Efficient Implementations of 2-D Noncausal IIR Filters

Michael M. Daniel, *Student Member, IEEE,* and Alan S. Willsky, *Fellow, IEEE*

***Abstract*— In this paper, we propose a framework for the efficient implementation of two-dimensional (2-D) *noncausal* infinite impulse response (IIR) filters, i.e., filter systems described implicitly by difference equations and *boundary conditions*. A number of common 2-D LSI filter operations, (including low-pass, high-pass, and zero-phase filters), are efficiently realized and implemented in this paper as noncausal IIR filters. The basic concepts involved in our approach include the adaptation of so-called direct methods for solving partial differential equations (PDE's), and the introduction of an approximation methodology that is particularly well suited to signal processing applications and leads to very efficient implementations. In particular, for an input and output with $N \times N$ samples, the algorithm requires only $\mathcal{O}(N^2)$ storage and computations (yielding a per pixel computational load that is independent of image size), and has a parallel implementation (yielding a per pixel computational load that decreases with increasing image size). Also, because our approach allows for the implementation of filters with space-varying coefficients on irregularly shaped domains, it should have applications in related areas like linear estimation, geophysical signal processing, or any field requiring approximate solutions to elliptic PDE's.**

I. Introduction

FOR two-dimensional (2-D) signal processing applications, finite impulse response (FIR) filters have been overwhelmingly preferred to infinite impulse response (IIR) filters [3], [10], [12]. Among the reasons for this preference are: a) FIR filters can be efficiently implemented in both one-dimensional (1-D) and 2-D through the use of the FFT; and b) FIR filters are always stable and do not require any notion of recursion or ordering of the sample points (in 1-D or in 2-D) in order to be implemented. In contrast, 1- and 2-D IIR filters appear to be dramatically different. First, there is usually no natural ordering of the sample points in 2-D, and 2-D IIR filters are difficult to test for stability. More importantly, for 2-D noncausal IIR filters the lack of efficient implementations has both limited the investigation of these filters and led many to argue that they cannot be implemented in practice [3], [10], [12].

To understand these issues, as well as our approach to dealing with them, consider an IIR filter, in 1- or 2-D, specified in terms of a difference equation. In either case, the difference equation by itself isn't sufficient to completely specify the filtering algorithm, as one must also specify a set of **auxiliary conditions**. In 1-D, for the most part these are specified as a set of initial conditions, leading to causally-recursive filtering algorithms with computational load per sample point proportional to the order of the difference equation. Moreover, even for noncausal 1-D filters like zero-phase IIR filters, implementation results in a per-sample computational load proportional to filter order or, equivalently, to the total number of auxiliary (initial and final) conditions, (assuming that the 1-D noncausal IIR filters are implemented through the combination of a causal recursion requiring initial conditions and an anticausal recursion requiring "final" conditions).

In contrast, the dimension of the required auxiliary conditions in 2-D depends not only on the order of the difference equation but also on the size of the boundary. Since the size of the boundary is proportional to the dimensions of the 2-D domain of interest, an apparently significant increase in computational complexity results. In addition, since in most 2-D applications there is no natural ordering of the sample points and no natural direction for recursion, there is no reason to expect that the auxiliary conditions would separate into anything that might resemble "initial" or "final" conditions, but rather would more naturally be distributed around the entire boundary of the 2-D domain, leading to 2-D noncausal IIR (2-DNC-IIR) filters that are not recursively computable.

On the other hand, if effective methods of implementation for 2-DNC-IIR filters were available, there would be numerous possibilities for their application. For example, one potential advantage retained in 2-D for IIR filters is that a given set of frequency response characteristics typically may be met by an IIR filter of considerably lower order than a corresponding FIR design. Moreover, 2-DNC-IIR filters arise naturally in applications such as the modeling of random fields for image processing [1], [2], [11] and computer vision [9]. In this paper we present an efficient implementation of 2-DNC-IIR filters that overcomes the difficulties we have described, thus offering the possibility of recapturing in 2-D the computational advantages and flexibility that IIR filters have in 1-D.

The key to our approach is the recognition of both the similarities and differences between the implementation of 2-DNC-IIR filters and the solution of finite-difference and finite-element approximations of partial differential equations (PDE's). In particular, the equations resulting from such methods are of the same form as those for 2-DNC-IIR filters, and thus the many methods that have been developed for

Manuscript received March 24, 1995; revised October 24, 1996. This work was supported by the Office of Naval Research under Grant N00014-91-J-1004, by the Air Force Office of Scientific Research under Grant F49620-95-1-0083, and by the Army Research Office under Grant DAAL03-92-G-0115 (Center for Intelligent Control Systems). This paper was recommended by Associate Editor W.-S. Lu.

The authors are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: willsky@mit.edu).

Publisher Item Identifier S 1057-7130(97)03652-5.

the efficient solution of PDE's can be used to implement 2-DNC-IIR filters. These methods by themselves, while offering considerable savings in computational complexity and storage, may not reduce these loads enough to make 2-DNC-IIR filters attractive. However, by taking advantage of a fundamental difference in objective between solving PDE's and performing 2-D filtering, we can reduce the computational complexity even further, resulting in implementations with complexity per 2-D data point independent of domain size—the same attractive feature as in 1-D. While the focus in PDE's is typically on obtaining numerically very accurate solutions to specific 2-D difference equations, and hence accurate solutions to the corresponding physical models, in 2-D filtering the difference equation is not the fundamental object. Instead, the initial criterion is a set of filter or frequency response specifications. A 2-D difference equation is then chosen to meet these specifications within some tolerance. Consequently, approximations to the solution of the difference equation are acceptable as long as they lead to filters that also meet the desired tolerances.

In the next section, we introduce the class of 2-DNC-IIR filters and discuss the role of boundary conditions in these systems. In Section III, we make the connection between implementations of 2-DNC-IIR filters and methods for solving sparse linear systems of equations, such as those which arise when solving PDE's. One of these methods involves organizing the 2-D data points into 1-D columns, whose dimensions are equal to the linear dimension of the filtering domain. The PDE or filter solution is then given by processing these columns sequentially. While this algorithm is generally inefficient, if we view each of these sequential processing steps as being itself a 1-D processing procedure **along** the 1-D data set, we are led to the idea of approximating this step using low-order IIR filtering methods. This idea, which is developed in Section IV, results in very efficient 2-DNC-IIR filtering procedures applicable to a large class of noncausal, nonseparable filtering applications. In Section V, the efficient implementation is applied to several 2-DNC-IIR filters, some of which are zero-phase. Zero-phase filters are of considerable interest in practice, and the apparent difficulty in implementing 2-D IIR filters with zero phase has often been cited as one of the reasons that FIR filters are commonly used [10]. We now can implement zero-phase IIR filters efficiently, removing a major obstacle to their use in practice.

II. Two-Dimensional IIR Filters and Boundary Conditions

For a rich class of 2-D IIR filters, the inputs $x[i, j]$ and outputs $y[i, j]$ satisfy linear constant-coefficient difference equations (LCCDE's) of the form

$$\sum_{l_1=-L_1}^{L_1} \sum_{l_2=-L_2}^{L_2} a_{l_1 l_2}\, y[i-l_1, j-l_2] = \sum_{m_1=-M_1}^{M_1} \sum_{m_2=-M_2}^{M_2} b_{m_1 m_2}\, x[i-m_1, j-m_2]. \quad (1)$$

The order of this difference equation is defined to be (L_1, L_2). However, (1) provides only a partial specification of a system, as it must be accompanied by a set of auxiliary conditions. If the filter whose input and output satisfies (1) is stable, then the auxiliary conditions cannot generally be organized as a simple set of "initial" or "final" conditions, as considered in [3], [12], but must be specified around the entire boundary of the 2-D filtering domain; these auxiliary conditions are referred to as boundary conditions (BC's). If BC's are specified, no simple recursive solution is possible, and all of the output values $y[i, j]$ must in principle be computed simultaneously. Algorithms for computing the outputs of such 2-DNC-IIR filters are discussed in Sections III and IV. The remainder of this section addresses the some of the issues raised by the imposition of boundary conditions on 2-D IIR filters.

A fundamental issue is the effect of the boundary conditions upon the response of 2-DNC-IIR filters. The effect of boundary conditions is an important issue for any system defined on a finite domain, even in 1-D and for FIR filters, although this issue is rarely addressed [3], [12]. For IIR filters, both in 1- and 2-D, the method for limiting the effect of the BC's upon the filter output is to require that the system be stable. Not only does stability guarantee that the effect of the BC's will be limited to the boundary regions and decay with distance from the boundary, stability implies a particular choice of BC's. To illustrate this subtle relationship between the choice of BC's and stability, first consider 1-D IIR filters, for which analysis is simpler. For the 1-D IIR filter whose input and output satisfy $y[n] = ay[n-1] + x[n]$, the imposition of stability implies that the boundary condition is either an initial rest or a final rest condition. A value of $|a| < 1$ implies an initial rest condition and $|a| > 1$ implies a final rest condition. For higher-order filters, stability leads to three possible boundary conditions, where again the exact choice of boundary conditions is determined by the filter coefficients: a) if all the poles are inside the unit circle, then the BC's are initial rest conditions; b) if all the poles are outside the unit circle, then the BC's are final rest conditions; and c) if the poles are both inside and outside the unit circle, then the system is noncausal and the BC's correspond to both initial and final rest conditions. The BC's for the third possibility can be seen by splitting the IIR filter into a parallel realization of a causal filter (with poles inside the unit circle) and an anticausal filter (with poles outside the unit circle), where the causal filter satisfies initial rest and the anticausal filter satisfies final rest. This parallel realization is given by a partial fraction expansion of the system's frequency response. The conclusion to be drawn from this discussion is that, for 1-D IIR filters, stability both determines the form of the boundary conditions and limits the effect of the boundary condition upon the system response, i.e., the transients, to the region in which the BC's are applied. However, the rate of decay of the transients is a function of the filter difference equation, e.g., the magnitude of a in the first-order example.

For 2-D IIR filters, the link between stability and boundary conditions is much the same. First of all, given a stable filter satisfying (1), the frequency response follows immediately from the difference equation. Stability in this case corresponds

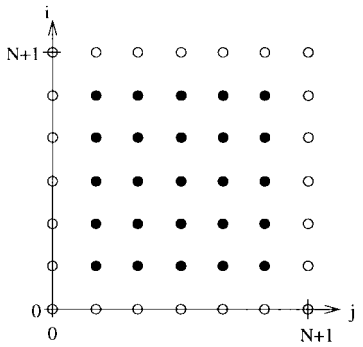

Fig. 1. The elements of Ω_N (denoted by $\bullet$) and $\partial\Omega_N^{(1,1)}$ (denoted by $\circ$), drawn for $N = 5$.

to the usual notion of bounded-input bounded-output stability, as well as to the concept that the effect due to the BC's on the filter response near the center of the 2-D domain decays to zero as the boundaries recede to infinity. Again, the form of the boundary conditions is completely determined by the imposition of stability, and the width of the annular region near the boundaries where the BC's significantly effect the filter response is determined by the coefficients of the 2-D difference equation. However, a major difference between IIR filters in 1- and 2-D is the ability to determine BC's which lead to stable systems. As noted earlier, determining such BC's for 1-D IIR filters is straightforward, and follows from a partial fraction expansion of the frequency response. For 2-D IIR filters, there is no general method for determining the BC's which lead to a stable system. The lack of such method is due to the inability to factor a 2-D system function and to the complexity of the boundaries in 2-D, which can cover large regions and have complicated geometries. However, as shown in the Section V, we can often find boundary conditions which lead to stable filters. As the algorithms proposed in Sections III and IV are motivated by numerical solutions to PDE's, the boundary conditions chosen in Section V are discrete equivalents of boundary conditions which lead to stable solutions of PDE's, where stability again refers to limiting the region in which the BC's significantly effect the solution of the PDE. Two such conditions are Dirichlet and Neumann conditions [15]. For a 2-DNC-IIR filter of order (1, 1), Dirichlet conditions correspond to specifying the value of $y[i, j]$ on the boundary of the filtering domain. For a filter of higher order, Dirichlet conditions correspond to setting the value of $y[n_1, n_2]$ on an annular ring around the boundary of the filtering domain. As will be described more clearly in the next section, the width of this annular ring is a function of the filter order.

Another issue raised by the imposition of boundary conditions is that traditional notions of shift-invariance do not easily extend to 2-DNC-IIR filters. To appreciate this subtle issue, first consider 1-D IIR filters. Shift-invariance for a 1-D difference equation requires that a) the locations of the boundary conditions are not fixed, but instead adjust to the location of the nonzero values of the input, e.g., initial rest conditions, and b) both the input and output are defined for all time, i.e., $-\infty < n < \infty$. As a consequence, the commonly defined property of shift-invariance cannot be applied to systems with inputs and outputs defined only over a finite interval, say $[n_1, n_2]$. For instance, how would one even define the shift of a signal $y[n]$ defined only on $[n_1, n_2]$. One option is to define $x[n]$ to be equal to zero outside $[n_1, n_2]$, and then to compute $y[n]$ for all $-\infty < n < \infty$. To guarantee shift-invariance, the IIR filter is implemented as a parallel realization of a causal IIR recursion initialized by initial rest conditions and an anticausal IIR recursion initialized by final rest conditions. However, while the resulting filter is shift-invariant, such parallel realizations do not generally exist for 2-DNC-IIR filters, unless the filter is separable. Furthermore, there are no analogous notions of initial and final rest for 2-D filters. A notion of shift-invariance which does extend to 2-DNC-IIR filters is given by comparing the following two 1-D signals: 1) the response $y_1[n]$ defined on $[n_1, n_2]$ to an input $x[n]$ defined on $[n_1, n_2]$, and 2) the response $y_2[n]$ defined on $[n_1 + n_0, n_2 + n_0]$ to the input $x[n - n_0]$ which is also defined on $[n_1 + n_0, n_2 + n_0]$. If the system is shift-invariant, then $y_2[n] = y_1[n - n_0]$ for $n \in [n_1 + n_0, n_2 + n_0]$. When extended to 2-D systems, this notion of shift-invariance applies to 2-DNC-IIR filters, i.e., filters satisfying (1) and constrained by boundary conditions.

III. Noncausal IIR Filters as Linear Systems of Equations

A. Direct versus Iterative Methods

In this section we make precise the connection between the problem of implementing 2-DNC-IIR filters and the general problem of solving large, sparse, sets of linear equations, in particular those arising in the solution of linear PDE's. The methods that result from this connection are quite broadly applicable. For example, our methodology can be used for linear difference equations which are **not constant-coefficient**, for regions of support Ω which are nonsquare and irregularly sampled, and for various types of boundary conditions. However, for notational simplicity in this and the following sections, we assume that the difference equation is LCCDE, that the filter domain is square, and that the boundary conditions are of the Dirichlet type. One square domain of $N \times N$ samples is $\Omega_N = \{(i, j) | 1 \le i, j \le N\}$. For an order (L_1, L_2) filter, the corresponding Dirichlet conditions are to set $y[i, j]$ to some known function $r[i, j]$ on the annular ring $\partial\Omega_N^{(L_1, L_2)} = \{(i, j) | (i, j) \notin \Omega_N, -L_1 + 1 \le i \le N + L_1, -L_2 + 1 \le j \le N + L_2\}$. Note that the width of this annular ring is determined by the order of the difference equation. Both Ω_N and $\partial\Omega_N$ are illustrated in Fig. 1 for a $N = 5$ and a filter of order (1, 1).

For clarity of exposition, we also assume that $b_{m_1 m_2} = \delta_{m_1 m_2}$ in (1), where $\delta_{m_1 m_2}$ is the Kronecker delta function. Since implementing the right-hand side of (1) is equivalent to implementing an FIR filter, a more complicated right-hand side adds only notational but not conceptual complexity. These assumptions lead to 2-DNC-IIR filters whose inputs and output satisfy the difference equation

$$\sum_{l_1=-L_1}^{L_1} \sum_{l_2=-L_2}^{L_2} a_{l_1 l_2}\, y[i - l_1, j - l_2] = x[i, j] \qquad (2)$$

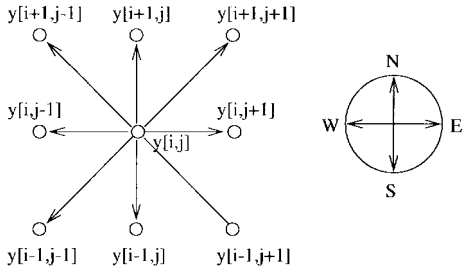

Fig. 2. The output mask for the 9-point NNM difference equation.

for all $(i, j) \in \Omega_N$ and satisfy the Dirichlet conditions $y[i, j] = r[i, j]$ for all $(i, j) \in \partial\Omega_N^{(L_1, L_2)}$.

Equation (2) can be cast in matrix form as

$$\begin{aligned} A\underline{y} &= \underline{x} + \underline{r} \\ &= \underline{b} \end{aligned} \tag{3}$$

where the nonzero elements of A are the filter coefficients $a_{l_1 l_2}$. Vectors $\underline{x}$ and $\underline{y}$ contain the filter input $x[i, j]$ and output $y[i, j]$, respectively, in Ω_N, and $\underline{r}$ contains the contribution of the Dirichlet conditions entering through the filter difference equation. The order in which the variables $y[i, k]$ appear in $\underline{y}$ is the *ordering* of Ω_N, or the ordering of A. For direct methods (see below), this ordering can drastically alter apparent complexity.

Note that A has dimension $N^2 \times N^2$. A nice property of IIR filters is that they generally require a small number of coefficients, so that $L_1 \ll N$ and $L_2 \ll N$. In other words, A will be very sparse. This obviously suggests the use of numerical methods developed for solving large sparse systems which take advantage of this sparsity to minimize computational and storage requirements. In particular, there are two distinct classes of methods for calculating the output $\underline{y}$ in (3)—iterative and direct methods. Iterative methods begin with an estimate $\underline{y}_0$ of $\underline{y}$, and produce at each step an estimate $\underline{y}_k$ which theoretically converges as $\lim_{k\to\infty} \underline{y}_k = \underline{y}$; however, in practice the series must converge within a tolerable error in a finite number of steps. Direct methods consist of variants of the LU factorization [6], [7], and produce the exact solution (disregarding numerical errors) in a finite number of steps.

For signal processing applications, the same filter is typically applied to a large number of inputs, and thus A must be factored only once for a direct method. This factorization can be done *off-line*, i.e., the factorization costs can either be considered part of the filter design process or amortized over the large number of inputs. This property of direct methods motivates us to focus here on direct implementations of 2-DNC-IIR filter systems. Iterative methods, such as preconditioned conjugate gradient or multigrid, might be just as or more effective for some applications, (especially for 3-D problems), but we show here that direct methods allow for very efficient implementations of classes of 2-D filters.

The LU factorization $A = LU$ yields a unit lower-triangular matrix L and an upper-triangular matrix U. Given L and U, the solution to $Ay = b$ can be found very efficiently by sequentially solving the following two triangular systems: $L\underline{\zeta} = \underline{b}$ and $U\underline{y} = \underline{\zeta}$. Solving for $\underline{\zeta}$ is called forward-substitution, while solving for $\underline{y}$ from $\underline{\zeta}$ is called back substitution. Assuming A has dimension $M \times M$, solving for $\underline{y}$ by explicitly computing A^{-1} and then computing $A^{-1}\underline{b}$ requires $2M^3 + 2M^2$ computations (measured in terms of floating point adds and multiplies). If A is dense, the LU factorization approach yields at most minimal computational savings, as the LU factorization alone requires $2M^3/3$ computations, while the substitutions require only $2M^2$ additional computations. However, if A is sparse, as for 2-DNC-IIR filter systems, the savings in both storage and computations can be tremendous, especially if proper orderings are used to minimize the amount of fill-in (loss of sparsity) that occurs during the factorization (see [4]). Amortizing the costs of the factorization over a large number of filter inputs further decreases the effective computation requirements for the LU approach. In our application, however, M is equal to the number of 2-D data points, N^2, and thus computations of order greater than linear in M can still make this approach prohibitive. Fortunately, as we will see, in the context of 2-D filtering there are natural and very accurate approximations to the LU factorization approach that do result in total complexity that is linear in M.

B. Columnwise Orderings

To make the following discussion explicit we focus here on a common 2-D LCCDE of order (1, 1), the 9-point nearest neighbor model (NNM). This difference equation also arises quite frequently in engineering applications [6], [9], [11], [15], most notably as the first-order and second-order finite-difference and finite-element approximations to elliptic PDE's. The constant-coefficient form of the 9-point NNM is given by

$$\begin{aligned} &cy[i, j] + ny[i+1, j] + sy[i-1, j] \\ &\quad + ey[i, j+1] + wy[i, j-1] + n_e y[i+1, j+1] \\ &\quad + n_w y[i+1, j-1] + s_e y[i-1, j+1] \\ &\quad + s_w y[i-1, j-1] = x[i, j]. \end{aligned} \tag{4}$$

The output mask of this difference equation is illustrated in Fig. 2. Note that the LSI system characterized by the frequency response from difference equation (4) is zero-phase if $n = s$,

$$\underbrace{\begin{bmatrix} D_1 & E_2 & & & \\ C_1 & D_2 & E_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & E_N \\ & & & C_{N-1} & D_N \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}}_{\underline{y}} = \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}}_{\underline{x}} + \underline{r} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \\ b_N \end{bmatrix}}_{\underline{b}}. \tag{5}$$

$e = w$, $n_e = s_w$, and $n_w = s_e$. Consider a 2-DNC-IIR filter which satisfies (4) on Ω_N and Dirichlet conditions on $\partial\Omega_N^{(1,1)}$. If $y[i, j]$ and $x[i, j]$ are ordered columnwise into $N \times 1$ dimensional vectors $y_j = [y[1, j], y[2, j], \cdots, y[N, j]]^T$ and $x_j = [x[1, j], x[2, j], \cdots, x[N, j]]^T$, respectively, (3) becomes[1] (5), as shown at the bottom of the previous page. The structure of A in (5) is block tridiagonal, and the $N \times N$ dimensional blocks C_j, D_j, and E_j are tridiagonal. Note that (5) allows for a space-varying NNM difference equation, but for the constant-coefficient difference equation the subscripts on the blocks of A can be dropped. In this case, the nonzero elements of C, D, and E are given by

$$[C]_{kl} = \begin{Bmatrix} s_w, & l = k+1 \\ w, & l = k \\ n_w, & l = k-1 \end{Bmatrix},$$

$$[D]_{kl} = \begin{Bmatrix} s, & l = k+1 \\ c, & l = k \\ n, & l = k-1 \end{Bmatrix},$$

$$[E]_{kl} = \begin{Bmatrix} s_e, & l = k+1 \\ e, & l = k \\ n_e, & l = k-1 \end{Bmatrix}.$$

For filters of order (L_1, L_2), a columnwise ordering of Ω_N leads to a matrix A which has block bandwidth L_2, while each of the blocks has bandwidth L_1. (A matrix D has bandwidth β if element $[D]_{ij}$ is nonzero only for $|i - j| \leq \beta$.)

A simple recursive algorithm can be invoked to compute the factorization of A in (5). This algorithm factors A "block-by-block," and is thus referred to as the block LU factorization [4], [7]. This algorithm recursively computes the matrices $\overline{D}_1, \cdots, \overline{D}_N$ and $\overline{E}_2, \cdots, \overline{E}_N$ using the following equations:[2]

$$\overline{D}_j \overline{E}_{j+1} = E_{j+1} \qquad \text{"compute } \overline{E}_{j+1}\text{"} \quad (6)$$

$$\overline{D}_{j+1} = D_{j+1} - C_j \overline{E}_{j+1}, \qquad \text{"compute } \overline{D}_{j+1}\text{"}. \quad (7)$$

The recursion is initialized with $\overline{D}_1 = D_1$. Solving (6) at each step is performed by an LU factorization on $\overline{D}_j$, and, as we discuss next, this factorization $\overline{D}_j = L_{jj}U_{jj}$ is needed on-line. For these recursions to be well-posed, the matrices $\overline{D}_j$ must be invertible for all $j = 1, \cdots, N$. Conditions which guarantee this are discussed in [7]. For the examples presented in Section V, the matrices $\overline{D}_j$ are invertible.

The block LU factorization resulting from the procedure just described yields

$$A = \begin{bmatrix} \overline{D}_1 & & & \\ C_1 & \overline{D}_2 & & \\ & \ddots & \ddots & \\ & & C_{N-1} & \overline{D}_N \end{bmatrix} \begin{bmatrix} I & \overline{E}_2 & & \\ & \ddots & \ddots & \\ & & I & \overline{E}_N \\ & & & I \end{bmatrix}. \quad (8)$$

While the recursion given by (6) and (7) is conceptually straightforward, its computational load can be overwhelming. This is a direct result of the columnwise ordering, which leads to a destruction of the sparsity of A during the factorization. Although C_j, D_j, and E_j are very sparse, the matrices $\overline{D}_j$ and $\overline{E}_j$ are generally full, with the exception of $\overline{D}_1 = D_1$. Storing each of these matrices requires $\mathcal{O}(N^2)$ storage elements and computing each $\overline{D}_j$ requires $\mathcal{O}(N^3)$ computations, leading to a total of $\mathcal{O}(N^3)$ storage elements and $\mathcal{O}(N^4)$ computations for the entire factorization.

The lack of sparsity in the blocks of (8) also implies a large computational burden for the *on-line solution*. First note that, since the factorization $\overline{D}_j = L_{jj}U_{jj}$ is needed to compute $\overline{E}_{j+1}$ at each step of the recursion, these factors can be stored in place of $\overline{D}_j$ in (8). The solution to (5) is then given by forward-substitution, (initialized by $C_0\zeta_0 = 0$),

$$L_{jj}U_{jj}\zeta_j = b_j - C_{j-1}\zeta_{j-1}, \qquad j = 1, \cdots, N \quad (9)$$

followed by back-substitution, (initialized by $y_N = \zeta_N$)

$$y_j = \zeta_j - \overline{E}_{j+1}y_{j+1}, \qquad j = N-1, \cdots, 1. \quad (10)$$

Since L_{jj} and U_{jj} are generally full, the solution of (9) and (10) requires $\mathcal{O}(N^2)$ computations per step and hence $\mathcal{O}(N^3)$ total computations. Thus, not only is the off-line computational load $\mathcal{O}(N^4)$, but the on-line storage and computations are both $\mathcal{O}(N^3)$, significantly greater than the $\mathcal{O}(N^2)$ goal.

However, if an approximate solution can be tolerated, the block LU factorization based on the columnwise ordering leads to an efficient approximation strategy which achieves the goal of $\mathcal{O}(N^2)$ storage elements and $\mathcal{O}(N^2)$ computations for both the off-line factorization and the on-line solution. In particular, note that (9) requires first solving the lower triangular equations

$$L_{jj}z_j = b_j - C_{j-1}\zeta_{j-1} \quad (11)$$

followed by solving the upper triangular system

$$U_{jj}\zeta_j = z_j. \quad (12)$$

Recall that we have organized our variables into 1-D columns, and thus the solution of the lower triangular system (11) can be thought as a causal 1-D recursion, beginning at the bottom of the column ($i = 1$) and proceeding recursively to the top of the column ($i = N$). The upper triangular system (12) corresponds to an anticausal recursion proceeding from top to bottom. The back-substitution filtering, (10), requires implementing an FIR filter, in the form of a matrix multiplication, along a single column of Ω_N.

Thus we can view (11) and (12) as 1-D recursive filtering operations, albeit shift-varying recursions, since in general L_{jj} and U_{jj} will not be Toeplitz. If L_{jj} and U_{jj} are full, then the order of these recursive filters equals the length N of the column, and it is the need to determine (off-line) and then implement (on-line) these high-order recursions that leads to the severe computational burden. However, if these recursive 1-D filters can be approximated by lower-order recursions—e.g., if $\overline{D}_j$ and hence L_{jj} and U_{jj} can be approximated by **banded** matrices, then both the storage and computational requirements for the forward-substitution phase of the on-line solution can be reduced to $\mathcal{O}(N^2)$. For the back-substitution, the computational burden is governed at each step by multiplication of the matrix $\overline{E}_{j+1}$ with y_{j+1}.

[1] For any matrix in this paper, such as A in (5), block entries not indicated are zero.

[2] The validity of the recursions (6) and (7) can be verified directly by equating A in (5) with the expression in (8).

Since $\overline{E}_{j+1}$ is generally full and not Toeplitz, this operation will require $\mathcal{O}(N^2)$ computations per step. However, if we similarly approximate $\overline{E}_{j+1}$ with a lower-order FIR filter, i.e., by approximating $\overline{E}_{j+1}$ with a banded matrix, the total computational and storage requirements for the on-line solution reduce to the $\mathcal{O}(N^2)$ goal.

Note, however, to reduce the **off-line** computational load to $\mathcal{O}(N^2)$, it is not sufficient that $\overline{D}_j$ and $\overline{E}_j$ are well approximated by matrices with narrow bandwidth; a method must exist for determining these approximate matrices in $\mathcal{O}(N)$ computations per stage. In the next section, we describe such an approximation procedure in detail.

IV. Efficient Implementations of 2-DNC-IIR Filters

A. Development of the Approximate Block LU Algorithm

The approximate implementation of 2-DNC-IIR filters described in this section is motivated by the fact that, for many filters, a small number of elements in the blocks C_j, $\overline{D}_j$, and $\overline{E}_j$ dominate the rest of the elements. An efficient approximation to the on-line solutions follows by setting to zero the insignificant elements of L_{jj}, U_{jj}, and $\overline{E}_j$. Recursions (9) and (10) then can be implemented very efficiently if one takes care to avoid operating on the zero elements.

However, the approach of simply discarding the insignificant elements of (8) is not enough. First, the off-line factorization will still require $\mathcal{O}(N^4)$ computations. Secondly, searching for the significant elements of each block in (8) and storing them in data structures for efficient implementation can be a costly procedure. Many filters, however, have a property which allows us to overcome these difficulties. For these filters, all of the matrices of interest—L_{jj}, U_{jj}, and $\overline{E}_j$—can be well approximated by **banded** matrices of some bandwidth $\beta \ll N$. Thus, we know *a priori* what elements of these matrices must be stored. Since β is not a function of N, each of these matrices has $\mathcal{O}(\beta N)$ nonzero elements.[3] Since there are $\mathcal{O}(N)$ such matrices, the total required storage is $\mathcal{O}(N^2)$, as desired. Furthermore, as we now describe, we can compute these approximations with an overall computational load of $\mathcal{O}(N^2)$.

The key assumption required for these approximations to yield good results is that, for $j = 1, \cdots, N$, the blocks $\overline{D}_j^{-1}$ are *approximately banded*, i.e., well approximated by setting to zero all the elements which do not fall within a small bandwidth of the main diagonal. If $\overline{D}_j^{-1}$ is approximately banded, then from the recursions (6) and (7), it is apparent that the blocks $\overline{D}_{j+1}$ and $\overline{E}_{j+1}$ will generally be approximately banded as well. Furthermore, as the following corollary states, if we have a banded approximation to $\overline{D}_j$, we can efficiently compute a banded approximation of its inverse.

Corollary of [5] (See the Appendix for Proof): If D is an $N \times N$ matrix with bandwidth β, the elements of D^{-1} which lie within the bandwidth β can be computed (exactly) in $\mathcal{O}(\beta^2 N)$ operations.

[3] By $\mathcal{O}(N^p)$ computations we mean that the number of computations divided by N^p tends to a constant as $N \to \infty$; writing $\mathcal{O}(\beta^q N^p)$ denotes that this constant is a polynomial in β of order q.

This leads to the following approximation to (6) and (7). Suppose that $\tilde{D}_j$ is an approximation to $\overline{D}_j$ which is β-*banded* (i.e., banded with bandwidth β). Note that $\overline{D}_1 = D_1$ is exactly banded, so we set $\tilde{D}_1 = \overline{D}_1$. We then compute a β-banded approximation of $\tilde{D}_j^{-1}$:

$$F(\tilde{D}_j, \beta) = \begin{cases} [\tilde{D}_j^{-1}]_{kl}, & |k-l| \le \beta \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where, from the corollary, F: $\mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ requires $\mathcal{O}(\beta^2 N)$ computations. Assuming that $\tilde{D}_j^{-1}$ is a good approximation to $\overline{D}_j^{-1}$ and that $F(\tilde{D}_j, \beta)$ is a good approximation to $\tilde{D}_j^{-1}$, then $F(\tilde{D}_j, \beta) \cdot E_{j+1} \approx \overline{E}_{j+1}$. Since E_{j+1} is tridiagonal, the product $F(\tilde{D}_j, \beta) \cdot E_{j+1}$ requires $\mathcal{O}(\beta N)$ computations. Note, however, that this product has bandwidth $(\beta+1)$, which will in turn require a growing bandwidth at each step. However, under the key assumption that the matrices of interest are approximately β-banded, we can neglect elements outside the β-bandwidth. Thus, define the truncation operator T_β: $\mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ as

$$T_\beta[H] = \begin{cases} H_{kl}, & |k-l| \le \beta, \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

which yields the following approximation to (6):

$$\tilde{E}_{j+1} = T_\beta[F(\tilde{D}_j, \beta) \cdot E_{j+1}]. \tag{15}$$

Similarly, substituting $\tilde{E}_{j+1}$ into (7) in lieu of $\overline{E}_{j+1}$ yields $D_{j+1} - C_j \tilde{E}_{j+1} \approx \overline{D}_{j+1}$, requiring $\mathcal{O}(\beta N)$ calculations. Once again, this yields a matrix that is $(\beta+1)$-banded, and applying our assumption of β-bandedness, we obtain our approximation to (7):

$$\tilde{D}_{j+1} = T_\beta[D_{j+1} - C_j \tilde{E}_{j+1}]. \tag{16}$$

Thus, (15) and (16) can be repeated iteratively, each stage requiring $\mathcal{O}(\beta^2 N)$ computations. For $j = 1, \cdots, N$, (15) and (16) form an approximation to (8) requiring a total of $\mathcal{O}(\beta^2 N^2)$ computations and $\mathcal{O}(\beta N^2)$ storage elements. This procedure results in the following approximation of (8):

$$A \approx \begin{bmatrix} \tilde{D}_1 & & & \\ C_1 & \tilde{D}_2 & & \\ & \ddots & \ddots & \\ & & C_{N-1} & \tilde{D}_N \end{bmatrix} \begin{bmatrix} I & \tilde{E}_2 & & \\ & \ddots & \ddots & \\ & & I & \tilde{E}_N \\ & & & I \end{bmatrix} \tag{17}$$

with β-banded $\tilde{L}_{jj}$ and $\tilde{U}_{jj}$ stored in place of $\tilde{D}_j$, where $\tilde{L}_{jj}\tilde{U}_{jj}$ is the LU factorization of $\tilde{D}_j$. The approximate on-line solution is given by substituting $\tilde{L}_{jj}$ and $\tilde{U}_{jj}$ into (9) for L_{jj} and U_{jj}, and $\tilde{E}_j$ into (10) for $\overline{E}_j$.

Note that this approximation method extends equally well to higher-order filters, with the elements in Ω_N are again ordered columnwise. Each of the blocks in the block LU factorization is analogously approximated by a banded matrix. This extension of the approximation algorithm is demonstrated for an order (2, 2) filter in Section V.

What follows is an example intending both to suggest the class of filters for which this approximate implementation will offer significant savings and to justify why in such cases we expect to be able to choose a small value of β independent of the size N of the image domain.

B. Analysis of the Block LU Approximation

Consider a 2-DNC-IIR filter satisfying (5). To make subsequent analysis simpler, we make a slight deviation from the constant-coefficient model of (4). Namely, assume that $c = (1+\alpha^2)$ and $n = s = \alpha$ for all $(i,j) \in \Omega_N$ and $(i,j) \neq (1,1)$. For $(i,j) = (1,1)$, we assume $c = 1$ and $n = s = \alpha$. For the point we wish to make, the values of the other NNM coefficients are of no consequence. The first block row of (5) follows as:

$$\underbrace{\begin{bmatrix} 1 & \alpha & 0 & 0 & \cdots & 0 \\ \alpha & 1+\alpha^2 & \alpha & 0 & \cdots & 0 \\ 0 & \alpha & 1+\alpha^2 & \alpha & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \alpha & 1+\alpha^2 & \alpha \\ 0 & \cdots & 0 & 0 & \alpha & 1+\alpha^2 \end{bmatrix}}_{\overline{D}_1} \cdot y_1 + E_2 y_2 = b_1. \tag{18}$$

The factorization of A in (5) begins by factoring $\overline{D}_1$:[4]

$$\overline{D}_1 = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \alpha & 1 & 0 & \cdots & 0 \\ 0 & \alpha & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha & 1 \end{bmatrix}}_{L_{11}} \underbrace{\begin{bmatrix} 1 & \alpha & 0 & \cdots & 0 \\ 0 & 1 & \alpha & \ddots & \vdots \\ 0 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \alpha \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}}_{U_{11}}. \tag{19}$$

The next step of the factorization is to compute $\overline{E}_2$. Note that, even though L_{11} and U_{11} are banded, this operation will require $\mathcal{O}(N^2)$ computations. Also note that $\overline{D}_1^{-1} = U_{11}^{-1} U_{11}^{-T}$, where

$$U_{11}^{-1} = \begin{bmatrix} 1 & -\alpha & \alpha^2 & \cdots & (-\alpha)^{N-1} \\ 0 & 1 & -\alpha & \ddots & \vdots \\ 0 & 0 & 1 & \ddots & \alpha^2 \\ \vdots & \vdots & \ddots & \ddots & -\alpha \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}. \tag{20}$$

The simple form of (20) leads to the expression $[\overline{D}_1^{-1}]_{kl} = (-\alpha)^{l-k} \sum_{l=0}^{N-l} (-\alpha)^{2l}$. From this expression, we see that for $|\alpha| < 1$ the values of $\overline{D}_1^{-1}$ decay geometrically away from the main diagonal. Thus, the smaller the value of $|\alpha|$, (equivalently, the greater the diagonal dominance of $\overline{D}_1$), the tighter the clustering of significant values about the main diagonal of $\overline{D}_1^{-1}$, and thus the smaller the bandwidth β needed to approximate $\overline{D}_1^{-1}$ at a desired level of accuracy.

While this example is particularly simple, it does serve to demonstrate the intuition and plausibility of our approximation. Moreover, a general guideline verified by extensive numerical simulations, some of which are given in Section V, is that the approximate factorization applies to order (1,1) filters for which $|n| + |s| < |c|$ and becomes more accurate (for fixed β) as the ratio $(|n| + |s|)/|c|$ decreases. These observations are consistent with the preceding example, since $(|n| + |s|)/|c| \to 0$ as $|\alpha| \to 0$. Also note that $|c|/(|n| + |s|)$ is a measure of the "degree" of diagonal dominance of the elements in the blocks D_j. In fact, for 2-DNC-IIR filters of any order, the diagonal dominance of the blocks D_j seems to be a useful guideline for determining which filters can be implemented by the algorithm in Section IV-A. Determining the exact class of filters which can be approximated by this algorithm is beyond the scope of this paper.

C. A Parallel Approximate Implementation

In this section, we briefly discuss a straightforward parallelization of the algorithm discussed in Section IV-A. In particular, a variant of the serial block LU factorization is cyclic block reduction [4], which is easily implemented in parallel. The block LU factorization proceeds by eliminating columns y_j sequentially from $j = 1$ to $j = N$. However, it is possible to eliminate columns in the interior of Ω independently. For example, consider again the block tridiagonal matrix A given in (5). Assume, for simplicity, that N is odd. If we order the even columns last and the odd columns first, (5) takes the form

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} y_{\text{odd}} \\ y_{\text{even}} \end{bmatrix} = \begin{bmatrix} b_{\text{odd}} \\ b_{\text{even}} \end{bmatrix}. \tag{21}$$

Because of the coupling implied by the NNM difference equation, the elimination of the odd columns of y_j for the block factorization of (21) can be performed in parallel. Upon completing this step, the second block equation of (21) becomes (22), shown at the bottom of the page, where the superscript $^{(l)}$ is used to relabel the variables after the l-stage of the cyclic block reduction. The blocks of $A_{22}^{(1)}$ are given by

$$\begin{aligned} D_j^{(1)} &= D_j - C_{j-1} D_{j-1}^{-1} E_j \\ &\quad - E_{j+1} D_{j+1}^{-1} C_j, \qquad j = 2, 4, \cdots, N-1, \\ C_j^{(1)} &= -C_{j+1} D_{j+1}^{-1} C_j, \qquad j = 2, 4, \cdots, N-3, \\ E_j^{(1)} &= -E_{j-1} D_{j-1}^{-1} E_j, \qquad j = 4, 6, \cdots, N-1. \end{aligned} \tag{23}$$

$$\underbrace{\begin{bmatrix} D_2^{(1)} & E_4^{(1)} & & & \\ C_2^{(1)} & D_4^{(1)} & E_6^{(1)} & & \\ & \ddots & \ddots & \ddots & \\ & & C_{N-5}^{(1)} & D_{N-3}^{(1)} & E_{N-1}^{(1)} \\ & & & C_{N-3}^{(1)} & D_{N-1}^{(1)} \end{bmatrix}}_{A_{22}^{(1)}} \underbrace{\begin{bmatrix} y_2 \\ y_4 \\ \vdots \\ y_{N-3} \\ y_{N-1} \end{bmatrix}}_{y_{\text{even}}} = \underbrace{\begin{bmatrix} b_2^{(1)} \\ b_4^{(1)} \\ \vdots \\ b_{N-3}^{(1)} \\ b_{N-1}^{(1)} \end{bmatrix}}_{b_{\text{even}}^{(1)}} \tag{22}$$

[4] The Toeplitz structure of L_{11} and U_{11} is a result of the particular choice of difference equation, but is not a general property even of filters described by constant-coefficient difference equations.

Since $A_{22}^{(1)}$ is again block tridiagonal, the odd-even reordering can continue recursively, where roughly one half of the remaining columns are eliminated in parallel at each stage of the algorithm. Rather than the N stages required for the block LU algorithm (and corresponding on-line solution), approximately $\log_2 N$ stages are required for block cyclic reduction, and each of the columns in each stage can be operated on in parallel. If more coarse-grained parallelism is required, Ω_N can be partitioned into M regions, where the columns in each region can be eliminated independent of the other regions [6]. (In the case of cyclic block reduction, $M = \lfloor (N+1)/2 \rfloor$.)

An approximate block cyclic reduction algorithm follows for any of these parallel structures by noting the strong similarities between implementing (23) and (6), (7). Namely, if a processor is allocated for each of the odd columns of Ω_N, the first stage of the (parallelized) approximate cyclic block reduction is

(a) factor $D_j$and compute $F(D_j, \beta)$
on processors $j = 1, 3, \cdots, N$;

(b) compute $\tilde{C}_{j-1} = F(D_j, \beta)C_{j-1}$
on processors $j = 3, 5, \cdots, N$;
compute $\tilde{E}_{j+1} = F(D_j, \beta)E_{j+1}$
on processors $j = 1, 3, \cdots, N-2$;

(c) compute $\tilde{E}_{j+1}^{(1)} = -T_\beta[E_j\tilde{E}_{j+1}]$
on processors $j = 3, 5, \cdots, N-2$;
compute $\tilde{C}_{j-1}^{(1)} = -T_\beta[C_j\tilde{C}_{j-1}]$
on processors $j = 3, 5, \cdots, N-2$;

(d) compute $G_{j+1} = C_j\tilde{E}_{j+1}$
on processors $j = 1, 3, \cdots, N-2$;
compute $J_{j-1} = E_j\tilde{C}_{j-1}$
on processors $j = 3, 5, \cdots, N$;

(e) compute $\tilde{D}_{j+1}^{(1)} = T_\beta[D_{j+1} - G_{j+1} - J_{j+1}]$
on processors $j = 1, 3, \cdots, N-2$.

Note that Step (e) requires communication between the processors, since G_{j+1} and J_{j+1} are computed on different, but "neighboring," processors. For the first stage, the computational load for each processor is $\mathcal{O}(\beta^2 N)$, where the asymptotic complexity is determined primarily by Step (a). The computational load for each processor will remain constant for subsequent stages of the algorithm. Ignoring interprocessor communication costs after each stage of the recursion, the total factorization will require $\mathcal{O}(\beta^2 N \log_2 N)$ computations. Since there are N^2 pixels, the per pixel computation time for the fully parallel implementation is $\mathcal{O}(\beta^2 \log_2 N/N)$, which **decreases** with increasing image size. This algorithm extends to higher-order filters just as cyclic reduction can be extended to matrices A with larger block bandwidth.

V. Examples

In this section, 2-DNC-IIR filters are implemented[5] using the approximation algorithm of Section IV-A. The 2-DNC-IIR filters are assumed to satisfy (2) on Ω_N and homogeneous Dirichlet conditions on $\partial\Omega_N^{(L_1, L_2)}$. The system functions of these filters have the form

$$H(z_1, z_2) = \frac{1}{\underline{z}_1^T J \underline{z}_2}$$
$$\underline{z}_1 = \begin{bmatrix} z_1^{L_1} \\ z_1^{L_1-1} \\ \vdots \\ z_1^{-L_1} \end{bmatrix}$$
$$J = \begin{bmatrix} a_{L_1,-L_2} & a_{L_1,-L_2+1} & \cdots & a_{L_1,L_2} \\ a_{L_1-1,-L_2} & a_{L_1-1,-L_2+1} & \cdots & a_{L_1-1,L_2} \\ \vdots & \vdots & & \vdots \\ a_{-L_1,-L_2} & a_{-L_1,-L_2+1} & \cdots & a_{-L_1,L_2} \end{bmatrix}$$
$$\underline{z}_2 = \begin{bmatrix} z_2^{-L_2} \\ z_2^{-L_2+1} \\ \vdots \\ z_2^{L_2} \end{bmatrix} \tag{24}$$

where z_1 is the frequency in the i-direction and z_2 is the frequency in the j-direction. Adding a polynomial $\underline{z}_1^T P \underline{z}_2$ to the numerator of (24) would obviously allow one to sharpen the filter frequency responses, such as by narrowing the transition regions or placing zeros in the stopbands [8], [16]; however, the purpose of the following examples is only to demonstrate the utility of the approximation given in Section IV-A and the viability of 2-DNC-IIR filters. Thus it makes sense to implement only the recursive portion of the difference equation.

Some example filters are given by the following coefficient matrices:

$$J_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$
$$J_2 = \tfrac{1}{9}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 5 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$
$$J_3 = \begin{bmatrix} -0.13 & 0.5 & -0.37 \\ -0.50 & 2.0 & -0.50 \\ -0.13 & 0.5 & -0.37 \end{bmatrix} \tag{25}$$

$$J_4 = \begin{bmatrix} -0.2304 & 0.3426 & 0.6967 & 0.3426 & -0.2304 \\ 0.3426 & -1.1575 & -2.0846 & -1.1575 & 0.3426 \\ 0.6967 & -2.0846 & 9.3618 & -2.0846 & 0.6967 \\ 0.3426 & -1.1575 & -2.0846 & -1.1575 & 0.3426 \\ -0.2304 & 0.3426 & 0.6967 & 0.3426 & -0.2304 \end{bmatrix} \tag{26}$$

[5] All of the Matlab files used to generate the examples in the section are available by anonymous **ftp** at **lids.mit.edu** in the directory **pub/ssg/outgoing**.

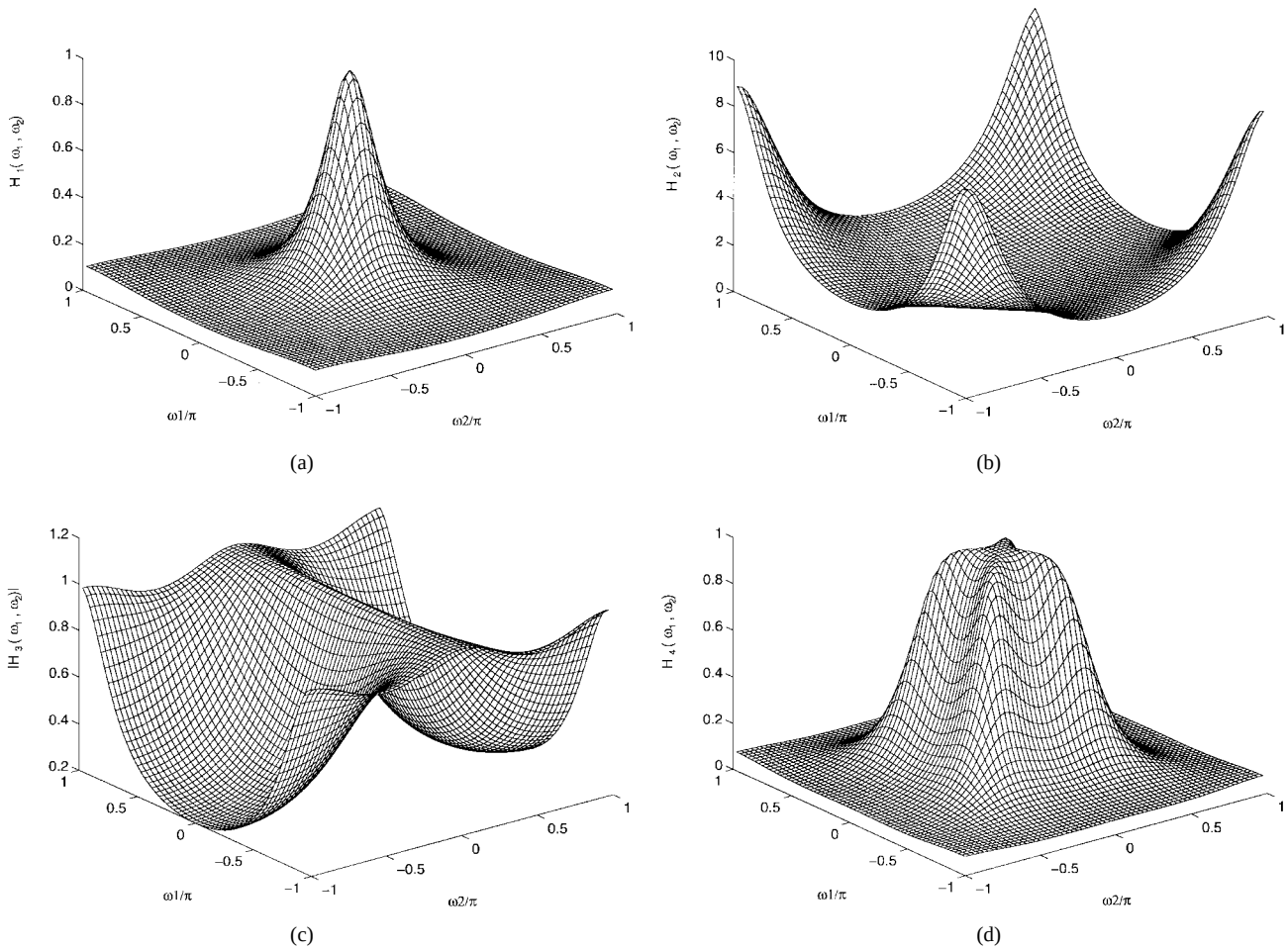

Fig. 3. 64×64 point sampling of $H_i(e^{j\omega_1}, e^{j\omega_2})$ for four filters. (a) Low-pass filter H_1 given by J_1. (b) High-pass filter H_2 given by J_2. (c) Fan filter H_3 given by J_3. (d) Low-pass filter H_4 given by J_4. Note that the only frequency with nonzero phase is H_3, in which case the magnitude of H_3 is plotted.

where $H_i(z_1, z_2) = (\underline{z}_2^T J_i \underline{z}_1)^{-1}$. The coefficients of each filter are normalized such that $H(1, 1) = 1$. The frequency responses $H_i(e^{j\omega_1}, e^{j\omega_2})$ of all four difference equations are illustrated in Fig. 3. Filters H_1 and H_4 are low-pass filters, H_2 is the edge-enhancing filter given in [13], and H_3 corresponds to a primitive (low-order) fan filter [8]. Note that filters H_1, H_2, and H_4 have zero phase.

Note that, while filters H_1, H_2, and H_3 have relatively nonsharp transition bands, they have many practical applications. Edge enhancement is one application requiring filters with nonsharp transition bands. In fact, all of the edge-enhancing filters given in [12], [13], including system H_2, have frequency responses with nonsharp transition bands. When enhancing edges, the width of the transition band is determined by the model of the smooth edges which are to be enhanced. This model usually implies a frequency response with a smooth transition band. The smooth transition band can also be used to avoid "over-enhancing" the image, or to make the enhanced image less sensitive to noisy data. Another application requiring filters with smooth frequency responses is optimal linear estimation. The low-pass filters given by H_1 and H_4 are of exactly the same order and structure as those commonly arising in the estimation of Markov random fields from noisy measurements [1], [14].

In the following examples, the four 2-DNC-IIR filters are implemented exactly (to within round-off error) using the nested dissection algorithm [6] and approximately using the algorithm of Section IV-A. For analyzing the errors introduced by the approximate implementation, define the error signal to be $e_\beta[i, j] = y[i, j] - y_\beta[i, j]$, where $y[i, j]$ is the exact filter output and $y_\beta[i, j]$ is the output obtained with an approximation bandwidth of β. Two of the error measures used are

$$\varepsilon_\beta = \frac{||\underline{e}_\beta||_2}{||\underline{y}||_2}$$

and

$$\eta_\beta = \frac{||\underline{e}_\beta||_1}{||\underline{y}||_1} \tag{27}$$

where $\underline{e}_\beta$ is a vector containing $e_\beta[i, j]$ for all $(i, j) \in \Omega_N$, and $\| \cdot \|_p$ is the standard l_p norm.

Example 1. The Response of 2-DNC-IIR Filters to Sinusoidal Inputs: In this example, we consider the response of the 2-DNC-IIR example filters to sinusoidal inputs of the form

$$x[i, j] = \frac{1}{N^2} \cos\left(\frac{2\pi k_1}{N} i\right) \cos\left(\frac{2\pi k_2}{N} j\right) \quad (i, j) \in \Omega_N. \tag{28}$$

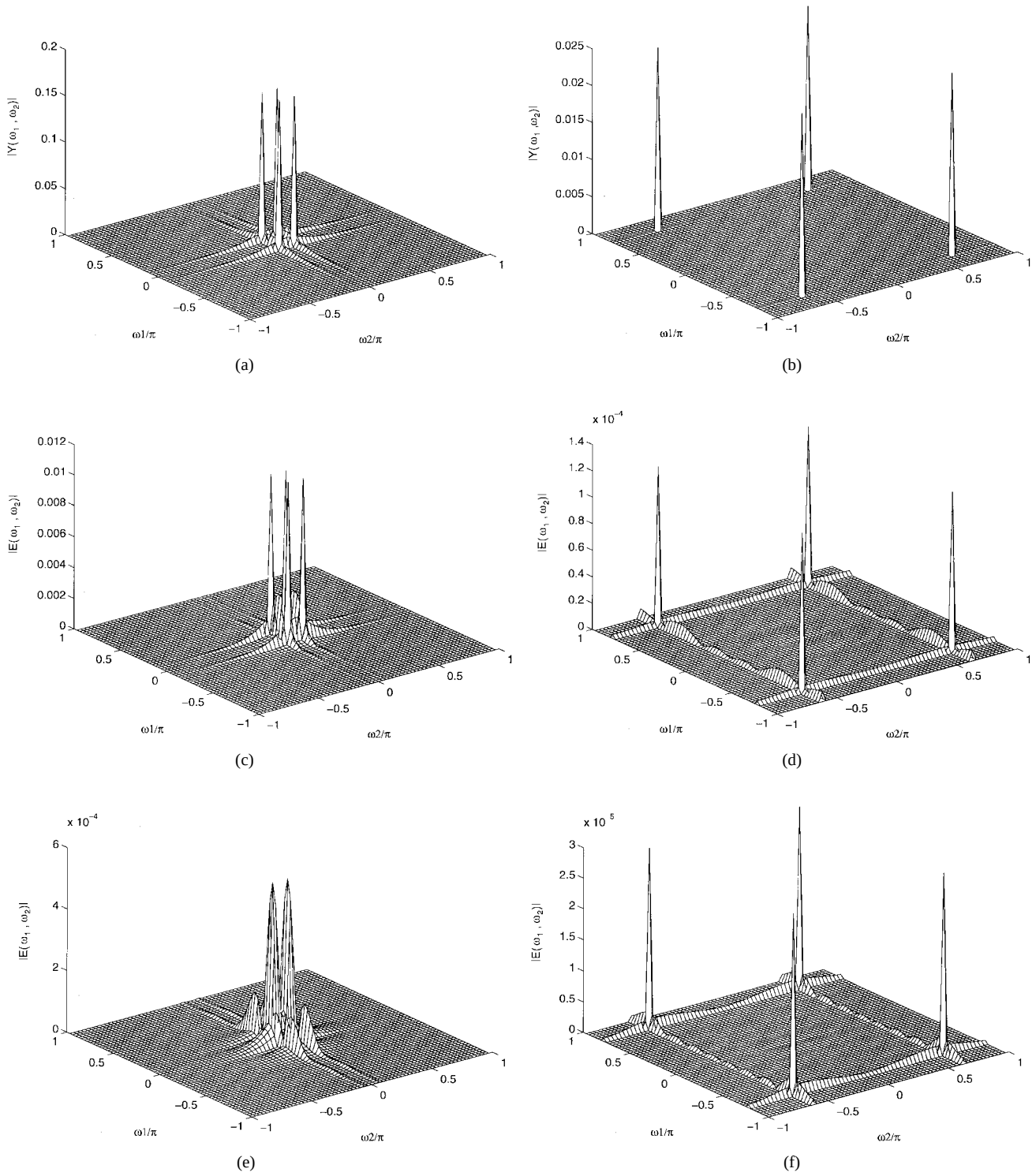

Fig. 4. The DFT magnitude of the exact responses and the approximation errors for a passband input (left column) and a stopband input (right column) to the 2-DNC-IIR **low-pass** filter given by $H_1(e^{j\omega_1}, e^{j\omega_2})$. The inputs are given by (28) for $(k_1, k_2) = (3, 2)$ and $(k_1, k_2) = (25, 20)$. Note the scalings of the vertical axes. (a) Response to pass-band signal. (b) Response to stop-band signal. (c) Error in pass-band response for $\beta = 2$. (d) Error in stop-band response for $\beta = 2$. (e) Error in pass-band response for $\beta = 4$. (f) Error in stop-band response for $\beta = 4$.

The $1/N^2$ term serves only to normalize the discrete Fourier transform (DFT) of $x[i, j]$. Assume for this example that $N = 64$. Consider first the low-pass filter H_1. For $(k_1, k_2) = (3, 2)$, the sinusoidal input lies in the filter's passband, while for $(k_1, k_2) = (25, 20)$ the input lies in the stopband. The exact responses to these two inputs are illustrated by Fig. 4(a) and (b). The Fourier transform $Y(e^{j\omega_1}, e^{j\omega_2})$ is shown in lieu of $y[i, j]$ in order to demonstrate the 9-to-1 (passband to stopband) selection ratio of the low-pass filter. The small ridges in the DFT's of the filter responses are due to the transients introduced by the Dirichlet boundary conditions. These ridges are barely noticeable for the response

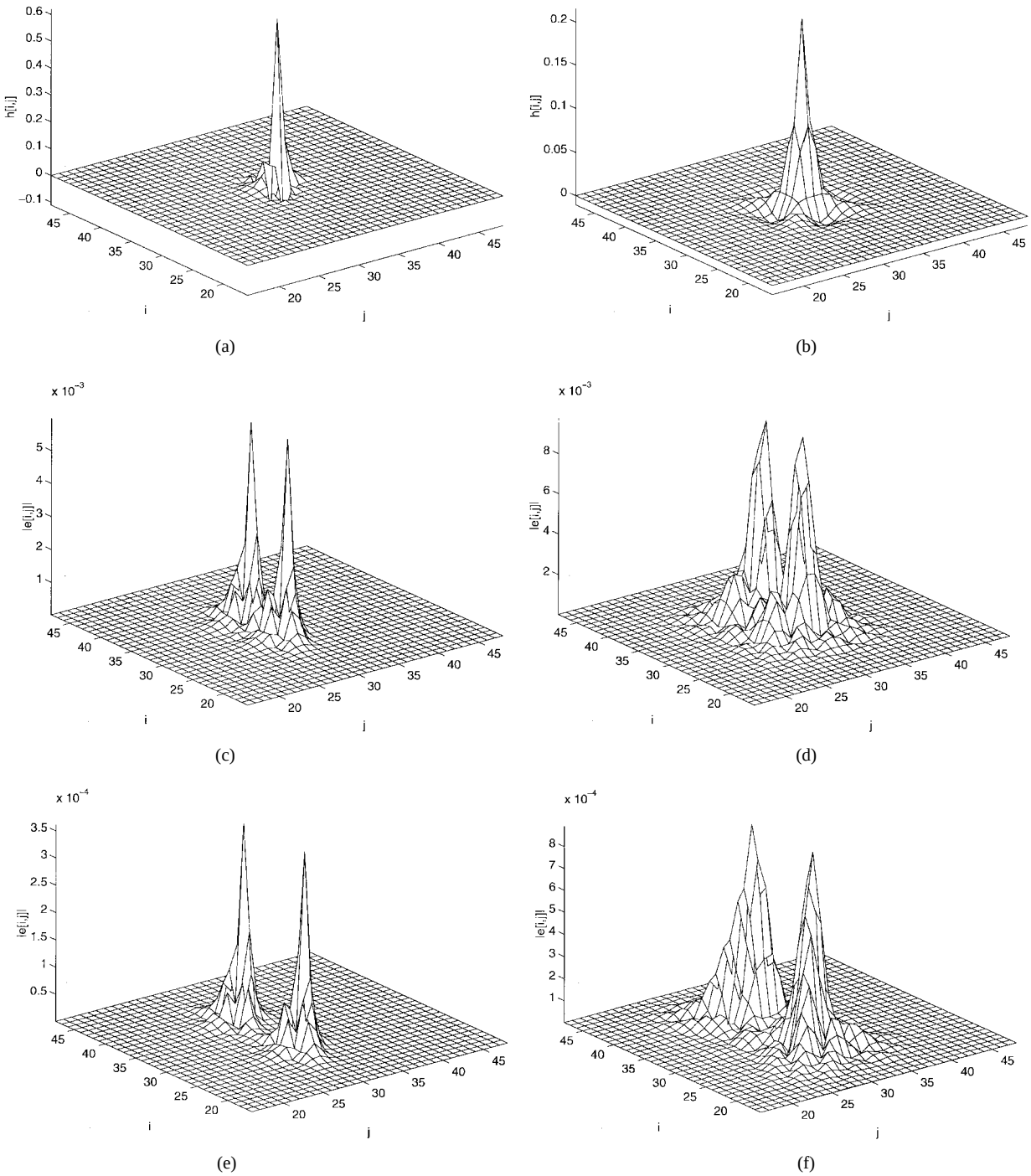

Fig. 5. (a) The impulse response of the **fan** filter H_3. (b) The impulse response of the **low-pass** filter H_4. (c) The error in the **fan** filter impulse response for $\beta = 2$. (d) The error in the **low-pass** filter impulse response for $\beta = 2$. (e) The error in the **fan** filter impulse response for $\beta = 4$. (f) The error in the **low-pass** filter impulse response for $\beta = 4$.

to the stopband input. The effect of these transient signals is negligible near the center of the filter domain Ω_N, and in fact becomes zero as the boundaries recede to infinity ($N \to \infty$).

The accuracy of the algorithm of Section IV-A is also demonstrated in Fig. 4 for approximation bandwidths $\beta = 2$ and 4. The DFT of the approximation error $e_\beta[i, j]$ is illustrated for both inputs and both values of β. Note that the approximation errors are small even when $\beta = 2$, and they decrease by an order of magnitude when β increases from 2 to 4. (In terms of the error metrics, $\varepsilon_2/\varepsilon_4 = 16$ and $\eta_2/\eta_4 = 22$ for the response to the passband input.) A similar geometric decrease in the error signal is obtained for larger values of β.

As verified by extensive numerical simulation, these results generalize to every sinusoidal signal input to each of the four example filters. Namely, for every sinusoidal input to the 2-DNC-IIR example filters, the filter response is given by a weighting of the input by the frequency response, plus some transients whose effect is limited to the boundaries of the filtering domain. The effect of these transients on the filter output near the center of Ω_N decreases to zero as the boundary

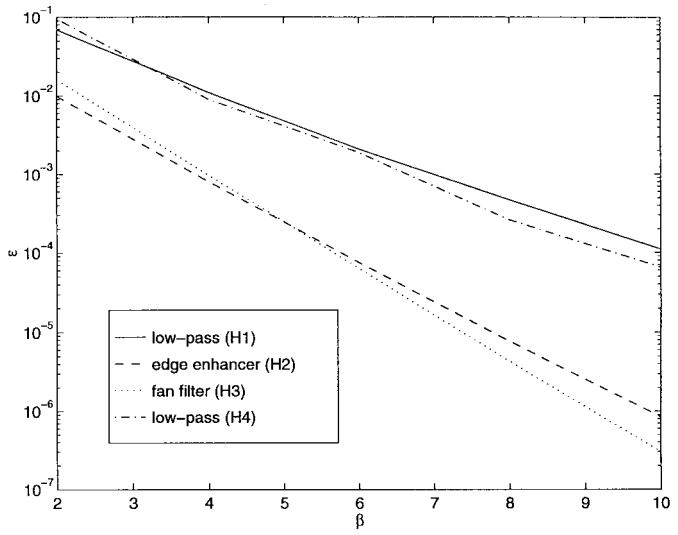

Fig. 6. Approximation errors ε_β versus β, for $N = 64$ and a unit impulse input applied at $(i, j) = (32, 32)$.

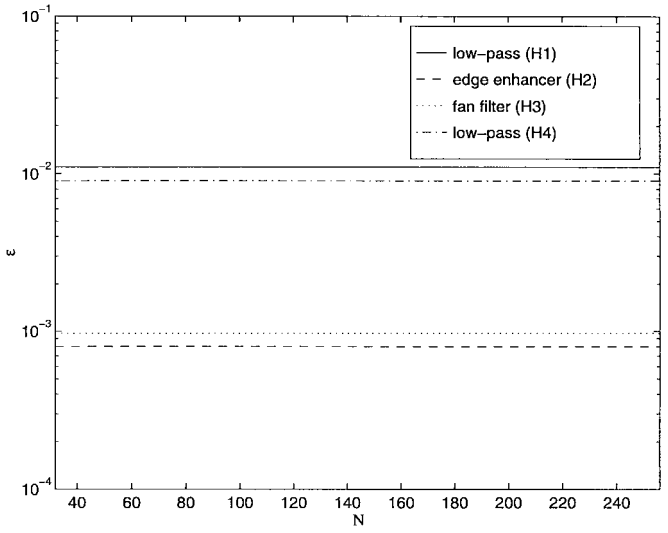

Fig. 7. Approximation errors ε_β versus N. The approximation bandwidth is fixed at $\beta = 4$ and the input is a unit impulse applied at $(i, j) = (N/2, N/2)$.

recedes to infinity ($N \to \infty$). Also, for each sinusoidal input, the approximate response is accurate even for small values of β. More importantly, the approximation errors decrease geometrically in magnitude with β. This geometric decrease is further demonstrated in the following examples.

Example 2. The Impulse Response of 2-DNC-IIR Filters: Because LSI filters are completely characterized by their impulse responses, we now consider the impulse responses of both the exact and the approximate implementations of the example 2-DNC-IIR filters. The impulse response $h[i, j]$ is the filter output in response to a unit impulse applied at the center of Ω_N. For $N = 64$, this input is $x[i, j] = \delta[i-32, j-32]$. The impulse response of both the fan filter H_3 and the order (2,2) low-pass filter H_4 are plotted at the top of Fig. 5. Because the impulse responses are essentially zero outside the region $\{(i, j) | 16 \le i, j \le 48\}$, the responses are not plotted outside this region. The error signals plotted in Fig. 5 correspond to the difference between the true impulse responses and those of the approximate implementations for $\beta = 2$ and $\beta = 4$. Like the approximate responses to the sinusoidal inputs given in Example 1, the approximate impulse responses are accurate even for $\beta = 2$, and the errors the decrease geometrically with increasing β. Note especially the accuracy of the β-banded approximation for the order (2,2) low-pass filter. While the frequency response of this system is very sensitive to the values of the coefficients stored in J_4, the approximate implementation is quite accurate even for small β. For $\beta = 4$, the maximum value of $|e_4[i, j]|$ is on the order of 10^{-4}.

To illustrate that the errors decrease geometrically as β increases beyond four, ε_β is plotted versus β in Fig. 6 for β ranging from two to ten. Fig. 6 also demonstrates that the geometric decrease in the error with increasing β applies to the other three example filters. [Recall that in Section IV-B we argued that the approximation errors will be small when $|n| + |s| < |c|$. For these order (1,1) filters, the elements in the blocks $\overline{D}_j$ and $\overline{E}_j$ decreased geometrically in magnitude with distance from the diagonal. Thus, for β-banded approximations of these blocks, one would expect the approximation errors to decrease geometrically with increasing approximation bandwidth.]

Example 3. The Independence of the Approximation Accuracy upon N: In Section IV-A, the computational and storage loads of the approximation algorithm were shown to be $\mathcal{O}(\beta^2 N^2)$ and $\mathcal{O}(\beta N^2)$, respectively, meaning that the **per pixel** computational and storage loads converge to polynomials in β of order two and one, respectively. However, if the per pixel computational and storage loads are to be truly constant, the approximation bandwidth needed for a desired approximation accuracy must not be an increasing function of N. For a unit impulse applied at the center of Ω_N, Fig. 7 shows that the approximation error metric ε_β remains constant over a wide range of N for a fixed value of $\beta = 4$. Identical results are obtained for other values of β. The independence of β upon N for a desired solution accuracy is consistent with the analysis of Section IV-B, where the rate of decrease with distance from the diagonal of the elements in the blocks $\overline{D}_j$ and $\overline{E}_j$ was shown to be independent of N. Thus, the approximation algorithm has constant per pixel computational and storage loads.

Example 4. Edge Enhancement of a Square Pulse: For this example, we examine the response of the edge-enhancing filter H_2 to a square pulse. The square pulse input and the response of the edge-enhancing filter are illustrated at the top of Fig. 8. Note that the response to the square input is analogous to the step response of the filter. The approximate responses for $\beta = 2$ and 4, and the corresponding approximation errors, are also illustrated in Fig. 8. Again, the approximation errors are small and visually unrecognizable even for these small approximation bandwidths, and the errors decrease by an order of magnitude as β increases from 2 to 4.

VI. Conclusion

In this paper we describe an approach to the efficient implementations of 2-DNC-IIR filters. In addition to efficiency, we were also motivated to consider filters specified by boundary rather than initial conditions, as the former are frequently the

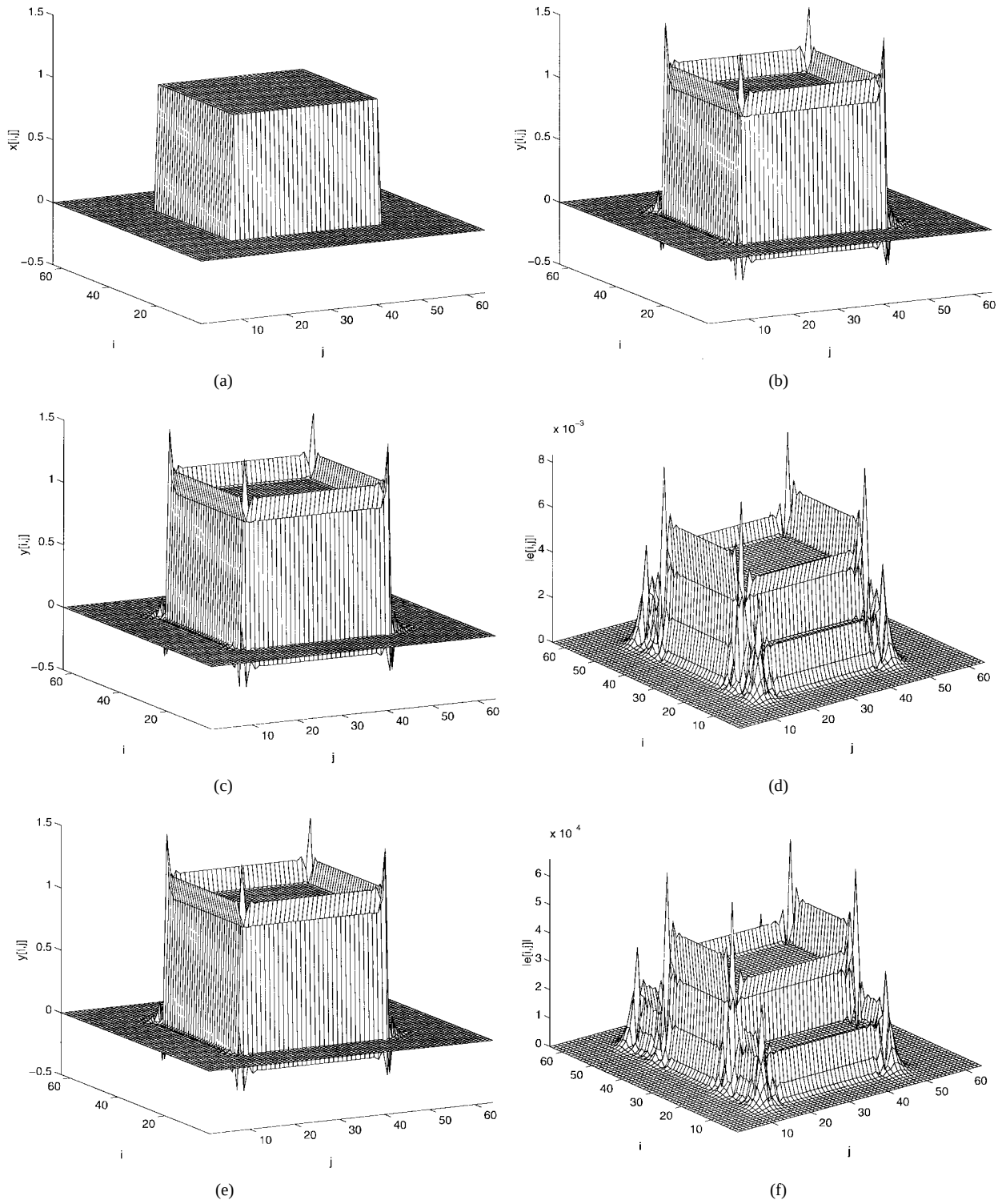

Fig. 8. (a) Square pulse input signal. (b) Exact response of edge-enhancing filter. (c)–(d) Filter output and error signals for $\beta = 2$. (e)–(f) Filter output and error signal for $\beta = 4$. Note the scalings of the vertical axes.

natural choice and are required, for example, if zero-phase filtering is desired. Indeed, some methodologies now exist for designing 2-D difference equations to meet desired frequency-selective specifications, and we demonstrated that imposing boundary conditions upon these difference equations can lead to the desired frequency selectivity.

The approach we developed for efficiently implementing 2-DNC-IIR filters involves a combination of two things: a) the application of concepts from the direct solution of PDE's to the calculation of the solution of a 2-D difference equation; and b) the development of new approximations, motivated by and appropriate for filtering applications, that reduce complexity to desired levels. In particular, the algorithms resulting from our procedure have constant computational complexity per pixel and, if implemented in maximally parallel form, have total computation time per pixel that decreases as image size

increases. In particular, our approximation is based on the columnwise ordering of data points and the block LU factorization of the linear system that results from this ordering. While exact factorization is still complex computationally, the observation that each successive block of computation could be viewed as a 1-D filtering operation **along** a column of the image led to the idea of a reduced-order approximation of each of these 1-D columnwise filters. In matrix terms, this corresponds to a banded approximation to each of the blocks in the block LU factorization, with bandwidth (and 1-D filter order) β. The resulting algorithm was shown both to achieve the computational levels mentioned previously and to yield excellent results using small values of β for a number of low-order frequency-selective filters.

The approach that we have described is, in principle, applicable to a broad range of filtering problems, e.g., higher-order or nonconstant-coefficient difference equations and irregularly shaped regions. Indeed, the success we have demonstrated here together with the guidelines we have described for situations in which our approximation should work well provide ample motivation for the application of this methodology and for a complete investigation of general conditions on the difference equation coefficients under which our approach is guaranteed to provide accurate answers for small values of β. However, the implementations of 2-DNC-IIR filters are certainly not limited to direct methods. Many of the efficient iterative algorithms developed for solving PDE's will likely provide efficient solutions to 2-DNC-IIR filters, and perhaps to a different class of filters than can be implemented by the algorithm of Section IV-A.

APPENDIX
COMPUTING CERTAIN ELEMENTS OF THE INVERSE OF A BANDED MATRIX

A special application of the results in [5] allows for the efficient computation of certain elements of the inverse of a banded matrix. Namely, if A_β is an $N \times N$ dimensional matrix with bandwidth β and $Z \triangleq A_\beta{}^{-1}$, then the elements of Z which lie within a bandwidth β of the diagonal can be computed in $\mathcal{O}(\beta^2 N)$ computations. The results of [5] are based on the following observation: given $A_\beta = LDU$ (where L and U are unit lower-triangular and upper-triangular, respectively), then

$$Z = D^{-1}L^{-1} + (I - U)Z \tag{29}$$

$$Z = D^{-1}U^{-1} + Z(I - L). \tag{30}$$

From these relations, $[Z]_{lk}$ for $|l-k| \le \beta$ is given by (31), which **does not depend upon computing any elements in L^{-1} and U^{-1}.**

$$[Z]_{lk} = \begin{cases} ([D]_{ll})^{-1} - \displaystyle\sum_{m=l+1}^{l+\beta} [U]_{lm}[Z]_{ml}, & l = k \\ -\displaystyle\sum_{m=l+1}^{l+\beta} [U]_{lm}[Z]_{mk}, & m < k. \\ -\displaystyle\sum_{m=l+1}^{l+\beta} [Z]_{lm}[L]_{mk}, & m > k \end{cases} \tag{31}$$

For certain matrices, such as those discussed in Section IV-A, the elements of Z which fall within the bandwidth β can be seen as a reasonable approximation to A_β.

To implement this algorithm, note that $[Z]_{NN}$ must be the first element computed of Z. Also, to compute any element $[Z]_{lk}$, all elements $[Z]_{mn}$ such that $m \ge l$, $n \ge k$, and $|l-k| \le \beta$ must already have been computed. With these restrictions placed on the recursion, the number of computations to compute Z within a bandwidth β is bounded above closely by $N[(\beta+1)+2\beta^2]$, where the first term represents the computations for the diagonal elements only. The computational complexity of the algorithm is thus $\mathcal{O}(\beta^2 N)$.

REFERENCES

[1] R. Chellappa and S. Chatterjee, "Classification of textures using gaussian Markov random fields," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-33, pp. 959–963, 1985.

[2] H. Derin and P. A. Kelly, "Discrete-index Markov-type random processes," *Proc. IEEE*, vol. 77, Oct. 1989.

[3] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1984.

[4] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*. Oxford, England: Oxford Univ. Press, 1987.

[5] A. M. Erisman and W. F. Tinney, "On computing certain elements of the inverse of a sparse matrix," *Commun. ACM*, vol. ACM-18, no. 3, 1975.

[6] A. George and J. W. Liu, *Computer Solution of Large and Sparse Positive Definite Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: John Hopkins Univ. Press, 1990.

[8] Q. Gu and M. N. S. Swamy, "On the design of a broad class of 2D recursive digitial filters with fan, diamond, and elliptically-symmetric responses," *IEEE Trans. Circuits Syst. II*, vol. 41, pp. 603–614, 1994.

[9] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artif. Intell.*, vol. 17, pp. 185–203, 1981.

[10] Mathworks Inc., *Matlab Image Processing Toolbox*, 4.1 ed. Natick, MA, June 1993.

[11] B. C. Levy, M. B. Adams, and A. S. Willsky, "Solution and linear estimation of 2-D nearest-neighbor models," *Proc. IEEE*, vol. 78, 1990.

[12] J. S. Lim, *Two-Dimensional Signal and Image Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1990.

[13] W. Lu and A. Antoniou, *Two-dimensional Digital Fitlers.* New York: Marcel Dekker, 1992.

[14] M. R. Luettgen, W. C. Karl, A. S. Willsky, and R. R. Tenney, "Multiscale representations of Markov random fields," *IEEE Trans. Signal Processing*, vol. 41, p. 3377, Dec. 1993.

[15] A. R. Mitchell and D. F. Griffiths, *The Finite Difference Method in Partial Differential Equations.* New York: Wiley, 1980.

[16] N. A. Pendergrass, S. K. Mitra, and E. I. Jury, "Spectral transformations for two-dimensional digital filters," *IEEE Trans. Circuits Syst.*, vol. CAS-23, pp. 26–35, 1976.

Michael M. Daniel (S'93) received the B.S. degree from the University of California, Berkeley, in 1990 and the Ph.D. degree from the Massachusetts Institute of Technology (MIT), Cambridge, in 1997.

He has been with Schlumberger–Doll Research, Ridgefield, CT, and the Cambex Corporation, Waltham, MA, and has been a teaching assistant for the Circuits, Signals, and Systems and the Discrete-Time Signal Processing courses at MIT. He recently coauthored the undergraduate-level book *Computer Explorations for Signals and Systems Using* MATLAB. His research interests include signal and image processing, stochastic processes and uncertainty analysis, and geophysical inverse problems.

Dr. Daniel is a member of SIAM, Sigma Xi, Tau Beta Pi, and Eta Kappa Nu.

Alan S. Willsky (S'70–M'73–SM'82–F'86) received the S.B. and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, in 1969 and 1973, respectively.

He joined the MIT faculty in 1973 and is currently a Professor of electrical engineering. From 1974 to 1981, he served as Assistant Director of the Laboratory for Information and Decision Systems, MIT. He is also a founder and member of the Board of Directors for Alphatech, Inc., Burlington, MA. He is the author of the research monograph *Digital Signal Processing and Control and Estimation Theory* and is coauthor of the undergraduate text *Signals and Systems*. He has held visiting positions at Imperial College, London, U.K., l'Université de Paris-Sud, and INRIA, France. His present research interests are in problems involving multidimensional and multiresolution signal processing and imaging, discrete-event systems, and the asymptotic analysis of control and estimation systems.

Dr. Willsky was program chair for the 17th IEEE Conference on Decision and Control, and has been an Associate Editor for several journals. He has served as a member of the Board of Governors and was Vice President for Technical Affairs of the IEEE Control Systems Society. He was program chairman for the 1981 Bilateral Seminar on Control Systems held in the People's Republic of China, and was a Special Guest Editor in 1992 for the IEEE TRANSACTIONS ON INFORMATION THEORY. In 1988, he was made a Distinguished Member of the IEEE Control Systems Society. He has given several plenary lectures at major scientific meetings, including the 1992 Inaugural Workshop for the National Center for Robust and Adaptive Systems, Canberra, Australia. In 1975, he received the Donald P. Eckman Award, in 1979 the Alfred Nobel Prize, and in 1980 the Broder S. Thompson Memorial Prize recognizing a paper excerpted from his monograph.