Approximate Gaussian process inference

Stephen Huan

cgdct.moe

10-624 guest lecture 2025-04-08

About me

Undergraduate at Georgia Tech $\Rightarrow$ CMU PhD @ CSD

Research interests: generative modeling (e.g. diffusion), statistical inference, PDEs, numerical computation

Homepage and contact: `https://cgdct.moe/`

Quick links

https://cgdct.moe/projects/cholesky/

https://theoryclub.github.io/files/gp1.pdf

https://kolesky.cgdct.moe/

https://misc.cgdct.moe/papers/undergrad_thesis.pdf

Mostly covering [Huan et al. 2023]

Overview

Introduction and background

Gaussian process approximation

Sparse Cholesky factorization

Conclusion

The problem

Covariance matrices from pairwise kernel function evaluations

i.e. $\Theta_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for points $\{\boldsymbol{x}_i\}_{i=1}^N$ and kernel function K

The problem

Covariance matrices from pairwise kernel function evaluations

i.e. $\Theta_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for points $\{\boldsymbol{x}_i\}_{i=1}^N$ and kernel function K

Kernel trick in machine learning

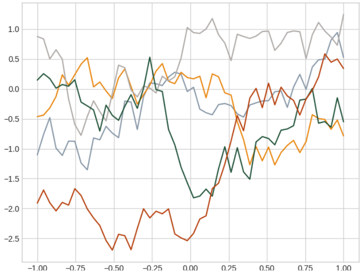
Statistical inference in Gaussian processes on $\boldsymbol{y} \sim \mathcal{N}(\mathbf{0}, \Theta)$

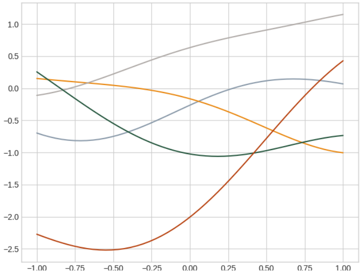
Matérn kernel functions

Matérn family of kernels with smoothness ν and length scale ℓ

$\nu = 1/2$ corresponds to the exponential kernel $\exp(-r/\ell)$

$\nu = \infty$ to the squared exponential kernel $\exp(-r^2/(2\ell^2))$





The problem

Gaussian process (GP) modeling $f \sim \mathcal{GP}(\mu(\cdot), K(\cdot, \cdot))$

The problem

Gaussian process (GP) modeling $f \sim \mathcal{GP}(\mu(\cdot), K(\cdot, \cdot))$

Posterior predictions

$$\begin{aligned}
\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] &= \boldsymbol{\mu}_{\mathsf{Pr}} + \Theta_{\mathsf{Pr},\mathsf{Tr}} \Theta_{\mathsf{Tr},\mathsf{Tr}}^{-1} (\boldsymbol{y}_{\mathsf{Tr}} - \boldsymbol{\mu}_{\mathsf{Tr}}) \\
\mathbb{C}\mathrm{ov}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] &= \Theta_{\mathsf{Pr},\mathsf{Pr}} - \Theta_{\mathsf{Pr},\mathsf{Tr}} \Theta_{\mathsf{Tr},\mathsf{Tr}}^{-1} \Theta_{\mathsf{Tr},\mathsf{Pr}}
\end{aligned}$$

Likelihood $-2 \log \pi(\boldsymbol{x}) = \operatorname{logdet}(\Theta) + \boldsymbol{x}^{\mathsf{T}} \Theta^{-1} \boldsymbol{x} + N \log(2\pi)$

Sampling $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Theta)$

The problem

Gaussian process (GP) modeling $f \sim \mathcal{GP}(\mu(\cdot), K(\cdot, \cdot))$

Posterior predictions

$$\begin{aligned} \mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] &= \boldsymbol{\mu}_{\mathsf{Pr}} + \Theta_{\mathsf{Pr},\mathsf{Tr}} \Theta_{\mathsf{Tr},\mathsf{Tr}}^{-1} (\boldsymbol{y}_{\mathsf{Tr}} - \boldsymbol{\mu}_{\mathsf{Tr}}) \\ \mathbb{C}\mathrm{ov}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] &= \Theta_{\mathsf{Pr},\mathsf{Pr}} - \Theta_{\mathsf{Pr},\mathsf{Tr}} \Theta_{\mathsf{Tr},\mathsf{Tr}}^{-1} \Theta_{\mathsf{Tr},\mathsf{Pr}} \end{aligned}$$

Likelihood $-2 \log \pi(\boldsymbol{x}) = \operatorname{logdet}(\Theta) + \boldsymbol{x}^{\mathsf{T}} \Theta^{-1} \boldsymbol{x} + N \log(2\pi)$

Sampling $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Theta)$

Direct computation scales as $\mathcal{O}(N^3)$, limiting data size (10^4)

Linear algebraic quantities

Quantities of interest

- Matrix-vector product $\Theta \boldsymbol{x}$
- Linear system solve $\Theta^{-1} \boldsymbol{x}$
- Log determinant $\operatorname{logdet}(\Theta)$
- Matrix square root $\Theta = L L^{\mathsf{T}}$

Linear algebraic quantities

Quantities of interest

- Matrix-vector product $\Theta \boldsymbol{x}$
- Linear system solve $\Theta^{-1} \boldsymbol{x}$
- Log determinant $\operatorname{logdet}(\Theta)$
- Matrix square root $\Theta = L L^{\top}$

Fast computation of the above suffices for our entire statistical pipeline (posterior mean, likelihood, gradients, etc.)

Linear algebraic quantities

Quantities of interest

- Matrix-vector product $\Theta \boldsymbol{x}$
- Linear system solve $\Theta^{-1} \boldsymbol{x}$
- Log determinant $\operatorname{logdet}(\Theta)$
- Matrix square root $\Theta = L L^{\top}$

Fast computation of the above suffices for our entire statistical pipeline (posterior mean, likelihood, gradients, etc.)

Robust (but inefficient) computation by Cholesky factorization

Schur complement

Block $\Theta = \left(\begin{smallmatrix} \Theta_{1,1} & \Theta_{1,2} \\ \Theta_{2,1} & \Theta_{2,2} \end{smallmatrix}\right)$ then perform a step of elimination, Block $\Theta = \left(\begin{smallmatrix} \Theta_{1,1} & \Theta_{1,2} \\ \Theta_{2,1} & \Theta_{2,2} \end{smallmatrix}\right)$ then perform a step of elimination,

$$\begin{pmatrix} \Theta_{1,1} & \Theta_{1,2} \\ \mathbf{0} & \Theta_{2,2} - \Theta_{2,1}\Theta_{1,1}^{-1}\Theta_{1,2} \end{pmatrix}$$

Schur complement

Block $\Theta = \begin{pmatrix} \Theta_{1,1} & \Theta_{1,2} \\ \Theta_{2,1} & \Theta_{2,2} \end{pmatrix}$ then perform a step of elimination,

$$\begin{pmatrix} \Theta_{1,1} & \Theta_{1,2} \\ \mathbf{0} & \Theta_{2,2} - \Theta_{2,1}\Theta_{1,1}^{-1}\Theta_{1,2} \end{pmatrix}$$

Denote the term in blue the *Schur complement* of Θ on $\Theta_{1,1}$,

$$\Theta = \begin{pmatrix} \mathsf{Id} & \mathbf{0} \\ \Theta_{2,1}\Theta_{1,1}^{-1} & \mathsf{Id} \end{pmatrix} \begin{pmatrix} \Theta_{1,1} & \mathbf{0} \\ \mathbf{0} & \Theta_{2,2|1} \end{pmatrix} \begin{pmatrix} \mathsf{Id} & \Theta_{1,1}^{-1}\Theta_{1,2} \\ \mathbf{0} & \mathsf{Id} \end{pmatrix}$$

Recursing finishes the construction

$$\begin{aligned}\operatorname{chol}(\Theta) &= \begin{pmatrix} \mathsf{Id} & \mathbf{0} \\ \Theta_{2,1}\Theta_{1,1}^{-1} & \mathsf{Id} \end{pmatrix} \begin{pmatrix} \operatorname{chol}(\Theta_{1,1}) & \mathbf{0} \\ \mathbf{0} & \operatorname{chol}(\Theta_{2,2|1}) \end{pmatrix} \\ &= \begin{pmatrix} \operatorname{chol}(\Theta_{1,1}) & \mathbf{0} \\ \Theta_{2,1} \operatorname{chol}(\Theta_{1,1})^{-\mathsf{T}} & \operatorname{chol}(\Theta_{2,2|1}) \end{pmatrix}\end{aligned}$$

Recursing finishes the construction

$$\operatorname{chol}(\Theta) = \begin{pmatrix} \mathsf{Id} & \mathbf{0} \\ \Theta_{2,1}\Theta_{1,1}^{-1} & \mathsf{Id} \end{pmatrix} \begin{pmatrix} \operatorname{chol}(\Theta_{1,1}) & \mathbf{0} \\ \mathbf{0} & \operatorname{chol}(\Theta_{2,2|1}) \end{pmatrix}$$
$$= \begin{pmatrix} \operatorname{chol}(\Theta_{1,1}) & \mathbf{0} \\ \Theta_{2,1} \operatorname{chol}(\Theta_{1,1})^{-\mathsf{T}} & \operatorname{chol}(\Theta_{2,2|1}) \end{pmatrix}$$

Efficient blocked cache-oblivious numerical algorithm!

Recursing finishes the construction

$$\mathrm{chol}(\Theta) = \begin{pmatrix} \mathsf{Id} & \mathbf{0} \\ \Theta_{2,1}\Theta_{1,1}^{-1} & \mathsf{Id} \end{pmatrix} \begin{pmatrix} \mathrm{chol}(\Theta_{1,1}) & \mathbf{0} \\ \mathbf{0} & \mathrm{chol}(\Theta_{2,2|1}) \end{pmatrix}$$
$$= \begin{pmatrix} \mathrm{chol}(\Theta_{1,1}) & \mathbf{0} \\ \Theta_{2,1}\,\mathrm{chol}(\Theta_{1,1})^{-\mathsf{T}} & \mathrm{chol}(\Theta_{2,2|1}) \end{pmatrix}$$

Efficient blocked cache-oblivious numerical algorithm!

Statistical interpretation of Cholesky factorization

$$L_{i,j} = \frac{\mathbb{C}\mathrm{ov}[y_i, y_j \mid y_{k<j}]}{\sqrt{\mathbb{V}\mathrm{ar}[y_j \mid y_{k<j}]}}$$

Our desired quantities, revisited

Matrix-vector product in $\mathcal{O}(N^2)$

$$\Theta \boldsymbol{x} = L(L^{\mathsf{T}} \boldsymbol{x})$$

Our desired quantities, revisited

Matrix-vector product in $\mathcal{O}(N^2)$

$$\Theta \boldsymbol{x} = L(L^{\mathsf{T}} \boldsymbol{x})$$

Linear system solve in $\mathcal{O}(N^2)$

$$\Theta^{-1} \boldsymbol{x} = L^{-\mathsf{T}} (L^{-1} \boldsymbol{x})$$

Our desired quantities, revisited

Matrix-vector product in $\mathcal{O}(N^2)$

$$\Theta \boldsymbol{x} = L(L^{\mathsf{T}} \boldsymbol{x})$$

Linear system solve in $\mathcal{O}(N^2)$

$$\Theta^{-1} \boldsymbol{x} = L^{-\mathsf{T}} (L^{-1} \boldsymbol{x})$$

Log determinant in $\mathcal{O}(N)$

$$\operatorname{logdet}(\Theta) = 2 \sum_{i=1}^{N} \log(L_{i,i})$$

Easy to sample from $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathsf{Id}_N)$

Easy to sample from $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathsf{Id}_N)$

Generalize to arbitrary Θ by affine ansatz $\boldsymbol{x} = L\boldsymbol{z} + \boldsymbol{\mu}$

$$\begin{aligned}
\mathbb{E}[\boldsymbol{x}] &= \mathbb{E}[L\boldsymbol{z} + \boldsymbol{\mu}] = L\,\mathbb{E}[\boldsymbol{z}] + \boldsymbol{\mu} = \boldsymbol{\mu} \\
\mathbb{C}\mathrm{ov}[\boldsymbol{x}] &= \mathbb{E}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])^\mathsf{T}] = \mathbb{E}[L\boldsymbol{z}(L\boldsymbol{z})^\mathsf{T}] \\
&= \mathbb{E}[L\boldsymbol{z}\boldsymbol{z}^\mathsf{T}L^\mathsf{T}] = L\,\mathbb{E}[\boldsymbol{z}\boldsymbol{z}^\mathsf{T}]L^\mathsf{T} = LL^\mathsf{T}
\end{aligned}$$

Moment matching

Easy to sample from $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathsf{Id}_N)$

Generalize to arbitrary Θ by affine ansatz $\boldsymbol{x} = L\boldsymbol{z} + \boldsymbol{\mu}$

$$\begin{aligned}
\mathbb{E}[\boldsymbol{x}] &= \mathbb{E}[L\boldsymbol{z} + \boldsymbol{\mu}] = L\,\mathbb{E}[\boldsymbol{z}] + \boldsymbol{\mu} = \boldsymbol{\mu} \\
\mathbb{C}\mathrm{ov}[\boldsymbol{x}] &= \mathbb{E}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])^\mathsf{T}] = \mathbb{E}[L\boldsymbol{z}(L\boldsymbol{z})^\mathsf{T}] \\
&= \mathbb{E}[L\boldsymbol{z}\boldsymbol{z}^\mathsf{T} L^\mathsf{T}] = L\,\mathbb{E}[\boldsymbol{z}\boldsymbol{z}^\mathsf{T}]L^\mathsf{T} = LL^\mathsf{T}
\end{aligned}$$

so $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, LL^\mathsf{T})$. We want $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Theta)$, so $\Theta = LL^\mathsf{T}$

Moment matching

Easy to sample from $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathsf{Id}_N)$

Generalize to arbitrary Θ by affine ansatz $\boldsymbol{x} = L\boldsymbol{z} + \boldsymbol{\mu}$

$$\begin{aligned} \mathbb{E}[\boldsymbol{x}] &= \mathbb{E}[L\boldsymbol{z} + \boldsymbol{\mu}] = L\,\mathbb{E}[\boldsymbol{z}] + \boldsymbol{\mu} = \boldsymbol{\mu} \\ \mathbb{C}\mathrm{ov}[\boldsymbol{x}] &= \mathbb{E}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])^{\mathsf{T}}] = \mathbb{E}[L\boldsymbol{z}(L\boldsymbol{z})^{\mathsf{T}}] \\ &= \mathbb{E}[L\boldsymbol{z}\boldsymbol{z}^{\mathsf{T}}L^{\mathsf{T}}] = L\,\mathbb{E}[\boldsymbol{z}\boldsymbol{z}^{\mathsf{T}}]L^{\mathsf{T}} = LL^{\mathsf{T}} \end{aligned}$$

so $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, LL^{\mathsf{T}})$. We want $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Theta)$, so $\Theta = LL^{\mathsf{T}}$

See [J. A. Tropp 2023] for a more rigorous argument

Moment matching

Easy to sample from $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathsf{Id}_N)$

Generalize to arbitrary Θ by affine ansatz $\boldsymbol{x} = L\boldsymbol{z} + \boldsymbol{\mu}$

$$\begin{aligned}
\mathbb{E}[\boldsymbol{x}] &= \mathbb{E}[L\boldsymbol{z} + \boldsymbol{\mu}] = L\,\mathbb{E}[\boldsymbol{z}] + \boldsymbol{\mu} = \boldsymbol{\mu} \\
\mathbb{C}\mathrm{ov}[\boldsymbol{x}] &= \mathbb{E}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])^{\mathsf{T}}] = \mathbb{E}[L\boldsymbol{z}(L\boldsymbol{z})^{\mathsf{T}}] \\
&= \mathbb{E}[L\boldsymbol{z}\boldsymbol{z}^{\mathsf{T}}L^{\mathsf{T}}] = L\,\mathbb{E}[\boldsymbol{z}\boldsymbol{z}^{\mathsf{T}}]L^{\mathsf{T}} = LL^{\mathsf{T}}
\end{aligned}$$

so $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, LL^{\mathsf{T}})$. We want $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Theta)$, so $\Theta = LL^{\mathsf{T}}$

See [J. A. Tropp 2023] for a more rigorous argument

Knothe-Rosenblatt rearrangement generalizes this idea to sample from non-Gaussians [Katzfuss and Schäfer 2022; Marzouk et al. 2016; Spantini, Bigoni, and Marzouk 2018]

Direct vs. iterative methods

Cholesky factorization example of *direct* method

Direct vs. iterative methods

Cholesky factorization example of *direct* method

- Numerically accurate and stable (up to floating point error)

Direct vs. iterative methods

Cholesky factorization example of *direct* method

- Numerically accurate and stable (up to floating point error)
- Requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory

Cholesky factorization example of *direct* method

- Numerically accurate and stable (up to floating point error)
- Requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory

Conjugate gradient: prototypical *iterative* method for $\Theta^{-1}\boldsymbol{x}$

Direct vs. iterative methods

Cholesky factorization example of *direct* method

- Numerically accurate and stable (up to floating point error)
- Requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory

Conjugate gradient: prototypical *iterative* method for $\Theta^{-1}\boldsymbol{x}$

- Accuracy steadily improves; can stop when desired

Direct vs. iterative methods

Cholesky factorization example of *direct* method

- Numerically accurate and stable (up to floating point error)
- Requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory

Conjugate gradient: prototypical *iterative* method for $\Theta^{-1}\boldsymbol{x}$

- Accuracy steadily improves; can stop when desired
- Only requires matrix-vector products ("matvecs") $\boldsymbol{x} \mapsto \Theta\boldsymbol{x}$

Direct vs. iterative methods

Cholesky factorization example of *direct* method

- Numerically accurate and stable (up to floating point error)
- Requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory

Conjugate gradient: prototypical *iterative* method for $\Theta^{-1}\boldsymbol{x}$

- Accuracy steadily improves; can stop when desired
- Only requires matrix-vector products ("matvecs") $\boldsymbol{x} \mapsto \Theta\boldsymbol{x}$
- Can use approximations like fast multipole method [Fong and Darve 2009; Wang et al. 2021] and $\mathcal{H}$-matrices [Geoga, Anitescu, and Michael L Stein 2020; Litvinenko 2019]

Direct vs. iterative methods

Cholesky factorization example of *direct* method

- Numerically accurate and stable (up to floating point error)
- Requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory

Conjugate gradient: prototypical *iterative* method for $\Theta^{-1}\boldsymbol{x}$

- Accuracy steadily improves; can stop when desired
- Only requires matrix-vector products ("matvecs") $\boldsymbol{x} \mapsto \Theta\boldsymbol{x}$
- Can use approximations like fast multipole method [Fong and Darve 2009; Wang et al. 2021] and $\mathcal{H}$-matrices [Geoga, Anitescu, and Michael L Stein 2020; Litvinenko 2019]
- Can accelerate with *preconditioning*: good guess for Θ^{-1}

Direct vs. iterative methods

Cholesky factorization example of *direct* method

- Numerically accurate and stable (up to floating point error)
- Requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory

Conjugate gradient: prototypical *iterative* method for $\Theta^{-1}\boldsymbol{x}$

- Accuracy steadily improves; can stop when desired
- Only requires matrix-vector products ("matvecs") $\boldsymbol{x} \mapsto \Theta\boldsymbol{x}$
- Can use approximations like fast multipole method [Fong and Darve 2009; Wang et al. 2021] and $\mathcal{H}$-matrices [Geoga, Anitescu, and Michael L Stein 2020; Litvinenko 2019]
- Can accelerate with *preconditioning*: good guess for Θ^{-1}

Plenty of great references, see
[Choi 2006; Golub and Van Loan 1996; Saad 2003]

Sampling & determinant

CG computes $\Theta^{-1}\boldsymbol{x}$, what about sampling and determinants?

Sampling & determinant

CG computes $\Theta^{-1}\boldsymbol{x}$, what about sampling and determinants?

Sampling

- Fast Fourier transform [Graham et al. 2018] for structured Θ
- Iterative methods: Adapt CG, Krylov subspace methods [Chow and Saad 2014; Parker and Fox 2012]

Sampling & determinant

CG computes $\Theta^{-1}\boldsymbol{x}$, what about sampling and determinants?

Sampling

- Fast Fourier transform [Graham et al. 2018] for structured Θ
- Iterative methods: Adapt CG, Krylov subspace methods [Chow and Saad 2014; Parker and Fox 2012]

Determinant: exploit

$$\operatorname{logdet}(\Theta) = \operatorname{trace}(\log(\Theta))$$

by forming $\boldsymbol{x} \mapsto \log(\Theta)\boldsymbol{x}$ and estimating trace from matvecs

- Krylov method [T. Chen and Hallman 2022; Higham 2008]
- trace estimator [Epperly, J. A. Tropp, and Webber 2024b; Meyer et al. 2021; Persson, Cortinovis, and Kressner 2022]

Quantities of interest

- Matrix-vector product $\Theta \boldsymbol{x}$
- Linear system solve $\Theta^{-1} \boldsymbol{x}$
- Log determinant $\operatorname{logdet}(\Theta)$
- Matrix square root $\Theta = L L^{\top}$

Compute by direct (Cholesky factor) and iterative methods

Purely linear-algebraic (with statistical interpretations)

No free lunch: cost-accuracy trade-offs abound

Goal: design algorithms at the Pareto frontier

Overview

Approximation strategies

Approximate the full process by *partial* information

Approximation strategies

Approximate the full process by *partial* information

For matrices, two natural ideas: low-rank and sparse

Approximation strategies

Approximate the full process by *partial* information

For matrices, two natural ideas: low-rank and sparse

Recall (Eckart-Young-Mirsky): Singular value decomposition
optimal low-rank approximation

$$A_* := \min_{\widehat{A}} \|A - \widehat{A}\|, \text{ s.t. } \operatorname{rank}(\widehat{A}) \leq r$$

Approximate the full process by *partial* information

For matrices, two natural ideas: low-rank and sparse

Recall (Eckart-Young-Mirsky): Singular value decomposition *optimal* low-rank approximation

$$A_* := \min_{\widehat{A}} \|A - \widehat{A}\|, \text{ s.t. } \operatorname{rank}(\widehat{A}) \le r$$

$$A_* = \sum_{i=1}^{r} \sigma_i u_i v_i^\mathsf{T} \text{ for SVD } A = U\Sigma V^\mathsf{T}$$

in both $\|\cdot\|_2$ and $\|\cdot\|_\mathsf{F}$.

Nyström method

Nyström low-rank approximation

$$A\langle X\rangle := (AX)(X^{\mathsf{T}}AX)^{-1}(AX)^{\mathsf{T}}$$
$$A\backslash X := A - A\langle X\rangle \approx 0$$

Nyström method

Nyström low-rank approximation

$$A\langle X\rangle \coloneqq (AX)(X^{\mathsf{T}}AX)^{-1}(AX)^{\mathsf{T}}$$
$$A\backslash X \coloneqq A - A\langle X\rangle \approx 0$$

$\boldsymbol{y} \sim \mathcal{N}(0, A)$, then $\boldsymbol{y} \mid X^{\mathsf{T}}\boldsymbol{y} \sim \mathcal{N}(0, A\backslash X)$.

Nyström method

Nyström low-rank approximation

$$A\langle X\rangle := (AX)(X^{\mathsf{T}}AX)^{-1}(AX)^{\mathsf{T}}$$
$$A\backslash X := A - A\langle X\rangle \approx 0$$

$\boldsymbol{y} \sim \mathcal{N}(0, A)$, then $\boldsymbol{y} \mid X^{\mathsf{T}}\boldsymbol{y} \sim \mathcal{N}(0, A\backslash X)$.

In practice: take $X = \begin{pmatrix}\mathsf{Id}_m & \mathbf{0}_{m\times(N-m)}\end{pmatrix}^{\mathsf{T}}$
$\Leftrightarrow A\langle X\rangle = A_{:,:m}A^{-1}_{:m,:m}A_{:m,:} = L_{:,:m}L^{\mathsf{T}}_{:,:m}$ for $L = \mathrm{chol}(A)$

Inducing point methods

Predictions (with noise $\sigma^2 \mathsf{Id}$) implied by new low-rank kernel

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \Theta_{\mathsf{Pr},:m}(\Theta_{:m,\mathsf{Tr}}\Theta_{\mathsf{Tr},:m} + \sigma^2\Theta_{:m,:m})^{-1}\Theta_{:m,\mathsf{Tr}}\boldsymbol{y}_{\mathsf{Tr}}$$

with computational complexity $\mathcal{O}(Nm^2)$ (same as Nyström)

Inducing point methods

Predictions (with noise $\sigma^2 \mathsf{Id}$) implied by new low-rank kernel

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \Theta_{\mathsf{Pr},:m}(\Theta_{:m,\mathsf{Tr}}\Theta_{\mathsf{Tr},:m} + \sigma^2\Theta_{:m,:m})^{-1}\Theta_{:m,\mathsf{Tr}}\boldsymbol{y}_{\mathsf{Tr}}$$

with computational complexity $\mathcal{O}(Nm^2)$ (same as Nyström)

Same as Subset of Regressors (SR), Projected Process (PP)

Inducing point methods

Predictions (with noise $\sigma^2 \mathsf{Id}$) implied by new low-rank kernel

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \Theta_{\mathsf{Pr},:m}(\Theta_{:m,\mathsf{Tr}}\Theta_{\mathsf{Tr},:m} + \sigma^2\Theta_{:m,:m})^{-1}\Theta_{:m,\mathsf{Tr}}\boldsymbol{y}_{\mathsf{Tr}}$$

with computational complexity $\mathcal{O}(Nm^2)$ (same as Nyström)

Same as Subset of Regressors (SR), Projected Process (PP)

Subset of Datapoints (SD) simply throws out other datapoints

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \Theta_{\mathsf{Pr},:m}(\Theta_{:m,:m} + \sigma^2\mathsf{Id})^{-1}\boldsymbol{y}_{:m}$$

Inducing point methods

Predictions (with noise $\sigma^2 \mathsf{Id}$) implied by new low-rank kernel

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \Theta_{\mathsf{Pr},:m}(\Theta_{:m,\mathsf{Tr}}\Theta_{\mathsf{Tr},:m} + \sigma^2\Theta_{:m,:m})^{-1}\Theta_{:m,\mathsf{Tr}}\boldsymbol{y}_{\mathsf{Tr}}$$

with computational complexity $\mathcal{O}(Nm^2)$ (same as Nyström)

Same as Subset of Regressors (SR), Projected Process (PP)

Subset of Datapoints (SD) simply throws out other datapoints

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \Theta_{\mathsf{Pr},:m}(\Theta_{:m,:m} + \sigma^2\mathsf{Id})^{-1}\boldsymbol{y}_{:m}$$

See [Krause and Hübotter 2025; Quiñonero-Candela and Rasmussen 2005; Rasmussen and Williams 2006]

How to select the inducing points?

Active set selection often information-theoretic, experimental design [Bartels et al. 2022; Krause, Singh, and Guestrin 2008]

How to select the inducing points?

Active set selection often information-theoretic, experimental design [Bartels et al. 2022; Krause, Singh, and Guestrin 2008]

Can use X from equivalence to the Nyström approximation

How to select the inducing points?

Active set selection often information-theoretic, experimental design [Bartels et al. 2022; Krause, Singh, and Guestrin 2008]

Can use X from equivalence to the Nyström approximation

Modern approach: randomized numerical linear algebra [Y. Chen, Epperly, et al. 2024; Epperly, J. A. Tropp, and Webber 2024a; Frangella, J. A. Tropp, and Udell 2021; Martinsson and J. Tropp 2021]

$\Theta = LL^{\top}$, L has N columns, s non-zero entries per column

Acceleration by sparsity

$\Theta = LL^{\mathsf{T}}$, L has N columns, s non-zero entries per column

$L\boldsymbol{v}$ and $L^{-1}\boldsymbol{v}$ both cost $\mathcal{O}(Ns)$

Acceleration by sparsity

$\Theta = LL^{\mathsf{T}}$, L has N columns, s non-zero entries per column

$L\boldsymbol{v}$ and $L^{-1}\boldsymbol{v}$ both cost $\mathcal{O}(Ns)$

Matrix-vector product: $N^2 \to Ns$

Acceleration by sparsity

$\Theta = LL^{\mathsf{T}}$, L has N columns, s non-zero entries per column

$L\boldsymbol{v}$ and $L^{-1}\boldsymbol{v}$ both cost $\mathcal{O}(Ns)$

Matrix-vector product: $N^2 \to Ns$

Solving linear system: $N^3 \to Ns$

$\Theta = LL^{\mathsf{T}}$, L has N columns, s non-zero entries per column

$L\boldsymbol{v}$ and $L^{-1}\boldsymbol{v}$ both cost $\mathcal{O}(Ns)$

Matrix-vector product: $N^2 \to Ns$

Solving linear system: $N^3 \to Ns$

Log determinant: $N^3 \to N$

Acceleration by sparsity

$\Theta = LL^\top$, L has N columns, s non-zero entries per column

$L\boldsymbol{v}$ and $L^{-1}\boldsymbol{v}$ both cost $\mathcal{O}(Ns)$

Matrix-vector product: $N^2 \to Ns$

Solving linear system: $N^3 \to Ns$

Log determinant: $N^3 \to N$

Sampling: $N^3 \to Ns$

Acceleration by sparsity

$\Theta = LL^{\mathsf{T}}$, L has N columns, s non-zero entries per column

$L\boldsymbol{v}$ and $L^{-1}\boldsymbol{v}$ both cost $\mathcal{O}(Ns)$

Matrix-vector product: $N^2 \to Ns$

Solving linear system: $N^3 \to Ns$

Log determinant: $N^3 \to N$

Sampling: $N^3 \to Ns$

We will take s to be $\mathcal{O}(\log^d(N/\epsilon))$!

Overview

Collaborators

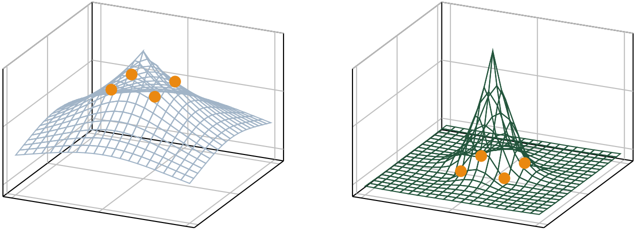
Joe Guinness, Cornell

Matthias Katzfuß, Texas A&M

Houman Owhadi, Caltech

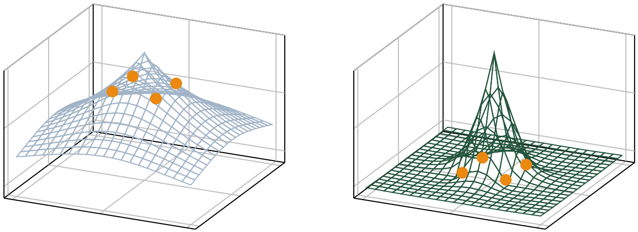
Florian Schäfer, Gatech

Screening effect



Conditional on points near a point of interest, far away points are almost independent [Michael L. Stein 2002]

Screening effect



Conditional on points near a point of interest, far away points are almost independent [Michael L. Stein 2002]

Suggests space-covering ordering and selecting nearby points

Statistical Cholesky factorization

Cholesky factorization $\Leftrightarrow$ iterative conditioning of process

$$L = \operatorname{chol}(\Theta)$$
$$L_{i,j} = \frac{\mathbb{C}\mathrm{ov}[y_i, y_j \mid y_{k<j}]}{\sqrt{\mathbb{V}\mathrm{ar}[y_j \mid y_{k<j}]}}$$

Statistical Cholesky factorization

Cholesky factorization $\Leftrightarrow$ iterative conditioning of process

$$\begin{aligned} L &= \operatorname{chol}(\Theta) \\ L_{i,j} &= \frac{\mathbb{C}\mathrm{ov}[y_i, y_j \mid y_{k<j}]}{\sqrt{\mathbb{V}\mathrm{ar}[y_j \mid y_{k<j}]}} \end{aligned}$$

Conditional (near)-independence $\Leftrightarrow$ (approximate) sparsity

Cholesky factorization recipe

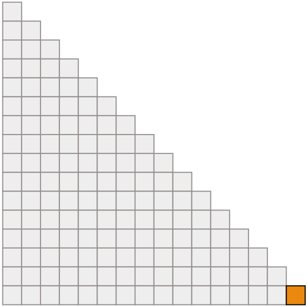
Implied procedure for computing $LL^{\mathsf{T}} \approx \Theta^{-1}$

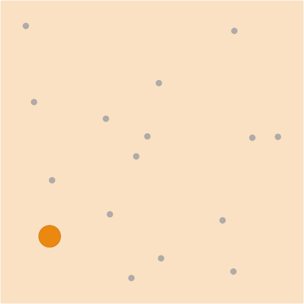
1. Pick an ordering on the rows/columns of Θ
2. Select a sparsity pattern lower triangular w.r.t. ordering
3. Compute entries by minimizing objective over all factors

Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho \ell_i$ from $\boldsymbol{x}_i$

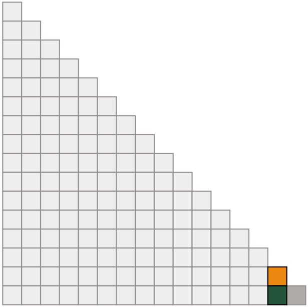


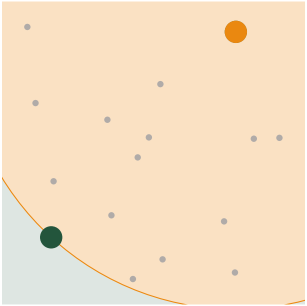


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

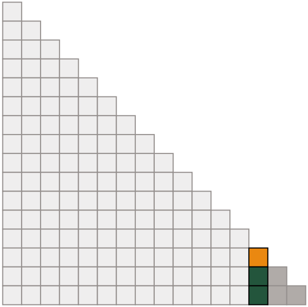


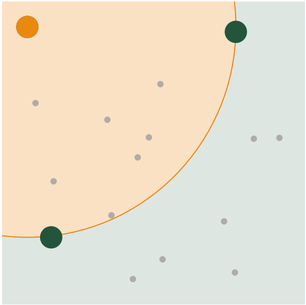


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

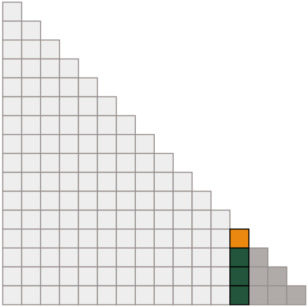


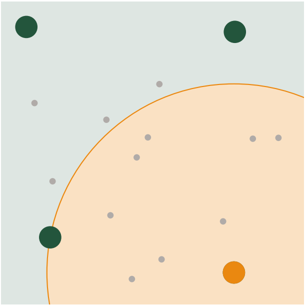


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

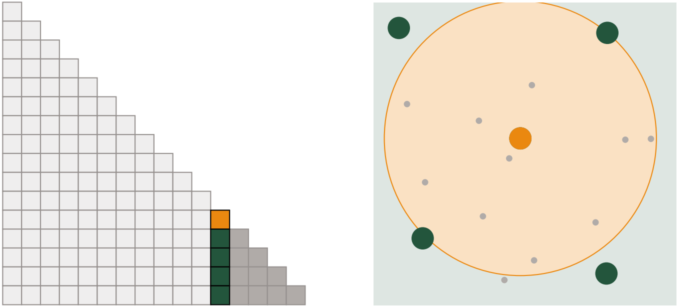




Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point x_i with largest distance ℓ_i to points selected before

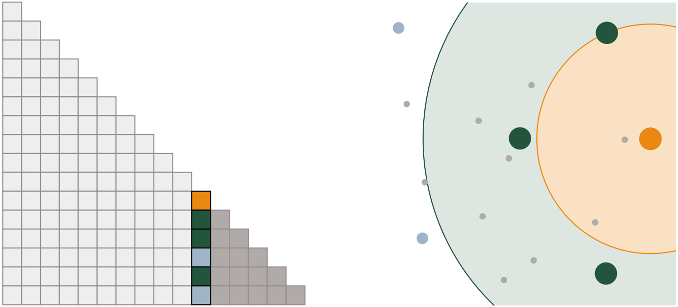
The i-th column selects points within a radius of $\rho\ell_i$ from x_i



Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

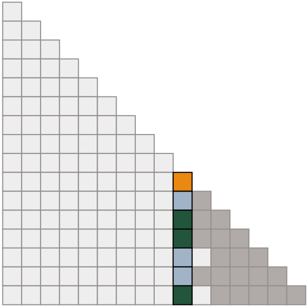
The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

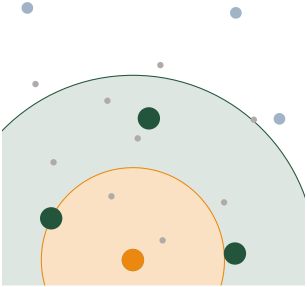


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

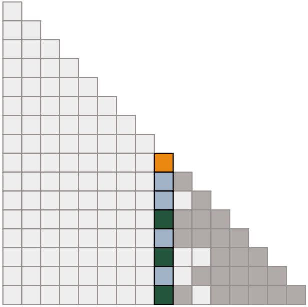


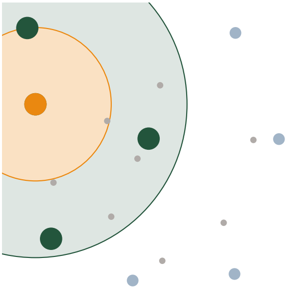


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

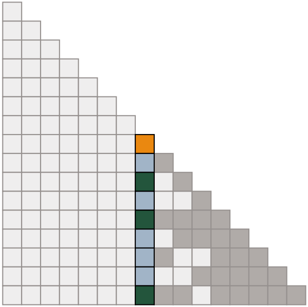


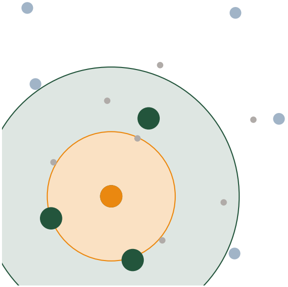


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

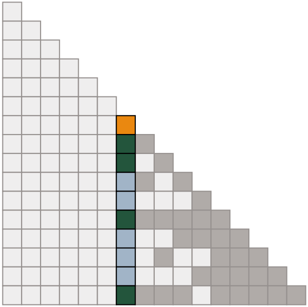


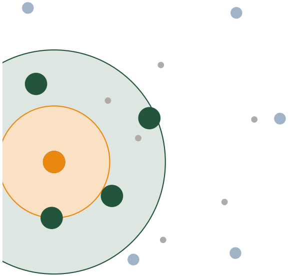


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

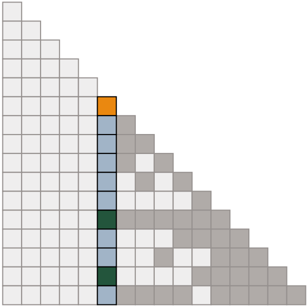


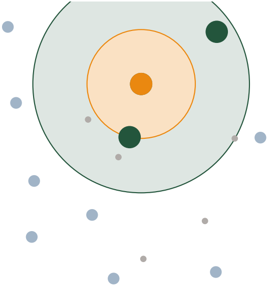


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

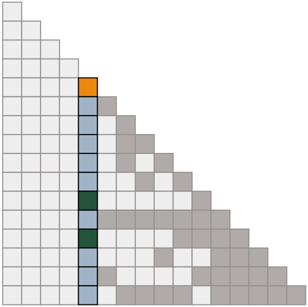


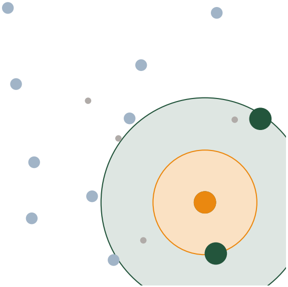


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

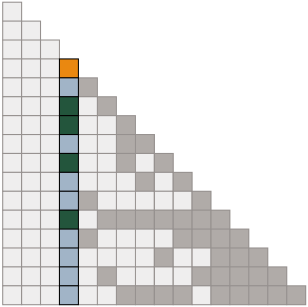


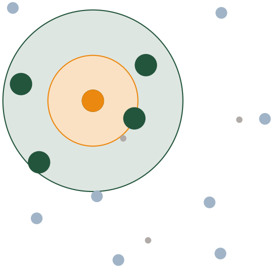


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

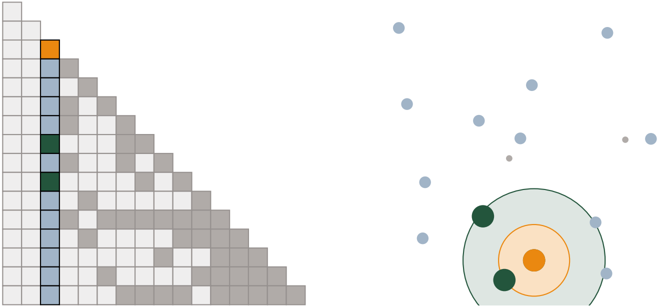
The i-th column selects points within a radius of $\rho \ell_i$ from $\boldsymbol{x}_i$





(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

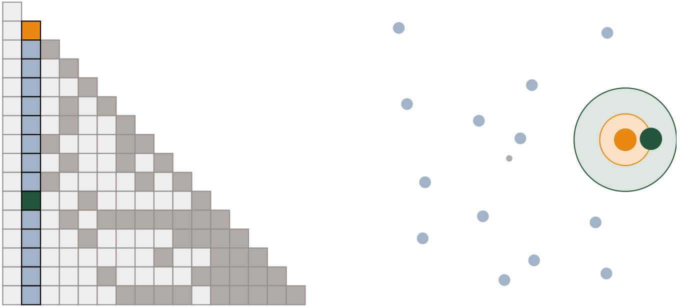
The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$



Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

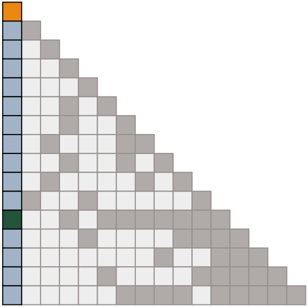
The i-th column selects points within a radius of $\rho\ell_i$ from $\boldsymbol{x}_i$

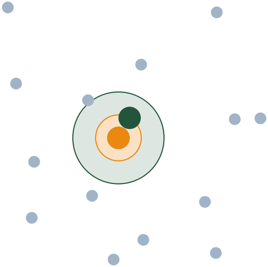


Ordering and sparsity pattern

(Reverse) maximin ordering [Guinness 2018] selects the next point $\boldsymbol{x}_i$ with largest distance ℓ_i to points selected before

The i-th column selects points within a radius of $\rho \ell_i$ from $\boldsymbol{x}_i$





Kullback-Leibler minimization

Compute entries by minimizing Kullback-Leibler divergence

$$L := \underset{\hat{L} \in \mathcal{S}}{\operatorname{argmin}} \; \mathbb{D}_{\mathrm{KL}}\Big(\mathcal{N}(\mathbf{0}, \Theta) \; \Big\| \; \mathcal{N}(\mathbf{0}, (\hat{L}\hat{L}^{\mathsf{T}})^{-1}) \Big)$$

Kullback-Leibler minimization

Compute entries by minimizing Kullback-Leibler divergence

$$L \coloneqq \underset{\hat{L} \in \mathcal{S}}{\operatorname{argmin}} \, \mathbb{D}_{\mathrm{KL}}\Big(\mathcal{N}(\mathbf{0}, \Theta) \, \Big\| \, \mathcal{N}(\mathbf{0}, (\hat{L}\hat{L}^{\top})^{-1}) \Big)$$

Efficient and embarrassingly parallel closed-form solution

$$L_{s_i, i} = \frac{\Theta_{s_i, s_i}^{-1} \boldsymbol{e}_1}{\sqrt{\boldsymbol{e}_1^{\top} \Theta_{s_i, s_i}^{-1} \boldsymbol{e}_1}}$$

Kullback-Leibler minimization

Compute entries by minimizing Kullback-Leibler divergence

$$L \coloneqq \operatorname*{argmin}_{\hat{L} \in \mathcal{S}} \mathbb{D}_{\mathrm{KL}}\Big(\mathcal{N}(\mathbf{0}, \Theta) \,\Big\|\, \mathcal{N}(\mathbf{0}, (\hat{L}\hat{L}^{\mathsf{T}})^{-1})\Big)$$

Efficient and embarrassingly parallel closed-form solution

$$L_{s_i, i} = \frac{\Theta_{s_i, s_i}^{-1} \boldsymbol{e}_1}{\sqrt{\boldsymbol{e}_1^{\mathsf{T}} \Theta_{s_i, s_i}^{-1} \boldsymbol{e}_1}}$$

Achieves state of the art ϵ-accuracy in time complexity $\mathcal{O}\left(N \log^{2d}\left(\frac{N}{\epsilon}\right)\right)$ with $\mathcal{O}\left(N \log^{d}\left(\frac{N}{\epsilon}\right)\right)$ nonzero entries [Schäfer, Katzfuss, and Owhadi 2021]

This work: KL-minimization, revisited

Plug optimal L back into the KL divergence

$$\mathbb{D}_{\mathrm{KL}}\left(\Theta \,\middle\|\, (LL^{\mathsf{T}})^{-1}\right) = \sum_{i=1}^{N} \left[\log\left(\Theta_{i,i|s_i \setminus \{i\}}\right) - \log\left(\Theta_{i,i|i+1:}\right)\right]$$

This work: KL-minimization, revisited

Plug optimal L back into the KL divergence

$$\mathbb{D}_{\mathrm{KL}}\left(\Theta \;\middle\|\; (LL^{\top})^{-1}\right) = \sum_{i=1}^{N} \left[\log\left(\Theta_{i,i \mid s_i \setminus \{i\}}\right) - \log\left(\Theta_{i,i \mid i+1:}\right)\right]$$

KL $\Leftrightarrow$ total error over independent regression problems

This work: KL-minimization, revisited

Plug optimal L back into the KL divergence

$$\mathbb{D}_{\mathrm{KL}}\left(\Theta \,\middle\|\, (LL^{\mathsf{T}})^{-1}\right) = \sum_{i=1}^{N} \left[\log\left(\Theta_{i,i|s_i \setminus \{i\}}\right) - \log\left(\Theta_{i,i|i+1:}\right)\right]$$

KL $\Leftrightarrow$ total error over independent regression problems

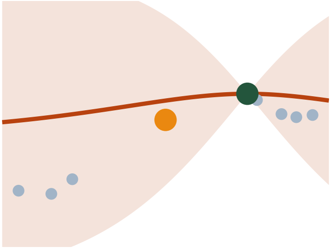
Goal: minimize posterior variance of i-th prediction point by selecting training points s_i *most informative* to that point

Variance $\Leftrightarrow$ mutual information $\Leftrightarrow$ mean squared error

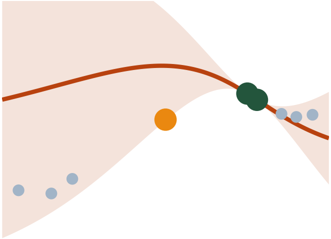
Conditional k-nearest neighbors

Sparse Gaussian process regression, experimental design, active set, etc.

Naive: select k closest points



Conditional k-nearest neighbors



Sparse Gaussian process regression, experimental design, active set, etc.

Naive: select k closest points

Chooses redundant information

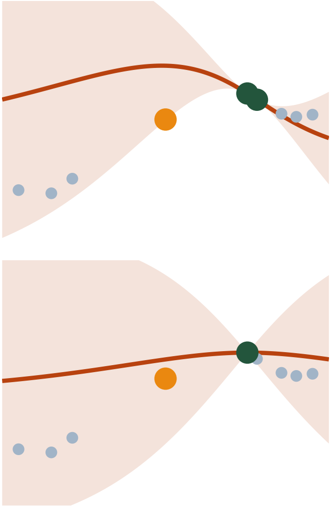
Conditional k-nearest neighbors

Sparse Gaussian process regression, experimental design, active set, etc.

Naive: select k closest points

Chooses redundant information

Maximize *mutual information*!



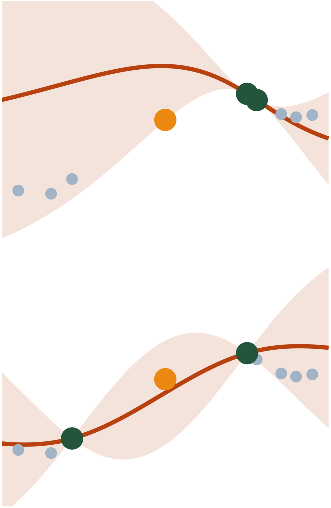
Conditional k-nearest neighbors

Sparse Gaussian process regression, experimental design, active set, etc.

Naive: select k closest points

Chooses redundant information

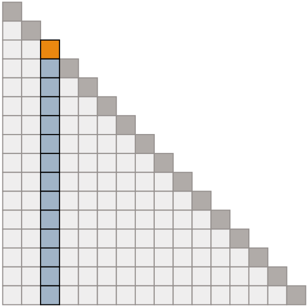
Maximize *mutual information*!

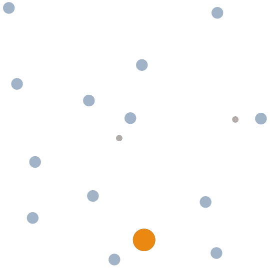


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

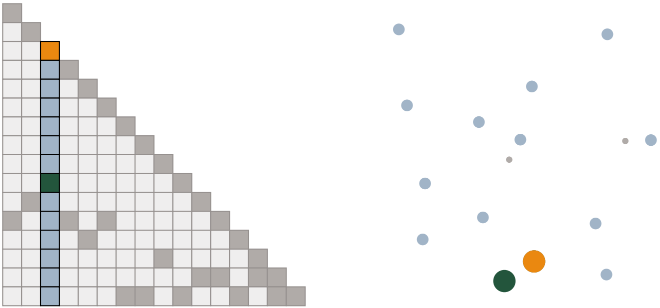




Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

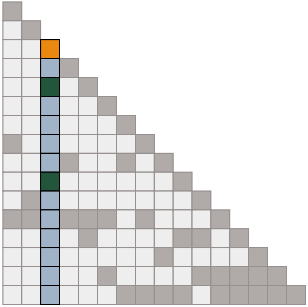
In practice, restrict candidate set to nearest neighbors, e.g.

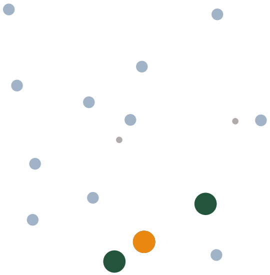


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

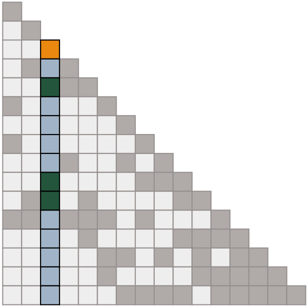


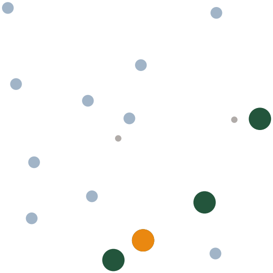


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

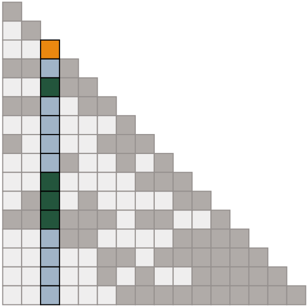


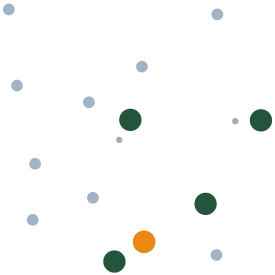


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

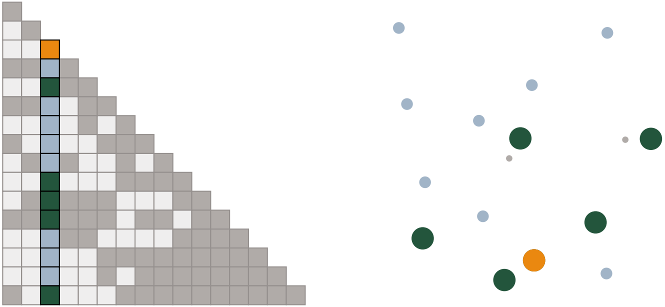




Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

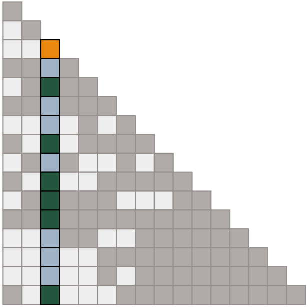
In practice, restrict candidate set to nearest neighbors, e.g.

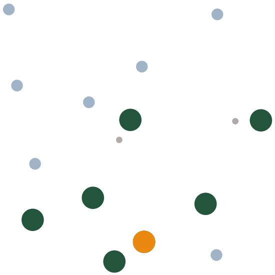


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

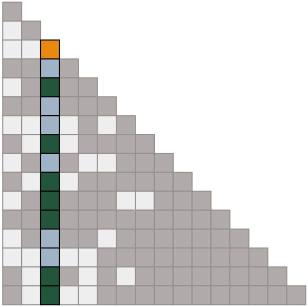


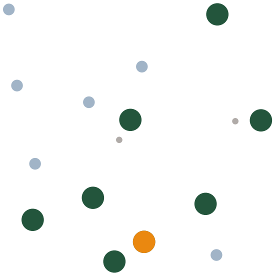


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

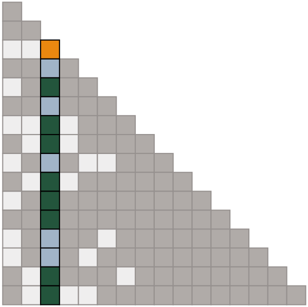


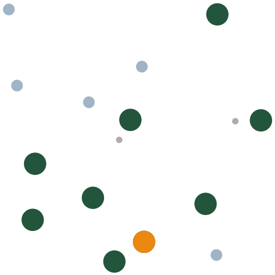


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

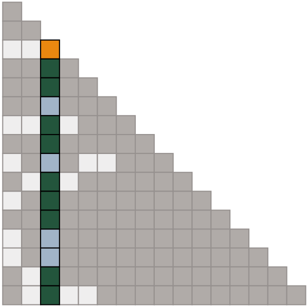


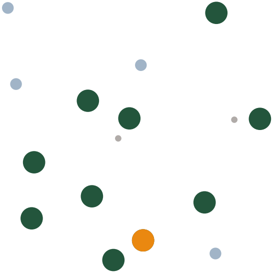


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

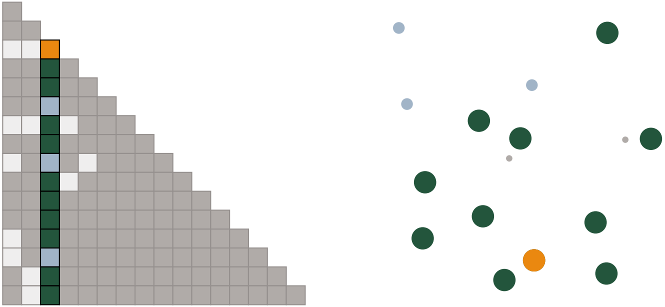




Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

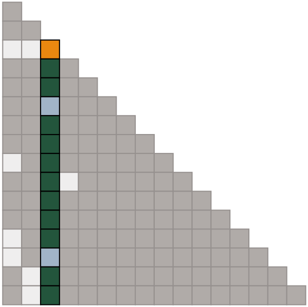
In practice, restrict candidate set to nearest neighbors, e.g.

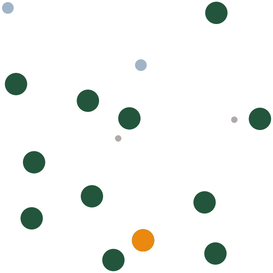


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

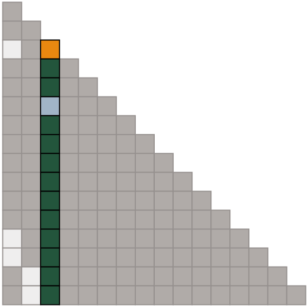


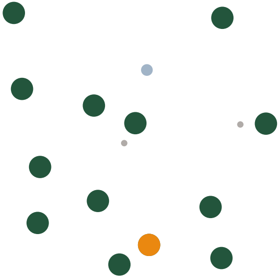


Cholesky factorization by greedy selection

Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.

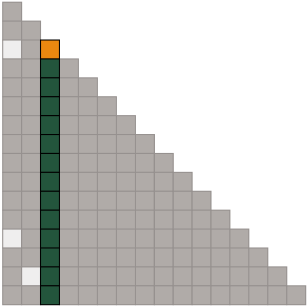


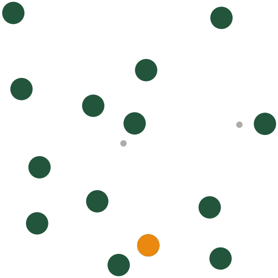


Cholesky factorization by greedy selection

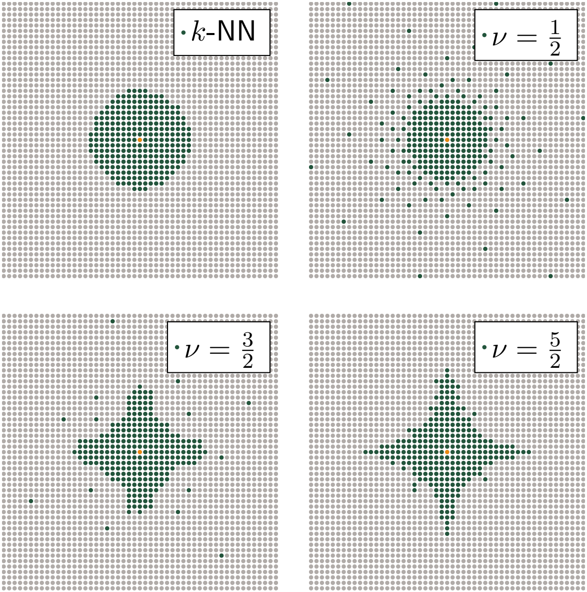
Identify target point as the diagonal entry, candidates are below it, and add selected entries to the sparsity pattern

In practice, restrict candidate set to nearest neighbors, e.g.





Conditional selection



Greedy conditional selection

Intractable to search over $\binom{N}{s}$ subsets, use greedy instead

Greedy conditional selection

Intractable to search over $\binom{N}{s}$ subsets, use greedy instead

Direct computation is $\mathcal{O}(Ns^4)$ to select s points out of N

Greedy conditional selection

Intractable to search over $\binom{N}{s}$ subsets, use greedy instead

Direct computation is $\mathcal{O}(Ns^4)$ to select s points out of N

Maintain partial Cholesky factor for $\mathcal{O}(Ns^2)$

Gaussian process regression

Recall: conditional predictions

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \boldsymbol{\mu}_{\mathsf{Pr}} + \Theta_{\mathsf{Pr},\mathsf{Tr}} \Theta_{\mathsf{Tr},\mathsf{Tr}}^{-1} (\boldsymbol{y}_{\mathsf{Tr}} - \boldsymbol{\mu}_{\mathsf{Tr}})$$
$$\mathbb{C}\mathrm{ov}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \Theta_{\mathsf{Pr},\mathsf{Pr}} - \Theta_{\mathsf{Pr},\mathsf{Tr}} \Theta_{\mathsf{Tr},\mathsf{Tr}}^{-1} \Theta_{\mathsf{Tr},\mathsf{Pr}}$$

Recall: conditional predictions

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \boldsymbol{\mu}_{\mathsf{Pr}} + \Theta_{\mathsf{Pr},\mathsf{Tr}} \Theta_{\mathsf{Tr},\mathsf{Tr}}^{-1} (\boldsymbol{y}_{\mathsf{Tr}} - \boldsymbol{\mu}_{\mathsf{Tr}})$$

$$\mathbb{C}\mathrm{ov}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = \Theta_{\mathsf{Pr},\mathsf{Pr}} - \Theta_{\mathsf{Pr},\mathsf{Tr}} \Theta_{\mathsf{Tr},\mathsf{Tr}}^{-1} \Theta_{\mathsf{Tr},\mathsf{Pr}}$$

Don't need to approximate kernel matrices directly

$$\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = -L_{\mathsf{Pr},\mathsf{Pr}}^{-\top} L_{\mathsf{Tr},\mathsf{Pr}}^{\top} \boldsymbol{y}_{\mathsf{Tr}}$$

$$\mathbb{C}\mathrm{ov}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] = L_{\mathsf{Pr},\mathsf{Pr}}^{-\top} L_{\mathsf{Pr},\mathsf{Pr}}^{-1}$$

$$\boldsymbol{e}_i^{\top} \mathbb{C}\mathrm{ov}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] \boldsymbol{e}_j = (L_{\mathsf{Pr},\mathsf{Pr}}^{-1} \boldsymbol{e}_i)^{\top} (L_{\mathsf{Pr},\mathsf{Pr}}^{-1} \boldsymbol{e}_j)$$

"Prediction points first" [Schäfer, Katzfuss, and Owhadi 2021]

Equivalent to Subset of Datapoints on each prediction point independently, also called `laGP` [Gramacy and Apley 2014; Gramacy and Haaland 2015]

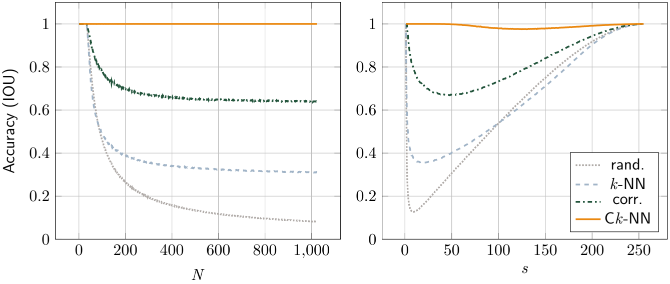
Equivalent to Subset of Datapoints on each prediction point independently, also called `laGP` [Gramacy and Apley 2014; Gramacy and Haaland 2015]

Main differences: supernodal aggregation, handling noise (incomplete Cholesky (ichol)) [Schäfer, Katzfuss, and Owhadi 2021; Schäfer, Sullivan, and Owhadi 2020]

Recovery of sparse factors

Randomly generate *a priori* sparse Cholesky factor L

Attempt to recover L given covariance matrix $\Theta = LL^{\mathsf{T}}$



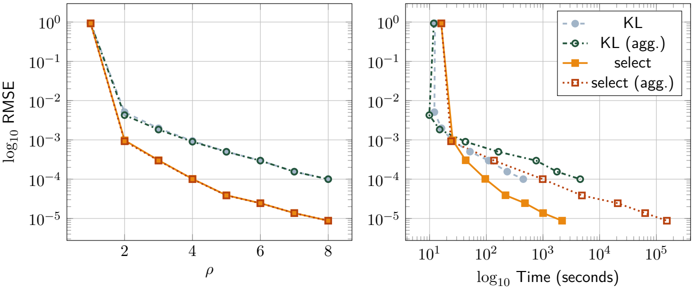
Gaussian process regression

Randomly sample 2^{16} points uniformly from $[0,1]^3$

Randomly partition into 90% training and 10% prediction

Matérn kernel with smoothness $\nu = \frac{5}{2}$ and length scale $\ell = 1$

Draw 10^3 realizations from the resulting Gaussian process



Summary

Sparse Cholesky factorization of *dense* kernel matrices from approximate conditional independence in Gaussian processes

Previous work exploits screening for ordering and sparsity

Replace pure geometry with information-theoretic criteria

More accurate factors at the same sparsity

Conditional selection is computationally efficient

Overview

Conclusion

Computation by direct and iterative methods

Conclusion

Computation by direct and iterative methods

Approximation by low-rank and sparse methods

Conclusion

Computation by direct and iterative methods

Approximation by low-rank and sparse methods

Sparse Cholesky interpolates between the two in a natural way

- direct/iterative: preconditioning strength
- low-rank/sparse: ordering & sparsity pattern

Conclusion

Computation by direct and iterative methods

Approximation by low-rank and sparse methods

Sparse Cholesky interpolates between the two in a natural way
- direct/iterative: preconditioning strength
- low-rank/sparse: ordering & sparsity pattern

Generalizes Nyström method, inducing points, `laGP`, ...

Future work

Solving elliptic PDEs and beyond, particularly for graphics [J. Chen, Schaefer, and Desbrun 2024; J. Chen, Schäfer, et al. 2021; Y. Chen, Owhadi, and Schäfer 2023]

Future work

Solving elliptic PDEs and beyond, particularly for graphics [J. Chen, Schaefer, and Desbrun 2024; J. Chen, Schäfer, et al. 2021; Y. Chen, Owhadi, and Schäfer 2023]

Sampling and inference with non-Gaussian distributions [Katzfuss and Schäfer 2022; Marzouk et al. 2016; Spantini, Bigoni, and Marzouk 2018]

Future work

Solving elliptic PDEs and beyond, particularly for graphics [J. Chen, Schaefer, and Desbrun 2024; J. Chen, Schäfer, et al. 2021; Y. Chen, Owhadi, and Schäfer 2023]

Sampling and inference with non-Gaussian distributions [Katzfuss and Schäfer 2022; Marzouk et al. 2016; Spantini, Bigoni, and Marzouk 2018]

Optimization (second-order, Hessian, natural gradient)

Future work

Solving elliptic PDEs and beyond, particularly for graphics [J. Chen, Schaefer, and Desbrun 2024; J. Chen, Schäfer, et al. 2021; Y. Chen, Owhadi, and Schäfer 2023]

Sampling and inference with non-Gaussian distributions [Katzfuss and Schäfer 2022; Marzouk et al. 2016; Spantini, Bigoni, and Marzouk 2018]

Optimization (second-order, Hessian, natural gradient)

Computational optimal transport [Cuturi 2013]

Future work

Solving elliptic PDEs and beyond, particularly for graphics [J. Chen, Schaefer, and Desbrun 2024; J. Chen, Schäfer, et al. 2021; Y. Chen, Owhadi, and Schäfer 2023]

Sampling and inference with non-Gaussian distributions [Katzfuss and Schäfer 2022; Marzouk et al. 2016; Spantini, Bigoni, and Marzouk 2018]

Optimization (second-order, Hessian, natural gradient)

Computational optimal transport [Cuturi 2013]

Machine learning cf. structured computation?

- HyperAttention [Han et al. 2023], Nyströmformer [Xiong et al. 2021], State space models [Dao and Gu 2024]

Thank You!

References I

Abedsoltan, Amirhesam, Mikhail Belkin, and Parthe Pandit (June 20, 2023). *Toward Large Kernel Models*. DOI: 10.48550/arXiv.2302.02605. arXiv: 2302.02605 [cs]. URL: http://arxiv.org/abs/2302.02605. Pre-published.

Bartels, Simon et al. (Feb. 23, 2022). "Adaptive Cholesky Gaussian Processes". arXiv: 2202.10769 [cs]. URL: http://arxiv.org/abs/2202.10769.

Charlier, Benjamin et al. (Apr. 8, 2021). *Kernel Operations on the GPU, with Autodiff, without Memory Overflows*. DOI: 10.48550/arXiv.2004.11127. arXiv: 2004.11127 [cs]. URL: http://arxiv.org/abs/2004.11127. Pre-published.

Chen, Jiong, Florian Schaefer, and Mathieu Desbrun (July 19, 2024). "Lightning-Fast Method of Fundamental Solutions". In: *ACM Transactions on Graphics* 43.4, pp. 1–16. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3658199. URL: https://dl.acm.org/doi/10.1145/3658199.

Chen, Jiong, Florian Schäfer, et al. (July 19, 2021). "Multiscale Cholesky Preconditioning for Ill-Conditioned Problems". In: *ACM Transactions on Graphics* 40.4, 81:1–81:13. ISSN: 0730-0301. DOI: 10.1145/3450626.3459851. URL: https://doi.org/10.1145/3450626.3459851.

Chen, Tyler and Eric Hallman (Nov. 10, 2022). *Krylov-Aware Stochastic Trace Estimation*. Version 2. DOI: 10.48550/arXiv.2205.01736. arXiv: 2205.01736 [math]. URL: http://arxiv.org/abs/2205.01736. Pre-published.

References III

Chen, Yifan, Ethan N. Epperly, et al. (Oct. 22, 2024). *Randomly Pivoted Cholesky: Practical Approximation of a Kernel Matrix with Few Entry Evaluations*. DOI: 10.48550/arXiv.2207.06503. arXiv: 2207.06503 [math]. URL: http://arxiv.org/abs/2207.06503. Pre-published.

Chen, Yifan, Houman Owhadi, and Florian Schäfer (Apr. 3, 2023). *Sparse Cholesky Factorization for Solving Nonlinear PDEs via Gaussian Processes*. arXiv: 2304.01294 [cs, math, stat]. URL: http://arxiv.org/abs/2304.01294. Pre-published.

Choi, Sou-Cheng (Dec. 2006). "Iterative Methods for Singular Linear Equations and Least-Squares Problems". URL: https://www-leland.stanford.edu/group/SOL/dissertations/sou-cheng-choi-thesis.pdf.

Chow, Edmond and Yousef Saad (Jan. 2014). “Preconditioned Krylov Subspace Methods for Sampling Multivariate Gaussian Distributions”. In: *SIAM Journal on Scientific Computing* 36.2, A588–A608. ISSN: 1064-8275. DOI: 10.1137/130920587. URL: https://epubs.siam.org/doi/abs/10.1137/130920587.

Cuturi, Marco (2013). “Sinkhorn Distances: Lightspeed Computation of Optimal Transport”. In: *Advances in Neural Information Processing Systems*. Vol. 26. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2013/hash/af21d0c97db2e27e13572cbf59eb343d-Abstract.html.

References V

Dao, Tri and Albert Gu (May 31, 2024). *Transformers Are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality*. DOI: 10.48550/arXiv.2405.21060. arXiv: 2405.21060 [cs]. URL: http://arxiv.org/abs/2405.21060. Pre-published.

Epperly, Ethan N., Joel A. Tropp, and Robert J. Webber (Oct. 4, 2024a). *Embrace Rejection: Kernel Matrix Approximation by Accelerated Randomly Pivoted Cholesky*. DOI: 10.48550/arXiv.2410.03969. arXiv: 2410.03969. URL: http://arxiv.org/abs/2410.03969. Pre-published.

Epperly, Ethan N., Joel A. Tropp, and Robert J. Webber (Mar. 31, 2024b). “XTrace: Making the Most of Every Sample in Stochastic Trace Estimation”. In: *SIAM Journal on Matrix Analysis and Applications* 45.1, pp. 1–23. ISSN: 0895-4798, 1095-7162. DOI: 10.1137/23M1548323. arXiv: 2301.07825 [math]. URL: http://arxiv.org/abs/2301.07825.

Fong, William and Eric Darve (Dec. 2009). “The Black-Box Fast Multipole Method”. In: *Journal of Computational Physics* 228.23, pp. 8712–8725. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2009.08.031.

Frangella, Zachary, Joel A. Tropp, and Madeleine Udell (Dec. 17, 2021). *Randomized Nyström Preconditioning*. DOI: 10.48550/arXiv.2110.02820. arXiv: 2110.02820 [cs, math]. URL: http://arxiv.org/abs/2110.02820. Pre-published.

References VII

Gardner, Jacob R. et al. (June 29, 2021). *GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration*. arXiv: `1809.11165 [cs, stat]`. URL: `http://arxiv.org/abs/1809.11165`. Pre-published.

Geoga, Christopher J, Mihai Anitescu, and Michael L Stein (2020). "Scalable Gaussian process computations using hierarchical matrices". In: *Journal of Computational and Graphical Statistics* 29.2, pp. 227–237.

Golub, Gene H. and Charles F. Van Loan (1996). *Matrix Computations*. 3rd ed. Johns Hopkins Studies in the Mathematical Sciences. Baltimore: Johns Hopkins University Press. 694 pp. ISBN: 978-0-8018-5413-2 978-0-8018-5414-9.

- Graham, Ivan G. et al. (Mar. 20, 2018). *Analysis of Circulant Embedding Methods for Sampling Stationary Random Fields*. DOI: 10.48550/arXiv.1710.00751. arXiv: 1710.00751 [math]. URL: http://arxiv.org/abs/1710.00751. Pre-published.
- Gramacy, Robert B. and Daniel W. Apley (Oct. 10, 2014). *Local Gaussian Process Approximation for Large Computer Experiments*. arXiv: 1303.0383 [stat]. URL: http://arxiv.org/abs/1303.0383. Pre-published.
- Gramacy, Robert B. and Benjamin Haaland (Jan. 5, 2015). *Speeding up Neighborhood Search in Local Gaussian Process Prediction*. arXiv: 1409.0074 [stat]. URL: http://arxiv.org/abs/1409.0074. Pre-published.

Guinness, Joseph (Oct. 2, 2018). "Permutation and Grouping Methods for Sharpening Gaussian Process Approximations". In: *Technometrics* 60.4, pp. 415–429. ISSN: 0040-1706, 1537-2723. DOI: 10.1080/00401706.2018.1437476. arXiv: 1609.05372 [stat]. URL: http://arxiv.org/abs/1609.05372.

Han, Insu et al. (Dec. 1, 2023). *HyperAttention: Long-context Attention in Near-Linear Time*. arXiv: 2310.05869 [cs]. URL: http://arxiv.org/abs/2310.05869. Pre-published.

Higham, Nicholas J. (Jan. 2008). *Functions of Matrices*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics. 431 pp. ISBN: 978-0-89871-646-7. DOI: 10.1137/1.9780898717778. URL: https://epubs.siam.org/doi/book/10.1137/1.9780898717778.

Huan, Stephen et al. (July 21, 2023). *Sparse Cholesky Factorization by Greedy Conditional Selection*. DOI: 10.48550/arXiv.2307.11648. arXiv: 2307.11648 [cs, math, stat]. URL: http://arxiv.org/abs/2307.11648. Pre-published.

Kaporin, I. E. (1990). “An Alternative Approach to Estimating the Convergence Rate of the CG Method”. In: *Numerical Methods and Software, Yu. A. Kuznetsov, ed., Dept. of Numerical Mathematics, USSR Academy of Sciences, Moscow*, pp. 55–72.

Kaporin, I. E. (1994). “New Convergence Results and Preconditioning Strategies for the Conjugate Gradient Method”. In: *Numerical Linear Algebra with Applications* 1.2, pp. 179–210. ISSN: 1099-1506. DOI: 10.1002/nla.1680010208. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.1680010208.

Katzfuss, Matthias and Florian Schäfer (Feb. 28, 2022). “Scalable Bayesian Transport Maps for High-Dimensional Non-Gaussian Spatial Fields”. arXiv: 2108.04211 [stat]. URL: http://arxiv.org/abs/2108.04211.

Krause, Andreas and Jonas Hübotter (Feb. 7, 2025). *Probabilistic Artificial Intelligence*. DOI: 10.48550/arXiv.2502.05244. arXiv: 2502.05244 [cs]. URL: http://arxiv.org/abs/2502.05244. Pre-published.

References XII

Krause, Andreas, Ajit Singh, and Carlos Guestrin (June 1, 2008). "Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies". In: *The Journal of Machine Learning Research* 9, pp. 235–284. ISSN: 1532-4435.

Lecun, Y. et al. (Nov. 1998). "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

Litvinenko, Alexander (May 2019). *HLIBCov: Parallel Hierarchical Matrix Approximation of Large Covariance Matrices and Likelihoods with Applications in Parameter Identification*. DOI: 10.48550/arXiv.1709.08625. arXiv: 1709.08625 [stat].

References XIII

Martinsson, Per-Gunnar and Joel Tropp (Mar. 15, 2021). *Randomized Numerical Linear Algebra: Foundations & Algorithms*. arXiv: 2002.01387 [cs, math]. URL: http://arxiv.org/abs/2002.01387. Pre-published.

Marzouk, Youssef et al. (2016). "An Introduction to Sampling via Measure Transport". DOI: 10.1007/978-3-319-11259-6_23-1. arXiv: 1602.05023 [math, stat]. URL: http://arxiv.org/abs/1602.05023.

Meyer, Raphael A. et al. (June 10, 2021). *Hutch++: Optimal Stochastic Trace Estimation*. arXiv: 2010.09649 [cs, math]. URL: http://arxiv.org/abs/2010.09649. Pre-published.

Parker, Albert and Colin Fox (Jan. 2012). "Sampling Gaussian Distributions in Krylov Spaces with Conjugate Gradients". In: *SIAM Journal on Scientific Computing* 34.3, B312–B334. ISSN: 1064-8275. DOI: 10.1137/110831404. URL: https://epubs.siam.org/doi/10.1137/110831404.

Persson, David, Alice Cortinovis, and Daniel Kressner (May 6, 2022). *Improved Variants of the Hutch++ Algorithm for Trace Estimation*. arXiv: 2109.10659 [cs, math]. URL: http://arxiv.org/abs/2109.10659. Pre-published.

Potapczynski, Andres et al. (June 28, 2021). *Bias-Free Scalable Gaussian Processes via Randomized Truncations*. arXiv: 2102.06695 [cs, stat]. URL: http://arxiv.org/abs/2102.06695. Pre-published.

Quiñonero-Candela, Joaquin and Carl Edward Rasmussen (2005). "A Unifying View of Sparse Approximate Gaussian Process Regression". In: *Journal of Machine Learning Research* 6.65, pp. 1939–1959. ISSN: 1533-7928. URL: http://jmlr.org/papers/v6/quinonero-candela05a.html.

Rasmussen, Carl Edward and Christopher K. I. Williams (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press. 248 pp. ISBN: 978-0-262-18253-9.

Rudi, Alessandro, Luigi Carratino, and Lorenzo Rosasco (Jan. 31, 2018). *FALKON: An Optimal Large Scale Kernel Method*. DOI: 10.48550/arXiv.1705.10958. arXiv: 1705.10958 [stat]. URL: http://arxiv.org/abs/1705.10958. Pre-published.

Saad, Yousef (Jan. 2003). *Iterative Methods for Sparse Linear Systems*. Second. Society for Industrial and Applied Mathematics. ISBN: 978-0-89871-534-7 978-0-89871-800-3. DOI: 10.1137/1.9780898718003. URL: http://epubs.siam.org/doi/book/10.1137/1.9780898718003.

Schäfer, Florian, Matthias Katzfuss, and Houman Owhadi (Oct. 22, 2021). “Sparse Cholesky Factorization by Kullback-Leibler Minimization”. arXiv: 2004.14455 [cs, math, stat]. URL: http://arxiv.org/abs/2004.14455.

Schäfer, Florian, T. J. Sullivan, and Houman Owhadi (Oct. 30, 2020). “Compression, Inversion, and Approximate PCA of Dense Kernel Matrices at near-Linear Computational Complexity”. arXiv: 1706.02205 [cs, math]. URL: http://arxiv.org/abs/1706.02205.

- Spantini, Alessio, Daniele Bigoni, and Youssef Marzouk (July 1, 2018). *Inference via Low-Dimensional Couplings*. DOI: 10.48550/arXiv.1703.06131. arXiv: 1703.06131 [stat]. URL: http://arxiv.org/abs/1703.06131. Pre-published.
- Stein, Michael L. (Feb. 2002). "The Screening Effect in Kriging". In: *The Annals of Statistics* 30.1, pp. 298–323. ISSN: 0090-5364, 2168-8966. DOI: 10.1214/aos/1015362194. URL: https://projecteuclid.org/journals/annals-of-statistics/volume-30/issue-1/The-screening-effect-in-Kriging/10.1214/aos/1015362194.full.
- Tropp, Joel A. (Dec. 14, 2023). "CMS/ACM 117: Probability Theory & Computational Mathematics". Version Accepted. In: DOI: 10.7907/Q75SZ-E1E79. URL: https://authors.library.caltech.edu/doi/10.7907/q75sz-e1e79.

Wang, Ruoxi et al. (Aug. 2021). "PBBFMM3D: A Parallel Black-Box Algorithm for Kernel Matrix-Vector Multiplication". In: *Journal of Parallel and Distributed Computing* 154, pp. 64–73. ISSN: 07437315. DOI: 10.1016/j.jpdc.2021.04.005. arXiv: 1903.02153 [cs].

Xiong, Yunyang et al. (Mar. 31, 2021). *Nyströmformer: A Nyström-Based Algorithm for Approximating Self-Attention*. DOI: 10.48550/arXiv.2102.03902. arXiv: 2102.03902 [cs]. URL: http://arxiv.org/abs/2102.03902. Pre-published.

Cholesky factorization

For a numerical algorithm (up-, down-, left-, right-)looking

https://theoryclub.github.io/files/cholesky_presentation.pdf

- up-looking: i-th iteration builds $\mathrm{chol}(\Theta_{:i,:i})$ in time $\mathcal{O}(i^2)$
- left-looking: i-th iteration builds $\mathrm{chol}(\Theta)_{:,:i}$ in time $\mathcal{O}(Ni)$

Computing the Cholesky Factorization

Down-looking

Like LU; Gaussian elimination downwards

```
def down_cholesky(theta: np.ndarray) -> np.ndarray:
    n = len(theta)
    M = np.copy(theta)
    L = np.identity(n)
    for i in range(n):
        for j in range(i + 1, n):
            L[j, i] = M[j, i] / M[i, i]
            # zero out everything below
            M[j] -= L[j, i] * M[i]
        # update L
        L[:, i] *= np.sqrt(M[i, i])
    return L
```

Let L' be blocked according to

$$L'L'^{\mathsf{T}} = \begin{pmatrix} L & \mathbf{0} \\ \boldsymbol{r}^{\mathsf{T}} & d \end{pmatrix} \begin{pmatrix} L^{\mathsf{T}} & \boldsymbol{r} \\ \mathbf{0}^{\mathsf{T}} & d \end{pmatrix} = \begin{pmatrix} LL^{\mathsf{T}} & L\boldsymbol{r} \\ \boldsymbol{r}^{\mathsf{T}}L^{\mathsf{T}} & \boldsymbol{r}^{\mathsf{T}}\boldsymbol{r} + d^2 \end{pmatrix}$$

So if we have a Cholesky factor for a principle submatrix of Θ, we can extend it inductively by reading off appropriate data!

$$\begin{pmatrix} LL^{\mathsf{T}} & L\boldsymbol{r} \\ \boldsymbol{r}^{\mathsf{T}}L^{\mathsf{T}} & \boldsymbol{r}^{\mathsf{T}}\boldsymbol{r} + d^2 \end{pmatrix} = \begin{pmatrix} \Theta & \boldsymbol{c} \\ \boldsymbol{c}^{\mathsf{T}} & D \end{pmatrix}$$

$$\boldsymbol{r} = L^{-1}\boldsymbol{c}$$

$$d = \sqrt{D - \boldsymbol{r}^{\mathsf{T}}\boldsymbol{r}}$$

Computing the Cholesky Factorization

Up-looking

```
def L_solve(L: np.ndarray, y: np.ndarray) -> np.ndarray:
    """Solves L x = y for lower triangular L."""
    n = len(y)
    x = np.zeros(n)
    for i in range(n):
        x[i] = (y[i] - np.dot(L[i, :i], x[:i])) / L[i, i]
    return x


def up_cholesky(theta: np.ndarray) -> np.ndarray:
    n = len(theta)
    L = np.zeros((n, n))
    for i in range(n):
        row = L_solve(L, theta[:i, i])
        L[i, :i] = row
        L[i, i] = np.sqrt(theta[i, i] - np.dot(row, row))
    return L
```

Write L in terms of its columns

$$LL^{\mathsf{T}} = \begin{pmatrix} \boldsymbol{l}_1 & \cdots & \boldsymbol{l}_N \end{pmatrix} \begin{pmatrix} \boldsymbol{l}_1^{\mathsf{T}} \\ \vdots \\ \boldsymbol{l}_N^{\mathsf{T}} \end{pmatrix} = \boldsymbol{l}_1 \boldsymbol{l}_1^{\mathsf{T}} + \cdots + \boldsymbol{l}_N \boldsymbol{l}_N^{\mathsf{T}} = \Theta$$

From lower triangularity, nested submatrices!

Computing the Cholesky Factorization

Right-looking

Read off first column

$$
\begin{aligned}
\boldsymbol{l}_1 \boldsymbol{l}_1^{\mathsf{T}} + \boldsymbol{l}_2 \boldsymbol{l}_2^{\mathsf{T}} + \cdots + \boldsymbol{l}_N \boldsymbol{l}_N^{\mathsf{T}} &= \Theta \\
\boldsymbol{l}_1 \boldsymbol{l}_1^{\mathsf{T}} &= \Theta_{:,1} \\
l_{1,1}^2 &= \Theta_{1,1};\ l_{1,1} = \sqrt{\Theta_{1,1}} \\
\boldsymbol{l}_1 &= \frac{\Theta_{:,1}}{l_{1,1}} = \frac{\Theta_{:,1}}{\sqrt{\Theta_{1,1}}} \\
\boldsymbol{l}_2 \boldsymbol{l}_2^{\mathsf{T}} + \cdots + \boldsymbol{l}_N \boldsymbol{l}_N^{\mathsf{T}} &= \Theta - \left(\frac{\Theta_{:,1}}{\sqrt{\Theta_{1,1}}} \right) \left(\frac{\Theta_{:,1}}{\sqrt{\Theta_{1,1}}} \right)^{\mathsf{T}} \\
&= \Theta - \frac{\Theta_{:,1} \Theta_{:,1}^{\mathsf{T}}}{\Theta_{1,1}}
\end{aligned}
$$

Proceed inductively on rank-one update

Computing the Cholesky Factorization

Right-looking

```
def right_cholesky(theta: np.ndarray) -> np.ndarray:
    n = len(theta)
    M = np.copy(theta)
    L = np.zeros((n, n))
    for i in range(n):
        L[:, i] = M[:, i] / np.sqrt(M[i, i])
        M -= np.outer(L[:, i], L[:, i])
    return L
```

Computing the Cholesky Factorization

Left-looking

Recall $\boldsymbol{l}_1 \boldsymbol{l}_1^\mathsf{T} + + \cdots + \boldsymbol{l}_N \boldsymbol{l}_N^\mathsf{T} = \Theta$; look at $\boldsymbol{l}_i \boldsymbol{l}_i^\mathsf{T}$

$$
\begin{aligned}
l_{i,i} \boldsymbol{l}_i &= \left(\Theta - (\boldsymbol{l}_1 \boldsymbol{l}_1^\mathsf{T} + \cdots + \boldsymbol{l}_{i-1} \boldsymbol{l}_{i-1}^\mathsf{T})\right) e_i \\
&= \Theta_{:,i} - (l_{1,i} \boldsymbol{l}_1 + \cdots + l_{i-1,i} \boldsymbol{l}_{i-1}) \\
&= \Theta_{:,i} - \begin{pmatrix} \boldsymbol{l}_1 & \cdots & \boldsymbol{l}_{i-1} \end{pmatrix} \begin{pmatrix} l_{1,i} \\ \vdots \\ l_{i,i-1} \end{pmatrix} \\
&= \Theta_{:,i} - L_{:,:i} L_{i,:i}
\end{aligned}
$$

Don't need to store modified Θ in memory!

```
def left_cholesky(theta: np.ndarray) -> np.ndarray:
    n = len(theta)
    L = np.zeros((n, n))
    for i in range(n):
        L[:, i] = theta[:, i] - L[:, :i] @ L[i, :i]
        L[:, i] /= np.sqrt(L[i, i])
    return L
```

Conjugate gradient

Solve $A\boldsymbol{x}^* = \boldsymbol{b}$, initial guess $\boldsymbol{x}_0$ and residual $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$

Optimization perspective: minimizing $\|\boldsymbol{x} - \boldsymbol{x}^*\|_A$ equivalent to

$$\mathcal{L}(\boldsymbol{x}) = \frac{1}{2}\langle \boldsymbol{x}, \boldsymbol{x}\rangle_A - \langle \boldsymbol{b}, \boldsymbol{x}\rangle = \frac{1}{2}\boldsymbol{x}^\mathsf{T} A\boldsymbol{x} - \boldsymbol{b}^\mathsf{T}\boldsymbol{x}$$

Like gradient descent ($\nabla\mathcal{L}(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}$), but pick directions (and learning rate) *optimally*, i.e. without any backtracking

$$\boldsymbol{x}_k := \min_{\boldsymbol{p}\in\mathcal{K}_k(A,\boldsymbol{r}_0)} \mathcal{L}(\boldsymbol{x}_0 + \boldsymbol{p})$$

for *Krylov subspace* $\mathcal{K}_k(A, \boldsymbol{r}_0) :=$ span$\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{k-1}\boldsymbol{r}_0\}$

Hence residuals $\boldsymbol{r}_k$ orthogonal and directions $\boldsymbol{p}_k$ A-conjugate

The polynomial perspective

Why pick Krylov subspace for search directions?

$$\boldsymbol{p}_k \in \mathcal{K}_k(A, \boldsymbol{r}_0) := \mathsf{span}\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{k-1}\boldsymbol{r}_0\}$$

Want $\mathcal{K}_{k+1}(A, \boldsymbol{r}_0)$ to include $\boldsymbol{x}_k$ and gradient $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$,

$$\boldsymbol{b} - A(\underbrace{\boldsymbol{x}_0 + \boldsymbol{p}_k}_{=\boldsymbol{x}_k}) = \underbrace{\boldsymbol{b} - A\boldsymbol{x}_0}_{=\boldsymbol{r}_0} - \underbrace{A\boldsymbol{p}_k}_{\in\mathcal{K}_{k+1}(A,\boldsymbol{r}_0)} \in \mathcal{K}_{k+1}(A, \boldsymbol{r}_0)$$

Naturally associated to (matrix) polynomials as

$$\boldsymbol{p}_k \in \mathcal{K}_k(A, \boldsymbol{r}_0) \iff \boldsymbol{p}_k = \varphi(A)\boldsymbol{r}_0$$

for some degree $k-1$ polynomial φ

User's notes on conjugate gradient

Convergence rate bounded by *condition number*

$$\kappa(A) := \|A\|_2 \|A^{-1}\|_2 = \frac{\lambda_{\mathsf{max}}(A)}{\lambda_{\mathsf{min}}(A)}$$

Rate of convergence $\approx (\sqrt{\kappa(A)} - 1)/(\sqrt{\kappa(A)} + 1)$, number of iterations to ε accuracy $\approx \sqrt{\kappa(A)} \log(\|\boldsymbol{e}_0\|_A / \varepsilon)$

- Convergence in n iterations only guaranteed in *exact arithmetic*
- Does not depend on full spectrum of A! (e.g. $\kappa(A) = \kappa(A^{-1})$)

Often *Kaporin condition number* [Kaporin 1990, 1994]

$$B(A) := \frac{\mathrm{trace}(A)/N}{\det(A)^{1/N}}$$

gives more accurate predictions of empirical progress

Preconditioning

As previously seen, rates depend critically on condition number

Idea: introduce *preconditioner* M s.t. $\kappa(M^{-1}A) \ll \kappa(A)$

- Caveat: need to be able to apply M^{-1} efficiently

CG on $M = FF^\mathsf{T}$, solve $(F^{-1}AF^{-\mathsf{T}})\boldsymbol{y} = F^{-1}\boldsymbol{b}$, $\boldsymbol{x} = F^{-\mathsf{T}}\boldsymbol{y}$

Happens all implicitly, don't need factored M, just need psd!

Jacobi, incomplete Cholesky, FSAI...

Randomized stopping to remove bias
[Potapczynski et al. 2021]

Conjugate residual

CG only works for symmetric + positive definite matrices

Conjugate residual/MINRES: only requires symmetry

- Minimize residual $\|\boldsymbol{b} - A\boldsymbol{x}_k\|_2$ instead of energy $\|\boldsymbol{x}^* - \boldsymbol{x}_k\|_A$
- *Residuals* conjugate and *search directions* orthogonal

Non-symmetric: GMRES, QMR, BiCG, CGS, BiCGSTAB
CGNR, CGNE, LSQR, LSMR

- Square the condition or re-orthogonalize

Practical implementation

Basic Linear Algebra Subprograms (BLAS) hierarchy

- Level 1: vector operations, e.g. `axpy`
  $\mathcal{O}(n)$ memops, $\mathcal{O}(n)$ flops
- Level 2: matrix-vector operations, e.g. `gemv`
  $\mathcal{O}(n^2)$ memops, $\mathcal{O}(n^2)$ flops
- Level 3: matrix-matrix operations, e.g. `gemm`
  $\mathcal{O}(n^2)$ memops, $\mathcal{O}(n^3)$ flops

"Kernel"-style programming especially important for GPUs

GPs on GPU

Ongoing line of work leveraging GPUs [Charlier et al. 2021]

Based on

- low-rank approximations [Abedsoltan, Belkin, and Pandit 2023; Gardner et al. 2021; Rudi, Carratino, and Rosasco 2018],
- gradient descent [Abedsoltan, Belkin, and Pandit 2023],
- conjugate gradient [Gardner et al. 2021; Rudi, Carratino, and Rosasco 2018]

Statistical Cholesky factorization

Factor covariance matrix Θ or precision matrix $Q = \Theta^{-1}$?

$$\Theta_{i,i} = \mathbb{V}\mathrm{ar}[y_i] \qquad Q^{-1}_{i,i} = \mathbb{V}\mathrm{ar}[y_i \mid y_{k \neq i}]$$

$$\Theta_{i,j} = \mathbb{C}\mathrm{ov}[y_i, y_j] \qquad \frac{-Q_{i,j}}{\sqrt{Q_{i,i} Q_{j,j}}} = \mathbb{C}\mathrm{orr}[y_i, y_j \mid y_{k \neq i,j}]$$

Cholesky factorization $\Leftrightarrow$ iterative conditioning of process

$$L = \mathrm{chol}(\Theta) \qquad R = \mathrm{chol}(Q)$$

$$L_{i,j} = \frac{\mathbb{C}\mathrm{ov}[y_i, y_j \mid y_{k<j}]}{\sqrt{\mathbb{V}\mathrm{ar}[y_j \mid y_{k<j}]}} \qquad -\frac{R_{i,j}}{R_{j,j}} = \frac{\mathbb{C}\mathrm{ov}[y_i, y_j \mid y_{k>j, k \neq i}]}{\mathbb{V}\mathrm{ar}[y_j \mid y_{k>j, k \neq i}]}$$

Covariance matrix encodes marginal independence

Precision matrix encodes conditional independence

Prefer precision matrix to attenuate density

Mutual information objective

Define *mutual information* or *information gain*

$$\mathbb{I}[\boldsymbol{y}_{\mathsf{Pr}}; \boldsymbol{y}_{\mathsf{Tr}}] = \mathbb{H}[\boldsymbol{y}_{\mathsf{Pr}}] - \mathbb{H}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}]$$

Entropy increasing with log determinant of covariance

Information-theoretic EV-VE identity

$$\begin{aligned} \mathbb{H}[\boldsymbol{y}_{\mathsf{Pr}}] &= \mathbb{H}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}] + \mathbb{I}[\boldsymbol{y}_{\mathsf{Pr}}; \boldsymbol{y}_{\mathsf{Tr}}] \\ \mathbb{V}\mathrm{ar}[\boldsymbol{y}_{\mathsf{Pr}}] &= \mathbb{E}[\mathbb{V}\mathrm{ar}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}]] + \mathbb{V}\mathrm{ar}[\mathbb{E}[\boldsymbol{y}_{\mathsf{Pr}} \mid \boldsymbol{y}_{\mathsf{Tr}}]] \end{aligned}$$

Orthogonal matching pursuit

Conditional selection can be seen as orthogonal matching pursuit in covariance rather than feature space

$$\Theta = F^{\mathsf{T}} F$$

where F's columns F_i are vectors in feature space and

$$\Theta_{i,j} = \langle F_i, F_j \rangle$$

Suppose F has QR factorization

$$F = QR$$

for Q orthonormal and R upper triangular. Then

$$\begin{aligned} \Theta = F^{\mathsf{T}} F &= (QR)^{\mathsf{T}} (QR) \\ &= R^{\mathsf{T}} Q^{\mathsf{T}} QR \\ &= R^{\mathsf{T}} R \end{aligned}$$

so R^{T} is a lower triangular Cholesky factor of Θ.

Fast conditional selection

Selecting candidate k is rank-one downdate to covariance Θ

$$\Theta_{:,:|I,k} = \Theta_{:,:|I} - \boldsymbol{u}\boldsymbol{u}^{\mathsf{T}} \qquad\qquad \boldsymbol{u} = \frac{\Theta_{:,k|I}}{\sqrt{\Theta_{k,k|I}}}$$

Corresponding decrease in posterior variance is

$$u_{\mathsf{Pr}}^2 = \frac{\mathbb{C}\mathrm{ov}[y_{\mathsf{Pr}}, y_k \mid I]^2}{\mathbb{V}\mathrm{ar}[y_k \mid I]} = \mathbb{V}\mathrm{ar}[y_{\mathsf{Pr}} \mid I]\,\mathbb{C}\mathrm{orr}[y_{\mathsf{Pr}}, y_k \mid I]^2$$

Compute $\boldsymbol{u}$ as next column of (partial) Cholesky factor

Replace $\mathcal{O}(N^2)$ update with $\mathcal{O}(Ns)$ by "left-looking"

$$L_{:,i} \leftarrow \Theta_{:,k} - L_{:,:i-1} L_{k,:i-1}^{\mathsf{T}}$$
$$L_{:,i} \leftarrow \frac{L_{:,i}}{\sqrt{L_{k,i}}}$$

Multiple prediction points

Select candidate for *multiple* prediction points jointly

Try to take advantage of "two birds with one stone"

Flipped objective allows efficient algorithm by single selection

$$\operatorname{logdet}(\Theta_{\mathsf{Pr},\mathsf{Pr}|I,k}) - \operatorname{logdet}(\Theta_{\mathsf{Pr},\mathsf{Pr}|I}) = \log(\Theta_{k,k|I,\mathsf{Pr}}) - \log(\Theta_{k,k|I})$$

$\mathcal{O}(Ns^2 + Nm^2 + m^3)$ to select s points out of N candidates for m targets, essentially m times faster than single selection

In aggregated (supernodal) Cholesky factorization, "partial" addition of candidates if candidate is between grouped targets

Conditional structure of partially conditioned covariance

$$\mathbb{C}\mathrm{ov}[\boldsymbol{y}_{\| k}] = \begin{pmatrix} L_{:p} L_{:p}^{\mathsf{T}} & L_{:p} {L'}_{p+1:}^{\mathsf{T}} \\ {L'}_{p+1:} L_{:p}^{\mathsf{T}} & {L'}_{p+1:} {L'}_{p+1:}^{\mathsf{T}} \end{pmatrix} = \begin{pmatrix} L_{:p} \\ {L'}_{p+1:} \end{pmatrix} \begin{pmatrix} L_{:p} \\ {L'}_{p+1:} \end{pmatrix}^{\mathsf{T}}$$

Efficient inductive algorithm matches complexity of multiple-target selection algorithm using rank-one downdating

$$\begin{aligned} \Theta_{i,i|:i-1} &= L_{i,i}^2 \\ \Theta_{j,i|:i-1} &= L_{j,i} \cdot L_{i,i} \\ \Theta_{i,i|:i-1,j} &= \Theta_{i,i|:i-1} - \Theta_{j,i|:i-1}^2 / \Theta_{j,j|:i-1} \\ \Theta_{j,j|:i-1,i} &= \Theta_{j,j|:i-1} - \Theta_{j,i|:i-1}^2 / \Theta_{i,i|:i-1} = \Theta_{j,j|:i} \end{aligned}$$

Partial selection

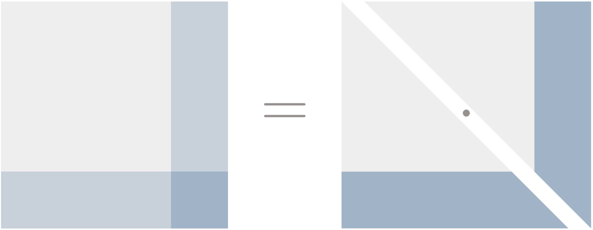


Figure: Cholesky factorization of a partially conditioned covariance matrix. Here grey denotes fully unconditional, blue denotes fully conditional, and the mixed color denotes interaction between the two.

Allocating nonzeros by global selection

It matters how many nonzeros each columns receives, especially for inhomogeneous geometries

Distributing evenly maximizes computational efficiency

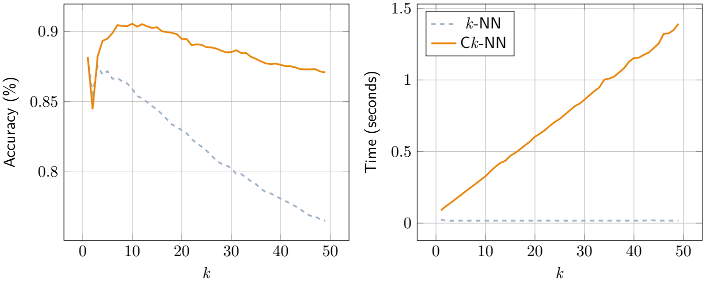
To maximize accuracy, maintain *global* priority queue that determines both the next candidate to select and its column

Priority queue implemented as array-backed binary heap, e.g.

Image classification by mode label of k-"nearest" neighbors

MNIST database of handwritten digits [Lecun et al. 1998]

Matérn kernel with smoothness $\nu = \frac{3}{2}$ and length scale 2^{10}



Cholesky factorization

Randomly sample $N = 2^{16}$ points uniformly from $[0, 1]^3$

Matérn kernel with smoothness $\nu = \frac{5}{2}$ and length scale $\ell = 1$