Approximating Gaussian processes with H2-matrices

Conference Paper · September 2007

DOI: 10.1007/978-3-540-74958-5_8 · Source: dx.doi.org

2 authors:

Steffen Börm
Christian-Albrechts-Universität zu Kiel
90 PUBLICATIONS **2,376** CITATIONS

SEE PROFILE

Jochen Garcke
University of Bonn
114 PUBLICATIONS **2,998** CITATIONS

SEE PROFILE

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

Steffen Börm[1] and Jochen Garcke[2]

[1] Max Planck Institute for Mathematics in the Sciences
Inselstraße 22–26, 04103 Leipzig, Germany, sbo@mis.mpg.de
[2] Technische Universität Berlin, Institut für Mathematik, MA 3-3
Straße des 17. Juni 136, 10623 Berlin, garcke@math.tu-berlin.de

Abstract. To compute the exact solution of Gaussian process regression one needs $\mathcal{O}(N^3)$ computations for direct and $\mathcal{O}(N^2)$ for iterative methods since it involves a densely populated kernel matrix of size $N \times N$, here N denotes the number of data. This makes large scale learning problems intractable by standard techniques.
We propose to use an alternative approach: the kernel matrix is replaced by a data-sparse approximation, called an $\mathcal{H}^2$-matrix. This matrix can be represented by only $\mathcal{O}(Nm)$ units of storage, where m is a parameter controlling the accuracy of the approximation, while the computation of the $\mathcal{H}^2$-matrix scales with $\mathcal{O}(Nm \log N)$.
Practical experiments demonstrate that our scheme leads to significant reductions in storage requirements and computing times for large data sets in lower dimensional spaces.

1 Introduction

In this paper we consider the regression problem arising in machine learning. A set of data points $\underline{x}_i$ in a d-dimensional feature space is given, together with an associated value y_i. We assume that a function f_* describes the relation between the predictor variables $\underline{x}$ and the (noisy) response variable y and want to (approximately) reconstruct the function f_* from the given data. This allows us to predict the function value for any newly given data point.

We apply Gaussian Process regression (GP) [1] for this task. In the direct application this method gives rise to a computational complexity of $\mathcal{O}(N^3)$; for iterative methods one has $\mathcal{O}(N^2)$ in each iteration. This makes large scale learning problems intractable for exact approaches.

One approximation approach for large problems is to use only a subset of the data for the iterative solver, i.e., to compute the inverse of a smaller matrix and extend that result in a suitable way to the whole data set, see [1] for an overview and further references. In [2] probabilistic sparse approximations are presented under a unified concept as "exact interference with an approximated prior".

One can interpret the above approaches also as "approximated interference with the exact prior", this view especially holds for recent approaches using Krylov subspace iteration methods with an approximation of the matrix-vector

product in the case of Gaussian kernels. In [3] this product is approximated using tree-type multiresolution data structures like kd-trees. [4, 5] compare several multipole approaches, these can show speedups of an order of magnitude or more, but the results heavily depend on the parameters of the problem.

In this paper we use hierarchical matrices ($\mathcal{H}$-matrices) [6] to derive a data-sparse approximation of the full kernel matrix. $\mathcal{H}$-matrices are closely related to panel-clustering and multipole techniques for the treatment of integral operators in two or three spatial dimensions. They reduce the storage requirements for N-by-N matrices to $\mathcal{O}(Nm \log N)$ by applying local rank-m-approximations and allow us to evaluate matrix-vector products in $\mathcal{O}(Nm \log N)$ operations. Other operations like multiplication or inversion can also be accomplished in almost linear complexity [6, 7].

For very large N it is a good idea to look for even more efficient techniques: $\mathcal{H}^2$-matrices [8] reduce the storage requirements to $\mathcal{O}(Nm)$, and using the recompression algorithm described in [9], the conversion of the original $\mathcal{H}$-matrix representation into the more efficient $\mathcal{H}^2$-matrix format can be accomplished with only a minor increase in run-time.

2 Gaussian process regression

Let us consider a given set of data (the training set) $S = \{(\underline{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^N$, with $\underline{x}_i$ representing data points in the feature space and y_i their associated response variable. Assume for now that these data have been obtained by sampling an unknown function f with additional independent Gaussian noise e_i of variance σ^2, i.e., $y_i = f(\underline{x}_i) + e_i$. The aim is now to recover the function f from the given data as well as possible. Following the Bayesian paradigm, we place a prior distribution on the function $f(\cdot)$ and use the posterior distribution to predict on new data points $\underline{x}$. In particular we assume a Gaussian process prior on the function $f(\underline{x})$, meaning that values $f(\underline{x})$ on points $\{\underline{x}_i\}_{i=1}^N$ are jointly Gaussian distributed with zero mean and covariance matrix $\mathcal{K}$. The kernel (or covariance) function $k(\cdot, \cdot)$ defines $\mathcal{K}$ via $\mathcal{K}_{i,j} = k(\underline{x}_i, \underline{x}_j)$.

It turns out that the solution $f(\underline{x})$ takes on the form of a weighted combination of kernel functions on training points $\underline{x}_i$ [1]

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

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

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

here $\mathcal{I}$ denotes the unit matrix. The variance σ^2 is in practice estimated via the marginal likelihood criterion, cross-validation, or similar techniques [1].

Directly solving this equation by inverting $\mathcal{K}+\sigma^2\mathcal{I}$ involves in general $\mathcal{O}(N^3)$ operations. Furthermore, the storage requirement of the covariance matrix $\mathcal{K}$

exceeds the memory of current workstations even for the moderate dimension $N \geq 15000$. A common approach is to use a smaller subset of the data and the associated kernel functions to represent the solution f [1, 2]. That way one can in principle achieve methods of order $\mathcal{O}(M^2 \cdot N + M^3)$ instead of $\mathcal{O}(N^3)$, but one loses the exact theoretical properties of the approach. Furthermore, these approximate methods are known to fail for some data sets or have costly subset selection strategies for competitive results, see [10] for results in the related case of regularised least squares classification. Note that these methods still scale with the square of M and therefore show a quadratic complexity in N if $M \geq \sqrt{N}$.

2.1 Iterative solution with Krylov subspace methods

The use of conjugate gradients (CG) for the solution of the linear equation system (2) is for example described in [1]. In Krylov subspace iteration methods the k-th approximate solution x^k of $Ax = b$ is searched in the Krylov space $\text{span}\{b, Ab, \ldots, A^{k-1}b\}$, i.e., the history of the already computed approximation is used, see, e.g., [11]. For symmetric positive definite matrices, the CG algorithm ensures that x^k minimises the A-norm of the error, i.e., $\langle Ae^k, e^k \rangle^{1/2}$ with $e^k := x - x^k$. In the case of general matrices, the GMRES algorithm minimises the Euclidean norm of the residual, i.e., $\|b - Ax^k\|$. Note that in both cases only the action of the matrix A on a vector is needed in the actual computation. As long as the number of iterations is bounded by a constant independent of the matrix size and the problem parameters under consideration, these iterative approaches result in a computational complexity that is proportional to the complexity of one matrix-vector multiplication, i.e., $\mathcal{O}(N^2)$ operations are required for the solution of (2) in the standard case.

Since $\mathcal{O}(N^2)$ is still too costly for large problems, approaches using Krylov subspace iteration methods with a more efficient approximation of the matrix-vector product in the case of Gaussian kernels have been studied recently. In [3] the matrix-vector product is approximated using tree-type multiresolution data structures like kd-trees. [4, 5] compare the fast Gauss transform, the improved fast Gauss transform (IFGT) and dual-tree approaches for fast Gauss transforms, see [3–5] for references with regard to the algorithms. In these articles empirical speedups of an order of magnitude or slightly more, in comparison with on-the-fly computation of the kernel matrix, are shown. The results heavily depend on the parameters, e.g., the number of dimensions, the width of the Gaussian kernel, the regularisation parameter, and the data distribution, see [5]. This is not always taken into proper account in experimental studies.

2.2 $\mathcal{H}$- and $\mathcal{H}^2$-Matrices

We propose to use a Krylov-based approach and rely on $\mathcal{H}$- (cf. [6, 7]) and $\mathcal{H}^2$-matrices (cf. [8, 9]) to speed up the computation of the matrix-vector multiplication by the full matrix $\mathcal{K}$. In order to reach this goal, we replace the matrix by a data-sparse approximation $\widetilde{\mathcal{K}}$ that allows us to perform matrix-vector multiplications (and other arithmetic operations) very efficiently.

$\mathcal{H}$- and $\mathcal{H}^2$-matrices have originally been developed for the approximation of matrices arising when treating elliptic partial differential equations. They require only $\mathcal{O}(Nm \log N)$ and $\mathcal{O}(Nm)$ units of storage, respectively, where m determines the accuracy. The matrix-vector multiplication is of the same order, since it requires not more than two operations per unit of storage.

In the following we will present the core ideas behind these matrix approximations and their computation. For more details we refer to the references, in particular the lecture notes [6].

The basic idea of $\mathcal{H}$- and $\mathcal{H}^2$-matrix techniques is to exploit the smoothness of the kernel function k: if k is sufficiently smooth in the first variable, it can be replaced by an interpolant

$$\tilde{k}(\underline{x}, \underline{z}) := \sum_{\nu=1}^{m} \mathcal{L}_\nu(\underline{x}) k(\underline{\xi}_\nu, \underline{z}),$$

where $(\underline{\xi}_\nu)_{\nu=1}^m$ are interpolation points and $(\mathcal{L}_\nu)_{\nu=1}^m$ are corresponding Lagrange polynomials. Replacing k by $\tilde{k}$ in the matrix $\mathcal{K}$ yields an approximation

$$\tilde{\mathcal{K}}_{ij} := \tilde{k}(\underline{x}_i, \underline{x}_j) = \sum_{\nu=1}^{m} \mathcal{L}_\nu(\underline{x}_i) k(\underline{\xi}_\nu, \underline{x}_j) = (AB^\top)_{ij} \tag{3}$$

with matrices $A, B \in \mathbb{R}^{N \times m}$ defined by $A_{i\nu} := \mathcal{L}_\nu(\underline{x}_i)$ and $B_{j\nu} := k(\underline{\xi}_\nu, \underline{x}_j)$. While the standard representation of $\mathcal{K}$ requires N^2 units of storage, only $2Nm$ units are sufficient for the factorised representation (3). If k is sufficiently smooth, m can be quite small, and the new representation will be far more efficient than the standard one.

Computing the factorised representation $\widetilde{\mathcal{K}} = AB^\top$ by interpolation will be relatively inefficient if the function k has special properties: if, e.g., k is a quadratic polynomial, interpolating it by seventh-order polynomials will lead to an unnecessary increase in computational complexity. In order to avoid this effect, we replace the interpolation by the heuristic *adaptive cross approximation* (ACA) algorithm [12]:

1: set $\widehat{\mathcal{K}} := \mathcal{K}$ and $\widetilde{\mathcal{K}} := 0$
2: estimate $\epsilon_0 := \|\widehat{\mathcal{K}}\|$
3: set $m := 0$
4: **while** $\epsilon_m > \epsilon_0 \epsilon_{\mathrm{ACA}}$ **do**
5: set $m := m + 1$
6: pick pivot indices $i^m, j^m \in \{1, \ldots, N\}$
7: compute $a_i^m := \widehat{\mathcal{K}}_{ij^m}$ for $i = 1, \ldots, N$
8: compute $b_j^m := \widehat{\mathcal{K}}_{i^m j} / a_{i^m}^m$ for $j = 1, \ldots, N$
9: set $\widetilde{\mathcal{K}} := \widetilde{\mathcal{K}} + a^m (b^m)^\top$ and $\widehat{\mathcal{K}} := \widehat{\mathcal{K}} - a^m (b^m)^\top$
10: estimate $\epsilon_m := \|\widehat{\mathcal{K}}\|$
11: **end while**

There are different strategies for picking the pivot indices in step 6 of the algorithm. A simple approach is to assume that j^m is given, compute a^m, pick i^m

such that $|a^m_{i^m}|$ is maximised, compute b^m, pick j^{m+1} such that $|b^m_{j^m}|$ is maximised, and repeat the procedure for $m+1$. In our experiments, we use the more refined strategies of the HLib package [13].

Since $a^m(b^m)^\top$ is a rough approximation of $\widehat{K}$, a reasonable strategy for estimating the remaining error in steps 2 and 10 of the algorithm is to use $\|\widehat{K}\|_2 \approx \|a^m(b^m)^\top\|_2 = \|a^m\|_2\|b^m\|_2$. Of course, problem-dependent norms can be used instead of the spectral norm, provided that they can be estimated efficiently.

The entries $\widehat{\mathcal{K}}_{ij^m}$ and $\widehat{\mathcal{K}}_{i^m j}$ used in steps 7 and 8 should not be computed explicitly, since this would require computing the entire matrix K. A more elegant approach is to use the definition of $\widehat{\mathcal{K}}$: due to

$$\widehat{\mathcal{K}}_{ij} = (\mathcal{K} - \widetilde{\mathcal{K}})_{ij} = \mathcal{K}_{ij} - \left(\sum_{\ell=1}^{m-1} a^\ell (b^\ell)^\top\right)_{ij} = \mathcal{K}_{ij} - \sum_{\ell=1}^{m-1} a^\ell_i b^\ell_j,$$

we can avoid storing $\widehat{\mathcal{K}}$ explicitly and reconstruct its entries as necessary.

At the end of each iteration of the inner loop, we have $\widehat{\mathcal{K}} + \widetilde{\mathcal{K}} = \mathcal{K}$, i.e., $\|\mathcal{K} - \widetilde{\mathcal{K}}\| = \|\widehat{\mathcal{K}}\|$, so the stopping criterion allows us to control the relative approximation error if the estimates for the norms of $\widehat{\mathcal{K}}$ are sufficiently accurate. The vectors $(a^\nu)_{\nu=1}^m$ and $(b^\nu)_{\nu=1}^m$ yield the desired representation $\widetilde{\mathcal{K}} = AB^\top$.

In many applications, e.g., if the function k is not globally smooth, the representation (3) will not be efficient for the global matrix $\mathcal{K}$. Typical functions k are *locally* smooth or decay rapidly, and the factorised representation can be applied for submatrices $\mathcal{K}|_{t\times s} = (\mathcal{K}_{ij})_{i\in t, j\in s}$ with suitable subsets $t, s \subseteq \{1,\ldots,N\}$.

The analysis in [14] shows that the approximation error will even decrease *exponentially* depending on m if k is locally analytic and if

$$\operatorname{diam}(B_t) \leq \eta \operatorname{dist}(B_t, B_s) \tag{4}$$

holds for a parameter $\eta \in \mathbb{R}_{>0}$ and two axis-parallel boxes B_t and B_s satisfying $\underline{x}_i \in B_t$ for all $i \in t$ and $\underline{x}_j \in B_s$ for all $j \in s$. Here the Euclidean diameter and distance are used, which can be computed easily for axis-parallel boxes.

We are faced with the task of splitting $\mathcal{K}$ (up to a sparse remainder) into a collection of submatrices which satisfy a condition of the type (4), i.e., can be approximated in the form (3). Since the data points $(\underline{x}_i)_{i=1}^N$ are embedded in $\mathbb{R}^d$, a hierarchy of *clusters* of indices assigned to boxes B_t can be constructed by binary space partitioning. If a cluster t is "too large", its box B_t is split in two equal parts along its longest edge, and this induces a splitting of the cluster t. Starting with $t = \{1,\ldots,N\}$ in the whole domain and applying the procedure recursively yields a cluster tree.

The corresponding hierarchical matrix structure is also constructed by recursion: if a block $t \times s$ is admissible, it is represented by a low-rank matrix. If t or s are leaves, the block is considered "small" and stored in the standard format. Otherwise, the sons of t and s are tested until no untested blocks remain. This procedure requires $\mathcal{O}(N \log N)$ operations [7], but the complexity will grow exponentially in d: each cluster t can be expected to "touch" at least $3^d - 1$

other blocks s, these blocks will not satisfy the admissibility condition (4) and have to be examined by recursion. Therefore the current implementation of the $\mathcal{H}$-matrix approach is attractive for up to four spatial dimensions, but will suffer from the "curse of dimensionality" if d becomes larger due to the use of binary space partitioning.

The $\mathcal{H}$-matrix approximation based on ACA or interpolation will have an algorithmic complexity of $\mathcal{O}(Nm \log N)$ [6]. If N is very large, the logarithmic factor can have a significant impact regarding the total storage requirements, and some problems may even become intractable given a fixed amount of storage.

The situation is different for the $\mathcal{H}^2$-matrix representation [8, 9], which relies on the same basic ideas as the $\mathcal{H}$-matrix approach: the matrix is split into blocks, and each block is approximated by a low-rank matrix. The main difference between the two methods is the choice of this low-rank matrix: for the $\mathcal{H}$-matrix, it is motivated by the interpolation in the x variable, for the $\mathcal{H}^2$-matrix, we interpolate in both variables and get

$$\tilde{k}(\underline{x}, \underline{z}) := \sum_{\nu=1}^{m} \sum_{\mu=1}^{m} \mathcal{L}_\nu(\underline{x}) k(\underline{\xi}_\nu, \underline{\zeta}_\mu) \mathcal{L}_\mu(\underline{z}),$$

which leads to the factorisation $\mathcal{K}|_{t \times s} \approx V S W^\top$ for a small *coupling matrix* $S \in \mathbb{R}^{m \times m}$ and *cluster bases* $V, W \in \mathbb{R}^{N \times m}$. If this approximation scheme is applied to submatrices of $\mathcal{K}$, the special structure of V and W, i.e., the fact that they are discretised Lagrange polynomials, can be used in order to reach a total complexity of $\mathcal{O}(Nm)$.

Using the quasi-optimal algorithm presented in [9], an $\mathcal{H}$-matrix can be converted into a more compact $\mathcal{H}^2$-matrix with only a minor run-time overhead while keeping the approximation error under close control.

2.3 Coarsening

The $\mathcal{H}$- and $\mathcal{H}^2$-matrix structures created by this procedure will suffer from the "curse of dimensionality", i.e., their complexity will grow exponentially in the spatial dimension d. In order to reduce this effect, the coarsening techniques described in [15] are employed to construct a far more efficient $\mathcal{H}$-matrix structure: even if t and s do not satisfy the admissibility condition, the block $\mathcal{K}|_{t \times s}$ may have an efficient low-rank approximation. Using the singular value decomposition of $\widetilde{\mathcal{K}}|_{t \times s}$, we can determine the *optimal* low-rank approximation for a given precision ϵ_{HC} and use it instead of the original one if it is more efficient.

1: **repeat**
2: pick t and s in such a way that $\widetilde{\mathcal{K}}_{t' \times s'}$ is represented by a low-rank matrix for all subblocks $t' \times s'$ of $t \times s$.
3: compute a low-rank approximation of $\widetilde{\mathcal{K}}_{t \times s}$ up to an error of ϵ_{HC}
4: **if** the new approximation is more efficient than the original ones **then**
5: replace the original low-rank representations by the new one
6: **end if**

7: **until** all pairs t and s have been checked

In most practical situations, the algorithm is relatively inexpensive compared to the initial approximation and yields a very significant compression.

See Figure 1 for an example of the resulting structure and the local ranks of an $\mathcal{H}^2$-matrix for a typical data set (using a suitable permutation of the data).

The full algorithm to compute the $\mathcal{H}^2$-matrix approximation consists of the following steps:

1: build hierarchy of clusters t by binary space partitioning
2: build blocks $t \times s$ using the admissibility criterion (4)
3: **for** all admissible blocks $t \times s$ **do**
4: use ACA to compute a low-rank approximation $\widetilde{\mathcal{K}}|_{t\times s} = AB^\top$
5: **end for**
6: **for** all remaining blocks $t \times s$ **do**
7: compute standard representation $\widetilde{\mathcal{K}}|_{t\times s} = \mathcal{K}|_{t\times s}$
8: **end for**
9: coarsen the block structure adaptively
10: convert the $\mathcal{H}$-matrix into an $\mathcal{H}^2$-matrix

Note that the blockwise errors introduced by ACA can be estimated by a simple heuristic approach. A rigorous error estimate can be provided if the approximation is based on interpolation and the growth rate of the derivatives of the kernel function can be bounded [14]. The blockwise errors introduced by the coarsening algorithm can be computed and controlled directly based on the singular value decomposition [15]. Given a bound for the condition number of $\mathcal{K}$, error estimates for the solution vector are possible [16, Theorem 2.7.2].

3 Experimental results

We employ the Gaussian RBF kernel $e^{-\|\underline{x}-\underline{y}\|^2/w}$ in the following experiments. The hyperparameters w and σ were found using a 2:1 split of the training data for each data set size. Note that many GP users prefer to use marginal likelihood as a criterion. Since we concentrate for now on the approximation properties of our approach we have not investigated in much detail how to (more) efficiently find good hyperparameters. Since one $\mathcal{H}^2$-matrix approximation can be used during the computation for several σ, this property, which allows the cheap solution for several σ, should be exploited, which we can by using the 2:1 split.

For the following experiments, we set both the tolerance for the cross approximation ϵ_{ACA} and the tolerance for the coarsening to ϵ_{HC} to 10^{-7}. The admissibility parameter in (4) is chosen as $\eta = 0.1$. In general these parameters should depend on the type of kernel function and its parameters. We choose these fixed values in this study to have a good approximation of the kernel matrix in all cases, they could be larger if chosen depending on the actual kernel and therefore result in less computation time. We aim to have a difference of less than 1% for the error on the test data from the 2:1 split of the training data due to the employed approximation.

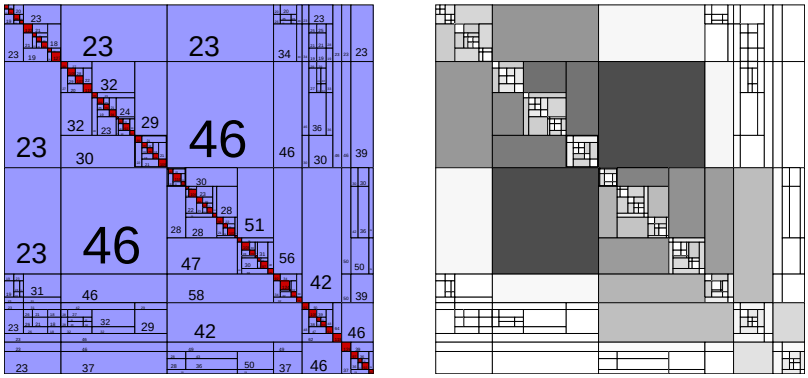

Fig. 1. Structure and rank of the $\mathcal{H}^2$-matrix approximation for the mote22 data set using 5000 data. On the right hand side error of the approximation, the darker the larger the error. The difference between the full matrix and the $\mathcal{H}^2$-matrix approximation is $3.79 \cdot 10^{-8}$ in the spectral norm for this example.

The computations are carried out with the HLib package [13] on an AMD Opteron 275 with about 4 GB of available memory. To solve the linear system (2) for the $\mathcal{H}$-matrix we use GMRES, since the use of the adaptive cross approximation algorithm disturbs the symmetry of the matrix $\widetilde{\mathcal{K}}$ slightly and gives somewhat unstable results with conjugate gradients. Note that we currently can not exploit the symmetry of the kernel matrix by only using the upper or lower half of the matrix due to limitations in the HLib, but the extension for this situation is in development. We also limit the number of iterations by 3000.

Two regression data sets are used, in the first one the data originates from a network of simple sensor motes[3] and the task is to predict the temperature at a mote from the measurements of neighbouring ones [17]. Mote22 consists of 30000 training / 2500 test data from two other motes and mote47 has 27000 training / 2000 test data from three nearby motes. The second data is from a helicopter flight project [18] and the task is to use the current state to predict subdynamics of the helicopter for one timestep later, in particular its yaw rate, forward velocity, and lateral velocity. We have 40000 training / 4000 test data in three dimensions in the case of the yaw rate, both velocities depend on two other measurements. For all these data sets we linearly transform the domain of the predictor variable $\underline{x}$ to $[0,1]^d, d = 2, 3$.

In Table 1 we present results of our experiments on these data sets, we use the mean absolute error (MAE) on the test set as the quality criterion. We give in each case results for 20000 data, here the matrix $(\mathcal{K} + \sigma^2\mathcal{I})$ can still be stored in the available memory of 4 GB, and for the full data set. The times (in seconds) presented here and in the following are for the computation using the given hyperparameters, i.e., solution of the equation system (2) and the evaluation on the data. In the case of the $\mathcal{H}^2$-matrix this includes the computation of the

[3] Intel Lab Data `http://berkeley.intel-research.net/labdata/`

Table 1. MAE and runtime (in seconds) for different data sets using the matrix in memory (stored), computing the matrix action in every iteration (on-the-fly) and using the $\mathcal{H}^2$-matrix approximation. Also given are the w and σ used.

			stored	on-the-fly (for both)		$\mathcal{H}^2$-matrix		
data set	#data	w/σ	time	time	error	time	error	KB/N
mote 22	20000	$2^{-9}/2^{-5}$	2183	21050	0.278530	230	0.278655	2.0
mote 22	30000	$2^{-11}/2^{-5}$	n/a	88033	0.257725	494	0.257682	3.7
mote 47	20000	$2^{-9}/2^{-5}$	3800	36674	0.132593	1022	0.132553	16.4
mote 47	27000	$2^{-9}/2^{-6}$	n/a	73000	0.128862	1625	0.128913	17.2
heliX	20000	$2^{-8}/2^{-6}$	4084	37439	0.015860	603	0.015860	2.9
heliX	40000	$2^{-10}/2^{-10}$	n/a	$> 50h$	n/a	1975	0.014748	9.5
heliY	20000	$2^{-7}/2^{-10}$	4053	37546	0.020373	724	0.020372	3.2
heliY	40000	$2^{-10}/2^{-10}$	n/a	$> 50h$	n/a	2303	0.018542	15.1
heliYaw	20000	$2^{-5}/2^{-6}$	1091	10781	0.009147	676	0.009154	2.3
heliYaw	40000	$2^{-7}/2^{-6}$	n/a	162789	0.008261	3454	0.008263	6.6

Table 2. Runtimes in seconds and MAE results for the mote22 data set for different data set sizes using 'optimal' parameters w / σ.

N=#data	w / σ	stored	on-the-fly	error	$\mathcal{H}^2$-matrix	error	size KB/N
1000	$2^{-3}/2^{-6}$	0.3	1.1	0.34979	1.56	0.34987	0.8
5000	$2^{-7}/2^{-7}$	30	296	0.31834	22.8	0.31938	1.1
10000	$2^{-7}/2^{-8}$	811	8502	0.30380	76.2	0.30743	1.1
20000	$2^{-9}/2^{-5}$	2183	19525	0.27853	230.1	0.27865	2.0
30000	$2^{-11}/2^{-5}$	n/a	88033	0.25772	494.8	0.25768	3.7

approximation matrix, the largest part of the total time for this approach. We observe a speedup between 1.6 and 9.5 against the stored matrix and between 16 and 91 measured against on-the-fly computation of the matrix-vector-product.

On the full version of the data set the matrix cannot be stored anymore and we can only compare against the on-the-fly computation, the speedup here is between 44 and 178, going from hours to minutes. Note that the additional data always results in an improvement on the test data, the mean absolute error is reduced by 3% to 11%. One also observes that the bigger data sets use different parameters w and σ, using the ones obtained on a smaller version of the data set often results in no improvement at all. The amount of memory needed, measured in KB per data points, can be reduced for the large helicopter data set from 156.25 down to 6.6, or in total from about 6 GB to about 250 MB. Note that [3] observe for these data sets speedups of 3.3 to 88.2 against on-the-fly computation of the matrix-vector-product using an approach based on *kd*-trees, although no mention of the employed parameters w and σ is made, which heavily influences the runtime as we will see in the following.

In Table 2 we show the results for the mote22 data set using different training set sizes, but the same test data. One can observe the different 'optimal' w /

Table 3. Runtimes in seconds, number of iterations and time per iteration for the mote22 data set using $w = 2^{-9}$ and $\sigma = 2^{-5}$ for different data set sizes.

		1000	5000	10000	20000	30000
$\mathcal{H}^2$-matrix	time (sec.)	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 (sec.)	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 (sec.)	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

Table 4. Using the 30000 data of mote22, the shown test results are from the 2:1 split using $w = 2^{-8}$ and different σ. Observe how the number of iterations needed by the GMRES solver depends on σ.

σ	2^{-7}	2^{-6}	2^{-5}	2^{-4}	2^{-3}	2^{-2}	2^{-1}	2^{0}
MAE 2:1 test	0.26447	0.26311	0.26349	0.26472	0.26818	0.27506	0.28929	0.32003
its	3000	2375	597	179	91	70	55	41

σ found for each data set size, which shows the need for parameter tuning on the full data instead of employing parameters found on a subset. Note that the runtime of the $\mathcal{H}^2$-matrix starts to make an improvement against the stored matrix already for 5000 data points.

To study the scaling behaviour with regard to N, the number of data, we present in Table 3 runtime results for one set of parameters. Since the number of iterations grows with the data set size we compare the runtime per iteration for the different values of N. For the on-the-fly computation one observes the expected $\mathcal{O}(N^2)$ scaling behaviour. For the full matrix it actually is even worse than $\mathcal{O}(N^2)$ from 10000 to 20000 data, this may be due to changes in the efficiency of the memory access in the dual core system after certain memory requirements. In the case of the $\mathcal{H}^2$-matrix the scaling is nearly like $\mathcal{O}(Nm\log(N))$.

We study in Table 4 how the number of iterations depends on the σ employed, it grows with smaller σ. This is due to the fact that the smaller σ is, the larger the condition of the matrix $(\mathcal{K} + \sigma^2\mathcal{I})$ becomes. But the smallest mean absolute errors are often achieved for small σ and one therefore typically needs quite a large number of iterations. Note also that with smaller w the number of iterations usually grows as well.

In Table 1 we also observe that typically with more data the best results are achieved with both decreasing w and σ, both cases result in more iterations of the Krylov solver. Therefore efficient computation of the matrix-vector-product gets even more important for large data sets, not just due to the number of data, but also because of the growing number of iterations in the solution stage.

Finally we use the $\mathcal{H}^2$-matrix with a different kernel, we tried the Matérn family [1] of kernels which is given by $\phi_\nu(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}(cr)^\nu K_\nu(cr)$, where K_ν is a modified Bessel function of the second kind of order $\nu > 0$ and $c > 0$. For $\nu = 1/2$ one obtains the 'random walk' Ornstein-Uhlenbeck kernel. We did experiments with the mote22 dataset for $\nu = 1/2, 1, 3/2$ and $5/2$ and achieved the best results with $\nu = 1/2$. Using all 30000 data the best parameters turn out to be $c = 8.0$ and $\sigma = 2^{-4}$. Computing the matrix-vector-product on-the-fly we need 5265 seconds, the approach with the $\mathcal{H}^2$-matrix is finished after 431 seconds using 3.3 KB/N. The result on the test data is 0.2004, a significant improvement over the use of the Gaussian kernel.

4 Conclusions and Outlook

We introduce the concept of hierarchical matrices for Gaussian Processes. Our approach scales with $\mathcal{O}(Nm\log(N))$, i.e., far less than quadratic in the number of data, which allows the efficient treatment of large data sets. On large data sets, where the kernel matrix cannot be stored, we observe speedups of up to two orders of magnitude compared to the on-the-fly computation of the matrix-vector-product during the iterative Krylov solution of the linear equation system.

Among the competing methods, in particular the probabilistic sparse approximations [2] are promising. These techniques have a complexity of $\mathcal{O}(NM^2)$, where the hyperparameter M, the number of data chosen for computational core, controls the accuracy of the approximation. Comparing this estimate to the estimate for our approach suggests that the latter will be preferable if $m \log N \leq M^2$. To our knowledge, the proper choice of M has not been investigated in detail up to now. Results in [1] suggest that even a choice of $M = 2000$ is not sufficient for a data set of size $N = 45000$ to achieve good accuracy.

The current implementation of the HLib is optimised for two and three spatial dimensions, the extension to higher dimensions is a topic of ongoing research. The basic structure of local rank-m- or more general tensor approximations should be usable in this case as well. In our case ideas like hierarchical clustering are most promising. It is also worthwhile to investigate if the ideas from probabilistic sparse approximations can be combined with the hierarchical matrix approach presented here.

In our experiments we use a simple 2:1 splitting of the training data for the hyperparameter fitting. For large data sets one advantage of the marginal likelihood criterion, that all data is employed in learning and fitting, is not as significant as for small data sets. Nevertheless we intend to adopt this criterion for hyperparameter fitting in the future. The goal is to use one approximation of the kernel matrix for several values of σ.

Till now we have not considered preconditioning at all. We could use a larger error tolerance ϵ_{HC} to compute a second but much coarser and therefore smaller $\mathcal{H}^2$-matrix. We then can cheaply compute its LU or Cholesky decomposition and use it as a preconditioner for GMRES [6]. In other application areas the

number of iterations typically goes from hundreds or thousands down to ten or twenty, depending on how coarse the second $\mathcal{H}^2$-matrix is.

This is especially worthwhile for the computation of the predictive variance on the test data. In our case it seems that this would necessitate the solution of a linear equation system for each test data point [1]. Since the kernel matrix would be the same in every case, additional computation to obtain a good and cheap preconditioner is easily compensated, since the solution would then need only a few matrix-vector-multiplications.

References

1. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
2. Quiñonero-Candela, J., Rasmussen, C.E.: A unifying view of sparse approximate gaussian process regression. J. of Machine Learning Research **6** (2005) 1935–1959
3. Shen, Y., Ng, A., Seeger, M.: Fast gaussian process regression using kd-trees. In Weiss, Y., Schölkopf, B., Platt, J., eds.: NIPS 18. MIT Press (2006)
4. Freitas, N.D., Wang, Y., Mahdaviani, M., Lang, D.: Fast krylov methods for n-body learning. In Weiss, Y., Schölkopf, B., Platt, J., eds.: Advances in Neural Information Processing Systems 18. MIT Press, Cambridge, MA (2006)
5. Lang, D., Klaas, M., de Freitas, N.: Empirical testing of fast kernel density estimation algorithms. Technical Report TR-2005-03, Department of Computer Science, University of British Columbia (2005)
6. Börm, S., Grasedyck, L., Hackbusch, W.: Hierarchical Matrices. Lecture Note 21 of the Max Planck Institute for Mathematics in the Sciences (2003)
7. Grasedyck, L., Hackbusch, W.: Construction and arithmetics of $\mathcal{H}$-matrices. Computing **70**(4) (2003) 295–334
8. Hackbusch, W., Khoromskij, B., Sauter, S.A.: On $\mathcal{H}^2$-matrices. In Bungartz, H., Hoppe, R., Zenger, C., eds.: Lect. on Applied Mathematics, Springer (2000) 9–29
9. Börm, S., Hackbusch, W.: Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. Computing **69** (2002) 1–35
10. Rifkin, R., Yeo, G., Poggio, T.: Regularized least-squares classification. In Suykens, J., Horvath, G., Basu, S., Micchelli, C., Vandewalle, J., eds.: Advances in Learning Theory: Methods, Models and Applications, IOS Press Amsterdam (2003) 131–153
11. Greenbaum, A.: Iterative methods for solving linear systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1997)
12. Bebendorf, M.: Effiziente numerische Lösung von Randintegralgleichungen unter Verwendung von Niedrigrang-Matrizen. PhD thesis, Uni. Saarbrücken (2000)
13. Börm, S., Grasedyck, L.: HLib – a library for $\mathcal{H}$- and $\mathcal{H}^2$-matrices (1999) Available at `http://www.hlib.org/`.
14. Börm, S., Grasedyck, L.: Low-rank approximation of integral operators by interpolation. Computing **72** (2004) 325–332
15. Grasedyck, L.: Adaptive recompression of $\mathcal{H}$-matrices for BEM. Computing **74**(3) (2004) 205–223
16. Golub, G.H., Van Loan, C.F.: Matrix Computations. Johns Hopkins U. P. (1996)
17. Buonadonna, P., Hellerstein, J., Hong, W., Gay, D., Madden, S.: Task: Sensor network in a box. In: Proc. of European Workshop on Sensor Networks. (2005)
18. Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., , Liang, E.: Autonomous inverted helicopter flight via reinforcement learning. In: International Symposium on Experimental Robotics. (2004)