Approximating Gaussian Processes with $\mathcal{H}^2$-matrices

Steffen Börm[1] Jochen Garcke[2]

[1]Max Planck Institute for Mathematics in the Sciences

MAX-PLANCK-GESELLSCHAFT

[2]Technische Universität Berlin and Matheon

Outline

Regression Problem Setup

- consider a given set of data (the training set)

$$S = \{(\underline{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^N$$

- $\underline{x}_i$ data points in feature space
- y_i associated response variable
- data obtained by sampling a function f with additional independent Gaussian noise e_i of variance σ^2, i.e., $y_i = f(\underline{x}_i) + e_i$
- recover function f from given data as well as possible

Gaussian Processes

- we assume a Gaussian process prior on $f(\underline{x})$,
- meaning that values $f(\underline{x})$ on points $\{\underline{x}_i\}_{i=1}^N$ are jointly Gaussian distributed with zero mean and covariance matrix $\mathcal{K}$
- kernel (or covariance) function $k(\cdot,\cdot)$ defines $\mathcal{K}$ via $\mathcal{K}_{i,j} = k(\underline{x}_i, \underline{x}_j)$.
- typical kernel: Gaussian RBF $k(\underline{x},\underline{y}) = e^{-\|\underline{x}-\underline{y}\|^2/w}$
- representer theorem:
  solution $f(\underline{x})$ is weighted combination of kernel functions on training points $\underline{x}_i$

$$f(\underline{x}) = \sum_{i=1}^{N} \alpha_i k(\underline{x}_i, \underline{x}),$$

- minimised least squares error on data points

Computing Gaussian Processes

- representer theorem:

$$f(\underline{x}) = \sum_{i=1}^{N} \alpha_i k(\underline{x}_i, \underline{x}),$$

- coefficient vector α is the solution of the linear equation system

$$(\mathcal{K} + \sigma^2 \mathcal{I})\alpha = y,$$

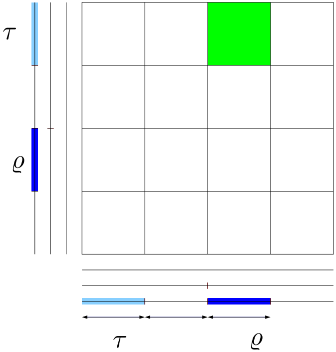
  ($\mathcal{I}$ denotes the unit matrix)
- full $N \times N$ matrix $\rightarrow \mathcal{O}(N^2)$ complexity, unfeasible for large data
- approximation needed
  - use subset in computational core $\rightarrow \mathcal{O}(M^2 \cdot N)$, see [Rasmussen.Williams:06] and [Quinonero-Candela.Rasmussen:05]
  - use iterative solver with approximation of matrix vector product $\mathcal{K}\alpha$ (references in the paper)

Hierarchical matrices

- data sparse approximation of kernel matrix
- $\mathcal{O}(Nm \log N)$ for storage, (local rank) m controls accuracy
- operations like matrix-vector product, matrix multiplication or inversion can now be computed efficiently
- efficient computation of $\mathcal{H}$-matrix approximation needed
- $\mathcal{H}$-matrix approach developed for efficient treatment of dense matrix arising from discretization of integral operators
- efficient computation for 2D, 3D problems exists
- strongly related to fast multipole, panel clustering, fast gauss transform

1D Model Problem

- in the following we present the underlying ideas in one dimension



we look at blocks in the (permuted) matrix whose corresponding subregions have a certain 1D-distance

- employ Taylor-expansion to approximate kernel
- note: Taylor-expansion only used for explanation, but not in algorithm

Panel clustering

Degenerate approximation: If k is sufficiently smooth in a subdomain $\tau \times \varrho$, we can approximate by a Taylor series:

$$\tilde{k}(x, y) := \sum_{\nu=0}^{m-1} \frac{(x - x_\tau)^\nu}{\nu!} \frac{\partial^\nu k}{\partial x^\nu}(x_\tau, y) \qquad (x \in \tau, y \in \varrho)$$

Factorization: For $i, j \in \mathcal{I}$ with $x_i \in \tau$ and $x_j \in \varrho$ we find

$$\begin{aligned} \mathcal{K}_{ij} = k(x_i, x_j) \approx \tilde{k}(x_i, x_j) &= \sum_{\nu=0}^{m-1} \underbrace{\frac{(x_i - x_\tau)^\nu}{\nu!}}_{=(A_{\tau,\varrho})_{i\nu}} \underbrace{\frac{\partial^\nu k}{\partial x^\nu}(x_\tau, x_j)}_{=(B_{\tau,\varrho})_{j\nu}} \\ &= \sum_{\nu=0}^{m-1} (A_{\tau,\varrho})_{i\nu} (B_{\tau,\varrho})_{j\nu} \end{aligned}$$

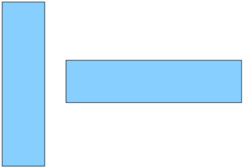
Panel clustering

Degenerate approximation: If k is sufficiently smooth in a subdomain $\tau \times \varrho$, we can approximate by a Taylor series:

$$\tilde{k}(x,y) := \sum_{\nu=0}^{m-1} \frac{(x - x_\tau)^\nu}{\nu!} \frac{\partial^\nu k}{\partial x^\nu}(x_\tau, y) \qquad (x \in \tau, y \in \varrho)$$

Factorization: For the sets $\hat{\tau} := \{i \; : \; x_i \in \tau\}$, $\hat{\varrho} := \{j \; : \; x_j \in \varrho\}$ we find

$$\mathcal{K}|_{\hat{\tau}\times\hat{\varrho}} \approx A_{\tau,\varrho} B_{\tau,\varrho}^\top$$



Storage $m(\#\hat{\tau} + \#\hat{\varrho})$ instead of $(\#\hat{\tau})(\#\hat{\varrho})$.

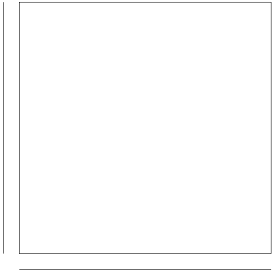
Result: Significant reduction of storage requirements if $m \ll \#\hat{\tau}, \#\hat{\varrho}$.

Cluster tree and block partition

Goal: Split $\Omega \times \Omega$ into subdomains satisfying the admissibility condition

$$\operatorname{diam}(\tau) \leq 2 \operatorname{dist}(\tau, \varrho)$$

(≤ 2 for demonstration purposes)

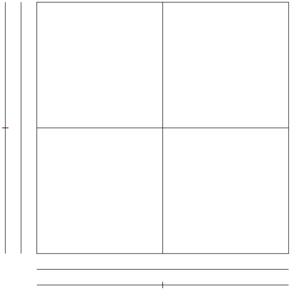


Start with $\tau = \varrho = \Omega$.
Nothing is admissible.

Cluster tree and block partition

Goal: Split $\Omega \times \Omega$ into subdomains satisfying the admissibility condition

$$\operatorname{diam}(\tau) \leq 2 \operatorname{dist}(\tau, \varrho)$$

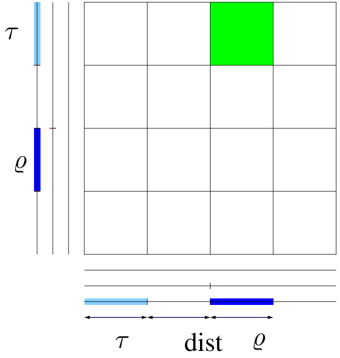


τ and ϱ are subdivided.
Still nothing is admissible.

Cluster tree and block partition

Goal: Split $\Omega \times \Omega$ into subdomains satisfying the admissibility condition

$$\operatorname{diam}(\tau) \leq 2 \operatorname{dist}(\tau, \varrho)$$

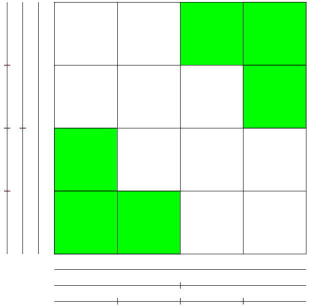


We split the intervals again.
And find an admissible block.

Cluster tree and block partition

Goal: Split $\Omega \times \Omega$ into subdomains satisfying the admissibility condition

$$\operatorname{diam}(\tau) \leq 2 \operatorname{dist}(\tau, \varrho)$$

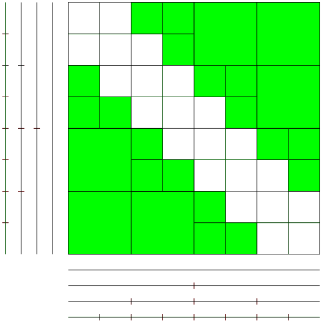


We find six admissible blocks on this level.

Cluster tree and block partition

Goal: Split $\Omega \times \Omega$ into subdomains satisfying the admissibility condition

$$\operatorname{diam}(\tau) \leq 2 \operatorname{dist}(\tau, \varrho)$$



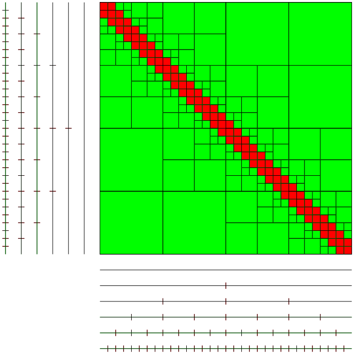
The procedure is repeated...

Cluster tree and block partition

Goal: Split $\Omega \times \Omega$ into subdomains satisfying the admissibility condition

$$\operatorname{diam}(\tau) \leq 2 \operatorname{dist}(\tau, \varrho)$$

(up to a small remainder).



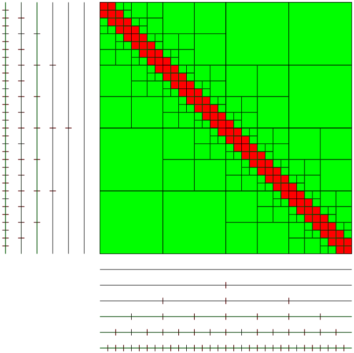
The procedure is repeated until only a small subdomain remains.

Result: Domain $\Omega \times \Omega$ partitioned into blocks $\tau \times \varrho$.

Clusters $\tau, \varrho \subseteq \Omega$ organized in a cluster tree.

Hierarchical matrix

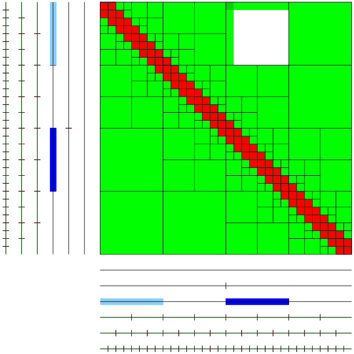
Idea: Use low-rank approximation in all admissible blocks $\hat{\tau} \times \hat{\varrho}$.



Standard representation of original matrix $\mathcal{K}$ requires N^2 units of storage.

Hierarchical matrix

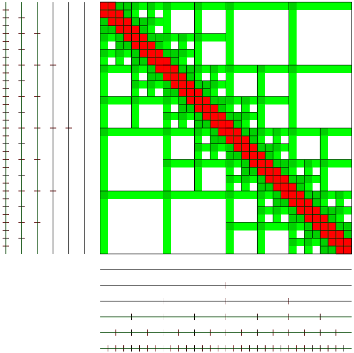
Idea: Use low-rank approximation in all admissible blocks $\hat{\tau} \times \hat{\varrho}$.



Replace admissible block $\mathcal{K}|_{\hat{\tau}\times\hat{\varrho}}$ by low-rank approximation $\widetilde{\mathcal{K}}|_{\hat{\tau}\times\hat{\varrho}} = A_{\tau,\varrho} B_{\tau,\varrho}^{\top}$.

Hierarchical matrix

Idea: Use low-rank approximation in all admissible blocks $\hat{\tau} \times \hat{\varrho}$.



Replace all admissible blocks by low-rank approximations, leave inadmissible blocks unchanged.

Result: Hierarchical matrix approximation $\widetilde{\mathcal{K}}$ of $\mathcal{K}$.

Storage requirements: One row of $\widetilde{\mathcal{K}}$ represented by only $\mathcal{O}(m \log N)$ units of storage, total storage requirements $\mathcal{O}(Nm \log N)$.

Second approach: Cross approximation

Observation: If M is a rank 1 matrix and we have pivot indices i^*, j^* with $M_{i^* j^*} \neq 0$, we get the representation

$$M = ab^\top, \qquad a_i := M_{ij^*} / M_{i^* j^*}, \qquad b_j := M_{i^* j}.$$

Idea: If M can be approximated by a rank 1 matrix, we still can find i^*, j^* with $M_{i^* j^*} \neq 0$ and $M \approx ab^\top$.

Higher rank: Repeating the procedure for the error matrix yields rank m approximation of arbitrary accuracy.

Efficient: If the pivot indices are known, only m rows and columns of M are required to construct a rank m approximation.

Problem: Selection of pivot indices.
Efficient strategies needed.
Provable in certain settings.
For our case it works (but till did not work on a proof)

Uniform hierarchical matrix

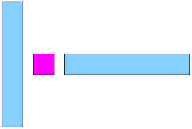
Goal: Reduce the storage requirements.

Approach: Expansion in both variables

$$k(x,y) \approx \sum_{\nu+\mu<m} \frac{\partial^{\nu+\mu} k}{\partial x^\nu \partial y^\mu}(x_\tau, y_\varrho) \frac{(x-x_\tau)^\nu}{\nu!} \frac{(y-y_\varrho)^\mu}{\mu!}$$

yields low-rank factorization

$$\mathcal{K}|_{\hat\tau\times\hat\varrho} \approx V_\tau S_{\tau,\varrho} V_\varrho^\top,$$

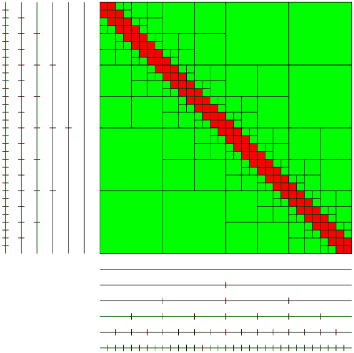


$$(V_\tau)_{i\nu} := \frac{(x_i - x_\tau)^\nu}{\nu!}\, dx, \qquad (S_{\tau,\varrho})_{\nu\mu} := \frac{\partial^{\nu+\mu} k}{\partial x^\nu \partial y^\mu}(x_\tau, y_\varrho).$$

Important: V_τ depends only on one cluster (τ).
Only the small matrix $S_{\tau,\varrho} \in \mathbb{R}^{m\times m}$ depends on both clusters.

$\mathcal{H}^2$-matrix

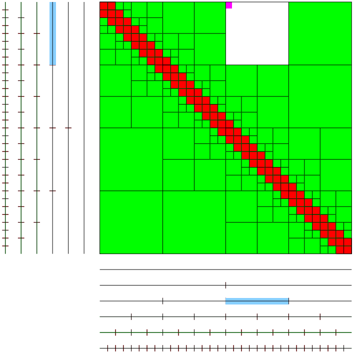
Idea: Use three-term factorization in all admissible blocks $\hat{\tau} \times \hat{\varrho}$.



Standard representation of original matrix $\mathcal{K}$ requires N^2 units of storage.

$\mathcal{H}^2$-matrix

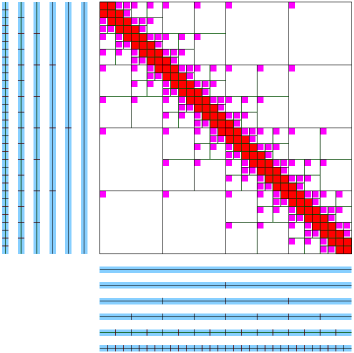
Idea: Use three-term factorization in all admissible blocks $\hat{\tau} \times \hat{\varrho}$.



Replace admissible block $\mathcal{K}|_{\hat{\tau}\times\hat{\varrho}}$ by low-rank approximation $\widetilde{\mathcal{K}}|_{\hat{\tau}\times\hat{\varrho}} = V_\tau S_{\tau,\varrho} V_\varrho^\top$.

$\mathcal{H}^2$-matrix

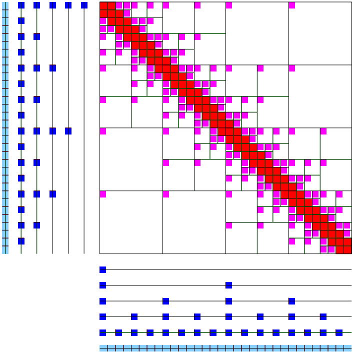
Idea: Use three-term factorization in all admissible blocks $\hat{\tau} \times \hat{\varrho}$.



Replace all admissible blocks by low-rank approximations, leave inadmissible blocks unchanged.

$\mathcal{H}^2$-matrix

Idea: Use three-term factorization in all admissible blocks $\hat{\tau} \times \hat{\varrho}$.
Use nested representation for the cluster basis.



Use transfer matrices $T_{\tau'} \in \mathbb{R}^{k \times k}$ with $V_\tau|_{\hat{\tau}' \times k} = V_{\tau'} T_{\tau'}$ for all sons $\tau' \in \operatorname{sons}(\tau)$ to handle cluster basis (V_τ) efficiently.

Result: $\mathcal{H}^2$-matrix approximation $\widetilde{\mathcal{K}}$ of $\mathcal{K}$.

Storage requirements: One row of $\widetilde{\mathcal{K}}$ represented by only $\mathcal{O}(m)$ units of storage, total storage requirements $\mathcal{O}(Nm)$.

Numerical Results

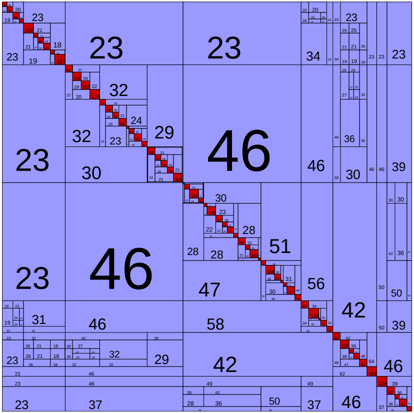
- data sets
  - network of simple sensor motes (Intel Lab Data)
  - predict the temperature at a mote from the measurements of neighbouring ones
  - mote22 consists of 30000 training / 2500 test from 2 other motes
  - mote47 has 27000 training / 2000 test from 3 nearby motes
  - helicopter flight project
  - predict yaw rate in next timestep based on 3 measurements
  - heliYaw has 40000 training / 4000 test data in 3 dimensions
- using Gaussian RBF kernel $e^{-\|\underline{x}-\underline{y}\|^2/w}$
- note: Matern family used in paper as well
- hyperparameters w and σ were found using a 2:1 split of the training data for each data set size
- note: $\mathcal{H}^2$-matrix approximation can be used for several σ

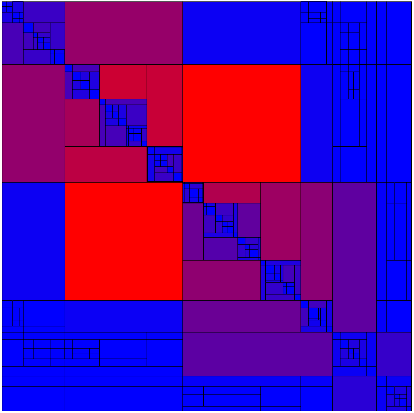
Numerical Results (Quality, Speedup)

		stored	o.t.fly	(f. both)	$\mathcal{H}^2$-matrix		
data set	#data	time	time	error	time	error	KB/N
mote 22	20000	2183	21050	0.2785	230	0.2787	2.0
mote 22	30000	n/a	88033	0.2577	494	0.2577	3.7
mote 47	20000	3800	36674	0.1326	1022	0.1326	16.4
mote 47	27000	n/a	73000	0.1289	1625	0.1289	17.2
heliYaw	20000	1091	10781	0.0091	676	0.0092	2.3
heliYaw	40000	n/a	162789	0.0083	3454	0.0083	6.6

- matrix for $N = 20000$ can (barely) be stored
- speedups of two orders of magnitude for large data sets
- twice to ten-times the speedup of related work
- storage reduction of more than one order of magnitude
- for helicopter data set from 156.25 down to 6.6, or in total from about 6 GB to about 250 MB

mote22: $\mathcal{H}^2$-matrix approximation for 5000 data





difference between full matrix and $\mathcal{H}^2$-matrix is $3.79 \cdot 10^{-8}$ in the spectral norm

mote 22: study scaling of approaches

- using $w = 2^{-9}$ and $\sigma = 2^{-5}$ for different data set sizes
- compare runtime per iteration for the different values of N
- on-the-fly computation expected $\mathcal{O}(N^2)$ scaling
- stored matrix even worse than $\mathcal{O}(N^2)$ from 10000 to 20000 data
- for $\mathcal{H}^2$-matrix scaling is nearly like $\mathcal{O}(Nm\log(N))$

		1000	5000	10000	20000	30000
$\mathcal{H}^2$-matrix	time	1.43	22.64	75.0	230.0	427.5
	its	284	688	1111	1599	2025
	time/its	0.00504	0.0329	0.0675	0.144	0.211
stored matrix	time	1.18	51.15	324.1	2183	
	its	284	689	1103	1596	n/a
	time/its	0.00415	0.0742	0.29383	1.368	
on-the-fly	time	9.13	565.2	3620.2	21050	60990
	its	284	689	1103	1596	2005
	time/its	0.032	0.82	3.282	13.189	30.42

mote22, different data set sizes using 'optimal' parameters w / σ

#data	w / σ	stored	o.t.fly	error	$\mathcal{H}^2$	error	KB/N
1000	$2^{-3}/2^{-6}$	0.3	1.1	0.350	1.56	0.350	0.8
5000	$2^{-7}/2^{-7}$	30	296	0.318	22.8	0.319	1.1
10000	$2^{-7}/2^{-8}$	811	8502	0.304	76.2	0.307	1.1
20000	$2^{-9}/2^{-5}$	2183	19525	0.279	230.1	0.279	2.0
30000	$2^{-11}/2^{-5}$	n/a	88033	0.258	494.8	0.258	3.7

- 'optimal' w / σ found via 2:1 split of training data
- observe different 'optimal' w / σ found for each data set size $\rightarrow$ need for parameter tuning on large data set
- runtime of $\mathcal{H}^2$-matrix starts to make an improvement against stored matrix after 5000 data points
- more data useful for better results

effect of σ on number of iterations

- using the 30000 data of mote22
- results are from 2:1 split using $w = 2^{-8}$ and different σ
- i.e. matrix size is 20000
- observe how number of iterations of GMRES depends on σ

σ	2^{-7}	2^{-6}	2^{-5}	2^{-4}	2^{-3}	2^{-2}	2^{-1}	2^{0}
MAE	0.265	0.263	0.264	0.265	0.268	0.275	0.289	0.320
its	3000	2375	597	179	91	70	55	41

- note: with smaller w the number of iterations usually grows as well

Conclusions

- $\mathcal{H}^2$-matrices for approximating Gaussian Processes
- time $\mathcal{O}(Nm\log(N))$, storage $\mathcal{O}(Nm)$
- speedups of up to two orders of magnitude for large data sets
- current work: use coarser $\mathcal{H}^2$-matrix for preconditioning
- open question: how to efficiently built $\mathcal{H}^2$-matrix in higher dim's
-
- HLib available at www.hmatrix.org
- code for GP with $\mathcal{H}^2$-matrices available on request