diagonalization and thresholding) transformation. This is important for hyperparameter inference; as the numerically determined rank of Σ_{xx} likely varies with hyperparameter values, the likelihood jumps as eigenvalues are added or removed from the pseudodeterminant, so with eigenvalue truncation it may be necessary to fix the rank once and for all at the start of the inference.

If the calculation of a pseudoinverse or a projector can not be avoided, use SVD (or diagonalization for symmetric matrices):

$$M = USV^\top, \qquad M^+ = VS^+U^\top, \qquad S^+_{ij} = \begin{cases} 1/S_{ji} & S_{ji} > 0, \\ 0 & S_{ji} = 0, \end{cases}$$
$$MM^+ = USS^+U^\top, \qquad M^+M = VS^+SV^\top. \tag{51}$$

Diagonalization costs 30 matmuls. For symmetric matrices, a faster approximate calculation can be computed by observing that

$$A^+ = \lim_{u\to 0^+} A(A^3 + uI)^{-1}A. \tag{52}$$

I tried this trick, also with a series like Jhurani and Demkowicz (2012), but it's not accurate (as expected). Try instead to use the pivoted Cholesky with some block formula, it should be faster and accurate.

In computing A^3, all the eigenvalues of A are raised to the same power, and thus also their ratios to the maximum eigenvalue. After inflating with uI, all eigenvalues below $\sqrt[3]{u}$ are "buried." Thus by fixing u to a small finite value (like the one in Equation 49), Equation 52 gives an approximation to the pseudoinverse with a symmetric formula. For example, to compute the second term of the conditioning:

$$\begin{aligned} \Sigma_{xx}^3 + uI &= LL^\top \text{ (Cholesky)},\\ \Sigma_{x^*x}\Sigma_{xx}^+\Sigma_{xx^*} &\approx \Sigma_{x^*x}\Sigma_{xx}L^{-\top}L^{-1}\Sigma_{xx}\Sigma_{xx^*} = (L^{-1}\Sigma_{xx}\Sigma_{xx^*})^\top(L^{-1}\Sigma_{xx}\Sigma_{xx^*}). \end{aligned} \tag{53}$$

This decomposition costs 1 Cholesky + 2 matmuls for the power + 1 matmul for the external term. Note: I made up this "Cholesky pseudoinverse" and I'm not sure how good or useful it is. I still have to try it thoroughly. In particular I don't know how to compute the log pseudo-determinant, although I guess 1/3 of $\sum_i 2\log L_{ii} - \operatorname{tr}(I - A^+A)\log u$ could work. See Jhurani and Demkowicz (2012) for a similar technique.

4.3 Hyperparameters

Apart from some very specific cases where inference over hyperparameters is analytic (conjugated priors), the posterior has to be maximized or sampled with an iterative numerical algorithm. The log posterior of the hyperparameters contains the log conditional probability of $f(\mathbf{x})$ (the likelihood) and the log prior of θ. We concentrate on the likelihood.

Regularizing the term $\delta((I - \Sigma\Sigma^+)(\mathbf{y} - \mathbf{m}))$ as a Normal with small but finite variance ε, the full minus log likelihood is

$$\begin{aligned} L = -\log p(f(\mathbf{x}) = \mathbf{y} \mid \theta) &= \frac{1}{2}\log\det(2\pi\tilde{\Sigma}) + \frac{1}{2}(\mathbf{y}-\mathbf{m})^\top\tilde{\Sigma}^{-1}(\mathbf{y}-\mathbf{m}),\\ \tilde{\Sigma} &= \Sigma + (I - \Sigma\Sigma^+)\varepsilon,\\ \mathbf{m} &= m(\mathbf{x};\theta), \quad \Sigma = \Sigma_{xx}(\theta). \end{aligned} \tag{54}$$

Most algorithms need the derivatives of the target function. Using the identities in section A.4, we obtain the gradient

$$\frac{\partial L}{\partial\theta} = \frac{1}{2}\operatorname{tr}\left(\tilde{\Sigma}^{-1}\frac{\partial\tilde{\Sigma}}{\partial\theta}\right) - (\mathbf{y}-\mathbf{m})^\top\tilde{\Sigma}^{-1}\frac{\partial\mathbf{m}}{\partial\theta} - \frac{1}{2}(\mathbf{y}-\mathbf{m})^\top\tilde{\Sigma}^{-1}\frac{\partial\tilde{\Sigma}}{\partial\theta}\tilde{\Sigma}^{-1}(\mathbf{y}-\mathbf{m}), \tag{55}$$

and the hessian

$$\frac{\partial^2 L}{\partial\theta\partial\theta'} = -\frac{1}{2}\operatorname{tr}\left(\tilde\Sigma^{-1}\frac{\partial\tilde\Sigma}{\partial\theta'}\tilde\Sigma^{-1}\frac{\partial\tilde\Sigma}{\partial\theta}\right) + \frac{1}{2}\operatorname{tr}\left(\tilde\Sigma^{-1}\frac{\partial^2\tilde\Sigma}{\partial\theta\partial\theta'}\right) + \\ + \frac{\partial\mathbf{m}^\top}{\partial\theta'}\tilde\Sigma^{-1}\frac{\partial\mathbf{m}}{\partial\theta} - (\mathbf{y}-\mathbf{m})^\top\tilde\Sigma^{-1}\frac{\partial^2\mathbf{m}}{\partial\theta\partial\theta'} + \\ + (\mathbf{y}-\mathbf{m})^\top\tilde\Sigma^{-1}\frac{\partial\tilde\Sigma}{\partial\theta'}\tilde\Sigma^{-1}\frac{\partial\mathbf{m}}{\partial\theta} + (\mathbf{y}-\mathbf{m})^\top\tilde\Sigma^{-1}\frac{\partial\tilde\Sigma}{\partial\theta}\tilde\Sigma^{-1}\frac{\partial\mathbf{m}}{\partial\theta'} + \\ - \frac{1}{2}(\mathbf{y}-\mathbf{m})^\top\tilde\Sigma^{-1}\frac{\partial^2\tilde\Sigma}{\partial\theta\partial\theta'}\tilde\Sigma^{-1}(\mathbf{y}-\mathbf{m}) + (\mathbf{y}-\mathbf{m})^\top\tilde\Sigma^{-1}\frac{\partial\tilde\Sigma}{\partial\theta}\tilde\Sigma^{-1}\frac{\partial\tilde\Sigma}{\partial\theta'}\tilde\Sigma^{-1}(\mathbf{y}-\mathbf{m}). \quad (56)$$

Some of the algorithms that use the Hessian require it to be positive definite. This is not the case in general. The Fisher information matrix is a nonnegative definite substitute for the Hessian. Using the Fisher matrix in a Newton-type minimizer is called "Fisher scoring." The Fisher matrix is:

$$E\left[\frac{\partial^2 L}{\partial\theta\partial\theta'}\,\middle|\,\theta\right] = \frac{1}{2}\operatorname{tr}\left(\tilde\Sigma^{-1}\frac{\partial\tilde\Sigma}{\partial\theta'}\tilde\Sigma^{-1}\frac{\partial\tilde\Sigma}{\partial\theta}\right) + \frac{\partial\mathbf{m}^\top}{\partial\theta'}\tilde\Sigma^{-1}\frac{\partial\mathbf{m}}{\partial\theta}. \quad (57)$$

It does not require to compute second derivatives. Expanding the definition of $\tilde\Sigma$ in Equations 54, 55 and 57, assuming $(I-\Sigma^+\Sigma)(\mathbf{y}-\mathbf{m}) = O(\sqrt{\varepsilon})$, and keeping only terms $O(\varepsilon^0)$ or larger, we obtain

$$L = \frac{n}{2}\log(2\pi) + \frac{1}{2}\log\operatorname{pdet}\Sigma + \frac{1}{2}(\mathbf{y}-\mathbf{m})^\top\Sigma^+(\mathbf{y}-\mathbf{m}) + \\ + \operatorname{tr}(I-\Sigma\Sigma^+)\log\varepsilon + \frac{1}{2}(\mathbf{y}-\mathbf{m})^\top\frac{I-\Sigma\Sigma^+}{\varepsilon}(\mathbf{y}-\mathbf{m}), \quad (58)$$

$$\frac{\partial L}{\partial\theta} = \frac{1}{2}\operatorname{tr}\left(\Sigma^+\frac{\partial\Sigma}{\partial\theta}\right) - (\mathbf{y}-\mathbf{m})^\top\left(\Sigma^+ + \frac{I-\Sigma\Sigma^+}{\varepsilon}\right)\frac{\partial\mathbf{m}}{\partial\theta} + \\ - \frac{1}{2}(\mathbf{y}-\mathbf{m})^\top\left(\Sigma^+ + 2\frac{I-\Sigma\Sigma^+}{\varepsilon}\right)\frac{\partial\Sigma}{\partial\theta}\Sigma^+(\mathbf{y}-\mathbf{m}) + O(\sqrt{\varepsilon}), \quad (59)$$

$$E\left[\frac{\partial^2 L}{\partial\theta\partial\theta'}\,\middle|\,\theta\right] = \frac{1}{2}\operatorname{tr}\left(\left(\Sigma^+ + 2\frac{I-\Sigma\Sigma^+}{\varepsilon}\right)\frac{\partial\Sigma}{\partial\theta}\Sigma^+\frac{\partial\Sigma}{\partial\theta'}\right) - 2\operatorname{tr}\left(\Sigma^+\frac{\partial\Sigma}{\partial\theta}(I-\Sigma\Sigma^+)\frac{\partial\Sigma}{\partial\theta'}\Sigma^+\right) + \\ + \frac{\partial\mathbf{m}^\top}{\partial\theta'}\left(\Sigma^+ + \frac{I-\Sigma\Sigma^+}{\varepsilon}\right)\frac{\partial\mathbf{m}}{\partial\theta} + O(\varepsilon). \quad (60)$$

If these formulas look too asymmetric to you, consider that it is possible to do the following replacement (which is not a standalone equality)

$$\left(\Sigma^+ + 2\frac{I-\Sigma\Sigma^+}{\varepsilon}\right)\frac{\partial\Sigma}{\partial\theta}\Sigma^+ \mapsto \left(\Sigma^+ + \frac{I-\Sigma\Sigma^+}{\varepsilon}\right)\frac{\partial\Sigma}{\partial\theta}\left(\Sigma^+ + \frac{I-\Sigma\Sigma^+}{\varepsilon}\right) \quad (61)$$

due to Equation 124. However I guess that numerically having only one $1/\varepsilon$ around is better.

Add a section on how to compute efficiently the gradient in reverse and forward mode.

Test this!

If the prior on θ is Normal with mean $\boldsymbol{\mu}$ and covariance matrix Θ, the gradient for the full minus log posterior gets an additional term $\Theta^{-1}(\theta-\boldsymbol{\mu})$, while the Hessian and

Fisher matrix get a Θ^{-1}. This is a case in which you are allowed to invert a matrix, since Θ is a small fixed known matrix completely under your control, e.g., you could fix directly Θ^{-1}.

But actually I'd decompose $\Theta = AA^\top$ and reparametrize as $\theta = A\tilde{\theta}$.

If the prior is not Normal, it can be convenient to express it as a Gaussian copula. Let $\theta = (\theta_1, \dots, \theta_d)$ and assume they are a priori independent with cumulative distributions $F_1, \dots, F_n$, i.e., $F_i(t) = P(\theta_i < t)$. Let v_i be i.i.d. standard Normal variables. Then θ can be written as $\theta_i = F_i^{-1}(\Phi(v_i))$, Φ being the Normal cdf. A dependence between the θ_i can be introduced by making the v_i correlated; provided the v_i have mean 0 and variance 1, the marginal distributions of the θ_i remain the same. The inference is performed on $\mathbf{v}$, and the final result is transformed back to θ.

The two main alternatives for summarizing the posterior are MCMC sampling and the Laplace approximation, i.e., approximating the distribution as Normal with mean equal to the maximum a posteriori (the "MAP") and precision (i.e., inverse covariance) matrix equal to the Hessian. It is convenient in any case to reparametrize the parameters to try to have a posterior more similar to a Normal distribution, particularly for Laplace. The Gaussian copula formulation is a good starting point. Unfortunately, sometimes the Laplace approximation fails particularly badly on GP hyperparameters, even if the model is correctly specified and the data abundant. In general it tends to be overconfident. If the hyperparameters are not of direct interest, i.e., they do not correspond to Physical quantities, but are just considered a way to implement a flexible model, this may not be a problem. The Fisher matrix tends to be less narrow than the Hessian, so a slight mitigation is using it as precision matrix in place of the Hessian.

Cite Basak? ⇒ Basak is not enough

Once the inference on the hyperparameters is complete, to keep into account the uncertainty on them when applying Equation 15, the easier way is sampling in turn the hyperparameters and then the predictive Normal distribution computed with the sampled hyperparameters, as indicated by Equation 20.

4.4 Latent Gaussian processes

If the data can not be expressed as a linear transformation of the Gaussian process plus Normal errors, then the joint distribution of the data and test points is not Normal and Equation 15 does not apply.

4.4.1 Non-Normal likelihood

If the data has continuous values (or discrete with many strata) with a non-Normal distribution where the Gaussian process represents a location parameter, then a convenient approximation is transforming the data with a variance-stabilizing transformation and treating it as Normal-distributed.

Consider a Poisson-distributed variable n. Its mean and variance are both μ. To approximate its distribution as Normal with a location parameter, we would need the variance to be fixed and not dependent on the mean. Let $n' = f(n)$. To first order, $E[n'] = f(\mu)$ and $\mathrm{Var}[n'] = f'(\mu)^2\mu$. If we decide to fix $\mathrm{Var}[n'] = 1/4$, then $f(\mu) = \sqrt{\mu}$ (plus an arbitrary constant). Thus the square root of Poisson data can

better be approximated as Normal, with the Gaussian process representing the mean $\sqrt{\mu(x)}$, and constant standard error $1/2$.

There are improved more sophisticated variance-stabilizing transformations for the Poisson distribution. However, the first-order one has the nice property that it can be modified to *not* be an approximation on bins with zero counts, the most problematic when approximating Poisson data as Normal, leading to the well known advice to make sure to have at least 5 counts per bin. Consider the original Poisson distribution for n and the Normal distribution on $\sqrt{n}$:

$$\begin{aligned} \mathcal{P}(n;\mu) &= \frac{\mu^n}{n!}e^{-\mu} \propto \mu^n e^{-\mu}, \\ \mathcal{N}(\sqrt{n};\sqrt{\mu},1/4) &= \frac{2}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}\frac{(\sqrt{n}-\sqrt{\mu})^2}{1/4}\right) \propto \exp(-2(\sqrt{n}-\sqrt{\mu})^2), \end{aligned} \tag{62}$$

where we have removed all factors which do not depend on μ, since they do not matter for inference on μ. If $n = 0$, the two likelihoods become

$$\mathcal{P}(n;\mu) \propto e^{-\mu}, \qquad \mathcal{N}(\sqrt{n};\sqrt{\mu},1/4) \propto e^{-2\mu}. \tag{63}$$

Thus if for bins with $n = 0$ we set the error on $\sqrt{\mu}$ to $1/\sqrt{2}$ instead of $1/2$, the contribution to the likelihood of that bins is exact even under the Normal approximation.

Small but not empty bins will still have an approximated likelihood. In particular, while the Poisson likelihood strongly disfavors $\mu \to 0$ if $n > 0$, the Normal approximation is chill even with $\mu = 0$ if n is small. So it might still be useful to aggregate small bins to reach 5 counts, while regions with many zero bins can be left as they are instead of excluding them or making a very large bin.

4.4.2 Nonlinear transformations

Even if the data has a conditional Normal distribution or can be approximated as such, it could be the case that the location parameter can not be expressed as a Gaussian process, for example if there are bounds. It is usually reasonable to write such location parameters as a nonlinear transformation of a Gaussian process.

With Normal data errors and a Gaussian process prior on the latent mean function, the inference becomes a nonlinear least squares problem, albeit unusually dense.

In nonlinear least-squares problems, the minus log distribution to be analyzed can be written as

$$Q(\boldsymbol{\beta}) = \frac{1}{2}\mathbf{r}(\boldsymbol{\beta})^\top \mathbf{r}(\boldsymbol{\beta}), \tag{64}$$

where $\mathbf{r}$ is called "residuals vector function." In Physics we usually call Q the chi-squared, but in statistics it's avoided because it suggests a distributional assumption that often is not made, and there are many possible likelihood ratio tests, each with its own chi-squared (Physicists usually deal only with the maximal LR test). Its gradient,

Hessian and Fisher information matrix are

$$\frac{\partial Q}{\partial \beta} = \mathbf{r}(\boldsymbol{\beta})^\top \frac{\partial \mathbf{r}}{\partial \beta}, \tag{65}$$

$$\frac{\partial^2 Q}{\partial \beta \partial \beta'} = \frac{\partial \mathbf{r}^\top}{\partial \beta'} \frac{\partial \mathbf{r}}{\partial \beta} + \mathbf{r}(\boldsymbol{\beta})^\top \frac{\partial^2 \mathbf{r}}{\partial \beta \partial \beta'}, \tag{66}$$

$$E\left[\frac{\partial^2 Q}{\partial \beta \partial \beta'} \,\middle|\, \boldsymbol{\beta} \right] = \frac{\partial \mathbf{r}^\top}{\partial \beta'} \frac{\partial \mathbf{r}}{\partial \beta}, \tag{67}$$

where the Fisher matrix is calculated assuming $E[\mathbf{r} \mid \boldsymbol{\beta}] = 0$, which is true if Q comes from a Normal distribution on $\mathbf{r}$. Otherwise, the "Fisher matrix" above is just a positive definite alternative to the Hessian.

Maybe the correct generalization is doing Fisher at each hierarchical level and summing.

The result of the inference is a Laplace approximation with the location of the maximum of Q as mean and the Fisher matrix as precision. I know that in particle Physics many people assume that likelihood maximization routines output the Hessian (called "*observed* Fisher information" in statistics), but Fisher is the standard for least squares. The maximum is searched by iteratively linearizing locally the problem and moving to the linear least squares solution

$$\Delta \boldsymbol{\beta} = X^+ \mathbf{r}, \qquad X_{ij} = \frac{\partial r_i}{\partial \beta_j}, \tag{68}$$

which is equivalent to Fisher scoring.

Our case translates to least squares in the following way. As before we indicate the Gaussian process values we are using with $f(\mathbf{x})$. Let $\mathbf{t}$ be the nonlinear transformation that maps $f(\mathbf{x})$ to the data mean, while the fixed data covariance matrix is V. The posterior on $f(\mathbf{x})$ is proportional to

Do the hierarchical version of this, with a graph of nonlinear transformations.

$$\mathcal{N}(\mathbf{y}; \mathbf{t}(f(\mathbf{x})), V) \cdot \mathcal{N}(f(\mathbf{x}); \mathbf{m}, \Sigma), \tag{69}$$

which gives a quadratic form

$$Q = \frac{1}{2}(\mathbf{y} - \mathbf{t})^\top V^+ (\mathbf{y} - \mathbf{t}) + \frac{1}{2}(f - \mathbf{m})^\top \Sigma^+ (f - \mathbf{m}), \tag{70}$$

implying our vectors of parameters and residuals are

I have to take into account the constraints $(I - \Sigma\Sigma^+)(f - \mathbf{m}) = 0$ and $(I - VV^+)(\mathbf{y} - \mathbf{t}) = 0$.

$$\boldsymbol{\beta} = f(\mathbf{x}), \qquad \mathbf{r}(\boldsymbol{\beta}) = \begin{pmatrix} F(\mathbf{y} - \mathbf{t}(\boldsymbol{\beta})) \\ G(\boldsymbol{\beta} - \mathbf{m}) \end{pmatrix}, \qquad \frac{\partial \mathbf{r}}{\partial \beta} = \begin{pmatrix} -F\, \partial \mathbf{t}/\partial \beta \\ G \end{pmatrix}, \tag{71}$$

where $V^+ = F^\top F$ and $\Sigma^+ = G^\top G$ are two symmetric decompositions of the pseudoinverses of the covariance matrices, determined with a method of choice (e.g., with Cholesky $V = LL^\top$, $V^{-1} = L^{-\top} L^{-1}$). The gradient and "Fisher matrix" are

Maybe in the degenerate case I should use directly the Schur complement instead of pseudoinverting Fisher?

$$\frac{\partial Q}{\partial \beta} = -(\mathbf{y} - \mathbf{t})^\top V^+ \frac{\partial \mathbf{t}}{\partial \beta} + (\boldsymbol{\beta} - \mathbf{m})^\top \Sigma^+, \tag{72}$$

$$E\left[\frac{\partial^2 Q}{\partial \beta \partial \beta'} \,\middle|\, \boldsymbol{\beta} \right] = \frac{\partial \mathbf{t}^\top}{\partial \beta} V^+ \frac{\partial \mathbf{t}}{\partial \beta'} + \Sigma^+. \tag{73}$$

If you use a nonlinear least squares routine, Equation 71 gives the required inputs.

In the following we assume the matrices are invertible. Let $J = \partial \mathbf{t}/\partial\beta$, and we intend J and $\mathbf{t}$ to be evaluated in the minimum $\boldsymbol{\beta}_{\text{min}}$. The Laplace-Fisher-approximated posterior can be written as a Matheron's rule using the linearization in the minimum:

Singular version?

$$\begin{aligned}
\begin{pmatrix} \boldsymbol{\varepsilon} \\ \boldsymbol{\beta} \end{pmatrix} &\sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} V & \\ & \Sigma \end{pmatrix}\right), \\
\tilde{\boldsymbol{\beta}} &= \Sigma J^\top (J\Sigma J^\top + V)^{-1}(\mathbf{y} - \mathbf{t} - J(\boldsymbol{\beta} - \boldsymbol{\beta}_{\text{min}}) - \boldsymbol{\varepsilon}) + \boldsymbol{\beta}, \\
E[\tilde{\boldsymbol{\beta}}] &= \boldsymbol{\beta}_{\text{min}}, \\
\text{Cov}[\tilde{\boldsymbol{\beta}}] &= E\left[\frac{\partial^2 Q}{\partial\beta\partial\beta'}\right]^{-1}.
\end{aligned} \tag{74}$$

To check the expected value, use the fact that $\boldsymbol{\beta}_{\text{min}}$ satisfies $\partial Q/\partial\beta = 0$. This rule allows an approximated version of the error propagation technique explained in sections 3.7 and 3.8, and is implemented by the Python package `lsqfit` (Lepage and Gohlke 2021).

To extend the posterior from $\boldsymbol{\beta} = f(\mathbf{x})$ to $f(\mathbf{x}^*)$, use the formulae of section 3.9.

4.4.3 Adding back the hyperparameters

With nonlinear data link and free hyperparameters with Normal prior, the posterior is proportional to

This is a model compatible with INLA, however INLA is focused on getting the marginals even if they are not Normal, while here we want a Normal approximation of the full posterior. With Poisson data and log link, it's a Log-Gaussian Cox Process.

$$p(f, \theta \mid \mathbf{y}) \propto \mathcal{N}(\mathbf{y}; \mathbf{t}(f(\mathbf{x})), V) \cdot \mathcal{N}(f(\mathbf{x}); \mathbf{m}, \Sigma) \cdot \mathcal{N}(\theta; \boldsymbol{\mu}, \Theta). \tag{75}$$

Assuming V does not depend on θ, the minus log posterior of Equation 70 becomes, by adding all terms which depend on θ,

$$\begin{aligned}
Q(f, \theta) = \frac{1}{2}\log \text{pdet}\,\Sigma(\theta) + \frac{1}{2}(f - \mathbf{m}(\theta))^\top \Sigma^+ (f - \mathbf{m}(\theta)) + \\
+ \frac{1}{2}(\mathbf{y} - \mathbf{t}(f))^\top V^+ (\mathbf{y} - \mathbf{t}(f)) + \frac{1}{2}(\theta - \boldsymbol{\mu})^\top \Theta^+ (\theta - \boldsymbol{\mu}),
\end{aligned} \tag{76}$$

where the free parameters are f and θ. The rank of Σ must not change over the allowed values of θ. Assuming Σ is invertible and $\mathbf{t}$ does not depend on θ, its gradient and "Fisher matrix" are

Constraints $(I - \Sigma\Sigma^+)(f - \mathbf{m}) = 0$, $(I - VV^+)(\mathbf{y} - \mathbf{t}) = 0$, and $(I - \Theta\Theta^+)\theta = 0$.

$$\begin{aligned}
\frac{\partial Q}{\partial f} &= -(\mathbf{y} - \mathbf{t})^\top V^+ \frac{\partial \mathbf{t}}{\partial f} + (f - \mathbf{m})^\top \Sigma^+, \\
\frac{\partial Q}{\partial \theta} &= \frac{1}{2}\,\text{tr}\left(\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta}\right) - (f - \mathbf{m})^\top \Sigma^+ \frac{\partial \mathbf{m}}{\partial \theta} - \frac{1}{2}(f - \mathbf{m})^\top \Sigma^{-1}\frac{\partial \Sigma}{\partial \theta}\Sigma^{-1}(f - \mathbf{m}) + (\theta - \boldsymbol{\mu})^\top \Theta^+,
\end{aligned} \tag{77}$$

$$
\begin{aligned}
E\left[\frac{\partial^2 Q}{\partial f \partial f'}\,\middle|\,\theta\right] &= \frac{\partial \mathbf{t}^\top}{\partial f} V^+ \frac{\partial \mathbf{t}}{\partial f'} + \Sigma^+, \\
E\left[\frac{\partial^2 Q}{\partial f \partial \theta}\,\middle|\,\theta\right] &= 0, \\
E\left[\frac{\partial^2 Q}{\partial \theta \partial \theta'}\,\middle|\,\theta\right] &= \frac{1}{2}\operatorname{tr}\left(\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta'}\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta}\right) + \frac{\partial \mathbf{m}^\top}{\partial \theta'}\Sigma^+\frac{\partial \mathbf{m}}{\partial \theta} + \Theta^+,
\end{aligned}
\tag{78}
$$

where the expected values are calculated under the distributions $\mathbf{y} - \mathbf{t}(f) \sim \mathcal{N}(0, V)$ and $f \sim \mathcal{N}(\mathbf{m}, \Sigma)$.

The dependence of Σ on θ makes this posterior less amenable to the Laplace approximation and forbids the definition of a Matheron's rule. It can be sampled with a Markov Chain algorithm. If Σ can be decomposed efficiently, the Fisher matrix may be used as a mass matrix for a Hybrid (a.k.a. Hamiltonian) Monte Carlo.

I'm not sure I can't do Matheron.

A different way to handle the presence of free covariance matrix hyperparameters is analogous to the one described in section 4.3: first detetermine the marginal posterior distribution of the hyperparameters, or an approximation of it, then sample it and for each value determine the conditional posterior distribution of the process.

We approximate the posterior on θ by marginalizing f from the posterior in Equation 76 with Laplace using the ff block of the Fisher matrix of Equation 78:

$$
\begin{aligned}
\int\limits_{AA^+x=x} \mathrm{d}x\, e^{-\frac{1}{2}x^\top A^+ x} &= \sqrt{\operatorname{pdet}(2\pi A)}, \\
f_{\min}(\theta) &= \arg\min_f Q(f, \theta), \\
A^+(\theta) &= \left.\frac{\partial \mathbf{t}^\top}{\partial f} V^+ \frac{\partial \mathbf{t}}{\partial f'}\right|_{f_{\min}} + \Sigma^+, \\
Q(f, \theta) &\approx Q(f_{\min}, \theta) + \frac{1}{2}(f - f_{\min})^\top A^+ (f - f_{\min}), \\
-\log p(\theta \mid \mathbf{y}) &= -\log \int \mathrm{d}f\, e^{-Q(f,\theta)} + \ldots \approx \\
&\approx Q(f_{\min}, \theta) + \frac{1}{2}\log \operatorname{pdet} A.
\end{aligned}
\tag{79}
$$

In practice this amounts to running the least squares procedure described in section 4.4.2 to determine $f_{\min}$ for each value of θ, and then compute $-\log p(\theta \mid \mathbf{y})$, to which a minimization or another generic algorithm is applied. This two-step procedure may be more convenient than dealing with the complete posterior Equation 76 at once because the specialized least squares algorithm takes advantage of the structure of the subproblem w.r.t. f.

Compute the Laplace approximation for θ with the implicit function theorem.

If θ is fixed to the value that maximizes $p(\theta \mid \mathbf{y})$, without taking into account its uncertainty, the result is an empirical Bayes approximation of the posterior and Matheron's rule can be applied to the "best" least squares subproblem. This is implemented in `lsqfit.empbayes_fit`.

4.5 Optimization

Computationally, the calculations with Gaussian processes are mostly linear algebra. This is true also when running iterative algorithms to deal with hyperparameters or nonlinearities. The key steps are:

Produce the covariance matrix Let n be the number of datapoints, or process points the data is linked to. Determining the covariance matrix takes up $O(n^2)$ time and memory. Complicated kernels can take a long time to compute. With large n, the matrix may not fit in memory. For example, with $n = 10\,000$, the covariance matrix is worth 0.8 GB using 64 bit floating point.

Decompose the covariance matrix Decomposing the prior covariance matrix of the datapoints is the most crucial step, since the number of datapoints is fixed by the problem and it appears at each iterative step when analyzing hyperparameters. Generic decomposition algorithms are $O(n^3)$. On my laptop, Cholesky with $n = 10\,000$ and fp64 takes 2.1 s, diagonalization 80 s.

Add section on computing the likelihood, mention that often decomposition is not the bottleneck

Sample predictions Let m be the number of points were the posterior of the process is evaluated. Since the values are strongly correlated, it is necessary to take into account the full joint distribution, instead of only computing the variance at each location. To sample from the posterior it is necessary to decompose the $m \times m$ covariance matrix. Often m can be chosen with some arbitrariness, so this may not be a problem.

4.5.1 Numerical operations

The first thing you need for numerical linear algebra is a software library. Most computing environments provide state of the art implementations for CPU, while to use the GPU it may be necessary to make a particular choice of library/GPU brand. Personally, I mostly use JAX on CPU.

The numerical type can either be 32 or 64 bit floating point. 32 bit is twice as fast as 64 bit, but the accuracy is so low that some white noise may be visible in the final result. Due to numerical degeneracy, the decomposition of the prior covariance matrix is particularly sensitive to numerical accuracy. In general, the accuracy required grows arbitrarily with the sample size, but 64 bit should be enough up to the currently unfeasible $n = 1 \times 10^9$.

The Google guy said this, find citation.

Most operations other than decompositions are matrix multiplications. On my installation, JAX matmuls are twice as fast as numpy matmuls.

4.5.2 Special decompositions

There are many special decomposition algorithms that take advantage of some properties of the matrix. Most of them do not apply to the PDF fit as currently set up, I list all I know of for completeness in case the evolution grid was significantly refined.

Circulant A circulant matrix has constant diagonals that wrap around the borders: $A_{ij} = A_{i+k \bmod n, j+k \bmod n}$. It arises from stationary periodic processes evaluated on an evenly-spaced grid aligned with the period. Any circulant matrix is diagonalized by the Discrete Fourier Transform, so the decomposition products can be computed in $O(n \log n)$ with FFT. The product of circulant matrices is circulant.

Toeplitz Toeplitz matrices have constant diagonals: $A_{ij} = A_{i+k,j+k}$. They typically arise from 1D stationary processes evaluated on evenly spaced points. ND stationary processes on evenly spaced axes-aligned grids produce nested block Toeplitz matrices, where the Toeplitz structure applies at the level of blocks, and each block is nested Toeplitz.

Toeplitz matrices can be stored in $O(n)$ space and decomposed at least as fast as $O(n^2)$. The decomposition can be computed and applied in a streaming fashion, to avoid storing $O(n^2)$ matrices. The algorithms are:

Schur $O(n^2)$, produces a column at a time of the Cholesky factor L.

Levinson-Durbin $O(n^2)$, produces a row at a time of L^{-1}, less numerically accurate than Schur.

Levinson-Durbin-Trench-Zohar $O(n^2)$, faster version of Levinson-Durbin specialized to solve the linear system, i.e., compute $L^{-\top}L^{-1}M$.

Generalized Schur $O(n \log^2 n)$, better complexity version of Schur to solve the linear system.

PCG $O(n \log n)$, preconditioned conjugate gradient iterative method to solve the linear system. Best complexity but less straightforward to apply due to the choice of preconditioning.

Moreover, Toeplitz matrix-vector multiplication is $O(n \log n)$ because it can be computed with FFT. The product of two Toeplitz matrices is not Toeplitz in general.

Any Toeplitz matrix can be embedded in a larger circulant matrix. However it may not be possible to embed a positive semidefinite Toeplitz matrix in a pos. sdef. circulant one. Since the FFT is fast, it may be worth trying to decompose a Toeplitz matrix in this way, and check numerically if the Fourier coefficients are nonnegative. Some stationary kernels always guarantee this property. Nested Toeplitz matrices can be embedded recursively too.

Sparse Sparse matrices can be decomposed with faster specialized algorithms. They typically arise from spatial processes, where far away points are assumed uncorrelated, by choosing an apposite finite support kernel.

What is the computational complexity of a sparse Cholesky in terms of the number of nonzero elements?

Sparse inverse Assumptions on the sparsity of the (pseudo-)inverse of the matrix can be very effective. They correspond to conditional independence assumptions, since they allow the Gaussian probability density to be factorized in terms that couple only subsets of the variables. They typically arise either from (1D) Markov processes, or from Gaussian Markov random fields (GRMF).

The sparsity pattern can be represented as a Bayesian Network. The decomposition can take advantage of the fact that if two (sets of) variables are separated by a third, conditioning must "pass through" the middle variable as in section 3.9. In the Markov chain case, the algorithm is called Kalman filter.

If inverse sparsity is not exact, it may be forced as an approximation, possibly by adding auxiliary points, called "inducing points."

Kronecker product If a ND kernel is separable as a product along axes, and the points form a grid aligned with the axes, then the covariance matrix is a kronecker product. Each factor can be decomposed separately.

Blockwise decomposition If the covariance matrix is block diagonal, each block can be decomposed separately. If it is not block diagonal, but a submatrix has a special property that allows a faster decomposition, the generic blockwise decomposition of Equation 115 allows to decompose it separately. This happens if most of the datapoints follow the requirements of one of the other special decompositions, but some don't.

Low rank approximation It is possible to extract a subset of r eigenspaces of a matrix in $O(rn^2)$, leading to a low-rank approximation of the matrix. The algorithms are iterative and use only matrix-vector multiplications. If the matrix properties allow faster matmul, the n^2 factor can be reduced accordingly.

For generic matrices, the algorithms are slower than generic decompositions, so they are convenient either for very small r and large n, or for special matrices, and if it makes sense to expect strong degeneracy.

Conjugate gradient Conjugate gradient algorithms solve positive definite linear systems only with matrix-vector multiplications. If the convergence is good, the number of iterations is independent of n, making the complexity proportional only to the matmul one. The convergence depends on the separation of the eigenvalues, which should be close to each other. Ill-conditioned matrices, like many covariance matrices, need preconditioning, which makes the method nontrivial to use. This is the state of the art method to handle large n without approximations.

Low rank matrices If a matrix can be written as $C = A\Sigma A^\top$ with A a tall $m \times n$ matrix, i.e., Σ smaller than C, then the pseudoinverse of C can be computed with the thin SVD of A, which takes $O(n^2 m)$:

$$\begin{aligned} A &= USW^\top, \qquad U(m \times n), S, W(m \times m), \\ C^+ &= US^{-1}W^\top \Sigma^{-1} WS^{-1}U^\top. \end{aligned} \tag{80}$$

Even if U is not a full orthogonal matrix, contraction along the longer side produces the identity: $U^\top U = I$. Alternatively with the thin QR:

$$\begin{aligned} A &= QR, \qquad Q(m \times n), R(n \times n), \\ C^+ &= QR^{-\top}\Sigma^{-1}R^{-1}Q^\top. \end{aligned} \tag{81}$$

If Σ is not invertible, decompose it as $\Sigma = LL^\top$ and fall back to the previous case with $C = (AL)I(AL)^\top$. The QR method will work only if L is full rank.

Woodbury If a full rank matrix is added to a low-rank one, the inversion can be reduced to the inversion of a smaller matrix using the Woodbury identity:

$$\begin{aligned} C &= A\Sigma A^\top + V, \\ C^{-1} &= V^{-1} - V^{-1}A^\top(A^\top V^{-1}A + \Sigma^{-1})^{-1}AV^{-1}. \end{aligned} \tag{82}$$

This formula is convenient if V is fixed and can be inverted once, while Σ, smaller than V, depends on parameters and its decomposition needs to be recomputed frequently. See section A.2 for alternatives when Σ and/or V are not invertible. This is the case of the PDF fit, since the grid is smaller than the number of datapoints, and the data error covariance is given.

4.6 Software

Unfortunately there is not a single software for Gaussian process regression that implements all the useful algorithms. There are many choices, each specialized to particular cases. The most general-purpose tool, which I consider as default choice, is the popular probabilistic programming tool PYMC (www.pymc.io), which has good GP support. GPyTorch (gpytorch.ai) instead is the state of the art for scaling to large datasets with brute force when no decomposition trick applies. For the rest, see the comparison page on Wikipedia (Comparison of Gaussian process software). If you know of another software not listed there, please drop me a line!

4.7 Positivity constraint

The positivity constraint on the PDFs does not play nicely with the Normal distribution because it is not linear. It is very not linear. If we wrote the PDFs as nonlinear positive transformations of latent Gaussian processes, then we would not be able anymore to impose exactly the sum rules using kernel derivatives.

If the final errors are small enough, such that the data pushes the functions mostly away from zero, the solution is sampling from the posterior on a finely spaced grid and removing all samples where one of the PDFs in one point is below zero. This works because truncating the support of the distribution can be equivalently done on the prior or on the posterior, Bayes can only change ratios of probabilities.

To make the sampling efficient it is important that the posterior PDFs are never below zero with nonnegligible probability. This will probably require excluding most of the region at low x with weak information. Samples are relatively cheap to generate once the covariance matrix has been decomposed, so it is convenient to sample many times for each hyperparameter sample.

Try this R packages: `tmvtnorm`, `TruncatedNormal`.

5 PDF GP prior

To express the fact that we expect the variation of the PDFs to increase arbitrarily as $x \to 0^+$, we use the Gibbs kernel (Rasmussen and Williams 2006, p. 93):

This kernel means we assume the PDFs are C^∞. What does the theory say? C^2?

$$k(x, x') = \sigma^2 \sqrt{\frac{2\ell(x)\ell(x')}{\ell^2(x) + \ell^2(x')}} \exp\left(-\frac{(x - x')^2}{\ell^2(x) + \ell^2(x')}\right), \tag{83}$$

where $\ell(x)$ gives the correlation length at x and σ^2 is a variance hyperparameter. An alternative could be transforming x to have uniform correlation length and then use a stationary kernel. We take the lengthscale to be uniform w.r.t. $\log(x + \varepsilon)$, so

$$\ell(x) = \frac{\ell_0}{\partial \log(x + \varepsilon)/\partial x} = (x + \varepsilon)\ell_0, \tag{84}$$

where ℓ_0 is a scale hyperparameter and ε is a small number to avoid a singularity in 0. We need to consider the point $x = 0$ because it is one of the endpoints of the sum rules, but we don't care about the shape of the function arbitrarily close to 0.

The hyperparameters σ and ℓ_0 are per-PDF.

The T_* functions do not have sum rules, so we assign an independent zero-mean process with kernel (83) to each of them:

$$T_i \sim \mathcal{GP}(0, k(x, x')), \qquad i \in \{3, 8, 15\}. \tag{85}$$

The sum rules over the V_* functions can be expressed in terms of their primitives, so we assign an independent process to each of the primitives:

$$V_i(x) = \tilde{V}_i'(x), \qquad \tilde{V}_i \sim \mathcal{GP}\big(0, \ell(x)k(x, x')\ell(x')\big), \qquad i \in \{\cdot, 3, 8, 15\}, \tag{86}$$

where we have rescaled the kernel with the lengthscale because the derivative of the unscaled process would scale in amplitude as $1/\ell(x)$. The sum rules then are the simple linear constraints

$$\begin{aligned} \tilde{V}(1) - \tilde{V}(0) &= 3, & \tilde{V}_3(1) - \tilde{V}_3(0) &= 1, \\ \tilde{V}_8(1) - \tilde{V}_8(0) &= 3, & \tilde{V}_{15}(1) - \tilde{V}_{15}(0) &= 3. \end{aligned} \tag{87}$$

For Σ and g we have a sum rule written in terms of the primitives of $x\Sigma(x)$ and $xg(x)$ and we want them to go like a power for $x \to 0^+$. So we define

$$\begin{aligned} x\Sigma(x) &= \tilde{\Sigma}'(x), & xg(x) &= \tilde{g}'(x), \\ \tilde{\Sigma}(x) &= \frac{x^{\alpha_\Sigma + 1}}{\alpha_\Sigma + 2}\tilde{\tilde{\Sigma}}(x), & \tilde{g}(x) &= \frac{x^{\alpha_g + 1}}{\alpha_g + 2}\tilde{\tilde{g}}(x), \\ \tilde{\tilde{\Sigma}} &\sim \mathcal{GP}(0, k(x, x')), & \tilde{\tilde{g}} &\sim \mathcal{GP}(0, k(x, x')), \end{aligned} \tag{88}$$

where α_Σ and α_g are hyperparameters. Since the lengthscale of $\tilde{\tilde{\Sigma}}$ is $\ell(x) \approx x$, and $\tilde{\Sigma}$ is rescaled with $x^{\alpha_\Sigma + 1}$, $x\Sigma(x)$ goes like x^{α_Σ}, with one x removed by the derivative due to the lengthscale. Analogously for g. The sum rule becomes

$$\tilde{\Sigma}(1) + \tilde{g}(1) - \tilde{\Sigma}(0) - \tilde{g}(0) = 1. \tag{89}$$

Finally, the constraints at $x = 1$ are

$$\Sigma(1) = 0, \qquad g(1) = 0, \qquad V_i(1) = 0, \qquad T_i(1) = 0. \tag{90}$$

To build the prior covariance matrix for this model, it is necessary to compute the various cross kernels implied by Equations 86 and 88, and the covariance blocks between the data and the quantities in Equations 87, 89 and 90. As an example, we show the calculation of the $1 \times n$ covariance block between $\tilde{V}(1) - \tilde{V}(0)$ and some hypothetical data which depends linearly on V as $\mathbf{y} = MV(\mathbf{x})$:

$$\begin{aligned}
\mathrm{Cov}[\tilde{V}(1) - \tilde{V}(0), MV(\mathbf{x})] &= \mathrm{Cov}[\tilde{V}(1), V(\mathbf{x})]M^\top - \mathrm{Cov}[\tilde{V}(0), V(\mathbf{x})]M^\top, \\
\mathrm{Cov}[\tilde{V}(1), V(\mathbf{x})] &= k_{\tilde{V},V}(1, \mathbf{x}), \\
\mathrm{Cov}[\tilde{V}(0), V(\mathbf{x})] &= k_{\tilde{V},V}(0, \mathbf{x}), \\
k_{\tilde{V},V}(x, x') &= \mathrm{Cov}[\tilde{V}(x), V(x')] = \mathrm{Cov}[\tilde{V}(x), \tilde{V}'(x')] = \\
&= \partial_{x'} k_{\tilde{V}}(x, x').
\end{aligned} \tag{91}$$

6 Models

I am developing a general purpose Python package, `lsqfitgp`[1], to implement the analysis (`https://github.com/Gattocrucco/lsqfitgp`). The software allows to express directly the model as written in section 5 and generates automatically the matrices. It implements the techniques described in the previous sections for inference. The interface is inspired by the `lsqfit` package by Peter G. Lepage (the inventor of the VEGAS Monte Carlo integrator) which he uses for the analysis of lattice QCD calculations (Lepage and Gohlke 2021).

Currently I am not using real data. I only use data simulated from the prior and check the consistency (what is known in Physics as "closure test"). I deem it prudent to only ever test the inference on fake data until I feel confident about the analysis. I know all too well the bias that can arise from human judgement, particularly here where the data has been analyzed and reanalyzed for years. I don't know myself QCD beyond "there are quarks, gluons and SU_3 somewhere and Jews," so I don't have personal expectations about what is the right result, and this is a feature.

The following table describes the tests and iterations I have done over time, listing their features. For each version of the analysis there is a pointer to a section below which explains the details, and a link to online documentation with the output figure and the complete code.

[1] No, -gp does not stand for my initials.

						Features:			
Sec.	Link	Data	Grid	Time	Works	Constr.	Hyp.	Nonlin.	Tables
1	pdf1	20	8×30	1 s	X	X			
2	pdf2	20	8×30	1 s	X	X			
3	pdf3	20	9×30	1 s	X	X			
4	pdf4	20	9×30	5 s	X	X	X		
5	pdf5	20	9×30	20 s	X	X		10	
6	pdf6	20	9×30	15 s	X	X		10	X
7	pdf7	20	9×30	35 min		X	X	10	X
8	pdf8	20	9×49	1 hour	X	X	X	10	X
9	pdf9	3000	9×34	1 min	~	X	X		
10	pdf10	3000	9×34	1 min	~	X	X		

Column	Meaning
Data	Total number of datapoints
Grid	Number of PDF functions × number of x points the data is linked to
Time	Total running time measured on my laptop
Constr.	Informative prior with sum rules
Hyp.	Free hyperparameters with inference
Nonlin.	Number of datapoints with nonlinear link to PDFs
Tables	Free FK tables with inference

Models 1 to 7 have an unrealistic simplified prior, different from the one described in section 5, and a small number of datapoints. They are proof of concepts that test each of the features in turn. Model 8 uses the realistic prior, but still with a small amount of data. Model 9 takes a step back and tries to scale to a full dataset with less features than model 8.

We always check that the sum rules are satisfied both in the fake PDFs drawn from the prior and in the posterior, using an approximate integration with the trapezoid rule. When hyperparameters are free, we represent their prior with a Gaussian copula, i.e., fixed transformations of the hyperparameters have a joint Normal distribution. The FK tables are generated randomly.

Model 1 A basic test with only Gaussian process regression and an unrealistic prior.

Model 2 An improvement over model 1 in the way the model is written, the rest being equal.

Model 3 GP regression with more realistic (but not correct) constraints written in the flavor basis.

Model 4 Adds scale, variance and correlation decay behavior as free hyperparameters to model 3.

Model 5 Uses a quadratic link for half the datapoints in model 3, without free hyperparameters.

Model 6 Adds errors on the FK tables to model 5, with covariance matrix rank 9.

Model 7 Adds a single free scale hyperparameter to model 6. The inference is very slow and overconfidently misses some of the true PDF functions.

Model 8 Uses the realistic prior in the evolution basis, with a small amount of datapoints, but with all features (inference over hyperparameters and FK tables, nonlinear observables). As model 7, it is very slow. The result is shown in Figure 1. The closure test is successful. The hyperparameters are fixed to their MAP, reported in the following table:

Transf. parameter	True	MAP	Parameter	True	MAP
log(scale)	-1.3	-1.2	scale	0.28	0.29
U(alpha_Sigma)	0.23	0.22	alpha_Sigma	0.090	0.087
U(alpha_g)	0.72	0.081	alpha_g	0.27	0.032

The inference does not seem to recover correctly α_g.

Model 9 Implements the prior of section 5, but with the same scale and variance parameters shared by all PDFs. The posterior of the hyperparameters is Hessian-Laplace-approximated. The x grid is the standard NNPDF grid, and the data is linked only to points with $x > 1 \times 10^{-4}$. Includes only data with linear link, but with a realistic amount of datapoints (3000). The FK tables are fixed, filled to resemble qualitatively the real FK tables, and scaled using the true values of the exponents for $x \to 0$ to avoid yielding very large observable values, which may give numerical problems. Mostly ok, but the uncertainty on the hyperparameters is a bit overconfident. Half of the running time, 30 s, is spent compiling the likelihood and its derivatives with JAX.

Model 10 Same as model 9, but with per-PDF scale and variance hyperparameters. Results similar to 9. Shown in Figures 2, 3, 4.

References

Aggarwal, Ritu et al. (2022). *New constraints on the up-quark valence distribution in the proton*. DOI: 10.48550/ARXIV.2209.06571.

Bernstein, Dennis S. (2018). *Scalar, Vector, and Matrix Mathematics: Theory, Facts, and Formulas*. Revised and expanded edition. Princeton: Princeton University Press. ISBN: 9780691151205.

Del Debbio, Luigi, Tommaso Giani, and Michael Wilson (2022). "Bayesian approach to inverse problems: an application to NNPDF closure testing". In: *The European Physical Journal C* 82.4, p. 330. DOI: 10.1140/epjc/s10052-022-10297-x.

Gbedo, Yémalin Gabin and Mariane Mangin-Brinet (July 2017). "Markov chain Monte Carlo techniques applied to parton distribution functions determination: Proof of concept". In: *Phys. Rev. D* 96 (1), p. 014015. DOI: 10.1103/PhysRevD.96.014015.

Gramacy, Robert B. (2020). *Surrogates. Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. Chapman and Hall/CRC. ISBN: 978-0-367-41542-6.

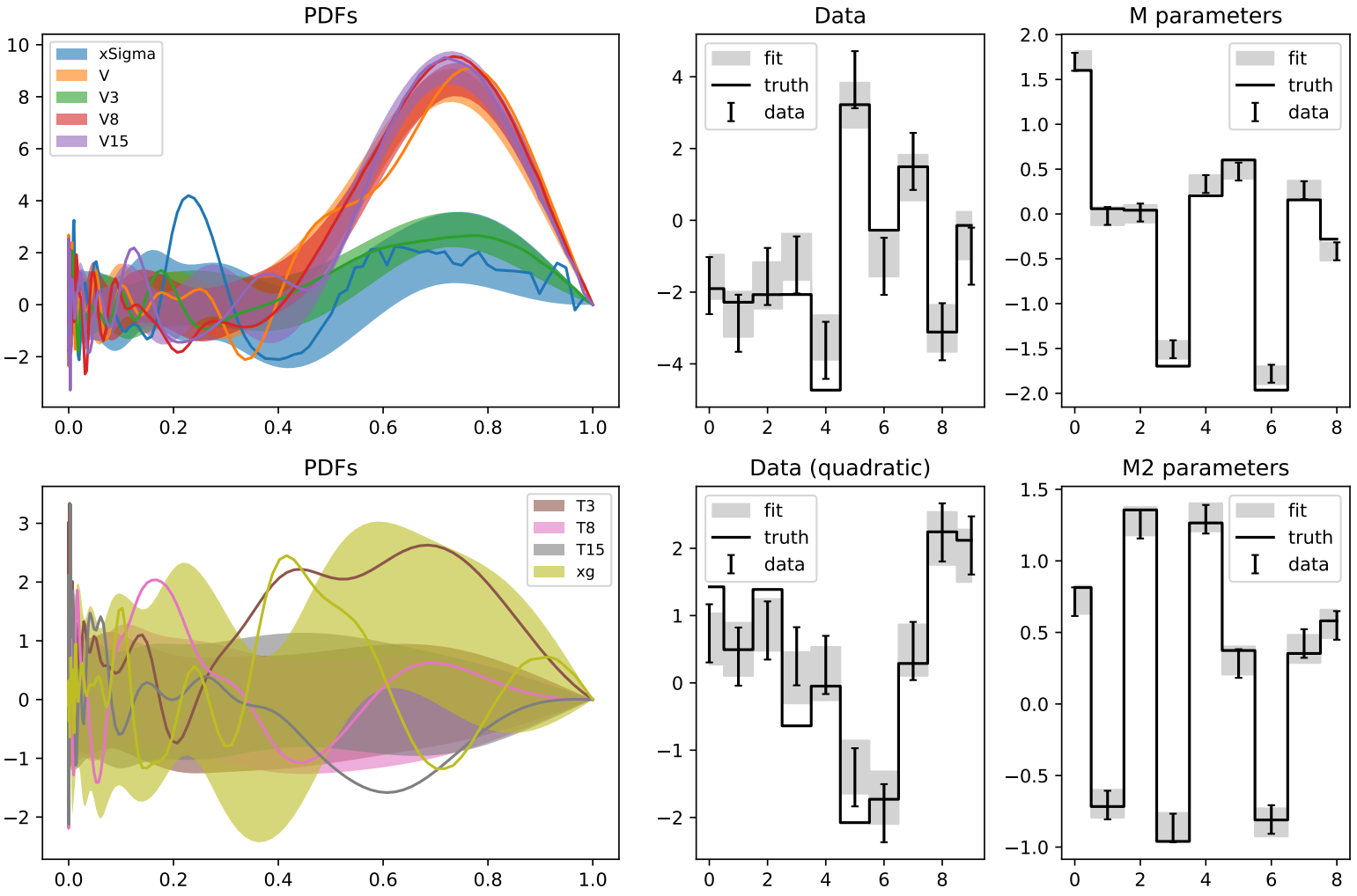

Figure 1: Results of model 8. The lines are the true (fake) PDFs used to generate the fake data, while the bands are the $\pm 1\sigma$ intervals of the posterior. The rightmost plots show the coefficients of the covariance eigenvectors of the FK tables for linear and quadratic observables.

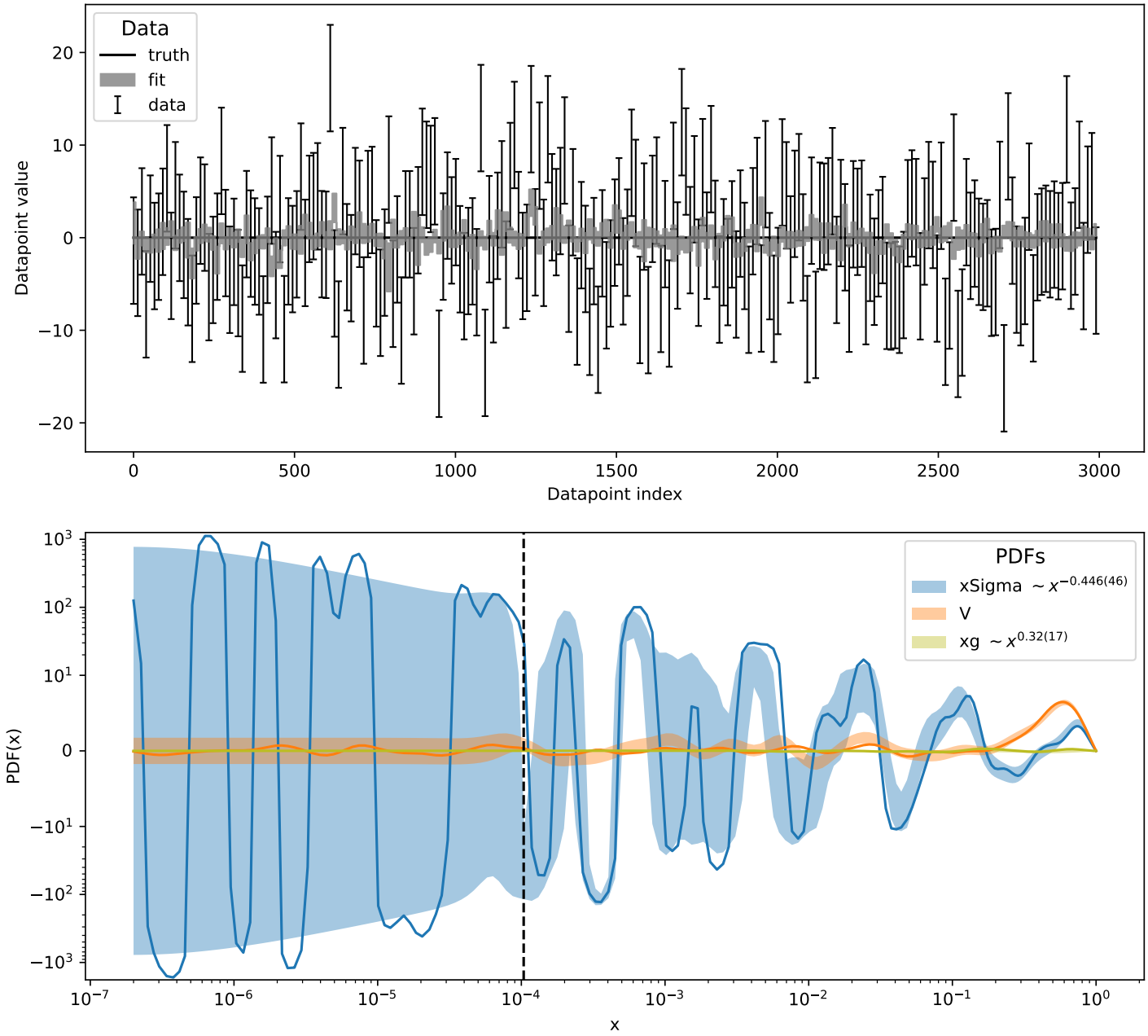

Figure 2: Results of model 10. Top plot: Data fit. Bottom plot: PDFs. The lines are the true (fake) PDFs used to generate the fake data, while the bands are the $\pm 1\sigma$ intervals of the posterior with hyperparameters fixed to the MAP. The vertical dashed line marks the start of the x values that are linked to the data.

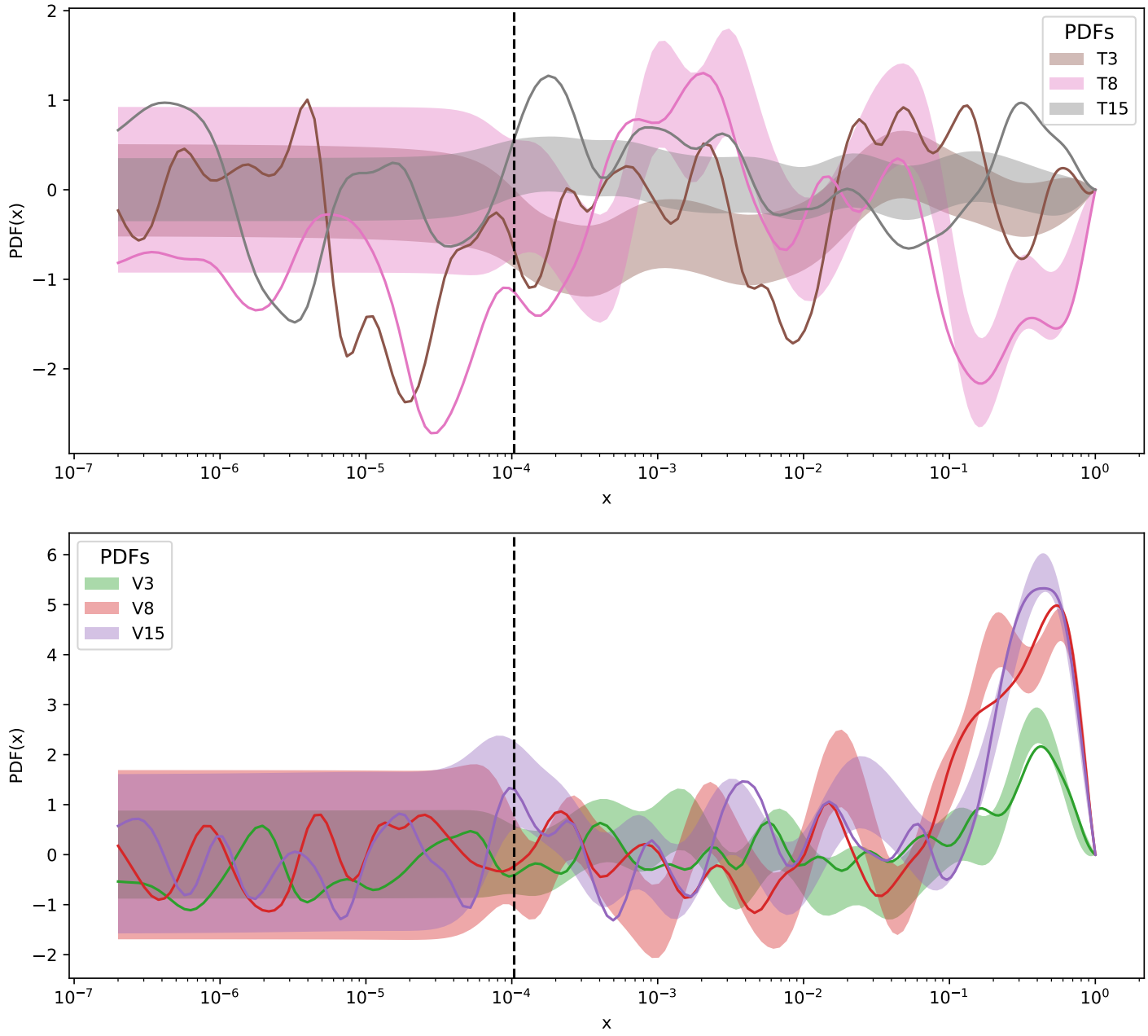

Figure 3: Results of model 10. The lines are the true (fake) PDFs used to generate the fake data, while the bands are the $\pm 1\sigma$ intervals of the posterior with hyperparameters fixed to the MAP. The vertical dashed line marks the start of the x values that are linked to the data.

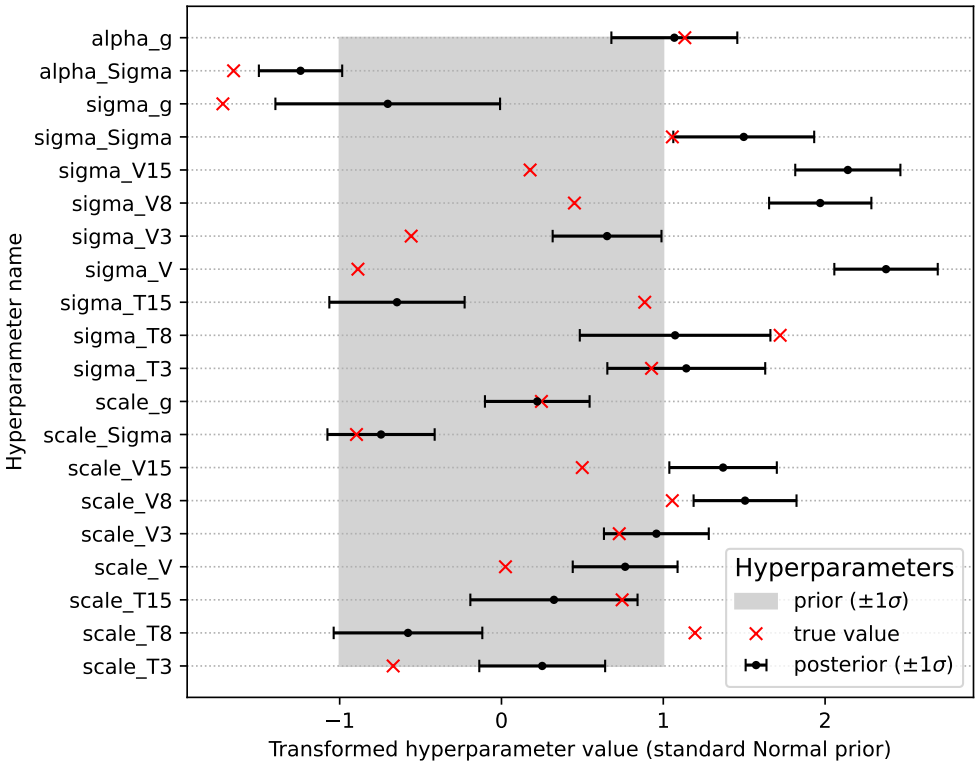

Figure 4: Results of model 10, inference on hyperparameters. Note the large discrepancies for the `sigma_V*` parameters, I guess they are caused by the interaction with the constraints.

Holbrook, Andrew (2018). "Differentiating the pseudo determinant". In: *Linear Algebra and its Applications* 548, pp. 293–304. ISSN: 0024-3795. DOI: 10.1016/j.laa.2018.03.018.

Jhurani, Chetan and Leszek Demkowicz (2012). "Multiscale modeling using goal-oriented adaptivity and numerical homogenization. Part II: Algorithms for the Moore–Penrose pseudoinverse". In: *Computer Methods in Applied Mechanics and Engineering* 213-216, pp. 418–426. ISSN: 0045-7825. DOI: 10.1016/j.cma.2011.06.003.

Lepage, Peter and Christoph Gohlke (Feb. 2021). *lsqfit*. Version 11.8. DOI: 10.5281/zenodo.4568470.

Lucas, Craig (2004). *LAPACK-Style Codes for Level 2 and 3 Pivoted Cholesky Factorizations. Numerical Analysis Reports No. 442*. 442. Department of Mathematics, University of Manchester, Manchester M13 9PL, England. eprint: https://www.maths.manchester.ac.uk/~higham/narep/narep442.pdf.

Mangin-Brinet, Mariane and Yémalin Gabin Gbedo (2017). "Markov Chain Monte Carlo techniques applied to Parton Distribution Functions determination: proof of concept". In: *Proceedings of XXV International Workshop on Deep-Inelastic Scattering and Related Subjects — PoS(DIS2017)*. Vol. 297, p. 213. DOI: 10.22323/1.297.0213.

Murphy, Kevin P. (Aug. 14, 2023). *Probabilistic Machine Learning: Advanced Topics*. MIT Press. URL: http://probml.github.io/book2.

Rasmussen, Carl Edward and Christopher K. I. Williams (2006). *Gaussian Processes for Machine Learning*. ISBN: 0-262-18253-X. URL: http://www.gaussianprocess.org/gpml/.

Schott, James R. (2017). *Matrix analysis for statistics*. 3rd ed. Hoboken, New Jersey: John Wiley & Sons. ISBN: 9781119092483.

Stein, Michael L. (1999). *Interpolation of Spatial Data. Some Theory for Kriging*. Springer Series in Statistics. Springer New York, NY. ISBN: 978-0-387-98629-6, 978-1-4612-7166-6, 978-1-4612-1494-6. DOI: 10.1007/978-1-4612-1494-6.

Tong, Y. L. (1990). *The Multivariate Normal Distribution*. 1st ed. Springer New York. ISBN: 978-1-4613-9657-4. DOI: 10.1007/978-1-4613-9655-0.

Wendland, Holger (2004). *Scattered Data Approximation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press. ISBN: 978-0-511-61753-9, 978-0-521-13101-8. DOI: 10.1017/CBO9780511617539.

A Matrix formulae

For textbook references see Bernstein (2018) and Schott (2017).

A.1 Pseudoinverse

The pseudoinverse of any matrix A (also rectangular) is the unique matrix A^+ that satisfies the following "Moore-Penrose conditions:"

$$AA^+A = A, \qquad A^+AA^+ = A^+, \qquad AA^+ = (AA^+)^\top, \qquad A^+A = (A^+A)^\top. \tag{92}$$

The order of these conditions is fixed by convention and relevant for some notations. A^+ can be computed with

$$A^+ = \begin{cases} A^\top (AA^\top)^{-1} & \text{if } AA^\top \text{ is invertible,} \\ (A^\top A)^{-1} A^\top & \text{if } A^\top A \text{ is invertible,} \\ US^+ V^\top & \text{where } A = VSU^\top \text{ is the SVD of } A. \end{cases} \tag{93}$$

The matrices

$$AA^+, \qquad A^+A, \qquad I - AA^+, \qquad I - A^+A \tag{94}$$

are the orthogonal projectors on the ranges and kernels of A and $A^\top$. You can remember which is which by looking at what space the matrix on the right eats, A input, A^+ output. Under orthogonal transformations, A^+ transforms like $A^\top$. Some useful properties and identities are

$$(A^\top)^+ = (A^+)^\top, \tag{95}$$

$$(AA^\top)^+ = A^{+\top} A^+, \tag{96}$$

$$A^+ = (A^\top A)^+ A^\top = A^\top (AA^\top)^+ = \tag{97}$$

$$= \lim_{\varepsilon \to 0^+} A^\top (AA^\top + \varepsilon I)^{-1} = \lim_{\varepsilon \to 0^+} (A^\top A + \varepsilon I)^{-1} A^\top = \tag{98}$$

$$= \sum_{n=1}^{\infty} A^\top (AA^\top + I)^{-n} = \underbrace{\sum_{n=1}^{\infty} A (A^3 + I)^{-n} A}_{\text{if } A \geq 0}. \tag{99}$$

A matrix A^- which satisfies the first Moore-Penrose condition, i.e., $AA^-A = A$, is called a generalized inverse of A. It is not unique, in particular any two generalized inverses A^- and $\tilde{A}^-$ are connected by the relation

$$\tilde{A}^- = A^- + C - A^- ACAA^- \tag{100}$$

for some C. A matrix A^L which satisfies the first and third conditions, $AA^L A = A$, $(AA^L)^\top = (AA^L)$, is called a least-squares inverse. Compared to A^-, it has the additional properties

$$AA^L = AA^+, \qquad A^L = (A^\top A)^- A^\top. \tag{101}$$

The pseudodeterminant pdet A of a symmetric matrix A is the product of its nonzero eigenvalues. An useful formula is

$$\log \operatorname{pdet} A = \log \det \tilde{A} - \operatorname{tr}(I - AA^+) \log \varepsilon, \qquad \tilde{A} = A + \varepsilon (I - AA^+),$$
$$\tilde{A}^{-1} = A^+ + \frac{I - AA^+}{\varepsilon}. \tag{102}$$

A.2 Woodbury identity

The Woodbury identity is

$$(A + LBR)^{-1} = A^{-1} - A^{-1} L (B^{-1} + RA^{-1} L)^{-1} RA^{-1}, \tag{103}$$

with A, B square invertible and L, R any matrices. If B is not invertible or also rectangular, by making the substitutions $B \mapsto I$ and $R \mapsto BR$ or $L \mapsto LB$, it becomes

$$\begin{aligned}(A+LBR)^{-1} &= A^{-1} - A^{-1}L(I+BRA^{-1}L)^{-1}BRA^{-1} = \\ &= A^{-1} - A^{-1}LB(I+RA^{-1}LB)^{-1}RA^{-1}. \end{aligned} \tag{104}$$

By making other substitutions we obtain the following useful particular cases:

$$(A^{-1}+B^{-1})^{-1} = A - A(A+B)^{-1}A, \tag{105}$$

$$(A^{-1}+B)^{-1} = A - A(I+BA)^{-1}BA = \tag{106}$$

$$= A - AB(I+AB)^{-1}A, \tag{107}$$

$$(A^{-1}+BB^{\top})^{-1} = A - AB(I+B^{\top}AB)^{-1}B^{\top}A. \tag{108}$$

Some related identities are

Pseudoinverse version of the second and third identities?

$$\det(A+LBR) = \det A \det B \det(B^{-1}+RA^{-1}L), \tag{109}$$

$$(B^{-1}+RA^{-1}L)BR = RA^{-1}(A+LBR), \tag{110}$$

$$BR(A+LBR)^{-1} = (B^{-1}+RA^{-1}L)^{-1}RA^{-1}. \tag{111}$$

If neither A nor B nor $A+LBR$ are invertible, there is not a general formula for the pseudoinverse, unless they are nonnegative definite matrices (i.e., covariance matrices), in which case

rewrite as $A + LL^{\top}$

$$\begin{aligned}(AA^{\top}+BB^{\top})^{+} &= (ZZ^{\top})^{+} + (I-BZ^{+})^{\top}A^{+\top}EA^{+}(I-BZ^{+}), \\ Z &= (I-AA^{+})B, \\ E &= I - A^{+}B(I-Z^{+}Z)F^{-1}(A^{+}B)^{\top}, \\ F &= I + (I-Z^{+}Z)B^{\top}(AA^{\top})^{+}B(I-Z^{+}Z). \end{aligned} \tag{112}$$

For the pseudodeterminant, if A is symmetric and $\operatorname{im} L \subseteq \operatorname{im} A$, then

What for arbitrary kernels? Also, I think this holds for nonsymmetric A with a pair of conditions. **Problem:** the formula is wrong, take $A = I_2$, $B = -I_1$, $L = [1, 0]$, then the pdet is 1 but the rhs gives $1 \cdot 1 \cdot (1-1) = 0$. I guess $B \geq 0$ fixes it.

$$\operatorname{pdet}(A+LBL^{\top}) = \operatorname{pdet} A \det B \det(B^{-1}+L^{\top}A^{+}L). \tag{113}$$

A.3 Blockwise operations

Given a matrix divided in blocks as

$$M = \begin{pmatrix} A & L \\ R & B \end{pmatrix},$$

the Schur complement of A in M is

$$M/A = B - RA^{-1}L. \tag{114}$$

Fixing $Q = M/A$, the inverse and determinant of M can be written as

$$\begin{pmatrix} A & L \\ R & B \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1}+A^{-1}LQ^{-1}RA^{-1} & -A^{-1}LQ^{-1} \\ -Q^{-1}RA^{-1} & Q^{-1} \end{pmatrix}, \tag{115}$$

$$\det \begin{pmatrix} A & L \\ R & B \end{pmatrix} = \det A \det Q. \tag{116}$$

If M is nonnegative definite, its block Cholesky decomposition is

$$M = \begin{pmatrix} A & R^\top \\ R & B \end{pmatrix} = c(M)c(M)^\top,$$
$$c(M) = \begin{pmatrix} c(A) & 0 \\ Rc(A)^{-\top} & c(B - RA^{-1}R^\top) \end{pmatrix}, \tag{117}$$

and Equation 115 is also valid replacing inverses with generalized inverses:

$$\begin{pmatrix} A & R^\top \\ R & B \end{pmatrix}^- = \begin{pmatrix} A^- + A^-R^\top Q^- RA^- & -A^-R^\top Q^- \\ -Q^- RA^- & Q^- \end{pmatrix}, \qquad Q = B - RA^-R^\top, \tag{118}$$

where Q is called the generalized Schur complement. The formula for the pseudoinverse is longer, see Schott (2017, th. 7.13, p. 300).

And the pseudodeterminant?

A.4 Derivatives

$$\partial A^{-1} = -A^{-1}\partial A A^{-1}, \tag{119}$$
$$\partial \log\det A = \operatorname{tr}(A^{-1}\partial A), \tag{120}$$
$$\partial A^+ = -A^+\partial A A^+ + A^+A^{+\top}\partial A^\top(I - AA^+) + (I - A^+A)\partial A^\top A^{+\top}A^+, \tag{121}$$
$$\partial(AA^+) = (I - AA^+)\partial A A^L + A^{L\top}\partial A^\top(I - AA^+), \tag{122}$$
$$\partial(A^+A) = A^+\partial A(I - A^+A) + (I - A^+A)\partial A^\top A^{+\top}, \tag{123}$$
$$0 = (I - AA^+)\partial A(I - A^+A), \tag{124}$$
$$\partial \log\operatorname{pdet} A = \operatorname{tr}(A^+\partial A) \qquad (A = A^\top), \tag{125}$$

where the formulas for degenerate A work for points where the rank of A does not change. Reference for the pseudodeterminant: Holbrook (2018, eq. 2.43, p. 302).

Proof of the pseudodeterminant derivative: start from $\partial \det(A + (I - AA^+))$.