

NUMERICS OF MACHINE LEARNING
LECTURE 02
NUMERICAL LINEAR ALGEBRA

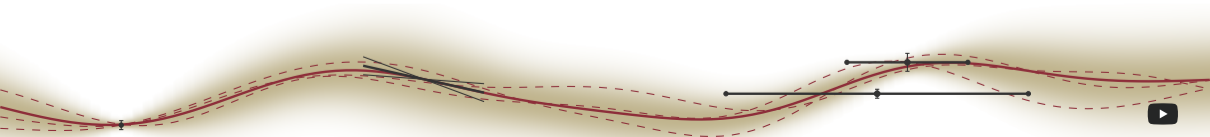
Jonathan Wenger
presented by Marvin Pförtner

27 October 2022

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



FACULTY OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
CHAIR FOR THE METHODS OF MACHINE LEARNING





Outlook

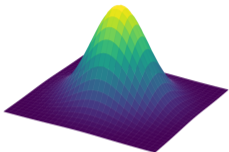
- ▶ Why numerical linear algebra?
- ▶ Implementation of a mathematical operation matters a lot!
- ▶ Fundamental tasks of numerical linear algebra for machine learning.
- ▶ Implementing Gaussian process regression.



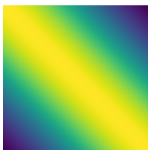
Numerical Linear Algebra is central to Machine Learning

Arguably, the most fundamental set of tools in (scientific) machine learning.

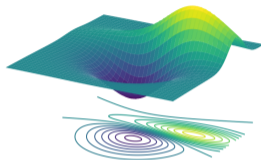
Basic Statistics



(Probabilistic) Kernel Methods



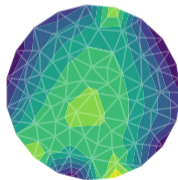
Optimization



Graphs and (Neural) Networks



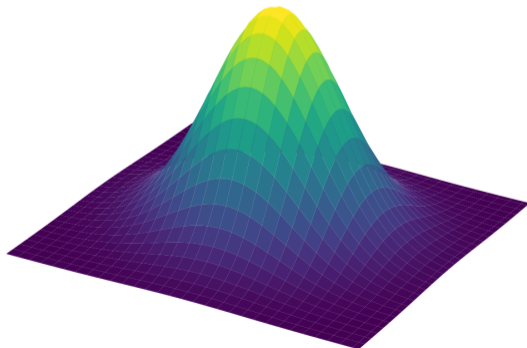
Differential Equations



...and many more (e.g. dimensionality reduction, generative models, ...).

Normal Distribution

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$
$$p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$



Numerical Linear Algebra is central to Machine Learning

Example: Statistics.

General Linear Model

$$y = \Phi(X)w + \varepsilon$$

$$\Phi^T \Phi =$$



$p \times p$

Special case: linear regression $w = (\Phi^T \Phi)^{-1} \Phi^T y = \Phi^\dagger y$

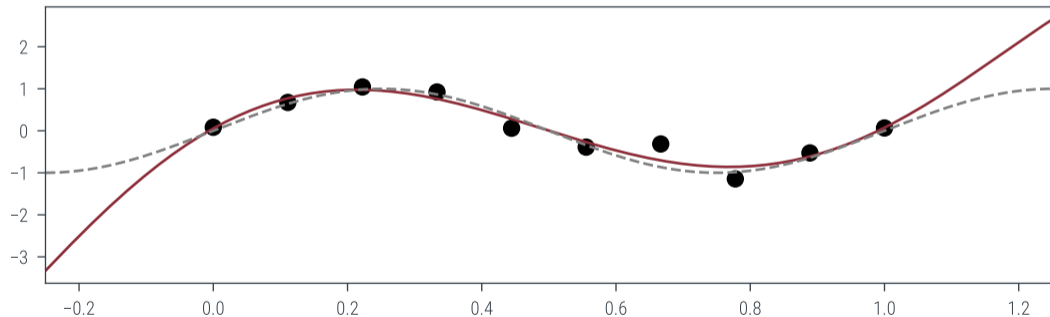


Figure: Linear regression with polynomial features $\Phi(X) = (1 \quad X \quad X^2 \quad \dots \quad X^5) \in \mathbb{R}^{n \times p}$.

Example: Dimensionality Reduction.

Principal Component Analysis (PCA)

$$X^T X = Q \Lambda Q^T \implies x \mapsto Q_{1:k}^T x$$



Labeled Faces in the Wild (LFW): Huang et al., 2009. Pre-processed by Samira Samadi

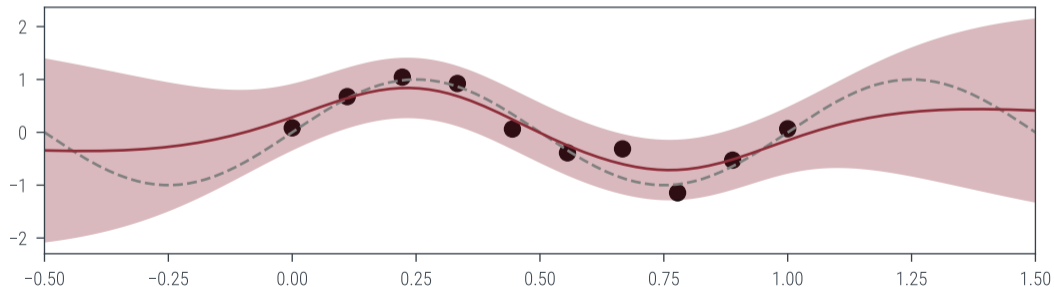
Gaussian Processes

$$f \sim \mathcal{GP}(\mu, k)$$

$$f \mid \mathbf{X}, \mathbf{y} \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$\mu_{\text{post}}(\mathbf{x}) = \mu(\mathbf{x}) + k(\mathbf{x}, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})^{-1}(\mathbf{y} - \mu(\mathbf{X}))$$

$$k_{\text{post}}(\mathbf{x}_0, \mathbf{x}_1) = k(\mathbf{x}_0, \mathbf{x}_1) - k(\mathbf{x}_0, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})^{-1}k(\mathbf{X}, \mathbf{x}_1)$$



Numerical Linear Algebra is central to Machine Learning



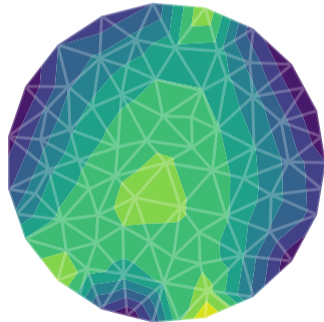
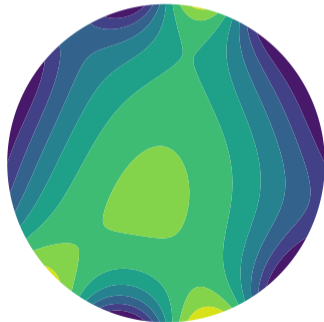
Example: Differential Equations.

Galerkin Method

$Du = f$
linear operator equation



$\hat{D}\hat{u} = \hat{f}$
finite dimensional linear system



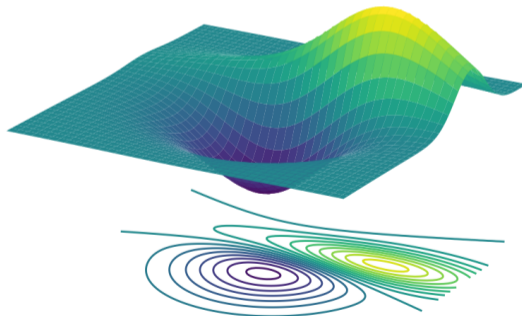
Example: Optimization.

Iterative Optimization Methods

$$\boldsymbol{\theta}_i \approx \arg \min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1} + \alpha_i \mathbf{M}_i \mathbf{d}_i$$

Examples: natural / conjugate / stochastic gradient descent, (Quasi-) Newton method, ...

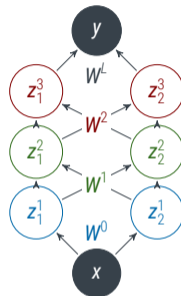


Feedforward Neural Network

$$z^0(x, \theta) = x$$

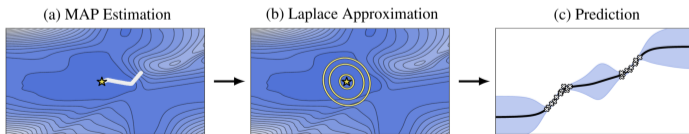
$$z^{\ell+1}(x, \theta) = \sigma(W^\ell z^\ell + b^\ell)$$

$$y := f(x, \theta) = z^L(x, \theta)$$



Backward pass also reduces to matrix-vector multiplication.

Bayesian deep learning via Laplace approximation: $p(\theta | \mathcal{D}) \approx \mathcal{N}(\theta; \theta_{\text{MAP}}, (\nabla_{\theta}^2 \mathcal{L}(\theta)|_{\theta_{\text{MAP}}})^{-1})$



Daxberger et al., "Laplace Redux – Effortless Bayesian Deep Learning."

I. Efficient Matrix-Vector Multiplication

$$v \mapsto Av$$

Examples: deep learning, optimization, kernel methods, ...

II. Solution of Linear Systems

$$Ax = b$$

Examples: Gaussian density, linear regression, Newton's method, Galerkin method, Bayesian deep learning, ...

III. Matrix Decomposition

$$A = LU \quad A = U\Sigma V^* \quad A = QR$$

Examples: (kernel) PCA, GP inference, Kalman filtering, ...

IV. Computation of Log-Determinants and Matrix Traces

$$\log \det(A) \stackrel{A \text{ spd}}{=} \text{tr}(\log(A)) \quad \text{tr}(f(A))$$

Examples: Gaussian density, model selection, GP hyperparameter optimization, ...



Efficient Matrix-Vector Multiplication



Implementation Matters!

Often there is more than one way to implement a mathematical expression. A cautionary tale.

Goal: Given $A \in \mathbb{R}^{n \times k}$ and $x \in \mathbb{R}^n$ with $n = 100000$ and $k = 5$, compute

$$AA^T x.$$

```

1 def matvec(A, x):
2     return A @ A.T @ x
3
4
5 A.shape
6 # (100000, 5)
7
8 matvec(A, x)

```

Implementation Matters!

Often there is more than one way to implement a mathematical expression. A cautionary tale.

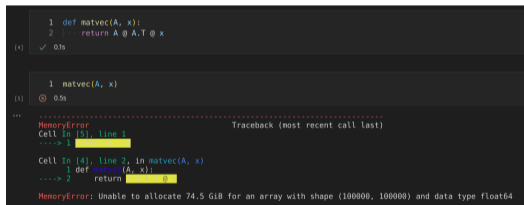
Goal: Given $A \in \mathbb{R}^{n \times k}$ and $x \in \mathbb{R}^n$ with $n = 100000$ and $k = 5$, compute

$$AA^T x.$$

```

1 def matvec(A, x):
2     return A @ A.T @ x
3
4
5 A.shape
6 # (100000, 5)
7
8 matvec(A, x)

```



```

1 def matvec(A, x):
2     return A @ A.T @ x
(4) ✓ 0.1s

1 matvec(A, x)
(5) Ⓜ 0.5s
...
MemoryError                                Traceback (most recent call last)
Cell In [5], line 1
----> 1 matvec(A, x)

Cell In [4], line 2, in matvec(A, x)
      1 def matvec(A, x):
----> 2     return A @ A.T @ x

MemoryError: Unable to allocate 74.5 GiB for an array with shape (100000, 100000) and data type float64

```

Implementation Matters!

Often there is more than one way to implement a mathematical expression. A cautionary tale.

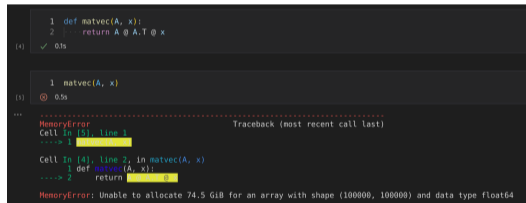
Goal: Given $A \in \mathbb{R}^{n \times k}$ and $x \in \mathbb{R}^n$ with $n = 100000$ and $k = 5$, compute

$$AA^T x.$$

```

1 def matvec(A, x):
2     return A @ A.T @ x
3
4
5 A.shape
6 # (100000, 5)
7
8 matvec(A, x)

```



```

1 def matvec(A, x):
2     return A @ A.T @ x
[4] ✓ 0.1s

1 matvec(A, x)
[5] ⓧ 0.5s
***
MemoryError                                Traceback (most recent call last)
Cell In [5], line 1
----> 1 matvec(A, x)

Cell In [4], line 2, in matvec(A, x)
----> 1 def matvec(A, x):
----> 2     return A @ A.T @ x

MemoryError: Unable to allocate 74.5 GiB for an array with shape (100000, 100000) and data type float64

```

Operator precedence: $(AA^T)x \implies n \times n$ matrix: $10^5 \cdot 10^5 \cdot 64 \text{ bits} = 80 \text{ gigabytes} = 74.5 \text{ GiB}$

Implementation Matters!

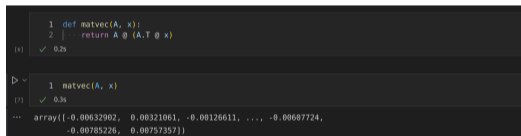
Often there is more than one way to implement a mathematical expression. A cautionary tale.

Goal: Given $A \in \mathbb{R}^{n \times k}$ and $x \in \mathbb{R}^n$ with $n = 100000$ and $k = 5$, compute

$$AA^T x.$$

```

1 def matvec(A, x):
2     return A @ (A.T @ x)
3
4
5 A.shape
6 # (100000, 5)
7
8 matvec(A, x)
    
```



```

1 def matvec(A, x):
2     return A @ (A.T @ x)
    
```

(6) ✓ 0.2s

```

1 matvec(A, x)
    
```

(7) ✓ 0.3s

```

array([-0.00632902, 0.00321061, -0.00126611, ..., -0.00607724,
       -0.00785226, 0.00757357])
    
```

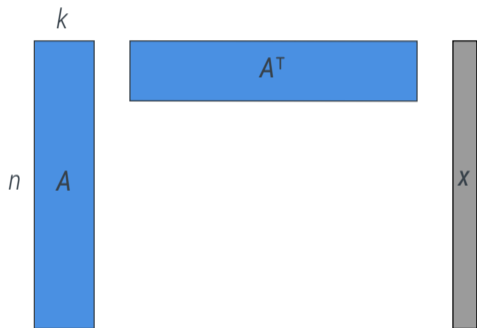
The algorithm implementing a mathematical operation can have a huge impact!

What's going on here?

Taking a closer look.

Goal: Given $A \in \mathbb{R}^{n \times k}$ and $x \in \mathbb{R}^n$ with $n = 100000$ and $k = 5$, compute

$AA^T x$.



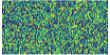




Operation	Time	Space
$(AA^T)x$	$\mathcal{O}(n^2k)$	$\mathcal{O}(n^2)$
$A(A^T x)$	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$

Exploiting structure in linear operations is fundamental to numerical linear algebra.

Efficient Matrix-Vector Multiplication

The importance of structure and lazy evaluation.

Goal: Evaluate $\mathbf{v} \mapsto \mathbf{A}\mathbf{v}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$.

Type	Expression	Time	Space	\mathbf{A}
None	\mathbf{A}	$\mathcal{O}(mn)$	$\mathcal{O}(mn)$	
Low Rank plus Diagonal	$\mathbf{A} = \mathbf{UV}^T + \mathbf{\Lambda}$	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	
Sparse	\mathbf{A}	$\mathcal{O}(\text{nnz}(\mathbf{A}))$	$\mathcal{O}(\text{nnz}(\mathbf{A}))$	
Kernel Matrix	$\mathbf{A} = k(\mathbf{X}, \mathbf{Z})$	$\mathcal{O}(mn)$	$\mathcal{O}((m+n)d)$	
Functional Form (e.g. fwd / rev mode autodiff)	$\mathbf{A} = f(\mathbf{A}_1, \dots, \mathbf{A}_\ell)$	often $o(mn)$	often $o(mn)$	

Running Example: Gaussian Process Regression



Gaussian Process Regression

An archetypical supervised machine learning model

Goal: Learn an unknown function $f_* : \mathbb{R}^d \rightarrow \mathbb{R}$.

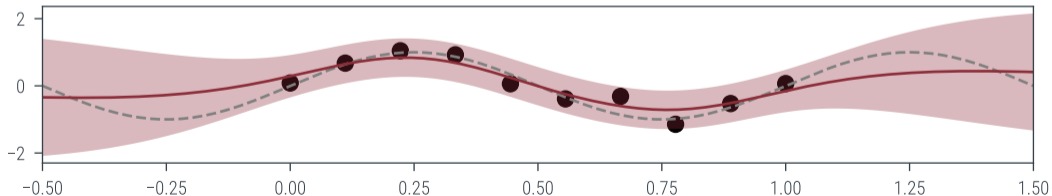
$$f \sim \mathcal{GP}(\mu, k)$$

$$y \mid f(X) \sim \mathcal{N}(f(X), \sigma^2 I)$$

$$f \mid X, y \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$\mu_{\text{post}}(x) = \mu(x) + k(x, X)(k(X, X) + \sigma^2 I)^{-1}(y - \mu(X))$$

$$k_{\text{post}}(x_0, x_1) = k(x_0, x_1) - k(x_0, X)(k(X, X) + \sigma^2 I)^{-1}k(X, x_1)$$



Model selection for Gaussian Processes

Finding the best kernel hyperparameters.

Model selection: Find kernel hyperparameters θ to maximize the log-marginal likelihood:

$$\begin{aligned}
 \theta_* &= \arg \max_{\theta} \mathcal{L}(\theta) \\
 &= \arg \max_{\theta} \log p(\mathbf{y} \mid \mathbf{X}, \theta) = \arg \max_{\theta} \log \int p(\mathbf{y} \mid f(\mathbf{X}) = \mathbf{z}, \theta) p(f(\mathbf{X}) = \mathbf{z} \mid \theta) d\mathbf{z} \\
 &= \arg \max_{\theta} \underbrace{-\frac{1}{2} (\mathbf{y} - \boldsymbol{\mu})^\top (k_{\theta}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})^{-1} (\mathbf{y} - \boldsymbol{\mu})}_{\text{model fit}} - \frac{1}{2} \underbrace{\log \det(k_{\theta}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})}_{\text{model complexity / Occam factor}}
 \end{aligned}$$

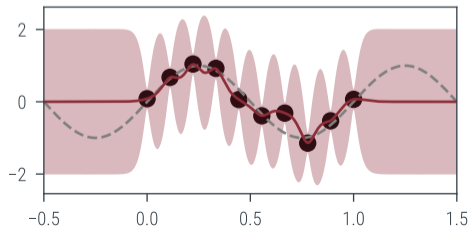
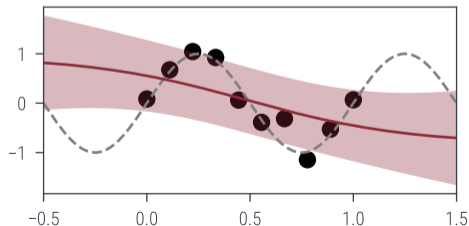


Model selection for Gaussian Processes

Finding the best kernel hyperparameters.

Model selection: Find kernel hyperparameters θ to maximize the log-marginal likelihood:

$$\begin{aligned}
 \theta_* &= \arg \max_{\theta} \mathcal{L}(\theta) \\
 &= \arg \max_{\theta} \log p(\mathbf{y} | \mathbf{X}, \theta) = \arg \max_{\theta} \log \int p(\mathbf{y} | f(\mathbf{X}) = \mathbf{z}, \theta) p(f(\mathbf{X}) = \mathbf{z} | \theta) d\mathbf{z} \\
 &= \arg \max_{\theta} \underbrace{-\frac{1}{2} (\mathbf{y} - \boldsymbol{\mu})^\top (k_{\theta}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})^{-1} (\mathbf{y} - \boldsymbol{\mu})}_{\text{model fit}} - \frac{1}{2} \underbrace{\log \det(k_{\theta}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})}_{\text{model complexity / Occam factor}}
 \end{aligned}$$



We need access to

- ▶ $v \mapsto (k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)^{-1} v$ (evaluated $m + 1$ times) and
- ▶ $\log \det(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)$.

The matrix $k(\mathbf{X}, \mathbf{X}) + \sigma^2 I$ is

- ▶ **large** ($n \times n$),
- ▶ **structured** (symmetric, positive definite), and
- ▶ **generative information** (the kernel function) about it is available.





Matrix Decompositions



Solution of Linear Systems via LU Decomposition

Forward and backward substitution.

Forward Substitution: Let $L_n = L \in \mathbb{R}^{n \times n}$ lower triangular, $\mathbf{y}_n = \mathbf{y} \in \mathbb{R}^n$, $\mathbf{b}_n = \mathbf{b} \in \mathbb{R}^n$ and partition recursively

$$L_i = \left(\begin{array}{c|c} L_{i-1} & \\ \hline I_{i-1}^\top & \lambda_i \end{array} \right), \quad \mathbf{y}_i = \begin{pmatrix} \mathbf{y}_{i-1} \\ \gamma_i \end{pmatrix}, \quad \mathbf{b}_i = \begin{pmatrix} \mathbf{b}_{i-1} \\ \beta_i \end{pmatrix}.$$

Setting $L_i \mathbf{y}_i = \mathbf{b}_i$ we obtain

$$I_{i-1}^\top \mathbf{y}_{i-1} + \lambda_i \gamma_i = \beta_i \iff \gamma_i = \frac{\beta_i - I_{i-1}^\top \mathbf{y}_{i-1}}{\lambda_i}$$

$$\gamma_1 = \frac{\beta_1}{\lambda_1}$$

Solution of Linear Systems via LU Decomposition

Forward and backward substitution.

Forward Substitution: Let $L_n = L \in \mathbb{R}^{n \times n}$ lower triangular, $\mathbf{y}_n = \mathbf{y} \in \mathbb{R}^n$, $\mathbf{b}_n = \mathbf{b} \in \mathbb{R}^n$ and partition recursively

$$L_i = \left(\begin{array}{c|c} L_{i-1} & \\ \hline I_{i-1}^\top & \lambda_i \end{array} \right), \quad \mathbf{y}_i = \begin{pmatrix} \mathbf{y}_{i-1} \\ \gamma_i \end{pmatrix}, \quad \mathbf{b}_i = \begin{pmatrix} \mathbf{b}_{i-1} \\ \beta_i \end{pmatrix}.$$

Setting $L_i \mathbf{y}_i = \mathbf{b}_i$ we obtain

$$I_{i-1}^\top \mathbf{y}_{i-1} + \lambda_i \gamma_i = \beta_i \iff \gamma_i = \frac{\beta_i - I_{i-1}^\top \mathbf{y}_{i-1}}{\lambda_i}$$

$$\gamma_1 = \frac{\beta_1}{\lambda_1}$$

Computational complexity: #flops $\simeq \sum_{i=1}^n \underbrace{2i}_{\text{Cost of } I_{i-1}^\top \mathbf{y}_{i-1}} \simeq n^2$

LU Decomposition: Let $A_1 = A \in \mathbb{R}^{n \times n}$, $L_1 = L \in \mathbb{R}^{n \times n}$, $U_1 = U \in \mathbb{R}^{n \times n}$ and partition recursively

$$A_i = \left(\begin{array}{c|c} \alpha_j & u_j^T \\ \hline b_i & B_i \end{array} \right), \quad L_i = \left(\begin{array}{c|c} 1 & \\ \hline l_j & L_{i+1} \end{array} \right), \quad U_i = \left(\begin{array}{c|c} \alpha_j & u_j^T \\ \hline & U_{i+1} \end{array} \right),$$

such that always $A_i = L_i U_i$, and therefore

$$l_j = \frac{1}{\alpha_j} b_i$$
$$A_{i+1} := L_{i+1} U_{i+1} = B_i - l_j u_j^T$$

LU Decomposition: Let $A_1 = A \in \mathbb{R}^{n \times n}$, $L_1 = L \in \mathbb{R}^{n \times n}$, $U_1 = U \in \mathbb{R}^{n \times n}$ and partition recursively

$$A_i = \left(\begin{array}{c|c} \alpha_i & u_i^T \\ \hline b_i & B_i \end{array} \right), \quad L_i = \left(\begin{array}{c|c} 1 & \\ \hline l_i & L_{i+1} \end{array} \right), \quad U_i = \left(\begin{array}{c|c} \alpha_i & u_i^T \\ \hline & U_{i+1} \end{array} \right),$$

such that always $A_i = L_i U_i$, and therefore

$$l_i = \frac{1}{\alpha_i} b_i$$
$$A_{i+1} := L_{i+1} U_{i+1} = B_i - l_i u_i^T$$

If all diagonal elements α_i , so called *pivots*, are non-zero, the recursion terminates and an LU decomposition of $A \in GL(n)$ exists. \implies choose element with largest abs. value per column as pivot.

Solution of Linear Systems via LU Decomposition

LU decomposition / Gaussian elimination.

LU Decomposition: Let $A_1 = A \in \mathbb{R}^{n \times n}$, $L_1 = L \in \mathbb{R}^{n \times n}$, $U_1 = U \in \mathbb{R}^{n \times n}$ and partition recursively

$$A_i = \left(\begin{array}{c|c} \alpha_i & \mathbf{u}_i^T \\ \mathbf{b}_i & B_i \end{array} \right), \quad L_i = \left(\begin{array}{c|c} 1 & \\ \mathbf{l}_i & L_{i+1} \end{array} \right), \quad U_i = \left(\begin{array}{c|c} \alpha_i & \mathbf{u}_i^T \\ \mathbf{u}_{i+1} & U_{i+1} \end{array} \right),$$

such that always $A_i = L_i U_i$, and therefore

$$l_i = \frac{1}{\alpha_i} \mathbf{b}_i$$

$$A_{i+1} := L_{i+1} U_{i+1} = B_i - l_i \mathbf{u}_i^T$$

If all diagonal elements α_i , so called *pivots*, are non-zero, the recursion terminates and an LU decomposition of $A \in GL(n)$ exists. \implies choose element with largest abs. value per column as pivot.

Computational complexity: #flops $\simeq \sum_{i=1}^n \underbrace{2(n-i)^2}_{\text{Cost of } A_{i+1}} = 2 \sum_{i=1}^{n-1} i^2 \simeq \frac{2}{3} n^3$

Solution of Linear Systems via LU Decomposition

Amortizing computational cost by reusing a known LU decomposition.

How do we solve a linear system given an LU decomposition? Decompose into two systems.

$$LUx = b \iff L(Ux) = b \iff Ly = b \wedge Ux = y$$

Then solve $Ly = b$ by forward substitution and $Ux = y$ by backward substitution.

Solution of Linear Systems via LU Decomposition

Amortizing computational cost by reusing a known LU decomposition.

How do we solve a linear system given an LU decomposition? Decompose into two systems.

$$LUx = b \iff L(Ux) = b \iff Ly = b \wedge Ux = y$$

Then solve $Ly = b$ by forward substitution and $Ux = y$ by backward substitution.

How much do k solves with the same system matrix A cost?

Solution of Linear Systems via LU Decomposition

Amortizing computational cost by reusing a known LU decomposition.

How do we solve a linear system given an LU decomposition? Decompose into two systems.

$$LUx = b \iff L(Ux) = b \iff Ly = b \wedge Ux = y$$

Then solve $Ly = b$ by forward substitution and $Ux = y$ by backward substitution.

How much do k solves with the same system matrix A cost?

Computational complexity: #flops $\simeq \frac{2}{3}n^3 + 2kn^2$

Given an LU decomposition $A = LU$, it holds that

$$\begin{aligned}\det(\mathbf{A}) &= \det(\mathbf{LU}) \\ &= \det(\mathbf{L}) \det(\mathbf{U}) \\ &= \prod_{i=1}^n L_{ii} \cdot \prod_{i=1}^n U_{ii}\end{aligned}$$

Computational complexity: #flops $\simeq 2n$

The Cholesky Decomposition

Hang on... My kernel Gram matrix is structured!

Assume $\mathbf{A} \in \mathbb{R}^{n \times n}$ is *symmetric positive definite* and partition $\mathbf{A}_1 := \mathbf{A}$ as follows

$$\mathbf{A}_i = \left(\begin{array}{c|c} \alpha_i & \mathbf{b}_i^\top \\ \hline \mathbf{b}_i & \mathbf{B}_i \end{array} \right), \quad \mathbf{L}_i = \left(\begin{array}{c|c} \lambda_i & \\ \hline \mathbf{l}_i & \mathbf{L}_{i+1} \end{array} \right), \quad \mathbf{L}_i^\top = \left(\begin{array}{c|c} \lambda_i & \mathbf{l}_i^\top \\ \hline & \mathbf{L}_{i+1} \end{array} \right),$$

such that always $\mathbf{A}_i = \mathbf{L}_i \mathbf{L}_i^\top$, and therefore

$$\lambda_i = \sqrt{\alpha_i} \quad \alpha_i > 0 \quad \text{by pos. def. assumption}$$

$$\mathbf{l}_i = \frac{1}{\lambda_i} \mathbf{b}_i$$

$$\mathbf{A}_{i+1} := \mathbf{L}_{i+1} \mathbf{L}_{i+1}^\top = \mathbf{B}_i - \mathbf{l}_i \mathbf{l}_i^\top$$

Theorem (Cholesky Decomposition)

Every spd matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ can be uniquely represented in the form $\mathbf{A} = \mathbf{L} \mathbf{L}^\top$, where \mathbf{L} is lower triangular with a positive diagonal.

$$f \mid \mathbf{X}, \mathbf{y} \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$\begin{aligned}\mu_{\text{post}}(\mathbf{x}) &= \mu(\mathbf{x}) + k(\mathbf{x}, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})^{-1}(\mathbf{y} - \mu(\mathbf{X})) \\ k_{\text{post}}(\mathbf{x}_0, \mathbf{x}_1) &= k(\mathbf{x}_0, \mathbf{x}_1) - k(\mathbf{x}_0, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})^{-1}k(\mathbf{X}, \mathbf{x}_1)\end{aligned}$$

GP Regression via Cholesky Decomposition

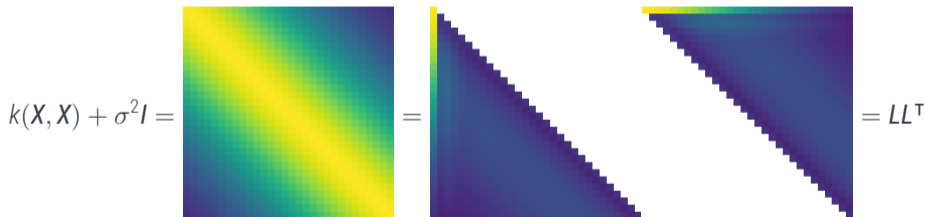
Putting it all together.

$$f | \mathbf{X}, \mathbf{y} \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$L = \text{CHOLESKY}(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)$$

$$\mu_{\text{post}}(\mathbf{x}) = \mu(\mathbf{x}) + k(\mathbf{x}, \mathbf{X})(L^\top)^{-1}L^{-1}(\mathbf{y} - \mu(\mathbf{X}))$$

$$k_{\text{post}}(\mathbf{x}_0, \mathbf{x}_1) = k(\mathbf{x}_0, \mathbf{x}_1) - k(\mathbf{x}_0, \mathbf{X})(L^\top)^{-1}L^{-1}k(\mathbf{X}, \mathbf{x}_1)$$





GP Regression via Cholesky Decomposition

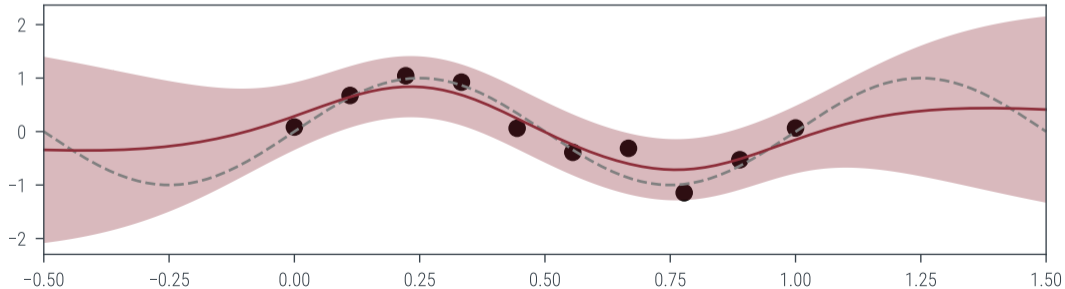
Putting it all together.

$$f | \mathbf{X}, \mathbf{y} \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$L = \text{CHOLESKY}(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)$$

$$\mu_{\text{post}}(\mathbf{x}) = \mu(\mathbf{x}) + k(\mathbf{x}, \mathbf{X})(L^T)^{-1}L^{-1}(\mathbf{y} - \mu(\mathbf{X}))$$

$$k_{\text{post}}(\mathbf{x}_0, \mathbf{x}_1) = k(\mathbf{x}_0, \mathbf{x}_1) - k(\mathbf{x}_0, \mathbf{X})(L^T)^{-1}L^{-1}k(\mathbf{X}, \mathbf{x}_1)$$



GP Regression via Cholesky Decomposition

Putting it all together.

$$f \mid \mathbf{X}, \mathbf{y} \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$L = \text{CHOLESKY}(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)$$

$$\mu_{\text{post}}(\mathbf{x}) = \mu(\mathbf{x}) + k(\mathbf{x}, \mathbf{X})(L^\top)^{-1}L^{-1}(\mathbf{y} - \mu(\mathbf{X}))$$

$$k_{\text{post}}(\mathbf{x}_0, \mathbf{x}_1) = k(\mathbf{x}_0, \mathbf{x}_1) - k(\mathbf{x}_0, \mathbf{X})(L^\top)^{-1}L^{-1}k(\mathbf{X}, \mathbf{x}_1)$$

$$\log \det(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I) = \log \left(\prod_{i=1}^n L_{ii} \cdot \prod_{i=1}^n L_{ii}^\top \right) = 2 \sum_{i=1}^n \log(L_{ii})$$

Baby Steps Towards Scalable Gaussian Process Regression

$$\mathbf{X} \in \mathbb{R}^{100000 \times d} \quad \Rightarrow \quad k(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{100000 \times 100000}$$

Oh oh...



Idea: Choose $m(\mathbf{x}) = \phi_{\mathbf{x}}^{\top} \boldsymbol{\mu}$ and $k(\mathbf{x}_1, \mathbf{x}_2) = \phi_{\mathbf{x}_1}^{\top} \boldsymbol{\Sigma} \phi_{\mathbf{x}_2}$ for some feature function $\phi_{\mathbf{x}} \in \mathbb{R}^p$, where $\Phi_{\mathcal{X}} \in \mathbb{R}^{n \times p}$ are the training set features.

$$\text{prior} \quad \mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \Rightarrow \quad f = \phi_{(\cdot)}^{\top} \mathbf{w} \sim \mathcal{GP}(m, k)$$

$$\text{likelihood} \quad \mathbf{y} \mid \mathbf{w}, \Phi_{\mathcal{X}} \sim \mathcal{N}(\Phi_{\mathcal{X}} \mathbf{w}, \sigma^2 \mathbf{I}) \quad \Rightarrow \quad \mathbf{y} \mid f(\mathcal{X}) \sim \mathcal{N}(f(\mathcal{X}), \sigma^2 \mathbf{I})$$

Idea: Choose $m(\mathbf{x}) = \phi_{\mathbf{x}}^{\top} \boldsymbol{\mu}$ and $k(\mathbf{x}_1, \mathbf{x}_2) = \phi_{\mathbf{x}_1}^{\top} \boldsymbol{\Sigma} \phi_{\mathbf{x}_2}$ for some feature function $\phi_{\mathbf{x}} \in \mathbb{R}^p$, where $\Phi_{\mathbf{X}} \in \mathbb{R}^{n \times p}$ are the training set features.

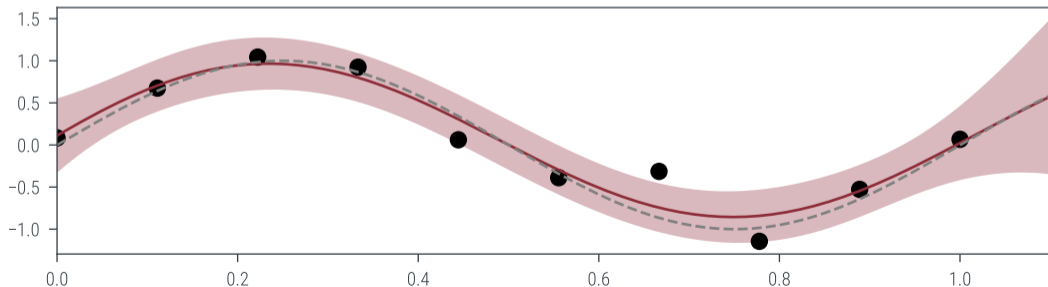
$$\begin{aligned} \text{prior} \quad \mathbf{w} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \Rightarrow \quad f = \phi_{(\cdot)}^{\top} \mathbf{w} \sim \mathcal{GP}(m, k) \\ \text{likelihood} \quad \mathbf{y} \mid \mathbf{w}, \Phi_{\mathbf{X}} &\sim \mathcal{N}(\Phi_{\mathbf{X}} \mathbf{w}, \sigma^2 \mathbf{I}) \quad \Rightarrow \quad \mathbf{y} \mid f(\mathbf{X}) \sim \mathcal{N}(f(\mathbf{X}), \sigma^2 \mathbf{I}) \\ \text{posterior} \quad \mathbf{w} \mid \mathbf{y}, \Phi_{\mathbf{X}} &\sim \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\Sigma} \Phi_{\mathbf{X}}^{\top} (\Phi_{\mathbf{X}} \boldsymbol{\Sigma} \Phi_{\mathbf{X}}^{\top} + \sigma^2 \mathbf{I})^{-1} (\mathbf{y} - \Phi_{\mathbf{X}} \boldsymbol{\mu}), \\ &\quad \boldsymbol{\Sigma} - \boldsymbol{\Sigma} \Phi_{\mathbf{X}}^{\top} (\Phi_{\mathbf{X}} \boldsymbol{\Sigma} \Phi_{\mathbf{X}}^{\top} + \sigma^2 \mathbf{I})^{-1} \Phi_{\mathbf{X}} \boldsymbol{\Sigma}) \\ f \mid \mathbf{X}, \mathbf{y} &\sim \mathcal{GP}(m(\cdot) + k(\cdot, \mathbf{X}) (\Phi_{\mathbf{X}} \boldsymbol{\Sigma} \Phi_{\mathbf{X}}^{\top} + \sigma^2 \mathbf{I})^{-1} (\mathbf{y} - m(\mathbf{X})), \\ &\quad k(\cdot, \cdot) - k(\cdot, \mathbf{X}) (\Phi_{\mathbf{X}} \boldsymbol{\Sigma} \Phi_{\mathbf{X}}^{\top} + \sigma^2 \mathbf{I})^{-1} k(\mathbf{X}, \cdot)) \end{aligned}$$

Scalable GP Regression by Exploiting Kernel Structure

Parametric Gaussian Process Priors

Idea: Choose $m(\mathbf{x}) = \phi_{\mathbf{x}}^T \boldsymbol{\mu}$ and $k(\mathbf{x}_1, \mathbf{x}_2) = \phi_{\mathbf{x}_1}^T \boldsymbol{\Sigma} \phi_{\mathbf{x}_2}$ for some feature function $\phi_{\mathbf{x}} \in \mathbb{R}^p$, where $\Phi_X \in \mathbb{R}^{n \times p}$ are the training set features.

$$f | X, y \sim \mathcal{GP}(m(\cdot) + k(\cdot, X)(\Phi_X \boldsymbol{\Sigma} \Phi_X^T + \sigma^2 I)^{-1}(y - m(X)), \\ k(\cdot, \cdot) - k(\cdot, X)(\Phi_X \boldsymbol{\Sigma} \Phi_X^T + \sigma^2 I)^{-1}k(X, \cdot))$$



Solution of Linear Systems by Exploiting Structure

The matrix inversion lemma.

$$(\Phi_X \Sigma \Phi_X^T + \sigma^2 I)^{-1} \in \mathbb{R}^{n \times n} \quad \Phi_X \in \mathbb{R}^{n \times p} \quad \Sigma \in \mathbb{R}^{p \times p}$$

Lemma (Matrix Inversion Lemma)

$$(UCV^T + A)^{-1} = A^{-1} - A^{-1}U(C^{-1} + V^T A^{-1}U)^{-1}V^T A^{-1}$$

Computational complexity: $\mathcal{O}(\underbrace{\text{inv}(A)}_{\text{often known}} + \underbrace{\text{inv}(C^{-1} + V^T A^{-1}U)}_{\text{often cheap, e.g. } UCV^T \text{ low rank}})$

Inversion of “low-rank plus diagonal” matrix in $\mathcal{O}(np^2)$ instead of $\mathcal{O}(n^3)$.

$$(\Phi_X \Sigma \Phi_X^T + \sigma^2 I)^{-1} \in \mathbb{R}^{n \times n} \quad \Phi_X \in \mathbb{R}^{n \times p} \quad \Sigma \in \mathbb{R}^{p \times p}$$

Lemma (Matrix Inversion Lemma)

$$(UCV^T + A)^{-1} = A^{-1} - A^{-1}U(C^{-1} + V^T A^{-1}U)^{-1}V^T A^{-1}$$

Lemma (Matrix Determinant Lemma)

$$\det(UCV^T + A) = \det(A) \det(C) \det(C^{-1} + V^T A^{-1}U)$$

Determinant of "low-rank plus diagonal" matrix in $\mathcal{O}(np^2)$ instead of $\mathcal{O}(n^3)$.

Teaser: Scalable GP Regression by Approximate Inversion

The Cholesky decomposition processes the data points in the given order.

Algorithm 1 Cholesky Decomposition

Input: spd matrix A

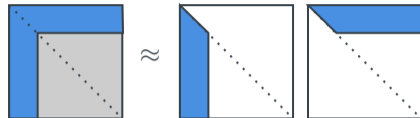
Output: lower triangular L_i , s.t. $L_i L_i^T = A$

```

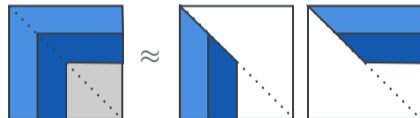
1 procedure CHOLESKY( $A$ )
2    $A' \leftarrow A$ 
3   for  $i \in \{1, \dots, n\}$  do
4      $l_i \leftarrow A'_{:i} / \sqrt{A'_{ii}} = A'(e_i / \|e_i\|_{A'})$ 
5      $A' \leftarrow A' - l_i l_i^T = A - L_i L_i^T$ 
6      $L_i = (L_{i-1} \quad l_i)$ 
7   end for
8   return  $L_i$ 
9 end procedure

```

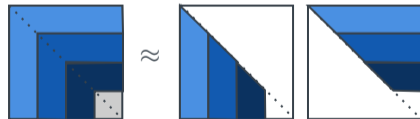
$i = 1$



$i = 2$



$i = 3$



A

L_i

L_i^T

Cholesky decomposition can also be seen as a low-rank approximation method.

Teaser: Scalable GP Regression by Approximate Inversion

The Cholesky decomposition processes the data points in the given order.

$$k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I} = \left(\begin{array}{c} \text{Heatmap 1} \end{array} \right) \approx \left(\begin{array}{c} \text{Heatmap 2} \end{array} \right) = \mathbf{L}_1 \mathbf{L}_1^T$$

The diagram illustrates the Cholesky decomposition of a kernel matrix. On the left, the matrix $k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}$ is shown as a heatmap with a diagonal band of high values (yellow) and a smooth gradient. This matrix is approximately equal to the product of a lower triangular matrix \mathbf{L}_1 and its transpose \mathbf{L}_1^T . The matrix \mathbf{L}_1 is shown as a heatmap where the diagonal elements are significantly larger than the off-diagonal elements, indicating a banded structure.

Teaser: Scalable GP Regression by Approximate Inversion

The Cholesky decomposition processes the data points in the given order.

$$k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I} = \left(\begin{array}{c} \text{Heatmap 1} \end{array} \right) \approx \left(\begin{array}{c} \text{Heatmap 2} \end{array} \right) = \mathbf{L}_5 \mathbf{L}_5^T$$

The image shows two heatmaps representing matrices. The first heatmap, labeled $k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}$, is a symmetric matrix with a color gradient from dark purple (low values) to bright yellow (high values), showing a smooth, diagonal-like pattern. The second heatmap, labeled \approx , is a lower triangular matrix with a similar color gradient, but with a distinct yellow-to-green diagonal band, indicating the Cholesky decomposition. The equation concludes with $= \mathbf{L}_5 \mathbf{L}_5^T$.

Teaser: Scalable GP Regression by Approximate Inversion

The Cholesky decomposition processes the data points in the given order.

$$k(\mathbf{X}, \mathbf{X}) + \sigma^2 I = \left(\begin{array}{c} \text{Heatmap 1} \\ \end{array} \right) \approx \left(\begin{array}{c} \text{Heatmap 2} \\ \end{array} \right) = L_{10} L_{10}^T$$

The diagram illustrates the Cholesky decomposition of the kernel matrix $k(\mathbf{X}, \mathbf{X}) + \sigma^2 I$. The matrix is shown as a heatmap with a color gradient from dark blue (low values) to yellow (high values). The matrix is symmetric and positive definite. The Cholesky decomposition is shown as a lower triangular matrix L_{10} , which is also represented as a heatmap. The matrix L_{10} is lower triangular, with the diagonal elements being the highest (yellow) and the off-diagonal elements being lower (green/blue). The decomposition is shown as $k(\mathbf{X}, \mathbf{X}) + \sigma^2 I \approx L_{10} L_{10}^T$.

Teaser: Scalable GP Regression by Approximate Inversion

The Cholesky decomposition processes the data points in the given order.

$$k(\mathbf{X}, \mathbf{X}) + \sigma^2 I = \left(\begin{array}{c} \text{Heatmap matrix} \end{array} \right) \approx \left(\begin{array}{c} \text{Heatmap matrix with diagonal} \end{array} \right) = L_{15} L_{15}^T$$

The image shows the Cholesky decomposition of a kernel matrix. On the left, the matrix $k(\mathbf{X}, \mathbf{X}) + \sigma^2 I$ is represented as a heatmap with a smooth, symmetric gradient from dark blue (low values) to yellow (high values). This matrix is approximately equal to the product of a lower triangular matrix L_{15} and its transpose L_{15}^T . The matrix L_{15} is also shown as a heatmap, where the diagonal elements are significantly brighter (yellow) than the off-diagonal elements, indicating that the Cholesky decomposition has been performed in a way that processes data points in a specific order.

Teaser: Scalable GP Regression by Approximate Inversion

The Cholesky decomposition processes the data points in the given order.

$$k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I} = \left(\begin{array}{c} \text{Heatmap matrix} \end{array} \right) \approx \left(\begin{array}{c} \text{Heatmap matrix with diagonal} \end{array} \right) = \mathbf{L}_{20} \mathbf{L}_{20}^T$$

The diagram illustrates the Cholesky decomposition of a kernel matrix. On the left, the matrix $k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}$ is shown as a heatmap with a smooth, symmetric pattern. This matrix is approximately equal to the product of a lower triangular matrix \mathbf{L}_{20} and its transpose \mathbf{L}_{20}^T . The matrix \mathbf{L}_{20} is also shown as a heatmap, where the diagonal elements are significantly brighter than the off-diagonal elements, indicating its lower triangular structure.

Teaser: Scalable GP Regression by Approximate Inversion

Changing the order of the data points drastically increases the low-rank approximation.

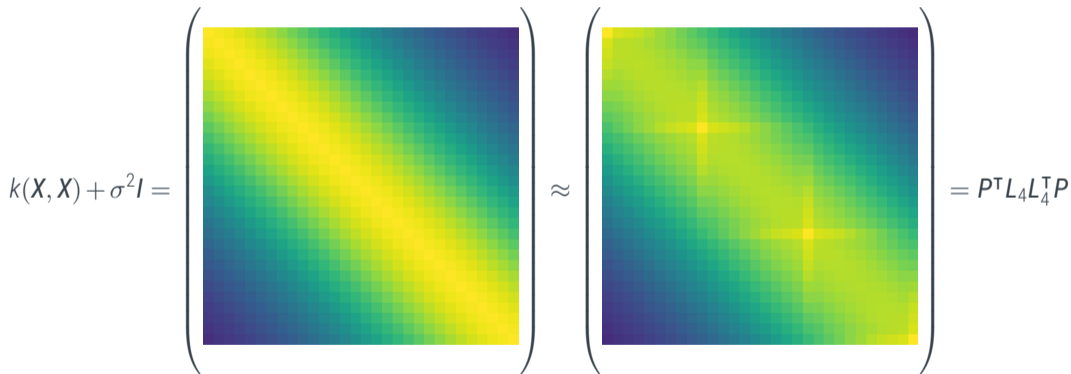
$$P(k(X, X) + \sigma^2 I)P^T = LL^T$$

Teaser: Scalable GP Regression by Approximate Inversion



Changing the order of the data points drastically increases the low-rank approximation.

$$P(k(X, X) + \sigma^2 I)P^T = LL^T$$



Summary

- ▶ Numerical linear algebra (NLA) is fundamental to ML.
- ▶ Why numerics? The algorithm implementing a mathematical operation matters a lot for performance!
- ▶ NLA for ML largely deals with four tasks:
 1. Efficient Matrix-Vector Multiplication
 2. Solution of Linear Systems
 3. Matrix Decomposition
 4. Computation of Log-Determinants and Matrix Traces
- ▶ Most linear algebra problems in machine learning are inherently structured.
- ▶ Structure can be leveraged for efficiency.

Please cite this course, as

```
@techreport{NoML22,
  title = {Numerics of Machine Learning},
  author = {N. Bosch and J. Grosse
    and P. Hennig and A. Kristiadi
    and M. Pförtner and J. Schmidt
    and F. Schneider and L. Tatzel
    and J. Wenger},
  series = {Lecture Notes in Machine Learning},
  year = {2022},
  institution = {Tübingen AI Center},
}
```

