Probabilistic Linear Solvers for Machine Learning

Jonathan Wenger, Philipp Hennig
University of Tübingen and Max Planck Institute for Intelligent Systems, Tübingen, Germany

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

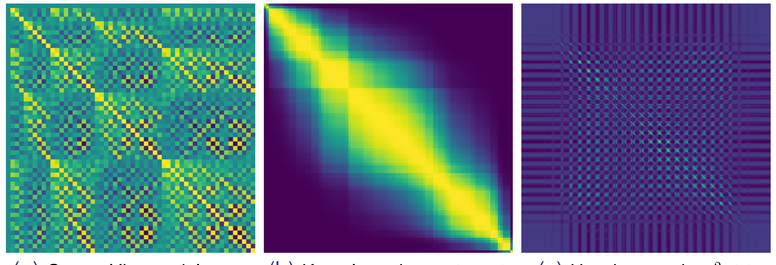
Contribution

- ▷ Unified probabilistic linear solver estimating $\boldsymbol{A}$, $\boldsymbol{A}^{-1}$ and the solution $\boldsymbol{x}_*$
- ▷ Incorporation of prior information on the matrix $\boldsymbol{A}$ or its inverse $\boldsymbol{A}^{-1}$
- ▷ Covariance class which enables uncertainty calibration via prior spectral information and recovers the conjugate gradient method

Linear Systems in Machine Learning

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

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ symmetric positive definite.



(a) General lin. model $\boldsymbol{X}^\intercal\boldsymbol{X}$ (b) Kernel matrix $k_{\mathsf{RBF}}(\boldsymbol{X},\boldsymbol{X})$ (c) Hessian matrix $\nabla^2 f(\boldsymbol{\theta})$

ML-specific Systems

▷ large-scale ▷ charact. structure ▷ generative information ▷ noise

Applications: regression, kernel methods, Kalman filtering, Gaussian processes, spectral graph theory, differential equations, (stochastic) second-order optimization

Probabilistic Linear Solvers

Probabilistic linear solvers (PLS) [1, 2, 3] iteratively build a model for the linear operator $\boldsymbol{A}$, its inverse $\boldsymbol{H} = \boldsymbol{A}^{-1}$ or the solution $\boldsymbol{x}_*$, represented by random variables $\mathbf{A}, \mathbf{H}$ or $\mathbf{x}$. In the framework of *probabilistic numerics* [4, 5] such solvers are Bayesian agents performing *inference* via linear *observations* $\boldsymbol{Y} = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_k] \in \mathbb{R}^{n\times k}$ resulting from *actions* $\boldsymbol{S} = [\boldsymbol{s}_1, \dots, \boldsymbol{s}_k] \in \mathbb{R}^{n\times k}$ given by a *policy* $\pi(\boldsymbol{s} \mid \mathbf{A}, \mathbf{H}, \mathbf{x}, \boldsymbol{A}, \boldsymbol{b})$. For a matrix-variate prior $p(\mathbf{A})$ or $p(\mathbf{H})$ our solver computes posterior beliefs over the matrix, its inverse and the solution.

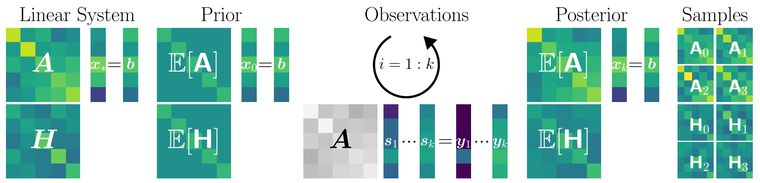


Figure: *Illustration of a probabilistic linear solver.*

Bayesian Inference Framework

$$\begin{aligned}
\text{Prior} \quad & p(\mathbf{A}) = \mathcal{N}(\mathbf{A}; \boldsymbol{A}_0, \boldsymbol{W}_0^{\mathbf{A}} \otimes \boldsymbol{W}_0^{\mathbf{A}}) \\
\text{Likelihood} \quad & p(\boldsymbol{Y} \mid \mathbf{A}, \boldsymbol{S}) = \lim_{\varepsilon \downarrow 0} \mathcal{N}(\boldsymbol{Y}; \boldsymbol{A}\boldsymbol{S}, \varepsilon^2 \boldsymbol{I} \otimes \boldsymbol{I}) = \delta(\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{S}) \\
\text{Posterior} \quad & p(\mathbf{A} \mid \boldsymbol{S}, \boldsymbol{Y}) = \mathcal{N}(\mathbf{A}; \boldsymbol{A}_k, \boldsymbol{\Sigma}_k) \\
& \boldsymbol{A}_k = \boldsymbol{A}_0 + \boldsymbol{\Delta}_0^{\mathbf{A}}\boldsymbol{U}^\intercal + \boldsymbol{U}(\boldsymbol{\Delta}_0^{\mathbf{A}})^\intercal - \boldsymbol{U}\boldsymbol{S}^\intercal\boldsymbol{\Delta}_0^{\mathbf{A}}\boldsymbol{U}^\intercal \\
& \boldsymbol{\Sigma}_k = \boldsymbol{W}_0^{\mathbf{A}}(\boldsymbol{I}_n - \boldsymbol{S}\boldsymbol{U}^\intercal) \otimes \boldsymbol{W}_0^{\mathbf{A}}(\boldsymbol{I}_n - \boldsymbol{S}\boldsymbol{U}^\intercal)
\end{aligned}$$

where $\boldsymbol{\Delta}_0^{\mathbf{A}} = \boldsymbol{Y} - \boldsymbol{A}_0\boldsymbol{S}$ and $\boldsymbol{U} = \boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S}(\boldsymbol{S}^\intercal\boldsymbol{W}_0^{\mathbf{A}}\boldsymbol{S})^{-1}$. Similarly for inverse model $\mathbf{H}$.

Algorithm 1: Probabilistic Linear Solver with Uncertainty Calibration

procedure ProbLinSolve($\boldsymbol{A}(\cdot), \boldsymbol{b}, \mathbf{A}, \mathbf{H}$) # prior for $\mathbf{A}$ or $\mathbf{H}$
  $\boldsymbol{x}_0 \leftarrow \mathbb{E}[\mathbf{H}]\boldsymbol{b}$ # initial guess
  $\boldsymbol{r}_0 \leftarrow \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$
  while $\min(\sqrt{\mathrm{tr}(\mathrm{Cov}[\mathbf{x}])}, \|\boldsymbol{r}_i\|_2) > \max(\delta_{\mathsf{rtol}}\|\boldsymbol{b}\|_2, \delta_{\mathsf{atol}})$ **do**
    $\boldsymbol{s}_i \leftarrow -\mathbb{E}[\mathbf{H}]\boldsymbol{r}_{i-1}$ # compute action via policy
    $\boldsymbol{y}_i \leftarrow \boldsymbol{A}\boldsymbol{s}_i$ # make observation
    $\alpha_i \leftarrow -\boldsymbol{s}_i^\intercal\boldsymbol{r}_{i-1}(\boldsymbol{s}_i^\intercal\boldsymbol{y}_i)^{-1}$ # optimal step size
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_{i-1} + \alpha_i\boldsymbol{s}_i$ # update solution estimate
    $\boldsymbol{r}_i \leftarrow \boldsymbol{r}_{i-1} + \alpha_i\boldsymbol{y}_i$ # update residual
    $\mathbf{A} \leftarrow$ Infer($\mathbf{A}, \boldsymbol{s}_i, \boldsymbol{y}_i$) # infer posterior distributions
    $\mathbf{H} \leftarrow$ Infer($\mathbf{H}, \boldsymbol{s}_i, \boldsymbol{y}_i$)
    $\boldsymbol{\Phi}, \boldsymbol{\Psi} \leftarrow$ Calibrate($\boldsymbol{S}, \boldsymbol{Y}$) # calibrate uncertainty
  $\mathbf{x} \leftarrow \mathcal{N}(\boldsymbol{x}_k, \mathrm{Cov}[\mathbf{H}\boldsymbol{b}])$ # belief over solution
  return ($\mathbf{x}, \mathbf{A}, \mathbf{H}$)

Theoretical Properties

- ▷ Conjugate directions method (convergence in $\leq n$ iterations)
- ▷ Recovers the conjugate gradient method [6] given certain priors.
- ▷ Time complexity $\mathcal{O}(kn^2)$
- ▷ Space complexity $\mathcal{O}(kn)$

Covariance Class and Uncertainty Calibration

Problem: Probabilistic Linear Solvers are typically miscalibrated.
Idea: Covariance class which connects $\mathbf{A}$ and $\mathbf{H}$ and enables calibration.

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

Experiments

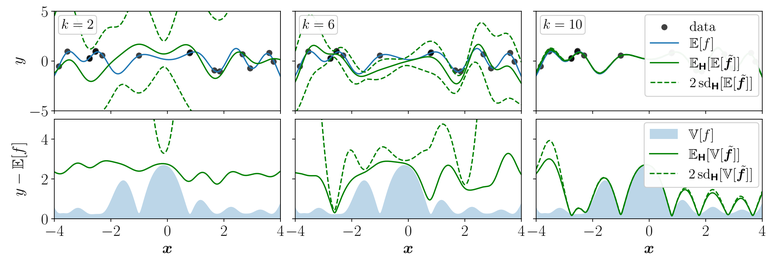


Figure: *Numerical uncertainty in GP inference.*

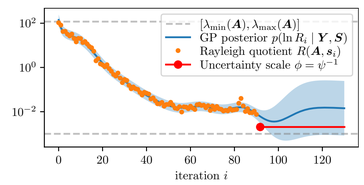


Figure: *Rayleigh regression.* Calibration via one-dim. GP regression on $\{\ln R(\boldsymbol{A}, \boldsymbol{s}_i)\}_{i=1}^k$ for an $n = 1000$ dim. Mátern32 kernel matrix.

Table: *Calibration on kernel matrices.* For $\bar{w} \approx 0$ the solver is well calibrated, for $\bar{w} \gg 0$ under- and for $\bar{w} \ll 0$ overconfident.

Kernel	n	none	Rayleigh	ε^2	$\bar{\lambda}_{k+1:n}$
Matérn32	10^2	−5.99	−0.24	0.32	0.09
Matérn32	10^3	−1.93	7.53	4.26	4.19
Matérn32	10^4	3.87	17.16	8.48	8.47
Matérn52	10^2	−7.84	−1.01	−0.76	−0.80
Matérn52	10^3	−4.63	1.43	−0.80	−0.81
Matérn52	10^4	−4.34	10.81	0.80	0.80
RBF	10^2	−7.53	−0.70	−0.84	−0.87
RBF	10^3	−4.94	6.60	0.77	0.77
RBF	10^4	0.14	21.32	2.92	2.92

Summary

- ▷ Linear systems in ML exhibit characteristic structure.
- ▷ Limited computational resources induce *numerical uncertainty*.
- ▷ Probabilistic linear solvers
  - . . . reinterpret classic linear solvers as Bayesian inference.
  - . . . quantify numerical uncertainty when solving linear systems.
  - . . . make use of prior generative information.
  - . . . allow propagation of information between problems.

ProbNum: Probabilistic numerical methods in Python.
https://github.com/probabilistic-numerics/probnum
or alternatively `pip install probnum`.

References

[1] Philipp Hennig (2015). "Probabilistic Interpretation of Linear Solvers". In: SIAM Journal on Optimization 25(1):pp. 234–260.
[2] Jon Cockayne et al. (2019). "Bayesian probabilistic numerical methods". In: SIAM Review 61(4):pp. 756–789.
[3] Simon Bartels et al. (2019). "Probabilistic linear solvers: A unifying view". In: Statistics and Computing 29(6):pp. 1249–1263.
[4] Philipp Hennig et al. (2015). "Probabilistic numerics and uncertainty in computations". In: Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 471(2179).
[5] Chris Oates et al. (2019). "A modern retrospective on probabilistic numerics". In: Statistics and Computing .
[6] Magnus Rudolph Hestenes et al. (1952). "Methods of conjugate gradients for solving linear systems". In: Journal of Research of the National Bureau of Standards 49.