Iterative polynomial interpolation and data compression [*]

Morten Dæhlen [‡] and Michael Floater

SINTEF-SI
P.O. Box 124, Blindern
0314 Oslo, NORWAY

Abstract In this paper we look at some iterative interpolation schemes and investigate how they may be used in data compression. In particular we use the pointwise polynomial interpolation method to decompose discrete data into a sequence of difference vectors. By compressing these differences, one can store an approximation to the data within a specified tolerance using a fraction of the original storage space (the larger the tolerance, the smaller the fraction).

We review the iterative interpolation scheme, describe the decomposition algorithm and present some numerical examples. The numerical results are that the best compression rate (ratio of non-zero data in the approximation to the data in the original) is often attained by using cubic polynomials and in some cases polynomials of higher degree.

1. Introduction A new method of interpolating a sequence of points was discovered by Dubuc (1986) and Dyn et al. (1987). The idea is to insert a new point between each pair of original points according to some local calculation (a *mask*). One thereby obtains a new sequence with twice as many points. Having doubled the number of points and halved the resolution one can repeat the procedure. If the mask is well chosen, the sequences of points will converge to a unique continuous curve. One way of choosing the mask is to fit a cubic polynomial through the nearest four points. Dubuc (1987) presented this scheme and studied various properties of the interpolating curve. In particular the curve was shown to be C^1. Dyn et al. (1987) presented a more general scheme (involving a tension parameter) which included the cubic case. They showed that there is a range of tension parameters which yield a C^1 curve.

Deslauriers & Dubuc (1989) have considered the possibility of using higher degree polynomials. Recently, Floater (1992) investigated the sequence of interpolating curves for odd degrees 1, 3, 5, 7 etc. It is proved there that the curve obtained from the quintic scheme is C^2. Moreover numerical computations, (Floater 1992), indicate the following table.

Degree of polynomial	1	3	5	7	9	11	13	15	17
Smoothness	C^0	C^1	C^2	C^3	C^4	C^4	C^5	C^5	C^6

Table 1. Numerical estimations of smoothness of the interpolations for low degrees

These figures have recently been confirmed by Rioul (1992) who calculates the exact Sobolev exponents for (odd) degree up to 19 (in general each curve is C^p where $p \in \mathbb{R}$ is not necessarily an integer).

In this paper we will investigate how this sequence of iterative interpolation schemes can be used in decomposition and compression of discrete data. Recently, numerous papers have been written on the subject of multiresolution analysis and decomposition techniques; see Mallat (1989), Meyer (1990), Chui (1992), Dæhlen and Lyche (1992) and references therein. Data compression

* This work was supported by The Royal Norwegian Council for Scientific and Industrial Research (NTNF).

‡ Partially at Institutt for Informatikk, Univ. of Oslo, P.O.Box 1080, Blindern, 0316 Oslo, Norway

using decomposition techniques is discussed in DeVore et al. (1991a) using box splines and DeVore et al. (1991b) using Haar wavelets. With special emphasis on decomposition of splines on non-uniform knots Dæhlen and Lyche (1992) give a general introduction to decomposition of discrete data. As more and more data is being collected in a wide variety of mathematical applications, data compression algorithms are becoming a very important tool. In this paper we introduce a computationally efficient method which compares favourably with other methods for the compression of uniform discrete data. Such data sets occur frequently as scalar or vector fields, terrain models, digital images, etc. The algorithm we propose is relatively simple and it illustrates how data decomposition and compression can be achieved without the need for a large armoury of mathematical tools. In particular, we confine ourselves to discrete spaces rather than B-spline or L^p spaces.

The basic concept behind the decomposition is as follows. We assume that a suitable tolerance is given. We begin by choosing a small subset of $\mathbb{Z}$ (on which the data is given), for example $2^N\mathbb{Z}$ for some positive integer N. The data points on $2^N\mathbb{Z}$ are stored. We then use one of the interpolating polynomials, described above, to predict each of the remaining points in $2^{N-1}\mathbb{Z}$. The difference between each predicted value and the given data value is then calculated. If any difference is less then the given tolerance it is reset to zero. The sequence of differences is now stored. The differences are added to the predicted values and the process is repeated from level $2^{N-1}\mathbb{Z}$ to $2^{N-2}\mathbb{Z}$. The procedure is continued until the whole data set is decomposed into the initial subset plus a sequence of differences.

Having obtained the decomposition, we can now compress the data since the sequences of differences will typically contain many zeros. The differences can be efficiently stored on the computer by using a bit map and various reversible coding techniques. To reconstruct the approximation (i.e. decompress the data) one simply repeats the process, starting with the points on $2^N\mathbb{Z}$ and reproducing points on each level from the differences.

The question now arises as to which polynomial will yield the greatest compression. Intuitively, the smoother the data, the higher the degree of polynomial which ought to be applied. In general it is difficult to make precise statements, but for certain data sets one can analyse the effect of the degree on the compression rate. In particular, by considering the polynomial-reproducing property of the iterative interpolation scheme, it is not difficult to see that, if the data is taken from a cubic polynomial, the cubic data compression scheme will yield a lower compression rate than the linear scheme. The numerical results in Section 5 indicate that the optimal polynomial to use in practice is often the cubic.

2. Iterative polynomial interpolation

Suppose we are given the sequence of values

$$\ldots, P_{-2}, P_{-1}, P_0, P_1, P_2, \ldots$$

in $\mathbb{R}$, infinite in both directions. Our goal is to find a smooth function $F : \mathbb{R} \to \mathbb{R}$ for which $F(i) = P_i$, for all $i \in \mathbb{Z}$. Consider the following scheme. We define the function $F(x)$ recursively, starting with the values of F at the integers themselves. Thus we set

$$F(i) = P_i \qquad \text{for } i \in \mathbb{Z}.$$

The next step is to define $F(x)$ at the mid-values 1/2, 3/2, etc. We define $F(i + \frac{1}{2})$ by

$$F(i + \tfrac{1}{2}) = -\tfrac{1}{16}P_{i-1} + \tfrac{9}{16}P_i + \tfrac{9}{16}P_{i+1} - \tfrac{1}{16}P_{i+2} \qquad \text{for } i \in \mathbb{Z},$$

which is the mid-value of the unique cubic polynomial interpolating the four neighbouring values, $P_{i-1}, P_i, P_{i+1}, P_{i+2}$, see Figure 1. We have now doubled the number of points at which F is defined,

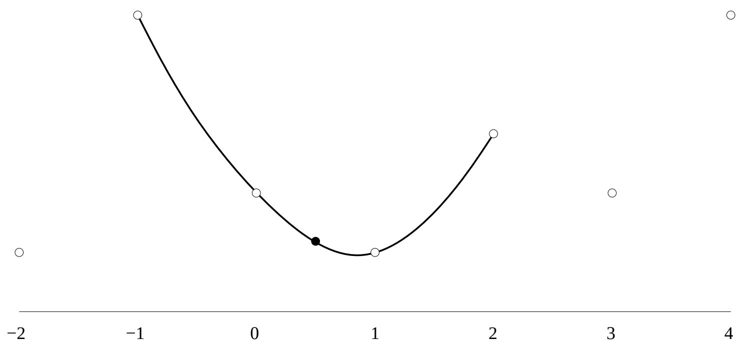

Figure 1. Cubic polynomial interpolation to define $F(1/2)$

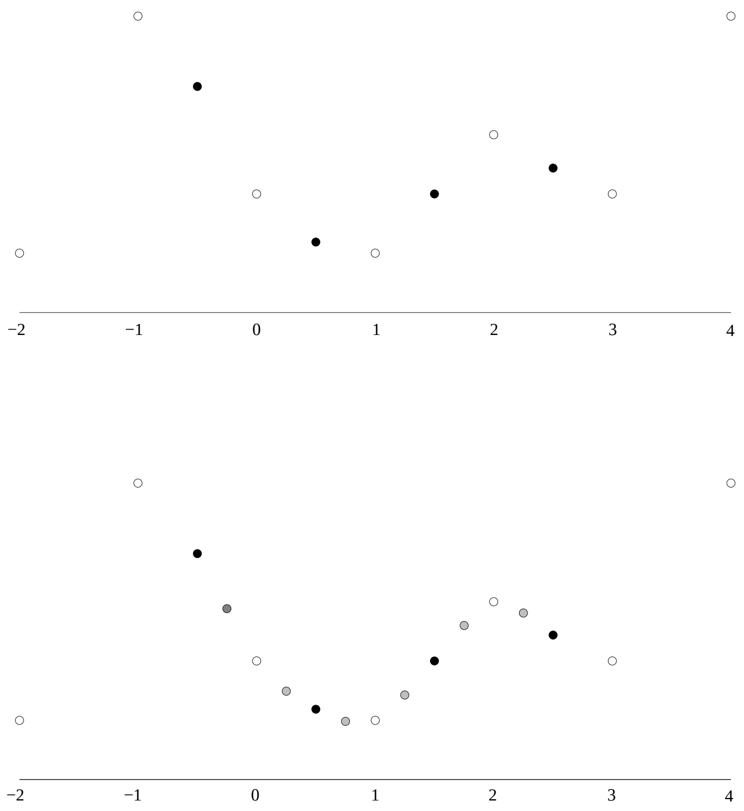

Figure 2. The first two iterations of the cubic interpolation algorithm

and this process is repeated indefinitely. Figure 2 illustrates the successive doubling of points. The white points are the given data, the black points are the first iteration and the grey ones are the second.

We have now defined $F : \Omega \to \mathbb{R}$, where Ω is the set of *dyadic rationals*, i.e.

$$\Omega = \{i/2^r : r \in \mathbb{N}, i \in \mathbb{Z}\}. \tag{1}$$

Dubuc (1986) showed that $F : \Omega \to \mathbb{R}$ has a unique continuous extension to $\mathbb{R}$ and is in fact C^1. In general, though, F is not twice differentiable. The only exception to this occurs when six values $P_i, \ldots, P_{i+5}$ lie on a cubic polynomial $Q(x)$. In this case, $F(x) = Q(x)$ for $x \in [i+2, i+3]$ and is therefore analytic on $[i+2, i+3]$.

Since F is constructed pointwise from cubic polynomials one can say that F is the *cubic pointwise interpolant* to the values P_i or F is the *pointwise polynomial interpolant of degree 3*. To indicate the use of cubic polynomials, we introduce the notation $F_3(x)$ and summarise by making the following formal definition.

Definition 1. Given the values $P_i \in \mathbb{R}$, for $i \in \mathbb{Z}$, we define $F_3 : \Omega \to \mathbb{R}$ by

$$F_3(i) = P_i \qquad \text{for } i \in \mathbb{Z},$$

and

$$F_3(\tfrac{i}{2^r} + \tfrac{1}{2^{r+1}}) = -\tfrac{1}{16}F_3(\tfrac{i-1}{2^r}) + \tfrac{9}{16}F_3(\tfrac{i}{2^r}) + \tfrac{9}{16}F_3(\tfrac{i+1}{2^r}) - \tfrac{1}{16}F_3(\tfrac{i+2}{2^r}), \tag{2}$$

for $i \in \mathbb{Z}$ and $r \in \mathbb{N}$, where Ω is defined by (1). The *pointwise cubic interpolant* $F_3 : \mathbb{R} \to \mathbb{R}$, of the values $\{P_i\}_{i\in\mathbb{Z}}$ is then the unique continuous extension of F_3 to $\mathbb{R}$.

The algorithm presented in Definition 1 is based on the cubic interpolation of four points. In a similar way we can base the algorithm on the interpolation of $n+1$ points by a degree n polynomial for any odd number $n \in \mathbb{N}$. For example, instead of fitting a cubic to define each new mid-value, as in (2), we could define it by fitting a quintic polynomial to the six values

$$F(\tfrac{i-2}{2^r}), F(\tfrac{i-1}{2^r}), F(\tfrac{i}{2^r}), F(\tfrac{i+1}{2^r}), F(\tfrac{i+2}{2^r}), F(\tfrac{i+3}{2^r}).$$

By this method, one can define the interpolating function $F_5(x)$. Set $F_5(i) = P_i$ and

$$\begin{aligned} F_5(\tfrac{i}{2^r} + \tfrac{1}{2^{r+1}}) = & \tfrac{3}{256}F_5(\tfrac{i-2}{2^r}) - \tfrac{25}{256}F_5(\tfrac{i-1}{2^r}) + \tfrac{150}{256}F_5(\tfrac{i}{2^r}) \\ & + \tfrac{150}{256}F_5(\tfrac{i+1}{2^r}) - \tfrac{25}{256}F_5(\tfrac{i+2}{2^r}) + \tfrac{3}{256}F_5(\tfrac{i+3}{2^r}) \end{aligned} \tag{3}$$

for $i \in \mathbb{Z}$ and $r \in \mathbb{N}$. This defines a function $F_5 : \Omega \to \mathbb{R}$. The *pointwise quintic interpolant* $F_5 : \mathbb{R} \to \mathbb{R}$, is then the unique continuous extension of $F_5 : \Omega \to \mathbb{R}$ to $\mathbb{R}$. The function F_5 is C^2 but not, in general, three times differentiable; see Floater (1992). Figure 3 shows an example. Four extra points were added at each end of the sequence to act as boundary conditions.

In order to compute the coefficients in (2) and (3) and to compute interpolating polynomials of general degree, we can use Lagrange polynomials; see, for example, Powell (1981). Let $Q(x)$ be the unique polynomial of degree n which fits the data values $P_0, \ldots, P_n$ at the parameter values $x_0, \ldots, x_n$. Then

$$Q(x) = \sum_{i=0}^{n} L_{i,n}(x)P_i, \qquad \text{where} \qquad L_{i,n}(x) = \frac{\prod_{j=0, j\neq i}^{n}(x - x_j)}{\prod_{j=0, j\neq i}^{n}(x_i - x_j)}.$$

We now define the interpolation algorithm for general odd degree $n = 2k + 1$, $k = 0, 1, 2, \ldots$ as follows. Note that $L_{i,n}(\frac{n}{2}) = L_{n-i,n}(\frac{n}{2})$ by symmetry.

Definition 2. Given the points $P_i \in \mathbb{R}$, for $i \in \mathbb{Z}$, we define $F_n : \Omega \to \mathbb{R}$ by

$$F_n(i) = P_i \qquad \text{for } i \in \mathbb{Z},$$

and

$$F_n(\tfrac{i}{2^r} + \tfrac{1}{2^{r+1}}) = \sum_{j=0}^{n} L_{j,n}(\tfrac{n}{2})F_n(\tfrac{i-k+j}{2^r}) \tag{4}$$

for $i \in \mathbb{Z}$ and $r \in \mathbb{N}$, where Ω is defined by (1). The *pointwise polynomial interpolant* $F_n : \mathbb{R} \to \mathbb{R}$, of the points $\{P_i\}_{i\in\mathbb{Z}}$ is then the unique continuous extension of $F_n : \Omega \to \mathbb{R}$ to $\mathbb{R}$ (assuming it exists).

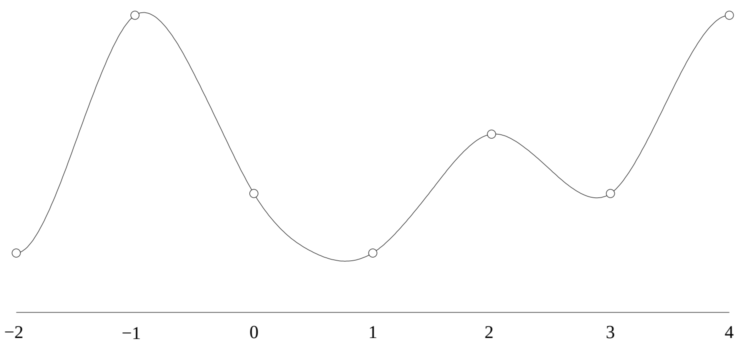

Figure 3. The degree 5 interpolation of a given set of points

Numerical evidence, (Floater 1992), indicates that each F_n in the sequence is not only continuous but is also smoother than the last one. In fact, the results imply that F_{2k+1} is C^k for $k = 0, 1, 2, 3, 4$. After that, the order of continuity ceases to increase linearly as shown in the Table 1.

It is now straightforward to derive the coefficients in (2) and (3) by setting $n = 3$ and $n = 5$ in (4). The linear case fits into the scheme naturally. The scheme for the linear case is

$$F_1(\tfrac{i}{2^r} + \tfrac{1}{2^{r+1}}) = \tfrac{1}{2}F_1(\tfrac{i}{2^r}) + \tfrac{1}{2}F_1(\tfrac{i+1}{2^r})$$

and the resulting function F_1 is simply the original polygon which is C^0.

3. Decomposition of discrete data In this section we adapt the iterative interpolation scheme, described in the previous section, in order to decompose discrete data in a fashion similar to multiresolution analysis; see Dæhlen and Lyche (1992). Suppose we are again given the sequence of values

$$\ldots, P_{-2}, P_{-1}, P_0, P_1, P_2 \ldots$$

in $\mathbb{R}$, infinite in both directions. Thus, we have a function $f : \mathbb{Z} \to \mathbb{R}$, $f(i) = P_i$. We wish to find an approximation $\hat{f} : \mathbb{Z} \to \mathbb{R}$ to f, which can be represented by as little data as possible. We can start by choosing a positive integer N, and define

$$\Omega_r = \{2^{N-r} i : i \in \mathbb{Z}\} \qquad \text{and} \qquad K_r = \{2^{N-r}(i + \tfrac{1}{2}) : i \in \mathbb{Z}\} \qquad \text{for } r = 0, \ldots, n.$$

We also define Then, we observe that

$$\begin{aligned} &\Omega_r \cap K_r = \emptyset, \\ &K_r \cap K_{r+1} = \emptyset, \\ &\Omega_r = \Omega_0 \cup K_0 \cup \cdots \cup K_{r-1}, \\ &\Omega_{r+1} = \Omega_r \cup K_r. \end{aligned}$$

Now, if $g : \Omega_r \to \mathbb{R}$, we define $R_n(g) : K_r \to \mathbb{R}$ by

$$R_n(g)(\tfrac{i}{2^{r-N}} + \tfrac{1}{2^{r-N+1}}) = \sum_{j=0}^{n} L_{j,n}(\tfrac{n}{2}) g(\tfrac{i-k+j}{2^{r-N}}) \qquad \text{for } i \in \mathbb{Z},$$

the pointwise interpolation scheme of odd degree n, described in the previous section (c.f. equation (4)).

Now, in order to describe the decomposition algorithm, we rewrite the given discrete function f, with $f(i) = P_i$, as a sequence of discrete functions $f_0, \ldots, f_N$ where f_0 is the projection of f onto Ω_0 and each f_r is the projection of f to K_{r-1}, for $r = 1, \ldots, N$. Thus, we have that

$$f = f_0 + f_1 + \cdots + f_N.$$

Each step of the algorithm consists of generating the next level of points using the cubic scheme, i.e. we apply the operator R_n and define the difference

$$e_r : K_{r-1} \to \mathbb{R} \qquad \text{as} \qquad e_r = f_r - R_n(f_0 + \ldots + f_{r-1}).$$

Hence, we obtain the decomposition sequence $f_0, e_1, \ldots, e_N$, where f is reconstructed by the following simple algorithm: For $r = 1, \ldots, N$, let $f_r = R_n(f_0 + \ldots + f_{r-1}) + e_r$.

In this fashion we say that the sequence $f_0, e_1, \ldots, e_N$ is an exact decomposition of the discrete function f. More details on this can be found in Dæhlen and Lyche (1992). We are now in a position to describe the data compression algorithm.

4. Compression by data decomposition The data compression problem can be stated as follows: Given a discrete function f, a tolerance $\epsilon > 0$ and a positive integer N, we find a sequence $\hat{f}_0, \hat{e}_1, \ldots \hat{e}_N$ with as few nonzero entries as possible, but so that the difference between the reconstruction $\hat{f}$ and the original f is less that the prescribed tolerance.

In order to describe the compression algorithm, we define the truncation

$$\hat{e}_1(x) = \begin{cases} e_1(x), & \text{if } |e_1(x)| > \epsilon; \\ 0, & \text{if } |e_1(x)| \le \epsilon; \end{cases} \qquad \text{for } x \in K_0.$$

First we set $\hat{f}_0 = f_0$. Then, as before, we set

$$e_1 = f_1 - R_n(f_0).$$

But instead of storing e_1, we store its truncation $\hat{e}_1$ which, if the estimate $R_n(f_0)$ of f_1 is good, will have many zero entries. We then define

$$\hat{f}_1 : K_0 \to \mathbb{R} \qquad \text{as} \qquad \hat{f}_1 = R_n(f_0) + \hat{e}_1.$$

Now, we can generate $\hat{f}_1$ from f_0 and $\hat{e}_1$ and the maximum difference between $\hat{f}_1$ and f_1 is within the tolerance, i.e.

$$|\hat{f}_1(x) - f_1(x)| \le \epsilon$$

for $x \in K_0$. We proceed in this way through all the levels until we have calculated all the $\hat{e}_r$. One defines

$$e_r : K_{r-1} \to \mathbb{R} \qquad \text{as} \qquad e_r = f_r - R_n(\hat{f}_0 + \hat{f}_1 + \cdots + \hat{f}_{r-1})$$

and

$$\hat{e}_r(x) = \begin{cases} e_r(x), & \text{if } |e_r(x)| > \epsilon; \\ 0, & \text{if } |e_r(x)| \le \epsilon; \end{cases}.$$

Then, we set

$$\hat{f}_r : K_{r-1} \to \mathbb{R} \qquad \text{as} \qquad \hat{f}_r = R_n(\hat{f}_0 + \hat{f}_1 + \cdots + \hat{f}_{r-1}) + \hat{e}_r.$$

It is easy to see that

$$|f(i) - (\hat{f}_0 + \hat{f}_1 + \cdots + \hat{f}_N)(i)| \le \epsilon$$

for all $i \in \mathbb{Z}$. One can also clearly regenerate the approximation to f, namely $\hat{f}_0 + \cdots + \hat{f}_N$, from $\hat{f}_0, \hat{e}_1, \ldots, \hat{e}_N$ Since the $\hat{e}_i$'s typically contain many zeros, they can be stored very efficiently on the computer. We have the following algorithm:

```
Algorithm 1
1. $\hat{f}_0 = f_0$
2. For $r = 1, \ldots, N$
    2.1. $e_r = f_r - R_n(\hat{f}_0 + \hat{f}_1 + \cdots + \hat{f}_{r-1})$
    2.2. $e_r \rightarrow \hat{e}_r$
    2.3. $\hat{f}_r = R_n(\hat{f}_0 + \hat{f}_1 + \cdots + \hat{f}_{r-1}) + \hat{e}_r$
```

One can experiment with different degrees n of the polynomial in the algorithm. If the data comes from a smooth function, one would expect that the cubic R_3 or quintic R_5 schemes would produce less non-zero data than the linear scheme R_1, i.e. the $\hat{e}_r$'s would contain more zeros, since the functions F_3 and F_5 are smoother than F_1, as shown in Table 1. The numerical evidence indicates that this is the case. In fact if the data comes from a cubic polynomial it is easy to see that the cubic scheme will perform much better than the linear one. In fact in this case, $\hat{e}_r = 0$ for all r. This corresponds to that fact that F_3 reproduces cubic polynomials.

The scheme has a straightforward extension to higher dimensions, where we are given the discrete function $f : \mathbb{Z}^d \rightarrow \mathbb{R}$. For $d = 2$ we define subsets

$$\Omega_r = \{2^{N-r} i : i \in \mathbb{Z}^2\}.$$

and

$$\begin{aligned} K_{r,1} &= \{2^{N-r}(i + (\tfrac{1}{2}, \tfrac{1}{2})) : i \in \mathbb{Z}^d\}, \\ K_{r,2} &= \{2^{N-r}(i + (0, \tfrac{1}{2})) : i \in \mathbb{Z}^d\}, \\ K_{r,3} &= \{2^{N-r}(i + (\tfrac{1}{2}, 0)) : i \in \mathbb{Z}^d\}, \end{aligned}$$

with $K_r = K_{r,1} \cup K_{r,2} \cup K_{r_3}$. Values on K_r are found by applying R_n first in the x-direction and the in the y-direction of the grid, or vise versa. The order in which we choose x and y does not matter. Hence, this can be regarded as a tensor product scheme.

However, there is an interesting extension of the scheme which can be investigated when $d = 2$. The black circles in Figure 4a represent the initial set Ω_r. By applying the scheme R_n as for the tensor product scheme above we find differences $e_{r,1} = f_r - R_n(\hat{f}_0 + \cdots + \hat{f}_{r-1})$ on $K_{r,1}$. The set $K_{r,1}$ is indicated by the grey circles in Figure 4a. Now, in order to estimate differences on the set $K_{r,2} \cup K_{r,3}$ we apply the same scheme R_n using the uniform grid $\Omega_r \cup K_{r,1}$ rotated through 45 degrees; see Figure 4b. The differences $e_{r,23}$ are then given on $K_{r,2} \cup K_{r,3}$, i.e. the index set indicated by open circles in Figure 4b. By rotating the grid back again through 45 degrees, we obtain Ω_{r+1}, the basis for the next level. This method has also been studied in the context of binary images by Arge and Dæhlen (1992). In the following we will denote this scheme by the modified tensor product scheme and refer to the operator as R_n^*.

5. Numerical experiments In this section we investigate three surface examples, discrete data sampled from an analytic function and two terrain models of different sizes and complexity, and we present the compression rates for various choices of degree of the scheme R_n and of the tolerance ϵ.

In order to simplify the notation we have so far assumed that the data has been given as infinite sequences on $\mathbb{Z}^d$ ($d = 1$ or $d = 2$). This is of course not the case in practice, where data is given on finite regions. In the following examples we assume that the data is given on

$\Psi = [0, 2^N m_1] \times [0, 2^N m_2] \cup \mathbb{Z}^2$. Hence, f_0 is defined on the $(m_1+1) \times (m_2+1)$ mesh given by $\Omega_0 = \Psi \cup 2^N \mathbb{Z}^2$, e_1 on $K_1 = (\Psi \cup 2^{N-1}\mathbb{Z}^2) \backslash \Omega_0$, etc..

Moreover, we have to redefine the interpolation scheme R_n along the boundary of Ω_r. This can be done by extending the mesh around Ω_r and using the scheme R_n as it is. However, there is another way of doing this which avoids extrapolation. For points near the boundary of Ψ we simply use a polynomial of lower degree. Recalling that $n = 2k+1$, we define an operator $\hat{R}_n$ adapted to Ψ as

$$\hat{R}_n(g)(\tfrac{i+1/2}{2^{r-N}}) = \begin{cases} R_n(g)(\frac{i+1/2}{2^{r-N}}), & \text{if } k \le i \le 2^r m_1 - k - 1; \\ R_{n-2(k-i)}(g)(\frac{i+1/2}{2^{r-N}}), & \text{if } 0 \le i < k; \\ R_{n-2(i-2^r m_1+k+1)}(g)(\frac{i+1/2}{2^{r-N}}), & \text{if } 2^r m_1 - k - 1 < i \le 2^r m_1 - 1. \end{cases}$$

Thus, for example, to estimate that outer two points on the rth level, we set $i = 0$ and $i = 2^r m_1 - 1$ in the definition and then

$$\hat{R}_n(g)(\tfrac{i+1/2}{2^{r-N}}) = R_1(g)(\tfrac{i+1/2}{2^{r-N}}) = g(\tfrac{i}{2^{r-N}}) + g(\tfrac{i+1}{2^{r-N}}))/2,$$

which is linear interpolation. This adaptation to finite domains is the method used to obtain the numerical results in the next section.

5.1. Franke's function

In the first example, we sample Franke's function, a well known standard function used in the investigation of methods for interpolating scattered data; see Foley (1987). The function $\omega : [0,1] \times [0,1] \to \mathbb{R}^2$ is defined as

$$\begin{aligned} \omega(x,y) =& 0.75e^{-0.25(9x-2)^2 - 0.25(9y-2)^2} + 0.75e^{-(9x-2)^2/49 - (9y-2)^2/10} \\ &+ 0.5e^{-0.25(9x-7)^2 - 0.25(9y-3)^2} - 0.2e^{-(9x-4)^2 - (9y-7)^2}. \end{aligned}$$

The range of ω is approximately $[-0.2, 1.05]$. We sampled ω on a uniform 257×257 grid in $[0,1] \times [0,1]$. Thus the nodes are (x_i, y_j) for $i = 0, \ldots, 256$ and $j = 0, \ldots, 256$, where $x_i = i/256$ and $y_j = j/256$. The number of decomposition levels, N, was chosen to be 6. Since $257 = 2^8 + 1$ and $2^8/2^6 = 2^2$, we find that Ω_0 has $2^2 + 1$ points in each direction, i.e. 5×5.

The data compression algorithm was carried out for tolerances 0.00001, 0.0001, 0.001, and 0.01 (these are absolute tolerances and so the range of ω should be borne in mind). The proportion of non-zero values in the decomposed data is shown in Table 2 as a percentage. Since the total number of original data points in either direction (257 in this example) is the same as in the decomposition sequence $(5 + 4 + 8 + 16 + 32 + 64 + 128)$, these percentages represent compression rates.

The tensor-product scheme for degrees 1,3,5,7 and 9 are shown, together with the modified tensor-product scheme of degree 3, denoted 3^*. Clearly, for low tolerances, higher degrees yield a lower compression rate. One finds, that as one increases the tolerance, the optimal degree goes down (from 9 for a tolerance of 0.0001 to 3 or 5 for a tolerances of 0.01). The modified scheme 3^* does not perform as well as the ordinary scheme with degree 3 for this particular example.

Tolerance	1	3	5	7	9	3^*
0.00001	79.97	6.50	3.67	3.36	3.43	6.89
0.0001	30.15	2.02	1.35	1.23	1.21	2.25
0.001	3.22	0.57	0.47	0.47	0.48	0.67
0.01	0.35	0.15	0.15	0.16	0.16	0.18

Table 2. Compression rates of Franke's function in percent

5.2. Map data

In the second example, the given data is discrete and represent heights above sea-level on a uniform 129×129 grid of coordinates in an area of land, 200×200 metres. The number of decomposition levels was chosen to be $N = 4$. Thus Ω_0 is the uniform 9×9 subgrid and the number of data points in each direction is decomposed into $129 = 9 + 8 + 16 + 32 + 64$. The range of z values is $[0, 50]$. Table 3 shows the compression rates for tolerances 0.25, 0.5, 1.0, and 2.0.

Here, the best ordinary scheme is the cubic for almost all tolerances. But the optimal scheme for all tolerances shown is the modified scheme 3^*.

Tolerance	1	3	5	7	3^*
0.25	11.44	9.58	9.53	9.67	8.55
0.5	6.63	5.35	5.41	5.54	4.63
1.0	3.25	2.88	2.96	3.01	2.41
2.0	1.62	1.51	1.51	1.48	1.37

Table 3. Compression rates of the map data in percent

5.3. Geological survey

In the third and final example, the given data is again discrete and comes from cartography. The data points represent depths below sea-level of a uniform 257×385 grid of coordinates in an area of seabed, 1.5×2.5 kilometres. The number of decomposition levels was chosen to be $N = 5$. Thus Ω_0 is the uniform 9×13 subgrid. The number of data points in the x direction is decomposed into $257 = 9+8+16+32+64+128$ and the number of data points in the y direction is decomposed into $385 = 13 + 12 + 24 + 48 + 96 + 192$ The range of z values is $[-3300, -1800]$ (metres). Table 4 shows the compression rates for tolerances 0.1, 1.0, 2.0, 4.0, 8.0 and 16.0.

Here, the best of the ordinary schemes is usually the linear one which indicates that the data is quite rough. But again, for the four highest chosen tolerances, the optimal scheme is 3^*.

Tolerance	1	3	5	7	3^*
0.1	30.67	83.21	85.35	86.32	85.68
1.0	20.25	26.34	28.62	29.36	23.04
2.0	13.13	13.54	14.69	15.22	11.58
4.0	7.41	6.48	6.83	7.13	5.50
8.0	3.68	3.32	3.35	3.40	2.75
16.0	1.64	1.61	1.63	1.66	1.28

Table 4. Compression rates of the seabed data in percent

5.4. Conclusions

If the data is rough, or has a lot of noise, the best scheme is generally the linear one. But in many cases the modified scheme 3^* produces less data. This is perhaps due to the fact that R_3^* uses data from a square grid as opposed to R_3 which takes data from lines. Thus R_3^* has the advantage of using information in all directions rather than just in the x and y directions.

If the data is relatively smooth, it is better to use a polynomial of higher degree. In this case there may be no advantage in using the modified scheme.

6. References

E. Arge & M. Dæhlen (1992) “ Data dependent subdivision”, *preprint.*

C. Chui (1992) “An introduction to wavelets”, *Academic Press, Boston.*

G. Deslauriers & S. Dubuc (1989) "Symmetric iterative interpolation processes", *Constr. Appro.*, **5**, 49–68.

R. A. deVore, B. Jawerth, & B. Lucier (1991a) " Surface compression", *preprint.*

R. A. deVore, B. Jawerth, & B. Lucier (1991b) " Image processing through wavelet transform coding", *preprint.*

S. Dubuc (1986) "Interpolation through an iterative scheme", *J. Math. anal. appl.*, **114**, 185–204.

N. Dyn, D. Levin, & J. Gregory (1987) "A 4-point interpolatory subdivision scheme for curve design", *Computer Aided Geometric Design*, **4**, 257–268.

M. Dæhlen & T. Lyche (1992) "Decomposition of Splines, Mathematical Methods CAGD and Image Processing, T. Lyche and L. Schumaker, 135–160", *Academic Press, Boston.*

M. S. Floater (1992) "Pointwise polynomial interpolation", *Research Report, SINTEF-SI, Oslo.*

T. A. Foley (1987) "Interpolation and approximation of 3-D and 4-D scattered data", *Comput. Math. Applic.*, **13**, 711–740.

S. Mallat (1989) "Multiresolution approximations and wavelet orthonormal bases of $L^2(\mathbb{R})$", *Trans. Amer. Math. Soc.*, **315**, 69–87.

Y. Meyer (1990) "Ondelettes et Opérateurs", *Hermann, Paris.*

M. J. D. Powell (1981) "Approximation theory and methods", *Cambridge University Press, Cambridge.*

O. Rioul (1992) "Simple regularity criteria for subdivision schemes", *to appear in SIAM J. Math. Anal..*

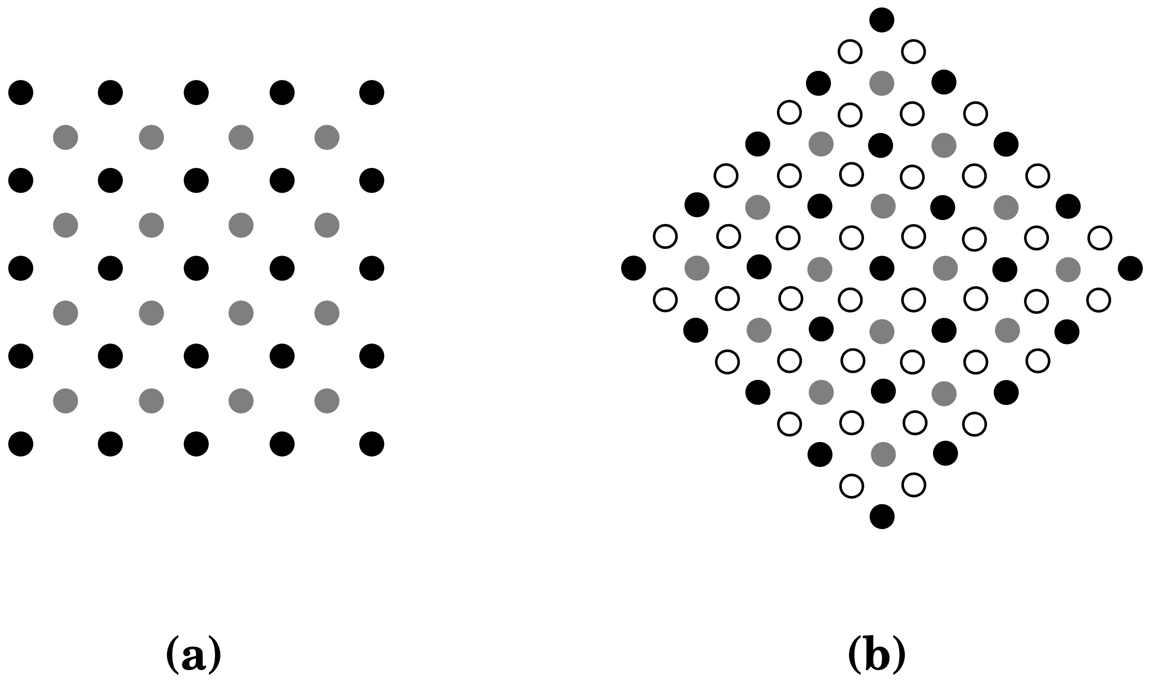

Figure 4. The modified tensor-product scheme.