The Gaussian perspective on Linear Algebra

computation as data processing

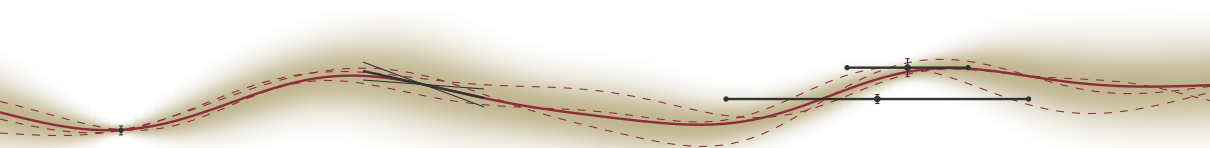
Philipp Hennig

Probnum Spring School 2024, Southampton

8 April 2024

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Tübingen AI Center
tuebingen.ai



Step 1
– Problems –

- It seems like there is a whole zoo of linear algebra problems.
  Can we put them all in one joint description?

Problem Setting

Starting from the basics

Want to find $x \in \mathbb{R}^M$ in

$$A^\intercal x = b,$$

with $A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^N$. For simple exposition, I will assume A has full rank (but can be rectangular).

What your teacher told you:

Underspecified If $M > N$ and $\mathrm{rk}(A) = N$, then there are are many possible solutions x_*. We then *regularize* to find

$$x_* = \arg\min_{x \in \mathbb{R}^M} \|A^\intercal x - b\|^2 + \|x - \mu\|^2_{\Sigma^{-1}} \quad \text{with } \|v\|^2_{\Sigma^{-1}} := v^\intercal \Sigma^{-1} v; \mu \in \mathbb{R}^M, \Sigma \succ 0 \in \mathbb{R}^{M \times M}$$

Nonsingular If $M = N$ and $\mathrm{rk}(A) = N$, then there is a unique solution $x_* = A^{-T} b$

Overspecified If $M < N$ and $\mathrm{rk}(A) = M$, then there are no solutions. We find the *least-squares* solution

$$x_* = \arg\min_{x \in \mathbb{R}^M} \|A^\intercal x - b\|^2$$

What your teacher didn't tell you

regularized least-squares unifies linear problems

Under- over- and well-specified problems can all be subsumed as special cases of the **regularized least-squares** problem

$$x_* = \arg\min_{x \in \mathbb{R}^M} \|A^\intercal x - b\|^2_{\Lambda^{-1}} + \|x - \mu\|^2_{\Sigma^{-1}} \qquad \text{with } \Sigma^{-1}, \Lambda^{-1} \succ 0$$

solved by

$$x_* = (A\Lambda^{-1}A^\intercal + \Sigma^{-1})^{-1}(A\Lambda^{-1}b + \Sigma^{-1}\mu) = \mu + \Sigma A(A^\intercal \Sigma A + \Lambda)^{-1}(b - A^\intercal \mu)$$

Underspecified: special case of $\Lambda^{-1} = I$.

Nonsingular: special case of $\Sigma^{-1} = 0$.

Over-specified: special case of $\Sigma^{-1} = 0, \Lambda^{-1} = I$.

Why this matters

beyond raw linear algebra expressions, the equations need interpretation

- $A^\intercal x = b$, whether under-, over-, or well-posed, can be solved as a regularized problem

$$x_* = \arg\min_{x \in \mathbb{R}^M} \|A^\intercal x - b\|^2_{\Lambda^{-1}} + \|x - \mu\|^2_{\Sigma^{-1}} \quad = (A\Lambda^{-1}A^\intercal + \Sigma^{-1})^{-1}(A\Lambda^{-1}b + \Sigma^{-1}\mu)$$

- Regularization and Pseudoinverses provide canonical solutions to the over- and underdetermined cases, respectively. But they leave questions on the table:
  - What is the *right* choice of μ, Σ? What is the interpretation of the regularized estimate?
  - What about the residuals $b - A^\intercal x_*$ of the least-squares solution? Shouldn't they, and thus Λ be part of the interpretation of the solution?
- In both cases, **uncertainty** provides an answer:
  - We need to regularize because the b does not provide enough information to pin down a unique x.
    We are **uncertain** about the "true" x.
  - If we actually believe a true x exists, then $A^\intercal x = b$ can not be perfectly true.
    We are **uncertain** about the "true" b, or the validity of the equation $A^\intercal x = b$.
- There are *many* possible ways to assign semantic meaning ("causes") to this uncertainty. But one universal way to *formalize* it…

The Probabilistic Interpretation

prior and likelihood lead to a universal solution

- The **regularizer** becomes a **prior** encoding *explicit uncertainty* about the true x:

$$\begin{aligned} p(x \mid \mu, \Sigma) &= \frac{1}{(2\pi)^{m/2}|\Sigma|} \exp(-\frac{1}{2}(x-\mu)^\intercal \Sigma^{-1}(x-\mu)) \\ &= \mathcal{N}(x; \mu, \Sigma) \end{aligned}$$

- The **least-squares problem** becomes a **likelihood** encoding that we are *uncertain* about its validity:

$$\begin{aligned} p(b \mid x, A, \Lambda) &= \frac{1}{(2\pi)^{n/2}|\Lambda|} \exp(-\frac{1}{2}(A^\intercal x - b)^\intercal \Lambda^{-1}(A^\intercal x - b)) \\ &= \mathcal{N}(b; A^\intercal x, \Lambda) \end{aligned}$$

- The regularized least-squares **solution** becomes the mode / mean of the **posterior**. Model mismatch is captured in the **evidence**:

$$\begin{aligned} p(x \mid b, A, \Lambda, \mu, \Sigma) &= \frac{p(b \mid x, A, \Lambda) p(x \mid \mu, \Sigma)}{p(b \mid A, \Lambda, \mu, \Sigma)} = \frac{\mathcal{N}(b; A^\intercal x, \Lambda)\mathcal{N}(x; \mu, \Sigma)}{\mathcal{N}(b; A^\intercal \mu, \Lambda + A^\intercal \Sigma A)} \\ &= \mathcal{N}(x; x_*, (A\Lambda^{-1}A^\intercal + \Sigma^{-1})^{-1}) \end{aligned}$$

Gaussian Inference

The basic mechanism used across not just this talk

Theorem: Let $f \sim \mathcal{N}(m, k)$ be a Gaussian random variable and A a bounded linear operator. Then

$$A^\intercal f \sim \mathcal{N}(A^\intercal m, A^\intercal k A)$$

Let $\varepsilon \in \mathcal{N}(\nu, \Lambda)$ be a $\mathbb{R}^N$ valued Gaussian random vector independent of f. Then, for any $b \in \mathbb{R}^N$,

$$f \mid (A^\intercal f + \varepsilon = b) \sim \mathcal{N}(m^{f|b}, k^{f|b}) \qquad \text{with conditional moments}$$
$$m^{f|b} = m + (A^\intercal k)^\intercal (A^\intercal k A + \Lambda)^\dagger (b - (A^\intercal m + \nu)) \qquad \text{and}$$
$$k^{f|b} = k - (A^\intercal k)^\intercal (A^\intercal k A + \Lambda)^\dagger A^\intercal k$$

Step 2
– Algorithms –

So far:

- All linear problems of the form $A^\intercal x = b$ can be phrased as Gaussian inference problems.
- All assumptions are spelled out in the generative model (prior and likelihood).
- Uncertainty from under-specification can be captured in the posterior.
- The errors necessary to explain residuals (especially in over-specified problems) can be captured in the likelihood.

However, to solve Gaussian inference problems computationally (and thus all linear problems!), we *still need to solve (albeit well-posed, symmetric positive definite) linear systems*. Does the probabilistic formulation also tell us how to do these computations?

The classic numerical perspective on $A^\intercal x = b$

- To solve a single problem once, we might call (for nonsingular and rectangular A, respectively)

  `x = np.linalg.solve(A.T,b)` or `x = np.linalg.lstsq(A.T,b)`

- To solve many problems with the same A but different b, we can save time by first computing a matrix decomposition of $A^\intercal$, e.g.

```python
import numpy as np

def solver(A): # for A in M x N
    Q,D,Ut = np.linalg.svd(A.T, full_matrices=False)
    return lambda b: Ut.T @ (Q.T @ b / D)

slv = solver(A) # pre-compute at O(MN^2)
x1 = slv(b1); x2 = slv(b2); x3 = slv(b3) # this is now cheap
```

- What is the relationship of these methods to probabilistic inference? There are many different decompositions (LU, Cholesky, QR, SVD, etc.) applying to special cases (square A, symmetric positive definite, etc.).

Setup

to simplify exposition

$$A^\intercal x = b, \qquad A \in \mathbb{R}^{M \times N}, x \in \mathbb{R}^M, b \in \mathbb{R}^N$$

In the following, I will assume

- full-rank A (i.e. the N columns of A are linearly independent). This is just for convenience. Redundant columns in A can be detected by solvers at runtime, but this makes the code messier.
- $N \leq M$ – this underspecified case has close connections to numerical methods. If $N > M$, the inference aspect is more obvious, and covered above. The standard least-squares solution ($\Lambda \to 0$) can be recovered (assuming full-rank A) by applying the following results to

$$(AA^\intercal)x = Ab$$

Sequential updates

a close look at Gaussian conditioning, cleaning up notation

Consider **a sequence of trustworthy observations** $a_i^\intercal x = b_i \in \mathbb{R}$, from prior $x \sim \mathcal{N}(\mu_0, \Sigma_0)$.

- After the first observation $a_1^\intercal x = b_1$, posterior mean and covariance are

$$\mu_1 = \mu_0 + \Sigma_0 a_1 (a_1^\intercal \Sigma_0 a_1)^{-1}(b_1 - a_1^\intercal \mu_0) \qquad = \mu_0 + u_1 \frac{b_1 - a_1^\intercal \mu_0}{L_{11}}$$

$$\Sigma_1 = \Sigma_0 - \Sigma_0 a_1 (a_1^\intercal \Sigma_0 a_1)^{-1} a_1^\intercal \Sigma_0 \qquad = \Sigma_0 - u_1 \frac{1}{L_{11}} u_1^\intercal$$

  using $u_1 := \Sigma_0 a_1$, $L_{11} := a_1^\intercal \Sigma_0 a_1 = a_1^\intercal u_1$.

- After the second observation $a_2^\intercal x = b_2$, we have (using $u_k := \Sigma_{k-1} a_k$, $L_{kk} := a_k^\intercal \Sigma_{k-1} a_k = a_k^\intercal u_k$.)

$$\mu_2 = \mu_1 + \Sigma_1 a_2 (a_2^\intercal \Sigma_1 a_2)^{-1}(b_2 - a_2^\intercal \mu_1) \qquad = \mu_1 + u_2 \frac{b_2 - a_2^\intercal \mu_1}{L_{22}}$$

$$\Sigma_2 = \Sigma_1 - \Sigma_1 a_2 (a_2^\intercal \Sigma_1 a_2)^{-1} a_2^\intercal \Sigma_1 \qquad = \Sigma_1 - u_2 \frac{1}{L_{22}} u_2^\intercal$$

Sequential updates

a close look at Gaussian conditioning, cleaning up notation

Consider **a sequence of trustworthy observations** $a_i^\intercal x = b_i \in \mathbb{R}$, from prior $x \sim \mathcal{N}(\mu_0, \Sigma_0)$.

- After the first observation $a_1^\intercal x = b_1$, posterior mean and covariance are

$$\mu_1 = \mu_0 + \Sigma_0 a_1 (a_1^\intercal \Sigma_0 a_1)^{-1}(b_1 - a_1^\intercal \mu_0) \qquad = \mu_0 + u_1 \frac{b_1 - a_1^\intercal \mu_0}{L_{11}}$$

$$\Sigma_1 = \Sigma_0 - \Sigma_0 a_1 (a_1^\intercal \Sigma_0 a_1)^{-1} a_1^\intercal \Sigma_0 \qquad = \Sigma_0 - u_1 \frac{1}{L_{11}} u_1^\intercal$$

  using $u_1 := \Sigma_0 a_1$, $L_{11} := a_1^\intercal \Sigma_0 a_1 = a_1^\intercal u_1$.
- After the i-th observation $a_i^\intercal x = b_i$, we have (using $u_i := \Sigma_{i-1} a_i$, $L_{ii} := a_i^\intercal \Sigma_{i-1} a_i = a_k^\intercal u_k$.)

$$\mu_i = \mu_{i-1} + \Sigma_{i-1} a_i (a_i^\intercal \Sigma_{i-1} a_i)^{-1}(b_i - a_i^\intercal \mu_{i-1}) \qquad = \mu_{i-1} + u_i \frac{b_i - a_i^\intercal \mu_{i-1}}{L_{ii}}$$

$$\Sigma_i = \Sigma_{i-1} - \Sigma_{i-1} a_i (a_i^\intercal \Sigma_{i-1} a_i)^{-1} a_i^\intercal \Sigma_{i-1} \qquad = \Sigma_{i-1} - u_i \frac{1}{L_{ii}} u_k^\intercal$$

It looks like we may get away without calling a linear solver. There must be a catch. What are the objects we need to store, and how expensive are the computations?

Solving one system, or all of them?

Precomputing solver structures

$$L_{ij} = a_i^\intercal \Sigma_{j-1} a_j = a_i^\intercal u_j \text{ for } j \leq i \qquad u_i = \frac{1}{\sqrt{a_i^\intercal \Sigma_{i-1} a_i}} \Sigma_{i-1} a_i = \frac{1}{\sqrt{a_i^\intercal \Sigma_{i-1} a_i}} \left(\Sigma_0 a_i - \sum_{j<i} u_j L_{jj} \right)$$

$$\Sigma_i = \Sigma_{i-1} - u_i u_i^\intercal \qquad = \Sigma_0 - \sum_{j \leq i} u_j u_j^\intercal$$

$$\mu_i = \mu_{i-1} + u_i \underbrace{\frac{b_i - a_i^\intercal \mu_{i-1}}{L_{ii}}}_{=:\delta_i} \qquad = \mu_0 + \sum_{j \leq i} u_j \delta_j$$

$$\text{with } \delta_i := \frac{b_i - a_i^\intercal \mu_{i-1}}{L_{ii}} \qquad = b_i - a_i^\intercal \mu_0 - \sum_{j<i} L_{ij} \delta_j$$

Observations:

- The sequence of u_i, L_{ij} has compute cost $\mathcal{O}(M(1/2 i(i-1)) = \mathcal{O}(Mi^2)$ (recall $a_i \in \mathbb{R}^M$).
- We can actually construct u_i and L_{ij} (thus, Σ_i) without touching b. Once the sequence has run, we can compute the posterior mean for *any* b in $\mathcal{O}(Mi)$.

Building a solver

Note: naïve implementation. In practice, we don't have to store Σ and μ explicitly; (U, L, δ) is enough.

```
Σ_0   = diagm(0 => ones(M)) # prior covariance

## construct solver structure in O(M*N^2)
L = LowerTriangular(zeros(N, N))    # storage
U = zeros(M, N)                     # storage
Σ = Σ_0
for i in 1:N
    y          = A[:,i]             # O(N)
    u          = Σ * y              # O(M^2) <-- !
    u         /= √(u' * y)          # O(M)
    U[:, i]    = u                  # O(M)
    L[i, 1:i] = y' * U[:, 1:i]      # O(Mi)
    Σ         -= (u * u')           # O(M^2) <-- !
end # whole loop is O(M^2N)

## solve A'x = b in O(M*N)
μ_0   = zeros(M)            # prior mean
δ     = zeros(N)   # storage for updates
μ     = μ_0
for i in 1:N
    δ[i] = (S[:, i]' * b - A[:, i]' * μ_0 - L[i, 1:i-1]' * δ[1:i-1]) / L[i,i] # O(M)
    μ   += U[:, i] * δ[i]           # O(M)
end # whole loop is O(MN)
```

Inference is any-time, if you track uncertainty

The value of the posterior

- Given $A \in \mathbb{R}^{M \times N}$, we can run one pre-processing step in $O(M^2N)$ to compute $U \in \mathbb{R}^{M \times M}$ and the lower triangular $L \in \mathbb{R}^{N \times N}$. Afterward, we can perform inference on any $x \in \mathbb{R}^M$ from $A^\intercal x = b \in \mathbb{R}^N$ in $O(MN)$ by building $\delta \in \mathbb{R}^N$.
- This is analogous to the notion of a matrix decomposition. (In fact, more below …)
- There is no problem with stopping the process at any $i < N$
- The posterior captures the resulting "uncertainty" about x. (We have ignored how to choose μ_0, Σ_0 so far, let's assume they are "reasonable")

– Demo –

Some special choices

Stability and efficiency

$$L_{ij} = a_i^\intercal \Sigma_{j-1} a_j = a_i^\intercal u_j \text{ for } j \leq i \qquad u_i = \frac{1}{\sqrt{a_i^\intercal \Sigma_{i-1} a_i}} \Sigma_{i-1} a_i = \frac{1}{\sqrt{a_i^\intercal \Sigma_{i-1} a_i}} \left(\Sigma_0 a_i - \sum_{j<i} u_j L_{ij} \right)$$

So far, we have assumed a general prior $\mathcal{N}(x; \mu_0, \Sigma_0)$. Let's consider the special choice $\Sigma_0 = I_M$.

- $u_1 = \Sigma_0 a_1 / \sqrt{a_1^\intercal \Sigma_0 a_1} = a_1 / \|a_1\|$, $L_{11} = a_1^\intercal u_1 = \|a_1\|$. Note that $\|u_1\| = 1$.
- $u_2 = \frac{1}{\sqrt{a_2^\intercal \Sigma_1 a_2}} (a_2 - u_1 (u_1^\intercal a_2))$, thus we find $u_1^\intercal u_2 = (u_1^\intercal a_2 - 1 \cdot u_1^\intercal u_2) / \sqrt{u_2^\intercal a_1} = 0$. Now, $L_{22} = a_2^\intercal u_2 \neq \|a_2\|$, but $\|u_2\| = 1$
- $u_i \propto a_i - \sum_{j<i} u_j L_{ij}$. By induction: $u_k^\intercal u_i = u_k^\intercal a_i - 1 \cdot u_k^\intercal a_i = 0$ and $\|u_k\| = 1$.

This is the **Gram-Schmidt** process. If simply rename $Q = U$ and $R = L^T$, then Q has orthonormal columns ($Q^\intercal Q = I$), R is upper triangular, and

$$[QR]_{ij} = [UL^\intercal]_{ij} = \sum_k u_{ik} a_j^\intercal u_k = A_{ij}$$

For $\Sigma_0 = I$, our algorithm computes the QR decomposition of A. Reassuring!
(For general spd Σ_0, the columns of Q are Σ_0^{-1} conjugate, rather than orthogonal)

A Generalization

Who says we have to use the columns of A as given?

- On second thought, it seems arbitrary to go through the columns of A in order. Why not choose the projections s_i in a way that makes the updates easier?
- More generally, we can *choose* to "observe" the sequence $s_i^\intercal A^\intercal x = s_i^\intercal b$ for any $s_i \in \mathbb{R}^N$. In the code, we replace $a_i \rightarrowtail As_i =: y_i$, and $b_i \rightarrowtail s_i^\intercal b = \beta_i$.

$$L_{ij} = y_i^\intercal \Sigma_{j-1} y_j = y_i^\intercal u_j \text{ for } j \leq i \qquad u_i = \frac{1}{\sqrt{y_i \Sigma_{i-1} y_i}} \Sigma_{i-1} y_i = \frac{1}{\sqrt{y_i \Sigma_{i-1} y_i}} \left(\Sigma_0 y_i - \sum_{j<i} u_j L_{ij} \right)$$

- This requires that we store the s_i in a matrix S (to compute the β_i), which seems like it may require extra storage. We also need to *construct* s_i somehow, following a *policy*.
- On the upside, maybe we can choose the s_i such that the process becomes
  - more stable to numerical errors (e.g., maybe we can improve the condition number of L), or easier to store
  - more efficient (e.g., maybe we can make the sequence of posterior means converge towards x faster)

Choice I

picking directions that improve L

Let's start again with $s_1 = e_1$, but then choose the following s_i such that L becomes diagonal:

$$s_1 := e_1, \qquad y_1 = As_1 = a_1 \qquad u_1 = \Sigma_0 y_1 \qquad L_{11} = y_1^\intercal \Sigma_0 y_1 = y_1^\intercal u_1$$

set $R_{12} = a_2^\intercal u_1$ and choose:

$$s_2 := e_2 - R_{11} e_1 \qquad y_2 = As_2 = a_2 - R_{11} y_1 \qquad u_2 = \Sigma_1 y_2 \qquad L_{21} = y_2^\intercal u_1 = a_2^\intercal u_1 - a_2^\intercal u_1 = 0$$

set $R_{ij} = a_j^\intercal u_i$ and choose:

$$s_i := e_i - \sum_{j=1}^{i-1} R_{ij} e_j \qquad y_i = As_i = a_i - \sum_{j=1}^{i-1} R_{ij} y_j \qquad u_i = \Sigma_{i-1} y_i \qquad L_{ii} = y_i^\intercal u_i$$

- Under this choice, L is diagonal. So we do not have to store it explicitly anymore (we can store the diagonal in the diagonal of R). However, now we have to store the upper triangular R.

Choice I

picking directions that improve L

$$s_1 := e_1 \qquad \text{set } R_{ij} = a_j^\mathsf{T} u_i \qquad s_i := e_i - \sum_{j=1}^{i-1} R_{ij} e_j$$

- Under this choice, L is diagonal. So we do not have to store it explicitly anymore (we can store the diagonal in the diagonal of R). However, now we have to store the upper triangular R.
- Note that S is evidently also triangular, with ones on the diagonal.
- Computing the posterior mean now simplifies to

$$\mu_i = \mu_0 + \sum_{j \leq i} u_j \delta_j \qquad \delta_i = \left(s_i^\mathsf{T} b - y_i^\mathsf{T} \mu_0 - \sum_{j<i} L_{ij} \delta_j \right) / R_{i,i} = s_i^\mathsf{T} b - y_i^\mathsf{T} \mu_0$$

  For $\mu_0 = 0$, it is now a two-part process, to compute $\beta_i = s_i^\mathsf{T} b$, and then $\mu_i = \sum_{j \leq i} u_j \beta_j$.
- We can also save a bit of space in U and store it as a lower triangular matrix (left as an exercise).
- For $\Sigma_0 = I$, this yields the **LU decomposition** of A. If A is symmetric positive definite, we can get rid of half the memory / compute, and get the **Cholesky** decomposition.

Choice II

picking directions that speed up convergence

- Given a particular b, our prior mean μ_0 yields an initial *residual*

$$r_0 = b - A^\intercal \mu_0$$

  which seems like an informative choice for the first direction $s_1 = r_0$.
- After the first iteration ($y_1 = As_1 = Ar_0, u_1 = \Sigma_0 y_1$), we have posterior mean

$$\mu_1 = \mu_0 + u_1 \frac{s_1^\intercal b - y_1^\intercal \mu_0}{y_1^\intercal u_1}$$

  and the new residual is orthogonal to the old one: $r_1 = b - A^\intercal \mu_1 = r_0 - s_1 s_1^\intercal r_0$. We can thus choose $s_2 = r_1$ (or $s_2 = r_1 - s_1 \frac{r_1^\intercal s_1}{s_1^\intercal s_1}$).
- For symmetric semidefinite A, $\Sigma_0 = I$, this is the **Conjugate Gradient** method (up to implementation details). In general, it is a **Krylov-Subspace** method.

Preconditioning

Prior knowledge helps stability and convergence

How should we choose Σ_0?

- We saw that $\Sigma_0 = I$ makes close connections to classic decompositions. More generally:
- The basic algorithm ($s_i = e_i$) computes a QR decomposition of A for $\Sigma_0 = I$.
- In conjugate gradients, the first update is

$$\mu_1 = \mu_0 + u_1 \cdot \frac{s_1^\intercal b - y_1^\intercal \mu_0}{y_1^\intercal u_1} = \mu_0 + \Sigma_0 A r_0 \cdot \frac{r_0^\intercal b - r_0^\intercal A \mu_0}{r_0^\intercal A \Sigma_0 A^\intercal r_0}$$

  If we manage to pick $\Sigma_0 = (AA^\intercal)^{-1}$, we get $\mu_1 = (AA^\intercal)^{-1}Ab$, the solution to the normal equations.
- Under the same choice, the prior uncertainty becomes scalable. Say A is spd. Then:

$$x^\intercal (AA^\intercal)^{-1} x = \|b\|^2$$

  So given a b, we can pre-scale it to have unit norm, and get meaningful uncertainty.

$\Sigma^{-1/2}$ is associated with the *pre-conditioner* in classic methods. We should aim to pick $\Sigma \approx (AA^\intercal)^{-1}$ for good uncertainty calibration *and* fast convergence of iterative solvers.

What I haven't told you

Calibrating uncertainty

- So far, nothing guarantees us that the posterior $p(x \mid A, b) = \mathcal{N}(\mu, \Sigma)$ is *calibrated*, i.e. that

$$(x_* - \mu)^\intercal \Sigma^\dagger (x_* - \mu) = \mathrm{tr}(\Sigma^\dagger (x_* - \mu)(x_* - \mu)^\intercal) \sim M$$

- Calibration – picking μ_0, Σ_0 – is a new, separate task for probabilistic solvers.
- Some pointers in papers listed at the end.

Step 3
– From vectors $[x_i]$ to functions $f(x)$ –

- Common matrix decompositions amount to specific **policies** for the collection of observations. In this sense, **matrix decompositions are data loaders**.
- Implemented as Gaussian inference, these algorithms become **anytime**, and quantify uncertainty.
- Projections s_i can be *actively* chosen from a policy to improve **stability** and **convergence**.
- The prior covariance Σ_0 amounts simultaneously to **pre-conditioning** and **uncertainty calibration**.

The final piece: all these methods can be implemented **lazily**, and thus also work on **functions**.

Gaussian Inference – again

general form

proof in Pförtner et al., 2022

Theorem: Let $f \sim \mathcal{N}(m, k)$ be a Gaussian random variable and A a bounded linear operator. Then

$$A^\intercal f \sim \mathcal{N}(A^\intercal m, A^\intercal k A)$$

Let $\varepsilon \in \mathcal{N}(\nu, \Lambda)$ be a $\mathbb{R}^N$ valued Gaussian random vector independent of f. Then, for any $b \in \mathbb{R}^N$,

$$f \mid (A^\intercal f + \varepsilon = b) \sim \mathcal{N}(m^{f|b}, k^{f|b}) \qquad \text{with conditional moments}$$

$$m^{f|b} = m + (A^\intercal k)^\intercal (A^\intercal k A + \Lambda)^\dagger (b - (A^\intercal m + \nu)) \qquad \text{and}$$

$$k^{f|b} = k - (A^\intercal k)^\intercal (A^\intercal k A + \Lambda)^\dagger A^\intercal k$$

Gaussian Inference – again

general form

proof in Pförtner et al., 2022

Theorem: Let $f \sim \mathcal{GP}(m, k)$ be a Gaussian process with index set $\mathbb{X}$ on the probability space $(\Omega, \mathcal{F}, \mathrm{P})$, whose paths lie in a real separable reproducing kernel Banach space $\mathbb{B} \subset \mathbb{R}^{\mathbb{X}}$, such that $\omega \mapsto f(\cdot, \omega)$ is a $\mathbb{B}$-valued Gaussian random variable. And let A be a bounded linear operator. Then

$$A^\intercal f \sim \mathcal{N}(A^\intercal m, A^\intercal k A)$$

Let $\varepsilon \in \mathcal{N}(\nu, \Lambda)$ be a $\mathbb{R}^N$ valued Gaussian random vector independent of f. Then, for any $b \in \mathbb{R}^N$,

$$\begin{aligned} f \mid (A^\intercal f + \varepsilon = b) &\sim \mathcal{N}(m^{f|b}, k^{f|b}) \qquad \text{with conditional moments} \\ m^{f|b} &= m + (A^\intercal k)^\intercal (A^\intercal k A + \Lambda)^\dagger (b - (A^\intercal m + \nu)) \qquad \text{and} \\ k^{f|b} &= k - (A^\intercal k)^\intercal (A^\intercal k A + \Lambda)^\dagger A^\intercal k \end{aligned}$$

where, for two bounded linear operators $A : \mathbb{B} \twoheadrightarrow \mathbb{R}^N, \tilde{A} : \mathbb{B} \twoheadrightarrow \mathbb{R}^{\tilde{N}}$, the $N \times \tilde{N}$ matrix $A^\intercal k \tilde{A}$ has entries

$$[A^\intercal k \tilde{A}]_{ij} = A[x \mapsto \tilde{A}[k(x, \cdot)]_j]_i$$

- So far, we considered a concrete vector $x \in \mathbb{R}^M$, for which we assume the prior $\mathcal{N}(x; \mu_0, \Sigma_0)$.
- Given a finite set of linear projections $A^\intercal x = b$, we pick a sequence of projections $y_i^\intercal x = s_i^\intercal A^\intercal x = s_i^\intercal b$ to iteratively update the posterior mean and covariance:

$$\mu_i = \mu_{i-1} + \Sigma_{i-1} A s_i \frac{1}{s_i^\intercal A^\intercal \Sigma_{i-1} A^\intercal s_i} (s_i^\intercal b - s_i^\intercal A^\intercal \mu_{i-1}) \qquad = \mu_{i-1} + u_i \frac{\delta_i(b)}{L_{ii}} \text{ and}$$

$$\Sigma_i = \Sigma_{i-1} - \Sigma_{i-1} A s_i \frac{1}{s_i^\intercal A^\intercal \Sigma_{i-1} A^\intercal s_i} s_i^\intercal A^\intercal \Sigma_{i-1} \qquad = \Sigma_{i-1} - u_i \frac{1}{L_{ii}} u_i^\intercal$$

- Recall that $[\Sigma_i]_{v,w} = \mathrm{cov}_{|A^\intercal_{:,:i} x = b_{:i}}(x_v, x_w)$. Consider a **covariance function** (aka. positive definite kernel) $k(v, w) = \mathrm{cov}(x_v, x_w)$ used to build Σ_0, and a **mean function** $\mu_0(v)$. Then constructing $\Sigma_0 A s_0$ is a *partial evaluation* (or *currying*) of k. The update is a *closure* of the function k. So long as we can compute this closure, we can construct a *functional* solver that lazily pre-computes the solution operator to **predict arbitrary elements of x, from arbitrary observations β.**
- **In particular, x does not have to be finite, but can be a function itself.**

$$\mu_i(\bullet) = \mu_{i-1}(\bullet) + u(\bullet)\delta_i(b) \text{ and}$$
$$\Sigma_i(\bullet, \circ) = \Sigma_{i-1}(\bullet, \circ) - u_i(\bullet)u_i^\intercal(\circ)$$

- For example, consider $As_i = \delta(z - z_i)$, the point evaluation function at z_i. Then $u_i(\bullet) = k(\bullet, z_i)/\sqrt{L_{ii}}$ with $L_{ii} = k(z_i, z_i) > 0$.

$$\mu_i(\bullet) = \mu_{i-1}(\bullet) + u(\bullet)\delta_i(b) \text{ and}$$
$$\Sigma_i(\bullet, \circ) = \Sigma_{i-1}(\bullet, \circ) - u_i(\bullet)u_i^\intercal(\circ)$$

- For example, consider $As_i = \delta(z - z_i)$, the point evaluation function at z_i.
  Then $u_i(\bullet) = k(\bullet, z_i)/\sqrt{L_{ii}}$ with $L_{ii} = k(z_i, z_i) > 0$.
- Consider a function $f(z)$ and $s_i^\intercal Af = \nabla^2 f(z_i) = \left.\frac{\partial^2 f(z)}{\partial z_1^2}\right|_{z=z_i} + \left.\frac{\partial^2 f(z)}{\partial z_2^2}\right|_{z=z_i} + \left.\frac{\partial^2 f(z)}{\partial z_3^2}\right|_{z=z_i}$. Then

$$u_i(\bullet) = L_{ii}^{-1} \sum_{d=1}^{3} \left.\frac{\partial k(\bullet, z)}{\partial z_d^2}\right|_{z=z_i} : \mathbb{R}^3 \to \mathbb{R} \quad \text{with} \quad L_{ii} = \sum_{c=1}^{3}\sum_{d=1}^{3} \left.\frac{\partial^2 \partial^2 k(z_i, z_i)}{\partial z_c^2 \partial z_d^2}\right|_{z=z_i} \in \mathbb{R}_+$$

  and we can encode that **Poisson's equation** $\nabla^2 f = g$ holds at point z_i, for arbitrary forces $g(z)$, without discretizing the solution f, and predict the solution at any point z.
- More in Pförtner et al. (2022), and the next talks.

Want to know more?

Some reading

- Philipp Hennig
  Probabilistic Interpretation of Linear Solvers
  SIAM JOpt, 25(1), 2015
- Jon Cockayne, Chris J. Oates, Ilse C.F. Ipsen, Mark Girolami
  A Bayesian Conjugate Gradient Method (with discussion)
  Bayesian Analysis, 14(3): 937-1012
- Jonathan Wenger, Philipp Hennig
  Probabilistic Linear Solvers for Machine Learning
  Adv. in NeurIPS 33 (2020)
- Jonathan Wenger, Geoff Pleiss, Marvin Pförtner, Philipp Hennig, John P. Cunningham
  Posterior and Computational Uncertainty in Gaussian Processes
  Adv. in NeurIPS 35 (2022)
- Chapter III in Philipp Hennig, Michael A. Osborne, Hans Kersting
  Probabilistic Numerics – Computation as Machine Learning
  Cambridge University Press, 2022

Summary: Linear Algebra and Gaussian Inference

- All linear problems (over- & under-specified, well-posed) are special cases of Gaussian inference.
- Inference itself realizes the solution operator – common matrix decompositions amount to specific **policies** for the collection of observations. In this sense, **matrix decompositions are data loaders**.
- When implemented as Gaussian inference, these algorithms become **anytime**, and quantify an uncertainty. Calibrating this uncertainty is about picking the right prior. It also speeds up convergence.
- All these methods can be implemented **lazily**, and thus also work on **functions**. This allows building meshless solvers for PDEs, and other numerical functional analysis tasks, with uncertainty quantification.

Download these slides: