Adaptive Greedy Techniques for Approximate Solution of Large RBF Systems

Robert Schaback and Holger Wendland

Abstract

For the solution of large sparse linear systems arising from interpolation problems using compactly supported radial basis functions, a class of efficient numerical algorithms is presented. They iteratively select small subsets of the interpolation points and refine the current approximative solution there. Convergence turns out to be linear, and the technique can be generalized to positive definite linear systems in general. A major feature is that the approximations tend to have only a small number of nonzero coefficients, and in this sense the technique is related to *greedy algorithms* and *best n–term approximation*.

1 Introduction

Let $\Omega \subset \mathbb{R}^d$ be a bounded domain, and let $\Phi : \Omega \times \Omega \to \mathbb{R}$ be a symmetric positive definite function. This means that for any finite set $X = \{x_1, \ldots, x_N\}$ of N different points in Ω the matrix

$$A_X := (\Phi(x_j, x_k))_{1 \le j,k < \le N}$$

is symmetric and positive definite. In particular, we think of Φ being a **radial** basis function generated by a compactly supported function $\phi : [0, h_0] \to \mathbb{R}$ via $\Phi(x, y) := \phi(\|x - y\|_2)$. In this case, the matrix A_X will be sparse for h_0 small enough.

The reconstruction of a function $f : \Omega \to \mathbb{R}$ from its discrete data $f_{|X} = (f(x_1), \ldots, f(x_N))^T$ on X can be done by an interpolant

$$s_{f,X} := \sum_{j=1}^{N} \alpha_j(f, X)\Phi(\cdot, x_j) \tag{1}$$

whose coefficients $\alpha(f,X) = (\alpha_1(f,X), \ldots, \alpha_N(f,X))^T$ satisfy the system

$$A_X \alpha(f,X) = f_{|X}$$

The main goal of this paper is to provide methods that efficiently produce approximate solutions of very large systems of the above form. In addition, we concentrate on approximate solutions with only few nonzero coefficients $\alpha_j(f,X)$. The reason is that the evaluation of a full sum in (1) on many points will be too costly, if the sum contains a term for each data value. In short, we try to approximate N data with $K << N$ terms, and we want to keep the storage and computational effort proportional to N. This implies that we try to avoid storage of the full matrix A_X.

2 Native Space Norm

A crucial tool will be the norm $\|.\|_\Phi$ defined via the inner product

$$(s_{f,X}, s_{g,Y})_\Phi = \sum_{i=1}^{M} \sum_{j=1}^{N} \alpha_i(f,X)\alpha_j(g,Y)\Phi(x_i,y_j).$$

For the special case $\Phi(x,y) = \|x-y\|_2 \log \|x-y\|_2$ in $I\!R^2$ the value $\|s_{f,X}\|_\Phi^2$ describes the bending energy of a thin plate described by the function $s_{f,X}$. Thus one should view this norm as kind of an energy. The closure of all functions of the form $s_{f,X}$ with respect to the above norm is a ("native") Hilbert space $\mathcal{N}_\Phi$ of functions in Ω. We do not want to pursue this topic any further (see e.g. [5] for a recent reference), but we need the orthogonality relation

$$(s_{f,X}, f - s_{f,X})_\Phi = 0$$

for all f from the native space. It is a consequence of the fact that $s_{f,X}$ has minimal norm under all functions in $\mathcal{N}_\Phi$ that interpolate f on X. The Pythagorean Theorem then implies

$$\|f\|_\Phi^2 = \|f - s_{f,X}\|_\Phi^2 + \|s_{f,X}\|_\Phi^2, \tag{2}$$

and we shall make frequent use of this equation.

3 Iteration on Residuals

The orthogonality relation (2) simply says that the "energy" of a function f can be split up into the "energy" of an interpolant $s_{f,X}$ plus the "energy" of the residual $f - s_{f,X}$. We shall apply this "energy split" recursively by interpolating the residual. More precisely:

Algorithm 1 *Start with a given function $f_0 := f \in \mathcal{N}_\Phi$ and iterate over an index $k = 0, 1, ...$ by interpolating f_k on some set $X_k \subset \Omega$ by $s_k := s_{f_k, X_k}$. The next iterate will then be $f_{k+1} := f_k - s_k$.*

Theorem 1 *The functions s_k of Algorithm 1 satisfy the summability condition*

$$\begin{aligned} \|f_0\|_\Phi^2 - \|f_{m+1}\|_\Phi^2 &= \sum_{k=0}^{m} \left(\|f_k\|_\Phi^2 - \|f_{k+1}\|_\Phi^2 \right) \\ &= \sum_{k=0}^{m} \|s_k\|_\Phi^2. \end{aligned} \tag{3}$$

Proof: Using Algorithm 1, equation (2) turns into

$$\begin{aligned} \|f_k\|_\Phi^2 &= \|f_k - s_{f_k, X_k}\|_\Phi^2 + \|s_{f_k, X_k}\|_\Phi^2 \\ &= \|f_{k+1}\|_\Phi^2 + \|s_k\|_\Phi^2 \end{aligned}$$

and by summation we get (3). □

We now want to look for conditions that imply convergence of the residuals f_k to zero, because then our accumulated interpolants

$$g_k := \sum_{j=0}^{k} s_j = f - f_{k+1} \tag{4}$$

converge to f for $k \to \infty$. This needs some further assumptions, since we have so far not excluded trivial cases like $X_k = X$ for all k.

4 Convergence Analysis

From the energy viewpoint, we should require that s_k picks up at least a certain fraction of the energy of f_k.

Theorem 2 *If there is some positive constant γ such that*

$$\|s_k\|_\Phi \geq \gamma \|f_k\|_\Phi \quad \textit{for all } k, \tag{5}$$

then the functions f_k and the accumulated interpolants g_k of (4) converge linearly to zero and f, respectively, in the native space.

Proof: The assertion is implied by

$$\|f_{k+1}\|_\Phi^2 = \|f_k\|_\Phi^2 - \|s_k\|_\Phi^2 \leq (1-\gamma^2)\|f_k\|_\Phi^2. \square$$

But since $\|f_k\|_\Phi$ is not easily accessible in practice, we prefer to use a weaker seminorm $|.|_*$, i.e.

$$|f|_* \leq C\|f\|_\Phi \quad \text{for all } f \in \mathcal{N}_\Phi. \tag{6}$$

Theorem 3 *If there is some positive constant γ such that*

$$|s_k|_* \geq \gamma |f_k|_* \quad \textit{for all } k, \tag{7}$$

then the seminorms $|f_k|_*$ and $|f-g_k|_*$ converge to zero for $k \to \infty$. More precisely, they form square summable sequences.

Proof: The assumptions (6) and (7) imply

$$\begin{aligned} \|f_0\|_\Phi^2 - \|f_{m+1}\|_\Phi^2 &= \sum_{k=0}^{m} \left(\|f_k\|_\Phi^2 - \|f_{k+1}\|_\Phi^2 \right) \\ &= \sum_{k=0}^{m} \|s_k\|_\Phi^2 \\ &\geq \frac{\gamma^2}{C^2} \sum_{k=0}^{m} |f_k|_*^2 \end{aligned} \tag{8}$$

and summability of $|f_k|_*^2 = |f - g_{k-1}|_*^2$. This is all we can hope for under our weak hypotheses. $\square$

But note that the seminorm $|.|_*$ can be a norm like $\|.\|_2$ or $\|.\|_\infty$ on Ω. Then we would get convergence in these norms, and the requirement (7) in each step still is manageable. We leave this interesting case and its consequences for calculating native space norms open for later work.

5 Interpolation on subsets

An important special case arises from a discrete norm $|.|_* = \|.\|_{L_p(X)}$ on a large subset $X = \{x_1, \ldots, x_N\} \subset \Omega$. By standard results on error bounds for radial basis function interpolation, this is a bounded seminorm on the native space. We now confine everything to X and use the above argument for $s(f, X)$ instead of f.

Algorithm 2 *Start with data* $f_{0|_X}$ *of some function* $f_0 := f \in \mathcal{N}_\Phi$ *and iterate over an index* $k = 0, 1, ...$ *by interpolating the data* $f_{k|_X}$ *of* f_k *on some subset* $X_k \subseteq X = \{x_1, \ldots, x_N\} \subset \Omega$ *satisfying*

$$|f_k|_{L_p(X_k)} \geq \gamma |f_k|_{L_p(X)}. \tag{9}$$

by $s_k := s_{f_k, X_k}$. *The next iterate will then be* $f_{k+1} := f_k - s_k$.

Theorem 4 *The functions* g_k *of* (4) *converge* **linearly** *in* $\mathcal{N}_\Phi$ *to* $s(f, X)$. *Furthermore, the norms* $|f_k|_{L_p(X)}$ *of residuals* f_k *converge linearly to zero.*

Proof: We first apply the results of Theorem 3 to $s(f, X)$ instead of f, noting that everything just works on the finite set X. At each step of Algorithm 2 we need $X_k \subseteq X$ and (7) in the form

$$|s_k|_{L_p(X)} \geq \gamma |f_k|_{L_p(X)} \quad \text{for all } k, \tag{10}$$

which is easily achievable, since we make s_k to coincide with f_k on $X_k \subseteq X$ by interpolation. In fact, due to

$$|s_k|_{L_p(X)} \geq |s_k|_{L_p(X_k)} = |f_k|_{L_p(X_k)} \geq \gamma |f_k|_{L_p(X)} \tag{11}$$

we only require X_k to satisfy (9).

Then the accumulated approximations g_k converge to $s(f, X)$ on X. But since functions of this form are bijectively mapped to their values on X, we have a convergent iterative scheme for solving large systems of the form (1).

But this is not the end of the story. Since we restrict everything to X and linear combinations s of $\Phi(\cdot, x_j)$ for $x_j \in X$, there are constants $c_1 = c_1(p, X, \Phi)$ and $C_1 = C_1(p, X, \Phi)$ with

$$c_1 |s|_{L_p(X)} \leq \|s\|_\Phi \leq C_1 |s|_{L_p(X)}$$

for all such s. But now

$$\|s_k\|_\Phi \geq c_1 |s_k|_{L_p(X)} \geq c_1 \gamma |f_k|_{L_p(X)} \geq \frac{c_1 \gamma}{C_1} \|f_k\|_\Phi \tag{12}$$

implies linear convergence by Theorem 2. □

For smooth radial basis functions and densely distributed points in X, the quotient c_1/C_1 can be extremely small, making the linear convergence statement a purely theoretical issue. The convergence behavior of $\|s_k\|_\Phi$ from (3) often shadows linear convergence within the numerically relevant range of iterations.

6 Iterative interpolation on single points

Let us look at the above argument for the case where X_k consists of a single point $x_{j_k} \in X = \{x_1, \ldots, x_N\}$. We get linear convergence via (9) in Theorem 4, if the condition

$$|f_k(x_{j_k})| \geq \gamma |f_k|_{L_\infty(X)} \tag{13}$$

holds at each step. This is clear for $p = \infty$ in (11), and for the other cases we have

$$|f_k|^p_{L_p(X)} \geq |f_k|^p_{L_p(X_k)} \geq |f_k(x_{j_k})|^p \geq \gamma^p |f_k|^p_{L_\infty(X)} \geq \frac{\gamma^p}{N} |f_k|^p_{L_p(X)}. \tag{14}$$

Picking the maximum absolute value of the residual at each stage means $\gamma = 1$, and then we have a "greedy" method. Since this extremely simple algorithm turns out to be unexpectedly useful in case of compactly supported radial basis functions, let us write it down in some detail. Everything is done on function or residual values on a large finite set $X = \{x_1, \ldots, x_N\}$. Storage is needed for X and the values $f_{|X} = (f(x_1), \ldots, f(x_N))^T$, which are later overwritten by residuals, i.e.the values of f_k on X. Furthermore, a vector of length N accumulates the coefficients α_j of the functions g_k for later use. Storage requirements thus are $N * (d + 2)$ in d dimensions.

Algorithm 3 *For initialization, the values of $f = f_0$ on X are generated and stored. The N coefficients are set to zero. For the startup iteration index $k = 0$ we further pick some dummy point $x_{j_0} \in X = \{x_1, \ldots, x_N\}$ and the dummy coefficient $\beta_{j_0} = 0$.*

The iteration at stage k then loops over all values of f_k on X and does two things on each value: it replaces $f_k(x_i)$ by the residual

$$f_{k+1}(x_i) := f_k(x_i) - \beta_{j_k} \Phi(x_i, x_{j_k})$$

and it keeps track of the maximum absolute value of the updated results. After this loop over N elements, there is some point $x_{j_{k+1}} \in X = \{x_1, \ldots, x_N\}$ where $|f_{k+1}(x_{j_{k+1}})| = |f_{k+1}|_{L_\infty, X}$, and the interpolant to this value on $x_{j_{k+1}}$ is the function

$$s_k := \Phi(\cdot, x_{j_{k+1}}) \frac{f_{k+1}(x_{j_{k+1}})}{\Phi(x_{j_{k+1}}, x_{j_{k+1}})}.$$

Thus we set

$$\beta_{j_{k+1}} := \frac{f_{k+1}(x_{j_{k+1}})}{\Phi(x_{j_{k+1}}, x_{j_{k+1}})}$$

and add this value to the current value of $\alpha_{j_{k+1}}$ to update the total approximation. Then we repeat the iteration for $k + 1$ instead of k.

Due to Theorems 1 and 4, the values $|f_k|_{L_\infty,X}$ generated by Algorithm 3 are square summable and converge linearly to zero. This proves linear convergence of the algorithm, measured in the native space norm or any discrete norm on X.

For curiosity, one can form the energy

$$\|s_k\|_\Phi^2 = \frac{f_{k+1}(x_{j_{k+1}})^2}{\Phi(x_{j_{k+1}}, x_{j_{k+1}})}$$

and monitor the monotonely convergent sum over these values according to (3). The values $|f_k|_{L_\infty,X}$ are also numerically available, and they must converge linearly (but not necessarily monotonely) to zero. Furthermore, their squares are summable, and they must converge to zero at least like $1/k$. Though being inferior to linear convergence, this convergence behaviour is the one that can be numerically observed in early stages of the iteration. These values can be used as a stopping criterion, but one can also choose any discrete norm $|f_k|_{L_p,X}$ for this purpose. In view of (3) and (8), a comparison of the sum of squares of $\|s_k\|_\Phi$ and $|f_k|_{L_p,X}$ reveals some information on the constants in the error analysis.

Convergence of the algorithm is rather slow, but its merits for extremely large problems rely on other properties:

- It brings in one coefficient at a time, and it produces approximations that have less than the full number of nonzero coefficients.
- It does not form any matrix–vector multiplications, and it does not even store the coefficient matrix.
- Compared to the convergence analysis in [3], its convergence (in theory) is linear with respect to the index k only, and does not require N such steps to form a successful iteration.

Let us do a very rough analysis of its performance, based on the weaker convergence behaviour like $1/k$. After k steps the order of magnitude of the residuals will be brought down by a factor of $1/k$, and this is achieved by using only k approximating functions. One can possibly expect 1% accuracy after 100 steps, using just 100 coefficients.

This strategy is not useful if one wants an **exact** solution of a system of, say, 100.000 data points. But it often does not make sense to use all 100.000 degrees of freedom to solve such a system exactly, coming up with a "solution"

with 100.000 coefficients, whose sheer size limits its usefulness. It seems to be much more reasonable to get away with 1000 nonzero parameters that fit the data to an accuracy of 0.1%. The above algorithm adaptively picks points (and corresponding coefficients) that are the best candidates for further treatment, and it turns out to be extendable to an algorithm that is the first to use radial basis functions of different scales adaptively. We shall address this in the next section.

Some comments towards other techniques seem appropriate at this point.

- The Faul-Powell [3] method will usually work on a full coefficient vector. Convergence of the latter is proven via steps that need a full sweep over a set of N directions, and thus each step contains a full coefficient vector. If just a part of the first sweep is considered, the technique gets comparable to ours, because it then does not work on a full coefficient vector. Linear convergence is not proven.

- Conjugate gradients have linear convergence like our technique, and in cases where its convergence rate is numerically reasonable, it outperforms our method. But it uses matrix–vector multiplications, and these (and the convergence rate) limit its applicability. For large and badly conditioned problems our technique will already produce some reasonable approximation before the conjugate gradient method has even finished its first step.

- The above technique is a special case of a greedy algorithm as described in [1], [4],[2], [6], and [7]. We use it here for solving a large linear system, but the analysis in section 3 shows that the notion of a dictionary is applicable here. Furthermore, it extends to cases with multiple instances of functions Φ, or with radial basis functions of varying scale. We shall exploit these possibilities later, without using results of the cited literature on greedy algorithms.

7 General Linear Systems

We now look at the above greedy algorithm in case of a general linear system $Ax = b$ with a symmetric and positive definite $N \times N$ coefficient matrix A. As usual in the theory of the conjugate gradient method, we define

$$\|x\|_A^2 := x^T Ax \text{ for all } x \in I\!R^N.$$

Algorithm 4 *For $j := 0$ start with $x^j := 0 \in I\!R^N$, $r^j := -b \in I\!R^N$. Then iterate for $j = 0, 1, 2, \ldots$ as follows:*

$$\begin{array}{rcl} \textit{stop} & \textit{if} & \|r^j\|_\infty \textit{ is small enough, else:} \\ |r^j_{k_j}| & := & \|r^j\|_\infty \\ \alpha_{k_j} & := & -r^j_{k_j}/a_{k_j,k_j} \\ x^{j+1} & := & x^j + \alpha_{k_j} e_{k_j} \\ r^{j+1} & := & r^j + \alpha_{k_j} A e_{k_j} \quad \textit{(in practice)} \\ & = & Ax^{j+1} - b \quad \textit{(by induction)} \end{array}$$

Note that the method introduces only the numerically relevant unknowns due to its pivoting strategy based on the right–hand side. Thus the technique is fundamentally different from the method of Gauss–Seidel or Jacobi. Furthermore, the method does not form any matrix–vector products. It pays for this by a low convergence rate.

Theorem 5 *The iterates x^j of Algorithm 4 converge linearly to the solution $x^* \in I\!R^N$ with $Ax^* = b$. The convergence rate can be bounded above via*

$$\|x^* - x^{j+1}\|_A^2 \leq \|x^* - x^j\|_A^2 \left(1 - \frac{\lambda_{min}(A)}{N \max_k a_{kk}}\right)$$

Proof: By a standard variational argument, the algorithm solves the minimization problem

$$\|x^* - x^{j+1}\|_A = \min_\alpha \|x^* - x^j - \alpha e_{k_j}\|_A.$$

By Pythagoras' theorem we then get

$$\|x^* - x^j\|_A^2 = \|x^* - x^{j+1}\|_A^2 + \alpha_{k_j}^2 \|e_{k_j}\|_A^2.$$

From $\|e_{k_j}\|_A^2 = a_{k_j,k_j}$ and $|\alpha_{k_j}| = \|r^j\|_\infty / a_{k_j,k_j}$ we conclude

$$\|x^* - x^{j+1}\|_A^2 = \|x^* - x^j\|_A^2 - \|r^j\|_\infty^2 / a_{k_j,k_j}.$$

We are done if we show

$$\|r^j\|_\infty^2 \geq \frac{\lambda_{min}(A)}{N} \|x^* - x^j\|_A^2.$$

But this follows from

$$\|x^* - x^j\|_A^2 = (x^* - x^j)^T A (x^* - x^j) = (x^* - x^j)^T r^j \leq \|r^j\|_\infty \|x^* - x^j\|_1$$

and

$$\begin{aligned}\lambda_{min}(A)\|x^* - x^j\|_1^2 &\leq N\lambda_{min}(A)\|x^* - x^j\|_2^2 \\ &\leq N(x^* - x^j)^T A(x^* - x^j) \\ &= N\|x^* - x^j\|_A^2.\end{aligned}$$

□

The above algorithm cannot be suggested as a general–purpose solver for symmetric positive definite linear systems. It makes sense only for cases where the application expects to get away with an approximative solution that has many zero coefficients. This, however, is the case as soon as bases with some hierarchical structure or a lot of built–in redundancy are considered. Since preconditioning can be seen as an appropriate change of basis, it makes sense to investigate how this algorithm behaves under some additional preconditioning. But we leave such things open here.

8 Adaptive Scaling

We now want to look at a modification of Algorithm 2 that uses **scaled** radial basis functions $\Phi_c(x, y) := \phi(\|x - y\|_2/c^2)$. In particular, we aim at functions ϕ that have support in $[0, 1]$, such that $\Phi_c(x, y)$ vanishes for $\|x - y\|_2 > c$.

Algorithm 5 *We fix real constants*

$$\alpha, \epsilon > 0 < \gamma < \beta < 1 < \sigma.$$

Furthermore, we use some discrete norm for residuals on a large data set X, and we need an iteration count $K \geq 1$ and a large starting scale c. In what follows, a successful try *is defined by a run of K steps of Algorithm 3 at a fixed scale c such that the discrete norm of residuals is reduced at least by a factor of α.*

- *The outermost loop runs over successful tries until the discrete norm of residuals falls below a prescribed bound ϵ. At each iteration, it uses the other loops to find a successful try by suitable variation of the values of K and c:*
  - *A middle loop tries larger and larger numbers $K, K\sigma, K\sigma^2, \ldots$ of iterations, and an inner loop*
    - *tries scales $c, c\beta, c\beta^2 > \ldots > c\gamma$*

  until a successful try is found.

Since we know that at any fixed scale Algorithm 3 must bring the residuals to zero after sufficiently many iterations, the middle loop must terminate at each of the finitely many scales allowed in the inner loop. It terminates using the scale that roughly takes the fewest number of new points to reach success. Since the middle loop reduces the residual norm by a certain factor smaller than 1, any prescribed accuracy can be reached after sufficiently many outer iterations.

Note that the algorithm tries first to get away with as few new points as possible, using the smallest possible iteration count that leads to a reduction of the residuals. For each iteration count, it tests large scales first, but priority is given to the iteration count over the scale.

Setting $K = 1$, using a large c and extremely small values of $\delta, 1-\beta, \sigma-1$ will lead to a very time–consuming optimization, trying hard to reconstruct the data with as few centers as possible. We shall call such a case an "optimizing" run of the algorithm in our examples. But there are some economizations that should be pointed out.

First, extremely small scales will have a very local effect and will not lead to any reasonable reduction in early stages of the algorithm. This means that the algorithm tends to prefer large scales over small scales at early stages, and extremely small values of δ need not be considered. We found $\delta = 0.5$ or $\delta = 0.25$ quite sufficient.

Second, if the scales c for successful cases are inspected, they tend to be decreasing steadily (but not monotonically). It therefore makes sense to use an update formula like

$$c_{new} := \rho \cdot c_{success}$$

with some factor $\rho \geq 1$ after each success.

Third, the necessary iterations to reach success have the tendency to increase. This suggests an update formula

$$K_{new} := \tau \cdot K_{success}$$

with some factor $\tau \geq 1$ after each success. The two values above are determined after a successful outer iterations, and used for starting the inner iterations.

A particularly efficient situation is given by $\rho = \tau = 1$, forcing successful iterations to have weakly monotone increasing or decreasing values of K and c, respectively. We shall call such a run of Algorithm 5 a "monotonic" run.

If applied for compactly supported radial basis functions, the algorithm in its above form reaches smaller and smaller scales, until the calculations can be localized and parallelized. This has not yet been fully exploited in the numerical examples following in the next section.

But we want to point out a further generalization. One can view the inner iteration just as a trial of M different radial basis functions, ignoring scale completely. Since the middle iteration increases the number of iterations for each function, it will automatically select the radial basis function that reaches success using the fewest centers. The inner loop must be finite, but after each success of the outer iteration one can come up with a different set of finitely many candidates for radial basis functions. It is easy to incorporate thin–plate splines or multiquadrics at early stages, and one can go over to compactly supported functions when it comes to resolving local details. Numerical experiments in this direction are still to be carried out. The notion of a dictionary with respect to a greedy algorithm in the sense of [1], [4],[2], [6], and [7] applies here, and it is an interesting research area to pursue this connection further.

9 Numerical Experiments

We start with a reproduction of the following Franke–type function:

$$f(x) = \sum_{j=0}^{3} a_j \exp(-b_j \|x - x_j\|_2^2)$$

with the values

j	a_j	b_j	x_j
0	1.0	-0.1	(0.0, 0.0)
1	1.0	-5.0	(0.5, 0.5)
2	1.0	-15.0	(-0.2,-0.4)
3	1.0	-9.0	(-0.8, 0.8)

To make it less smooth, we introduced a singularity of lower-order derivatives along the line $\eta - \xi = -1.0$ by taking $f(\xi, \eta) - (\eta - \xi + 1.0)\eta$ instead of $f(\xi, \eta)$ for $\eta - \xi < -1.0$. The function plot is given in Figure 1, and one can clearly see the modification in the front right corner. We then picked 40000 random centers on $[-1, +1]^2$ and constructed approximate solutions of the corresponding interpolation problem, consisting of up to 500 centers. In all examples to follow, we concentrate on three cases that reduce the maximum

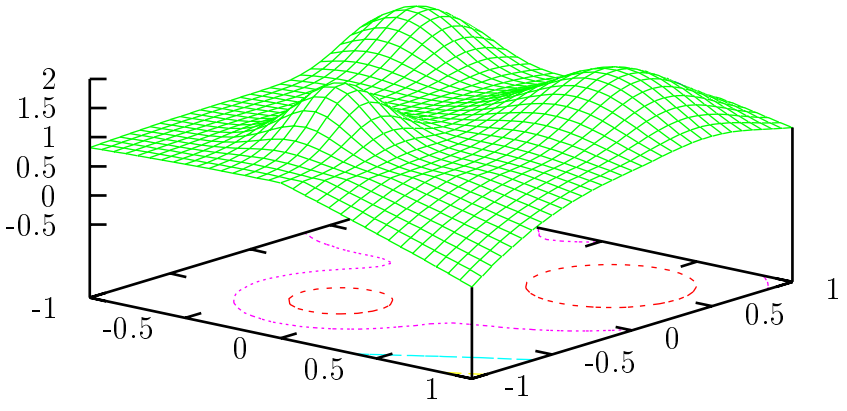

Figure 1: Franke–type function

absolute value of the residuals to 10%, 5%, and 1%, respectively. Further reduction should be done by local techniques provided by a forthcoming paper. The following table shows how many of the 40000 data locations are necessary to reach the prescribed accuracy:

%	monotone	optimized
10%	41	27
5%	61	45
1%	125	143

These two runs were made with $\alpha = \beta = 0.9, \gamma = 0.5, \sigma = 2$, and the starting scale was $c = 10$. A more detailed plot of the error as a function of the used centers is in Figure 2, while the corresponding scales are in Figure 3. Note how close the monotone run is to the optimized run in both cases, in particular for large numbers of centers. The error for the monotone run does not lead to a monotone decreasing error curve, because monotonicity is only attained for the outer iteration. Since later iterations use large values of K, there are clearly visible non–monotonic sections in the curve for the monotonic run in Figure 2. The decrease of the optimized scale in Figure 3 clearly shows that the optimizing algorithm has a strong tendency to "localize" automatically.

Both figures strongly support our suggestion to prefer the monotonic run over the optimizing run, if one just wants a quick approximation of 1% accuracy.

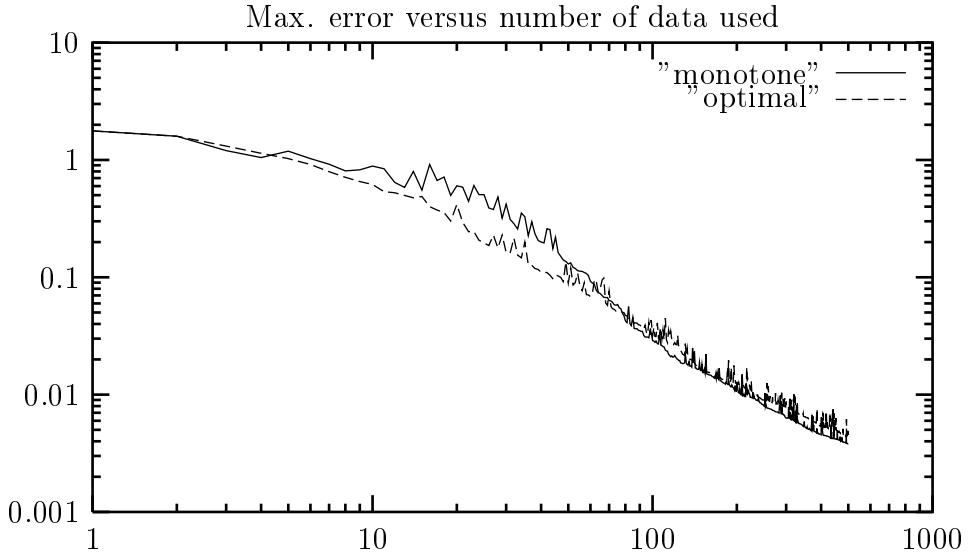

Figure 2: Error behavior

In particular, the calculation time for up to 500 actually used centers out of 40000 on a notebook with a 350MHz AMD–K6 under Linux was about 1 hour for the full optimization, as opposed to 100 seconds for the monotone run. If just the 1% accurate solution based on 125 points is needed, the monotone run needs 30 seconds.

Figure 4 shows how our adaptive technique automatically selects crucial points near the discontinuity line, if we let the monotone run extend up to 500 centers. The 1% accurate approximations from the table above do not yet discover the discontinuity precisely.

The `mbay.dat` data from R. Franke's webpage are rather difficult to handle, though they have only 1669 data points. The main problem is their infinite variation in relative scale. In the NE area of Figure 5 there is an area having data values exactly zero, and near the origin there is a single sharp positive peak. Both of these are defined by rather few data values, but there are many and dense data with small positive values describing a "shallow" area with small positive data values. The problem is to avoid negative values of the reconstruction in the zero area, and to avoid errors from the fitting of the peak to propagate into the shallow area. An exact solution with $c = 1.0$ is given in Figure 6. Note that there are areas with negative function values (the solid contour line describes the zero level), and there is some visible undulation near the NE corner. The coarse approximations with our

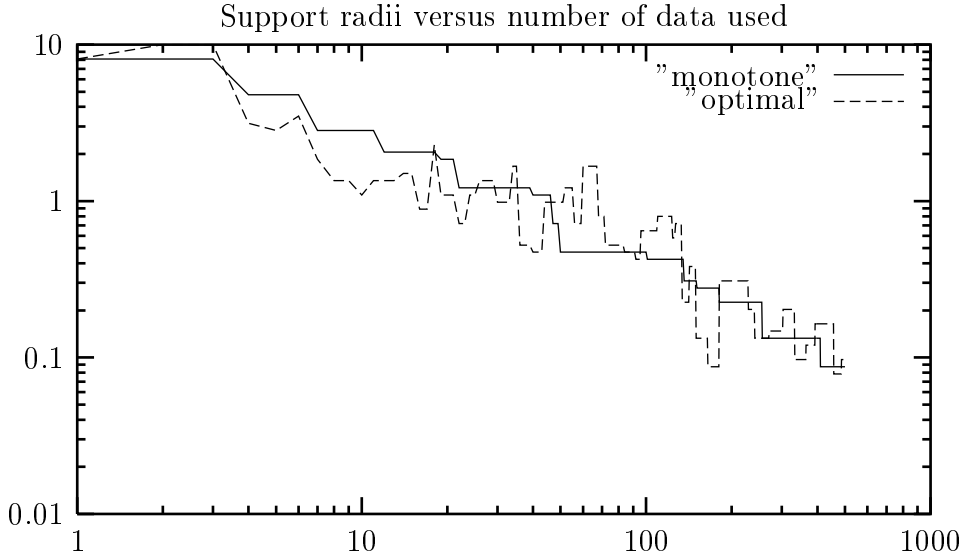

Figure 3: Scale behavior

algorithm, starting with $c = 0.3$, yielded the following numbers of centers for a prescribed relative accuracy:

%	monotone	optimized
10%	48	28
5%	109	55
1%	430	335

Even the optimized approximation of Figure 7 is calculated rather quickly (48 seconds on the aforementioned notebook computer) compared to an exact solution of a full system with 1669 equations.

In all cases one can observe how the residuals and the scales go down proportionally to $1/k$, when k centers are introduced. The summability of the squares of the residuals supports this behaviour, but asymptotic linear convergence is not visible at this distance from the full solution.

All examples will be provided and documented on the Internet via URL `http://www.num.math.uni-goettingen.de/schaback`, including some ray–traced reproductions showing more details.

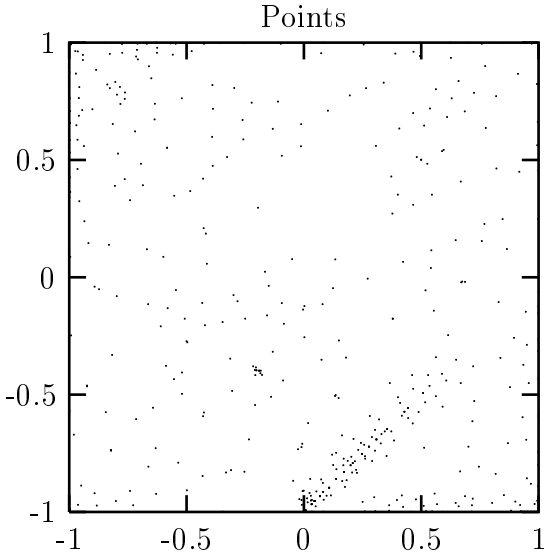

Figure 4: First 500 center locations

References

[1] Davis, G., S. Mallat, and M. Avallaneda, Adaptive greedy approximations, Constr. Approx. **13** (1997) 737–785

[2] DeVore, R.A., and V.N. Temlyakov, Some remarks on greedy algorithms, Adv. in Comp. Math. **5** (1996) 173–187

[3] Faul, A., and M.J.D. Powell, Proof of convergence of an iterative technique for thin–plate spline interpolation in two dimensions, Preprint DAMTP 1998/NA08

[4] Jones, L. A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training, Ann. Statist. **20** (1992) 608–613

[5] Schaback, R., Native Spaces of Radial Basis Functions I, to appear in the proceedings of IDoMat98.

[6] Temlyakov, V.N., The best m–term approximation and greedy algoriths, Adv. in Comp. Math. **8** (1998) 249–265

[7] Temlyakov, V.N., Greedy Algorithms and M–Term Approximation with Regard to Redundant Dictionaries, J. of Approx. Th. **98** (1999) 117–145

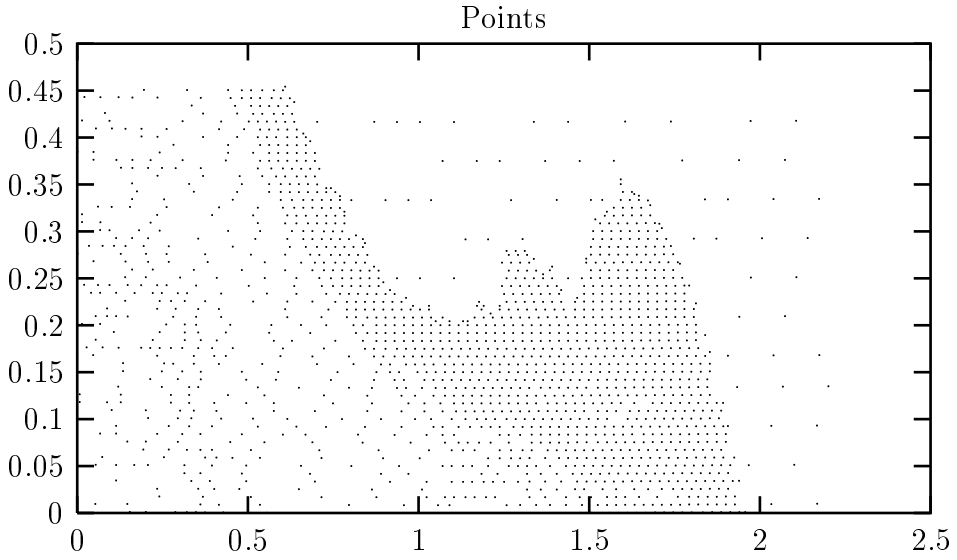

Figure 5: Data locations for mbay.dat

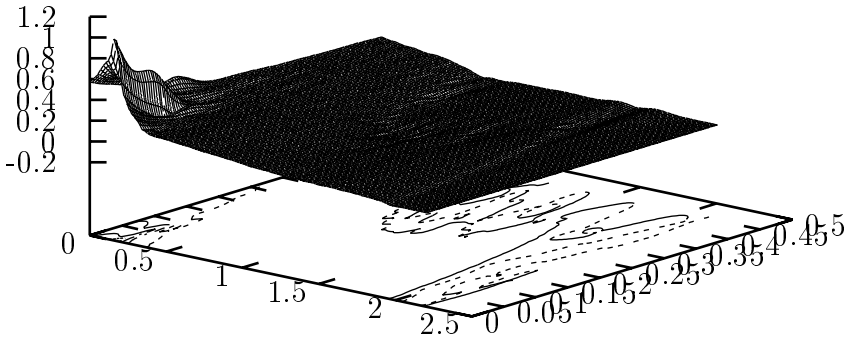

Figure 6: Exact solution with $c = 1.0$

Figure 7: Optimized run with 335 centers

Figure 8: Monotonic run with 430 centers