LIDS-P-2294

EFFICIENT IMPLEMENTATIONS OF 2-D NONCAUSAL IIR FILTERS [*]

Michael M. Daniel [†] *Alan S. Willsky* [‡]

March 10, 1995

Abstract

In 2-D signal processing, FIR filters are the standard choice for implementing linear shift-invariant (LSI) systems, as they retain all of the advantages of their 1-D counterparts. The same is not true in general for IIR filters. In particular, with the exception of the class of *causal* recursively computable filters, implementation issues for 2-D IIR filters have received little attention. Moreover, the forced causality and apparent complexity in implementing even recursively computable filters has severely limited their use in practice as well. In this paper, we propose a framework for implementing 2-D LSI systems with 2-D *noncausal* IIR filters, i.e., filter systems described implicitly by a difference equation and *boundary conditions*. This framework avoids many of the drawbacks commonly associated with 2-D IIR filtering. A number of common 2-D LSI filter operations, (such as low-pass, high-pass, and fan 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). In addition to its uses in 2-D filtering, we believe that this approach also has applications in related areas, such as geophysical signal processing and linear estimation, or any field requiring approximate solutions to elliptic PDE's.

1 Introduction

For two-dimensional signal processing applications, finite impulse response (FIR) filters have been overwhelmingly preferred to infinite impulse response (IIR) filters [3,10,12]. This preference is markedly different from

[*]This research was supported by the Office of Naval Research under Grant N00014-91-J-1004, the Air Force Office of Scientific Research under Grant F49620-95-1-0083, and the Army Research Office under grant DAAL03-92-G-0115 (Center for Intelligent Control Systems).

[†]Massachusetts Institute of Technology, Room 35-427, Cambridge, MA, 02139, cafepre@mit.edu.

[‡]Massachusetts Institute of Technology, Room 35-435, Cambridge, MA 02139, willsky@mit.edu.

that encountered in 1-D signal processing, in which 1-D IIR filtering plays a significant role. In 1-D, an extensive array of techniques exists for optimally designing both FIR and IIR filters. Both FIR and IIR filters have efficient, stable implementations. FIR filters are always stable and can be implemented with the FFT. 1-D IIR filters can be implemented in a recursive manner, and often require less memory, lower order, and fewer computations per sample than their FIR counterparts.

In 2-D, however, the design, analysis, and implementation of filters is complicated by a number of factors, and for the most part practical 2-D filter design and implementation has focused on FIR filters. Among the reasons for this preference are: (a) efficient implementation of FIR filters can be accomplished equally well in 1-D or 2-D through the use of the FFT; and (b) FIR filters 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, for IIR filters the 1-D and 2-D cases appear to be dramatically different, and, in particular, both points (a) and (b) become serious obstacles that have limited the investigation of 2-D IIR filters and led many to argue that they cannot be implemented in practice (see [3, 10, 12] for more on these and related issues).

To understand these issues, as well as our approach to dealing with them, consider an IIR filter, in 1-D or 2-D, specified in terms of a difference equation. In either case, of course, the difference equation by itself isn't sufficient to completely specify the filtering algorithm, as one must also specify a set of **boundary 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 or, equivalently, the number of initial conditions required. Moreover, even for noncausal 1-D filters — e.g., zero-phase IIR filters — implementation, accomplished in this case through the combination of a causal recursion (with associated initial conditions) and an anti-causal recursion (with "final" conditions), results in a per-sample computational load proportional to filter order or, equivalently, to the total number of boundary (initial and final) conditions.

In contrast, in 2-D the dimension of the required boundary conditions depends not only on the order of the difference equation but also on the size of the boundary — i.e., the dimensions of the 2-D domain of interest — implying an apparently significant increase in computational complexity. 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 boundary conditions would separate into anything that might resemble "initial" or "final" conditions, but rather would more naturally be distributed around the entire 2-D domain, leading to 2-D noncausal IIR (2DNC-IIR) filters that are not recursively computable.

On the other hand, if effective methods of implementation for 2DNC-IIR filters were available, there would be numerous possibilities for their application. For example, one potential advantage of IIR filters that is retained in 2-D 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, 2DNC-IIR filters arise naturally in applications such as the modeling of random fields for image processing [2, 16] and computer vision [1, 9]. With potential applications like these as our motivation, in this paper we present an approach to the efficient implementation of 2DNC-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 2DNC-IIR filters and the solution of finite difference and finite element approximations of PDE's. In particular, the equations resulting from such PDE methods are equivalent to the difference equations for 2DNC-IIR filters, and thus the many methods that have been developed for the efficient solution of such PDE's can be brought to bear in developing implementation concepts for 2DNC-IIR filters. These methods by themselves, while offering considerable savings in computational complexity and storage, may not reduce these loads enough to make 2DNC-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. In particular, 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 fluid flows or similar physical models, in 2-D the difference equation is **not** the fundamental object. That is, in filtering we generally begin with filter or frequency response specifications and design or choose a 2-D difference equation that meets these specifications to within some given tolerances. Consequently, approximations to the solution of the difference equation are acceptable as long as they lead to filters that also meet the desired tolerances, and in this paper we describe a method for approximating solutions to 2-D difference equations that both meets the desired filtering objectives and also results in very efficient implementations.

In the next section, we introduce the class of 2DNC-IIR difference equations and briefly discuss the issue of the choice of boundary conditions for such an equation. In Section 3, we make explicit the connection of the problem of interest here and the methods for solving sparse linear systems of equations such as those arising in finite difference methods for PDE's. In Section 3, we also introduce one of the constructs used in direct solution of such equations, namely, the organization of 2-D data points into 1-D arrays or columns and the ordering of these 1-D data sets in ways that lead to efficient solution procedures through the sequential processing of these 1-D sets of variables. The dimensionality of these 1-D data sets, of course, is quite large, corresponding to the linear dimension (width or length) of the 2-D domain. However, 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 4, results in very efficient 2DNC-IIR filtering procedures applicable to a large class of noncausal, nonseparable filtering applications. In particular, in Section 5 we illustrate the efficient implementation of several zero-phase 2DNC-IIR filters. Zero-phase filters are of considerable interest in practice, and the apparent difficulty in implementing IIR filters with zero phase has often been cited as one of the reasons that FIR filters are commonly used. As our results here indicate, we now can implement zero-phase IIR filters efficiently, removing a major obstacle to their use in practice.

2 Two-Dimensional IIR Filters Given by Difference Equations

A rich class of 2DNC-IIR filters can be described implicitly through 2-D linear constant-coefficient difference equations (LCCDE's) of the form

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

Such an equation corresponds to a 2-D system function of the form

$$H(z_1, z_2) = \frac{\sum_{m_1=-M_1}^{M_1} \sum_{m_2=-M_2}^{M_2} b_{m_1 m_2}\, z_1^{-m_1} z_2^{-m_2}}{\sum_{l_1=-L_1}^{L_1} \sum_{l_2=-L_2}^{L_2} a_{l_1 l_2}\, z_1^{-l_1} z_2^{-l_2}} . \tag{2}$$

Equation (1), of course, provides only a partial specification of a system, as it must be accompanied by a set of boundary conditions (BC's), and it is here that we see a significant difference from the 1-D case. In particular, for a 1-D difference equation with system function $H(z)$, the nature of the boundary conditions is completely and very simply specified if we require the system to be stable so that it has a well-defined frequency response. In this case, poles of $H(z)$ inside the unit circle correspond to causal parts of the system requiring initial conditions, while poles outside the unit circle correspond to anticausal parts of the system requiring final conditions. Thus, in general, the difference equation corresponding to $H(z)$ will require conditions on both boundary points of the data interval of interest. In analogy, the specification of a stable system corresponding to (1) requires the imposition of BC's around the entire boundary of the 2-D domain of interest. Of course the lack of a factorization theorem for polynomials in two variables makes the specification of the appropriate BC's a more challenging problem than in 1-D. Nevertheless, in Sections 4 and 5, we illustrate a number of important examples for which appropriate BC's are easily specified. For these examples and in general, the BC's **cannot** be organized simply as sets of "initial" or "final" conditions, e.g., as considered in [3, 12]. Consequently, no simple recursive solution is possible, and all of the output values $y[i, j]$ must be computed, in principle, simultaneously.

As in any practical application in 1-D or 2-D, the data domain of interest to us is assumed to be bounded. While our approach can be applied to a non-square and non-rectangular regions, for simplicity in the presentation here we focus on the case in which the filter difference equation is supported on the square region of $N \times N$ samples

$$\Omega_N = \{(i, j) \mid 1 \leq i \leq N,\ 1 \leq j \leq N\} . \tag{3}$$

Boundary conditions must then be specified around the boundary $\partial\Omega_N$ of Ω_N, which in analogy with finite difference methods can be thought of as an annular region of some width around Ω_N, where, roughly speaking, the width of the annular region depends upon both the order of the difference equation and the nature of the boundary conditions. For example, in 1-D a second-order difference equation might require two initial conditions, $y[-1]$ and $y[0]$, (and hence a "boundary" of width two at the left end of the interval of interest), or one initial and one final condition, $y[0]$ and $y[N+1]$, respectively, (corresponding to a boundary of width one at either end). For 2DNC-IIR filters, two types of boundary conditions follow directly from finite difference approximations to Dirichlet and Neumann conditions, which are commonly imposed on elliptic PDE's to guarantee unique solutions. The discrete Dirichlet conditions ("of width one") set the value of $y[i,j]$ on $\partial\Omega_N^1$, the boundary of Ω_N of width one defined by

$$\partial\Omega_N^1 = \{(i,j) \mid i \in \{0, N+1\},\ j \in [0, N+1]\} \ \cup\ \{(i,j) \mid i \in [0, N+1],\ j \in \{0, N+1\}\},$$

as illustrated in Figure 1. The discrete Neumann conditions correspond to a finite-difference approximation of the derivative of the filter output normal to the boundary of Ω_N. Thus Neumann conditions locally constrain the values of $y[i,j]$ as follows:

$$y[i,j] - y[p(i,j), q(i,j)] = r[i,j]\,, \qquad (i,j) \in \partial\Omega_N^1, \tag{4}$$

where $(p(i,j), q(i,j))$ is the closest point in Ω_N to the boundary point (i,j). As can be seen from Figure 1, Equation (4) provides a local constraint between each of the output values in the outermost square (represented by $\circ$'s) and their nearest neighbor in Ω_N (represented by the outermost square of $\bullet$'s). Although we will focus upon Dirichlet and Neumann conditions in this paper, the number of possible choices is much greater. The suitability of particular choices for the BC's, however, is a function of the filter characteristics; (the same is true in 1-D, but the choices for BC's are far more limited).

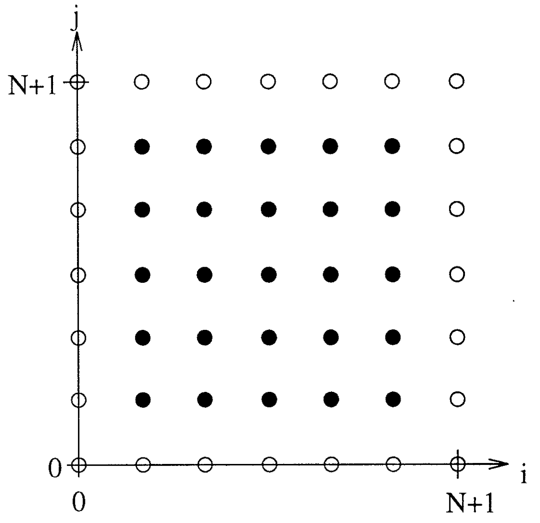


Figure 1: The elements of Ω_N (denoted by $\bullet$) and $\partial\Omega_N^1$ (denoted by $\circ$).

3 Implementing Noncausal IIR Filters as Linear Systems of Equations

3.1 Direct vs. Iterative Methods

In this section we examine the problem of implementing 2DNC-IIR filters and, in particular, make precise the connection between this problem 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 non-square and irregularly sampled, and for various types of boundary conditions. However, for notational simplicity in this and the following sections, unless stated otherwise, we assume that the difference equation is LCCDE, that $\Omega = \Omega_N$, and that the boundary conditions are of the Dirichlet type. Some of these assumptions are relaxed in the examples of Section 5. Also, since our focus here is on the IIR nature of filters, for clarity of exposition we make one final assumption, namely, 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 Equation (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 a 2DNC-IIR filter system given algebraically by

$$
\begin{aligned}
&\textstyle\sum_{l_1=-L_1}^{L_1} \sum_{l_2=-L_2}^{L_2} a_{l_1 l_2}\, y[i-l_1, j-l_2] = x[i,j]\,, && (i,j) \in \Omega_N \\
&y[i,j] = r[i,j]\,, && (i,j) \in \partial\Omega_N^1
\end{aligned}
\tag{5}
$$

where $r[i,j]$ is known.

Equation (5) can be cast in matrix form as

$$
A\underline{y} = \underline{x} + \underline{r} = \underline{b}\,, \tag{6}
$$

where the non-zero 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. Since the focus here is on LSI systems, we set $\underline{r}$ equal to zero to preserve linearity. 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 (defined in the following paragraph), this ordering can drastically alter the apparent complexity of the implementation.

Note that the matrix 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, like Equation (6), that 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 (6), i.e., for implicitly calculating $A^{-1}\underline{b}$. This calculation can be performed by either iterative or 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] (Gaussian elimination followed by back-substitution), and produce the exact solution (disregarding numerical errors) in a finite number of steps. Assuming both methods produce a solution within a desired level of accuracy, their relative performance is measured in terms of storage and computational requirements.

A comparison of direct and iterative methods is impossible without knowing the specific applications. Iterative methods require little storage beyond that to store A, $\underline{b}$, and $\underline{y}_0$, implying much less storage than direct methods usually require. The storage and computational complexity of direct methods is determined by the amount of *fill-in* which occurs during the factorization, i.e., the loss of sparsity that occurs in the Gaussian elimination process (see [4] for a complete discussion of fill-in). The amount of fill-in is strongly dependent upon the ordering of A, and the usefulness of a particular ordering depends upon the particular application.

For signal processing applications, the same filter is typically applied to a large number of inputs. Thus while $\underline{b}$ varies in (6), A remains fixed. For such applications, A must be factored only once. This factorization can then be done off-line, in which case 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 2DNC-IIR filter systems. Iterative methods, such as preconditioned conjugate gradient or multi-grid, 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 a large number of filters.

The primary reason for using the LU factorization of A for direct methods is that triangular systems are easy to solve [4]. The LU factorization

$$A = LU \tag{7}$$

yields a unit lower-triangular matrix L and an upper-triangular matrix U. The LU factorization of a dense $M \times M$ matrix requires $2M^3/3$ computations, measured in terms of floating point adds and multiplies. Storage can be done in place of A, such that no extra storage is needed beyond that required for A. Given the LU factorization of A, the solution to $Ay = b$ can be found very efficiently. In particular, if A has dimension $M \times M$, solving the following two triangular systems requires only $2M^2$ computations:

$$L\underline{\zeta} \;=\; \underline{b}, \qquad \text{“Forward substitution”} \tag{8}$$

$$U\underline{y} \;=\; \underline{\zeta}. \qquad \text{"Back substitution"} \tag{9}$$

Note that if we solved for $\underline{y}$ by explicitly computing A^{-1} and then computing $A^{-1}\underline{b}$, $2M^3+2M^2$ computations would be required. Thus the computational savings which result from the LU decomposition solution to $A\underline{y} = \underline{b}$ is modest for dense A. However, if A is sparse, as for 2DNC-IIR filter systems, the savings in both storage and computations can be tremendous (orders of magnitude) [4]. In particular, if A is sparse, then L and U will be sparse, especially if proper orderings are used. Amortizing the costs of the factorization over a large number of filter inputs further decreases the effective computation requirements for the LU approach. However, note that in our application $M = N^2$, the number of 2-D data points, 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 just described that do result in total complexity that is linear in M, (or, equivalently, quadratic in N).

Finally, note that a simple generalization of the LU factorization is the block LU factorization, in which L and U are, respectively, lower and upper **block** triangular. We will make use of such a block factorization in the next subsection, in which the block size is proportional to the linear dimension of the domain of interest. In this case, of course, the complexity of the calculations implied by (8) and (9) increase with increasing block size, and it is this fact that motivates the approximation introduced in Section 4.

3.2 Columnwise Orderings

The particular ordering choice that we make here is motivated by the structure of the 2-D difference equation (5), which will be exploited in Section 4.1 for an approximate solution. To make the discussion explicit we focus here on a common low-order 2-D LCCDE, the 9-point nearest neighbor model (NNM). This difference equation also arises quite frequently in engineering applications, most notably as the first-order and second-order finite difference and finite element approximations to elliptic PDE's [6,9,11,14]. The constant-coefficient form of the 9-point NNM is given by

$$c\,y[i,j] = n\,y[i,j+1] \;+\; s\,y[i,j-1] \;+\; e\,y[i+1,j] \;+\; w\,y[i-1,j] \tag{10}$$

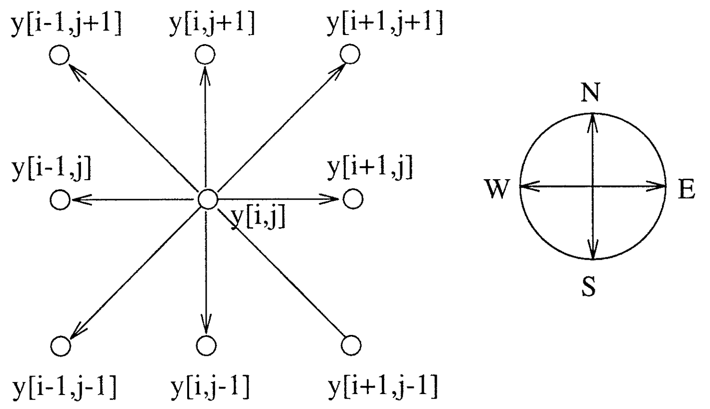


Figure 2: The output mask for the 9-point NNM difference equation.

$$+ \; n_e \, y[i+1, j+1] \; + \; n_w \, y[i-1, j+1] \; + \; s_e \, y[i+1, j-1] \; + \; s_w \, y[i-1, j-1]$$

$$+ \; x[i, j] \, ,$$

where the coefficients are labeled according to the directions of a compass. The output mask of this difference equation is illustrated in Figure 2. Note that the LSI system characterized by the frequency response from difference equation (10) is zero-phase if $n = s$, $e = w$, $n_e = s_w$, and $n_w = s_e$. Most of the filters implemented in Section 5 are zero-phase. This model will be the used for the algorithmic development and simulations in this paper, but the results readily generalize to filters of higher-order or shift-varying difference equations.

Consider a 2DNC-IIR filter supported on a square grid Ω_N, with the difference equation given by (10) and boundary conditions in Dirichlet form, i.e., $y[i,j] = r[i,j]$ on $\partial\Omega_N^1$. If the filter input and output variables defined on Ω_N are ordered columnwise into $N \times 1$ dimensional vectors $y_i = [y[i,1], y[i,2], \ldots, y[i,N]]^T$ and $x_i = [x[i,1], x[i,2], \ldots, x[i,N]]^T$, respectively, the 2-D filter system is represented algebraically in the form of Equation (6) as[1]

$$\underbrace{\begin{bmatrix} D_1 & E_2 & & & \\ C_1 & D_2 & E_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & E_N \\ & & & C_{N-1} & D_N \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}}_{\underline{y}} = \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}}_{\underline{x}} + \underline{r} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \\ b_N \end{bmatrix}}_{\underline{b}} . \tag{11}$$

As in Equation (6), the vector $\underline{r}$ contains the Dirichlet boundary values which enter through the difference equation. The structure of A in (11) is block tridiagonal, and the $N \times N$ dimensional blocks C_i, D_i, and E_i are tridiagonal. Note that Equation (11) 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 non-zero elements of C, D, and E are given by

$$[C]_{kl} = \left\{ \begin{array}{ll} -s_w, & l = k+1 \\ -w, & l = k \\ -n_w, & l = k-1 \end{array} \right\}, \quad [D]_{kl} = \left\{ \begin{array}{ll} -s, & l = k+1 \\ c, & l = k \\ -n, & l = k-1 \end{array} \right\}, \quad [E]_{kl} = \left\{ \begin{array}{ll} -s_e, & l = k+1 \\ -e, & l = k \\ -n_e, & l = k-1 \end{array} \right\},$$

for $1 \leq k, l \leq N$. For higher-order difference equations, a block tridiagonal ordering similar to that of (11) can be created by allowing y_i and x_i to contain filter output and input values, respectively, for more than one column of Ω_N. In this case the blocks C_i, D_i, and E_i are no longer tridiagonal, but will have small bandwidths with the Reverse-Cuthill-McKee ordering (used for ordering the nodes of a "thin" graph [4]). Another possibility, more appropriate for the algorithms discussed in Section 4, is to increase the number of blocks in A according to the filter order. For a 25-point NNM difference equation, which has an output mask with one more square layer of output values beyond that for the 9-point difference equation illustrated in Figure 2, A would be block penta-diagonal

While the block LU factorization of a block tridiagonal system is not unique [4, 7], one particular

[1]For any matrix in this paper, such as A in (11), block entries not indicated are zero.

choice leads to a simple recursive algorithm for the computation of the successive blocks of the factorization. Specifically, the quantities $\overline{D}_1, \ldots, \overline{D}_N$ and $\overline{E}_2, \ldots, \overline{E}_N$ needed in this factorization are recursively computed from the following equations[2]:

$$\begin{aligned} \overline{D}_i \overline{E}_{i+1} &= E_{i+1} & (12) \\ \overline{D}_{i+1} &= D_{i+1} - C_i \overline{E}_{i+1}, & (13) \end{aligned}$$

where the recursion is initialized with $\overline{D}_1 = D_1$. Thus the recursion begins by solving (12) for E_2, then computing $\overline{D}_2$ from (13), etc. Note that for these recursions to be well-posed, $\overline{D}_i$ for $i = 1, \ldots, N$ must be invertible. Conditions which guarantee this are discussed in [7]. In the filtering examples presented in Section 5, the matrices $\overline{D}_i$ are always invertible. Also, solving (12) at each step is performed by an LU factorization on $\overline{D}_i$, and indeed, as we discuss next, this factorization $\overline{D}_i = L_{ii}U_{ii}$ is needed on-line.

The block LU factorization resulting from the procedure just described is given by

$$A = \begin{bmatrix} \overline{D}_1 & & & \\ C_1 & \overline{D}_2 & & \\ & \ddots & \ddots & \\ & & C_{N-1} & \overline{D}_N \end{bmatrix} \begin{bmatrix} I & \overline{E}_2 & & \\ & \ddots & \ddots & \\ & & I & \overline{E}_N \\ & & & I \end{bmatrix}. \tag{14}$$

While implementing the recursions (12) and (13) is conceptually straightforward, the number of computations and storage elements needed to implement the *off-line* factorization (14) can be overwhelming. This burden is a direct result of the columnwise ordering, which leads to a destruction of the sparsity of A during the factorization. Although C_i, D_i, and E_i are very sparse, with the exception of $\overline{D}_1 = D_1$, the matrices $\overline{D}_i$ and $\overline{E}_i$ are generally full. The storage of these matrices required for the on-line solution is thus $\mathcal{O}(N^3)$. Also, each step of the recursion requires the factorization of the full matrix $\overline{D}_i$ in order to compute $\overline{E}_{i+1}$. Each such factorization requires $\mathcal{O}(N^3)$ computations, leading to $\mathcal{O}(N^4)$ computations overall.

The lack of sparsity in the blocks of (14) implies a large computational burden for the *on-line solution.*

[2]The validity of the recursions (12) and (13) can be verified directly by equating A in (11) with the expression in (14).

First note that, since the factorization $\overline{D}_i = L_{ii}U_{ii}$ is needed at each step of the recursion (12)-(13) to compute $\overline{E}_{i+1}$, these factors can be stored in place of $\overline{D}_i$ in (14). The solution to (11) is then given by forward-substitution, (initialized by $C_0\zeta_0 = 0$),

$$L_{ii}U_{ii}\zeta_i = b_i - C_{i-1}\zeta_{i-1} \qquad i = 1, \ldots, N \tag{15}$$

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

$$y_i = \zeta_i - \overline{E}_{i+1}y_{i+1} \qquad i = N-1, \ldots, 1. \tag{16}$$

Since L_{ii} and U_{ii} are generally full, the solution of (15) and (16) 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 large ($\mathcal{O}(N^4)$), but the on-line storage and computations are both $\mathcal{O}(N^3)$, significantly greater than the $\mathcal{O}(N^2)$ goal that corresponds to constant per-pixel computational and storage burden.

The $\mathcal{O}(N^4)$ growth in computations and $\mathcal{O}(N^3)$ growth in storage makes the columnwise ordering procedure, as we have described it, infeasible for even modestly sized 2-D filtering problems. For exact solutions to (11), alternative orderings or iterative implementations must be employed (see Section 6). 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 <u>both</u> the off-line factorization and the on-line solution. Before describing the approximation strategy in detail (see Section 4), we can motivate its development by closely examining a single stage of the on-line solution. Equation (15) requires first solving the lower triangular equations

$$L_{ii}z_i = b_i - C_{i-1}\zeta_{i-1}\,, \tag{17}$$

followed by solving the upper triangular system

$$U_{ii}\zeta_i = z_i\,. \tag{18}$$

Recall that we have organized our variables into 1-D columns, and thus the solution of the lower triangular system (17) can be thought as a causal 1-D recursion, beginning at the bottom of the column ($j = 1$) and proceeding recursively to the top of the column ($j = N$). The upper triangular system (18) thus corresponds to an anticausal recursion proceeding from top to bottom. The back-substitution filtering, Equation (16), requires implementing an FIR filter, in the form of a matrix multiplication, along a single column of Ω_N.

Thus we can view (17) and (18) as 1-D recursive filtering operations, albeit shift-varying recursions, since in general L_{ii} and U_{ii} will not be Toeplitz. If L_{ii} and U_{ii} 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 $\overline{D}_i$ and hence L_{ii} and U_{ii} can be approximated by **banded** matrices, i.e., by matrices that are zero except for a band of preferably small bandwidth around the main diagonal, 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 $\overline{E}_{i+1}$ with y_{i+1}. Since $\overline{E}_{i+1}$ is generally full and not Toeplitz, this operation will require $\mathcal{O}(N^2)$ computations per step. However, if we similarly approximate $\overline{E}_{i+1}$ with a lower-order FIR filter, i.e., by approximating $\overline{E}_{i+1}$ with a banded matrix, the total computational and storage requirements for the on-line solution reduce to the $\mathcal{O}(N^2)$ goal we desire.

Note, however, to reduce the **off-line** computational load to $\mathcal{O}(N^2)$, it is not sufficient that $\overline{D}_i$ and $\overline{E}_i$ are well approximated 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 and also discuss conditions under which we would expect the approximation to be an excellent one. These conditions are typically satisfied in 2-D filtering applications, and we illustrate several of these in Section 5.

4 Efficient Implementations of 2DNC-IIR Filters

In this section, we develop an approximate implementation of 2DNC-IIR filters which requires $\mathcal{O}(N^2)$ computations and storage. Both the factorization of A given by Equations (12)-(13) and the on-line solution given by Equations (15)-(16) are approximated in $\mathcal{O}(N^2)$ computations. A parallel approximate implementation is outlined in Section 4.3. While the necessary and sufficient conditions under which the approximation is valid are not prescribed, some analytical guidelines are provided. These guidelines are used to develop a number of useful, low-order filters in Section 5, which are successfully implemented with the approximation algorithm in Section 5. The generality of the guidelines and the success of the approximation algorithm demonstrated in Section 5 leads us to believe that the approximation is valid for a significant class of filters.

4.1 Development of the Approximate Block LU Algorithm

The approximate implementation of 2DNC-IIR filters described in this subsection is motivated by a simple observation regarding the structure of the LU factorization (14). Namely, for many filters, a small number of elements in the blocks C_i, $\overline{D}_i$, and $\overline{E}_i$ dominate in magnitude the rest of the elements. (This dominance is obvious for the tridiagonal blocks C_i in Equation (11).) An efficient approximation to the on-line solutions follows by setting to zero the insignificant elements of L_{ii}, U_{ii}, (remember that L_{ii} and U_{ii} are stored in place of $\overline{D}_i$), and $\overline{E}_i$. Recursions (15) and (16) then can be implemented very efficiently if one takes care to avoid operating on the zero elements. Also, if the number of significant elements in each block is $\mathcal{O}(N)$, then in general only $\mathcal{O}(N^2)$ elements of (14) will need to be stored, implying that the on-line solution can be approximated in $\mathcal{O}(N^2)$ operations.

Unfortunately, the approach of simply discarding the insignificant elements of (14) has significant drawbacks. First, the off-line factorization will still require $\mathcal{O}(N^4)$ computations. Secondly, searching for the significant elements of each block in (14) and storing them in data structures necessary for efficient implementation of the on-line solution can be a costly procedure. A large class of filters, however, have a property which allows us to overcome these difficulties. In particular, for filters such as those described in this paper, all of the matrices of interest — $\overline{D}_i$, its LU factors L_{ii} and U_{ii}, and E_i — 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. Moreover, since each of these matrices has $\mathcal{O}(\beta N)$ non-zero elements, there are $\mathcal{O}(N)$ such matrices, and β is independent of N, the total required storage is $\mathcal{O}(N^2)$, as desired. Furthermore, as we now describe, we can compute each of these approximations in $\mathcal{O}(\beta^2 N)$ or $\mathcal{O}(\beta N)$ computations, resulting in an overall computational load of $\mathcal{O}(N^2)$, again as desired.

The key assumption required for these approximations to yield good results is that, for $i = 1, \dots, N$, the blocks $\overline{D}_i^{-1}$ are *approximately banded*, i.e., well approximated by setting to zero all the elements which do not fall within a small (relative to N) bandwidth of the main diagonal. For example, as holds for many of the filters considered in this paper, the elements of $\overline{D}_i^{-1}$ decay geometrically in magnitude with distance from the main diagonal. If $\overline{D}_i^{-1}$ is approximately banded, then from the recursions (12)-(13), it is apparent that the blocks $\overline{D}_i$ and $\overline{E}_i$ will generally be approximately banded as well. Furthermore, as the following corollary states, if we have a banded approximation to $\overline{D}_i$, we can efficiently compute a banded approximation of its inverse.

Corollary of [5] (See 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.*

This leads to the following approximation procedure, replacing Equations (12) and (13). Specifically, suppose that $\tilde{D}_i$ is an approximation to $\overline{D}_i$ which is *β-banded* (i.e., banded with bandwidth β). Note that $\tilde{D}_1 = \overline{D}_1 = D_1$ is exactly banded. We then compute a β-banded approximation to $\tilde{D}_i^{-1}$:

$$F(\tilde{D}_i, \beta) = \begin{cases} \left[\tilde{D}_i^{-1}\right]_{kl}, & |k-l| \le \beta \\ 0, & \text{otherwise} \end{cases}, \tag{19}$$

where $F : \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ requires $\mathcal{O}(\beta^2 N)$ computations and is called the approximate inverse operator. Assuming that $\tilde{D}_i^{-1}$ is a good approximation to $\overline{D}_i^{-1}$ and that $F(\tilde{D}_i, \beta)$ is a good approximation to $\tilde{D}_i^{-1}$, then

$$F(\tilde{D}_i, \beta) \cdot E_{i+1} \approx \overline{E}_{i+1} . \tag{20}$$

Since E_{i+1} is tridiagonal, the matrix multiplication in (20) requires $\mathcal{O}(\beta N)$ computations. Note, however, that the approximation of $\overline{E}_{i+1}$ in (20) has bandwidth $(\beta+2)$, 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 operator $T_\beta : \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$

$$T_\beta[H] = \begin{cases} H_{kl}, & |k-l| \leq \beta \\ 0, & \text{otherwise} \end{cases} . \tag{21}$$

We thus have the following approximation to Equation (12):

$$\tilde{E}_{i+1} = T_\beta[F(\tilde{D}_i, \beta) \cdot E_{i+1}] . \tag{22}$$

Similarly, substituting $\tilde{E}_{i+1}$ into (13) in lieu of $\overline{E}_{i+1}$ yields

$$D_{i+1} - C_i \tilde{E}_{i+1} \approx \overline{D}_{i+1} , \tag{23}$$

requiring $\mathcal{O}(\beta N)$ calculations. Once again, the quantity in (23) is $(\beta+2)$-banded, and applying our assumption of β-bandedness, we obtain our approximation to (13):

$$\tilde{D}_{i+1} = T_\beta[D_{i+1} - C_i \tilde{E}_{i+1}] . \tag{24}$$

Equations (22) and (24) can then be repeated iteratively, each stage requiring $\mathcal{O}(\beta^2 N)$ computations. For $i = 1, \ldots, N$, Equations (22) and (24) form an approximation to (14) requiring a total of $\mathcal{O}(\beta^2 N^2)$ computations and $\mathcal{O}(\beta N^2)$ storage elements. Furthermore, the LU factorization of each β-banded $\tilde{D}_i$ also requires only $\mathcal{O}(\beta^2 N)$ computations, yielding β-banded lower and upper-triangular matrices $\tilde{L}_{ii}$ and $\tilde{U}_{ii}$. This process is summarized as follows:

(i) choose $\beta \ll N$

(ii) for $i = 1, \ldots, N-1$

(a)	factor $\tilde{D}_i = \tilde{L}_{ii}\tilde{U}_{ii}$, $(\tilde{D}_1 = D_1)$;	**computations/step**: $2\beta^2 N$
	store LU factors in place of $\tilde{D}_i$;	**storage/step**: $(2\beta+1)N$
(b)	compute $G_i = F(\tilde{D}_i, \beta)$;	**computations/step**: $2\beta^2 N$
(c)	compute $\tilde{E}_{i+1} = T_\beta[G_i E_{i+1}]$;	**computations/step**: $3(2\beta+1)N$
	store $\tilde{E}_{i+1}$;	**storage/step**: $(2\beta+1)N$
(d)	compute $\tilde{D}_{i+1} = T_\beta[D_{i+1} - C_i\tilde{E}_{i+1}]$;	**computations/step**: $3(2\beta+3)N$
	set to zero values outside the bandwidth β;	

(iii) factor $\tilde{D}_N = \tilde{L}_{NN}\tilde{U}_{NN}$ ($2\beta^2 N$ computations), and store factors in place of $\tilde{D}_N$ ($2\beta+1$ storage elements)

This procedure results in the following approximation of (14)

$$A \approx \begin{bmatrix} \tilde{D}_1 & & & \\ C_1 & \tilde{D}_2 & & \\ & \ddots & \ddots & \\ & & C_{N-1} & \tilde{D}_N \end{bmatrix} \cdot \begin{bmatrix} I & \tilde{E}_2 & & \\ & \ddots & \ddots & \\ & & I & \tilde{E}_N \\ & & & I \end{bmatrix}, \tag{25}$$

with $\tilde{L}_{ii}$ and $\tilde{U}_{ii}$ stored in place of $\tilde{D}_i$. An approximate on-line solution follows by substituting $\tilde{L}_{ii}$ and $\tilde{U}_{ii}$ into (15) for L_{ii} and U_{ii}, and $\tilde{E}_i$ into (16) for $\overline{E}_i$.

While the approximation outlined by steps **(i)**-**(iii)** is straightforward, the utility and applicability of the algorithm has not been demonstrated. In particular, for step **(i)**, the approximation bandwidth β must be chosen according to the desired level of accuracy of the solution and also according to the set of filter coefficients (difference equation and BC's). How one chooses β to achieve a desired level of approximation accuracy depends, of course, on the application. If β is too large, the computational savings obviously are lost. What follows is an example intending both to suggest the class of filters for which the 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.

4.2 Analysis of the Block LU Approximation

Consider a 2DNC-IIR filter on Ω_N described implicitly by a 9-point NNM difference equation and homogeneous, Dirichlet BC's (i.e., Equation (11) with $\underline{r} = 0$). To make subsequent analysis more simple, we make a slight deviation from the constant-coefficient model of Equation (10). Namely, assume that $c = (1 + \alpha^2)$ and $n = s = -\alpha$ for all $(i, j) \in \Omega_N$ and $i \neq 1$. For $i = 1$ and $(i, j) \in \Omega_N$, the corresponding coefficients become $c = 1$ and $n = s = -\alpha$. For the point we wish to make, the values of the other NNM coefficients are of no consequence. Instead, we are interested in the utility of the block LU factorization for $|\alpha| < 1$. The first block row of Equation (11) follows as

$$\underbrace{\begin{bmatrix} 1 & \alpha & 0 & 0 & \cdots & 0 \\ \alpha & 1{+}\alpha^2 & \alpha & 0 & \cdots & 0 \\ 0 & \alpha & 1{+}\alpha^2 & \alpha & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \alpha & 1{+}\alpha^2 & \alpha \\ 0 & \cdots & 0 & 0 & \alpha & 1{+}\alpha^2 \end{bmatrix}}_{\overline{D}_1} y_1 + E_2 y_2 = x_1 , \tag{26}$$

The factorization of A in (11) begins by factoring $\overline{D}_1$, i.e.,

$$\overline{D}_1 = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \alpha & 1 & 0 & \cdots & 0 \\ 0 & \alpha & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha & 1 \end{bmatrix}}_{L_{11}} \underbrace{\begin{bmatrix} 1 & \alpha & 0 & \cdots & 0 \\ 0 & 1 & \alpha & \ddots & \vdots \\ 0 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \alpha \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}}_{U_{11}} . \tag{27}$$

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.

The next step of the factorization is to compute $\overline{E}_2$. Note that, even though D_1, L_{11}, and U_{11} are banded, this operation will require $\mathcal{O}(N^2)$ computations and $\mathcal{O}(N^2)$ storage elements for $\overline{E}_2$. To approximate the first stage of the factorization, we only need to compute the elements of $\overline{D}_1^{-1}$ within a pre-specified bandwidth. For this simple example, some simple algebraic manipulations will allow us to determine for which values of α such an approximation is valid. Note that since $n = s$, $\overline{D}_1 = D_1$ is symmetric. Thus, $L_{11} = U_{11}^T$ and $\overline{D}_1^{-1} = U_{11}^{-1}U_{11}^{-T}$. Also, since we can write $U_{11} = I - F$, where F is strictly upper-triangular, the matrix identity $(I-F)^{-1} = I + F + \cdots + F^{N-1}$ gives

$$U_{11}^{-1} = \begin{bmatrix} 1 & -\alpha & \alpha^2 & \cdots & (-\alpha)^{N-1} \\ 0 & 1 & -\alpha & \ddots & \vdots \\ 0 & 0 & 1 & \ddots & \alpha^2 \\ \vdots & \vdots & \ddots & \ddots & -\alpha \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} . \tag{28}$$

The simple form of Equation (28) leads to a single expression for $d_{kl} \triangleq \left[\overline{D}_1^{-1}\right]_{kl}$, i.e.,

$$d_{kl} = (-\alpha)^{l-k} \sum_{l=0}^{N-l} (-\alpha)^{2l} .$$

From this expression, the "clustering" of the significant elements d_{kl} about the main diagonal of $\overline{D}_1^{-1}$ can be determined as follows (for $0 < |\alpha| \leq 1$):

$$\begin{array}{ll} |d_{k-1,l}| = |\alpha||d_{kl}| & \\ & k \leq l \,, \\ |d_{k,l+1}| < |\alpha||d_{kl}| & \end{array} \tag{29}$$

Thus, the smaller the value of $|\alpha|$, (equivalently, the greater the diagonal dominance of $\overline{D}_1$), the tighter the clustering about the main diagonal of $\overline{D}_1^{-1}$, and thus the smaller the approximation bandwidth β necessary to obtain an approximation to $\overline{D}_1^{-1}$ at a desired level of accuracy. Also, Equation (29) shows that the elements of $\overline{D}_1^{-1}$ essentially decrease in magnitude geometrically with distance from the main diagonal, where the rate of decay depends only upon $|\alpha|$ and not N.

How might these observations be used to screen or suggest filters appropriate for the approximate block LU implementation? In analyzing only the first stage of the factorization, the effect of the NNM coefficients other than c, n, and s has not been accounted for. In essence, in cases where the matrices D_i have inverses for which all the significant elements are tightly clustered about the main diagonal, the same property does not necessarily hold for the inverses of the matrices $\overline{D}_i$ ($i = 2, \ldots, N$), which depend upon all the filter coefficients. However, a general guideline verified by extensive numerical simulations, some of which are given in Section 5, is that the approximate factorization becomes more accurate (for fixed β) as the ratio $(|n| + |s|)/|c|$ decreases. Furthermore, the approximation appears to be very useful, i.e., β is small for any desired level of accuracy, for $|n| + |s| \leq |c|/2$. These observations are consistent with the preceding example, since $(|n|+|s|)/|c| \to 0$ as $|\alpha| \to 0$. Note that $|c|/(|n|+|s|)$ is a measure of the "degree" of diagonal dominance of the elements in the blocks D_i. For the non-constant coefficient filters, analogous observations apply, i.e., the approximate block LU factorization becomes more accurate as the degree of diagonal dominance in the blocks D_i increases. The issue of how to select the value of β is discussed in Section 5, where we show that this approach can be successfully applied to a variety of filtering tasks ranging from low-pass filtering to fan filtering to high-pass operations such as edge enhancement.

4.3 A Parallel Approximate Implementation

In this section, we briefly discuss a straightforward parallelization of the algorithm discussed in Section 4.1. Many of the details of the implementation are omitted, since they depend in part upon the available computer architectures.

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_i sequentially from $i = 1$ to $i = N$. However, it is possible to eliminate columns in the interior of Ω independently. For example, consider again the block tridiagonal matrix A given in (11), which corresponds to a 2DNC-IIR filter described by a 9-point NNM difference equation and Dirichlet BC's. Assume for simplicity of notation that N is odd. If we order the even columns (y_i for $i = 2, 4, \ldots, N-1$) last and the odd columns (y_i for $i = 1, 3, \ldots, N$) first, Equation (11) takes the form

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} y_{odd} \\ y_{even} \end{bmatrix} = \begin{bmatrix} b_{odd} \\ b_{even} \end{bmatrix}. \tag{30}$$

Because of the coupling of the NNM difference equation, the elimination of the odd columns of y_i for the block factorization of (30) can be performed in parallel. Upon completing this step, the second block equation of (30) becomes

$$\underbrace{\begin{bmatrix} D_2^{(1)} & E_4^{(1)} & & & \\ C_2^{(1)} & D_4^{(1)} & E_6^{(1)} & & \\ & \ddots & \ddots & \ddots & \\ & & C_{N-5}^{(1)} & D_{N-3}^{(1)} & E_{N-1}^{(1)} \\ & & & C_{N-3}^{(1)} & D_{N-1}^{(1)} \end{bmatrix}}_{A_{22}^{(1)}} \underbrace{\begin{bmatrix} y_2 \\ y_4 \\ \vdots \\ y_{N-3} \\ y_{N-1} \end{bmatrix}}_{y_{even}} = \underbrace{\begin{bmatrix} b_2^{(1)} \\ b_4^{(1)} \\ \vdots \\ b_{N-3}^{(1)} \\ b_{N-1}^{(1)} \end{bmatrix}}_{b_{even}^{(1)}}, \tag{31}$$

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

$$\begin{aligned} D_i^{(1)} &= D_i - C_{i-1}D_{i-1}{}^{-1}E_i - E_{i+1}D_{i+1}{}^{-1}C_i, && i = 2,4,\ldots,N-1 \\ C_i^{(1)} &= -C_{i+1}D_{i+1}{}^{-1}C_i, && i = 2,4,\ldots,N-3 \\ E_i^{(1)} &= -E_{i-1}D_{i-1}{}^{-1}E_i. && i = 4,6,\ldots,N-1 \end{aligned} \tag{32}$$

Note that, if the difference equation is constant-coefficient, each of the three equations in (32) only needs to be computed for single value of i.

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. Thus, 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 corresponding on-line solution), 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. (In the case of cyclic block reduction, $M = \lfloor (N+1)/2 \rfloor$.) An example of partitioning for four

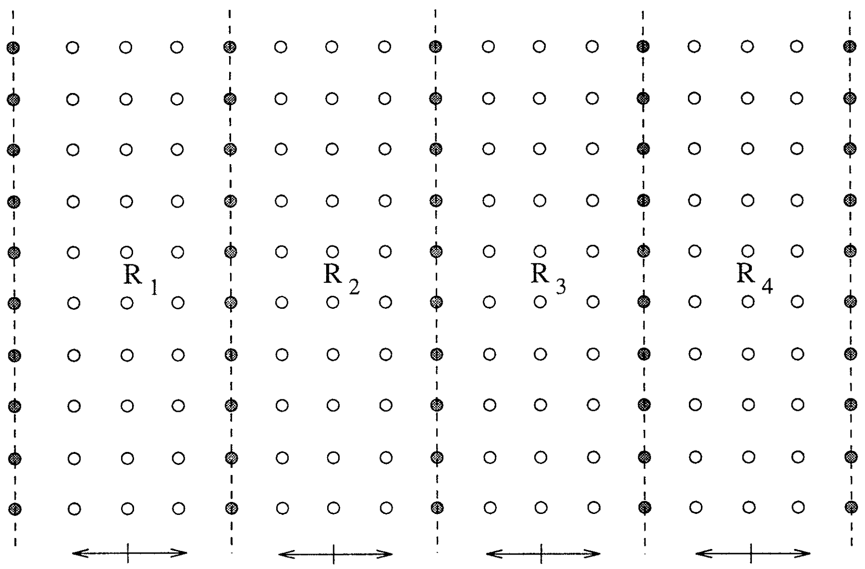


Figure 3: Ω partitioned into four regions R_i which overlap with neighboring regions by one column (variables in the regions of overlap are denoted by $\bullet$). For a 9-point filter, the internal variables of each region (denoted by $\circ$) are independent of the internal variables of other regions, allowing parallel Gaussian elimination. The arrows indicate the direction of elimination for each of the four processors.

processors, $M = 4$, is illustrated in Figure 3. Note that the four sub-regions R_i have overlapping domains, so that all but the boundaries can be eliminated independently. As shown by the arrows in Figure 3, the direction of elimination for each region R_i is from the center outwards to the boundary columns. In essence, the boundaries are ordered last, and the resulting submatrix for the boundaries after the interiors of R_i are eliminated is block tridiagonal.

An approximate block cyclic reduction algorithm follows for any of these parallel structures by noting the strong similarities between implementing Equation (32) and Equations (12)-(13). 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_i and compute $F(D_i, \beta)$	on processors $i = 1, 3, \ldots, N$;
(b)	compute $\tilde{C}_{i-1} = F(D_i, \beta)\, C_{i-1}$	on processors $i = 3, 5, \ldots, N$;
	compute $\tilde{E}_{i+1} = F(D_i, \beta)\, E_{i+1}$	on processors $i = 1, 3, \ldots, N-2$;
(c)	compute $\tilde{E}^{(1)}_{i+1} = -T_\beta[E_i\, \tilde{E}_{i+1}]$	on processors $i = 3, 5, \ldots, N-2$;
	compute $\tilde{C}^{(1)}_{i-1} = -T_\beta[C_i\, \tilde{C}_{i-1}]$	on processors $i = 3, 5, \ldots, N-2$;
(d)	compute $G_{i+1} = C_i\, \tilde{E}_{i+1}$	on processors $i = 1, 3, \ldots, N-2$;
	compute $J_{i-1} = E_i\, \tilde{C}_{i-1}$	on processors $i = 3, 5, \ldots, N$;
(e)	compute $\tilde{D}^{(1)}_{i+1} = T_\beta[D_{i+1} - G_{i+1} - J_{i+1}]$	on processors $i = 1, 3, \ldots, N-2$;

Note that step **(e)** requires communication between the processors, since G_{i+1} and J_{i+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 inter-processor communication costs after each stage of the recursion, the total factorization will require a total computation time of $\mathcal{O}(\beta^2 N \log_2 N)$. 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. The on-line calculations also can be performed in parallel with similar computational savings.

5 Examples and Simulations

In this section, a number of 2DNC-IIR filters are presented. The filter difference equations, along with easily specified BC's, yield frequency responses corresponding to canonical and widely used frequency-selective filter classes, e.g., low-pass, high-pass, and fan filters. As we will see, the frequency-selective characteristics of these filters can be achieved quite efficiently by implementing the approximation algorithm of Section 4.1. Each 2DNC-IIR filter considered here has a difference equation in the form of Equation (10). The system function for such difference equations is given by

$$H(z_1, z_2) = \frac{1}{\underline{z_2}^T J \underline{z_1}}, \tag{33}$$

$$\underline{z_2} = \begin{bmatrix} z_2 \\ 1 \\ z_2^{-1} \end{bmatrix}, \quad J = \begin{bmatrix} -n_w & -n & -n_e \\ -w & c & -e \\ -s_w & -s & -s_e \end{bmatrix}, \quad \underline{z_1} = \begin{bmatrix} z_1^{-1} \\ 1 \\ z_1 \end{bmatrix}.$$

Note that the system function (33) has no numerator. Adding a polynomial $\underline{z_2}^T P \underline{z_1}$ to the numerator of (33) would obviously allow one to improve the filter frequency responses, such as by narrowing the transition regions; however, our primary interest is to implement the recursive portion of the difference equation, in order to justify both the viability of 2DNC-IIR filters and the utility of the approximation given in Section 4.1.

If $n = s$, $e = w$, $n_e = s_w$, and $n_w = s_e$, the frequency response is real and follows as

$$H(e^{j\omega_1}, e^{j\omega_2}) = \frac{1}{c - 2[n \cos\omega_2 + e \cos\omega_1 + n_e \cos(\omega_1 + \omega_2) + n_w \cos(\omega_1 - \omega_2)]}. \tag{34}$$

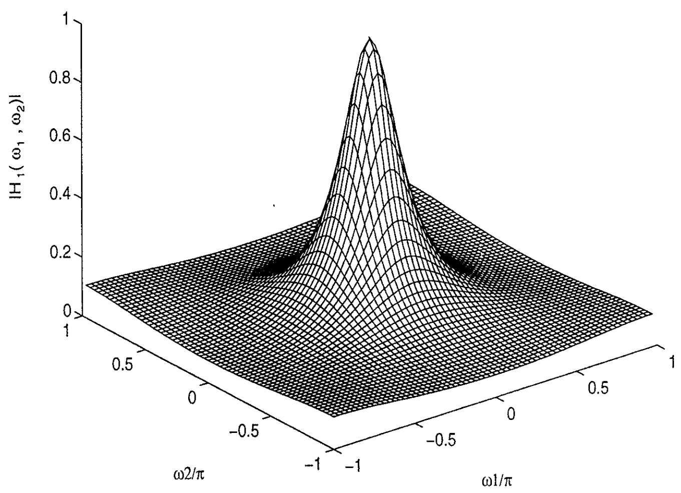
These filters have *zero-phase*. Three examples of zero phase filters are given by

$$J_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 9 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad J_2 = \tfrac{1}{9}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad J_3 = \tfrac{1}{5}\begin{bmatrix} -1 & 1 & -1 \\ 1 & 5 & 1 \\ -1 & 1 & -1 \end{bmatrix}, \tag{35}$$

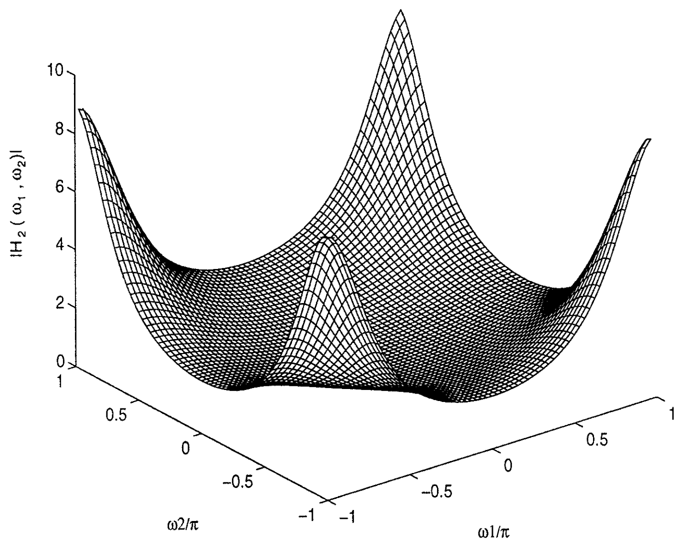
where $H_i(z_1, z_2) = (\underline{z_2}^T J_i \underline{z_1})^{-1}$. A fourth example, whose frequency response is not zero-phase, is

$$J_4 = \begin{bmatrix} 0.13 & -0.5 & 0.37 \\ 0.50 & 2.0 & 0.50 \\ 0.13 & -0.5 & 0.37 \end{bmatrix}. \tag{36}$$

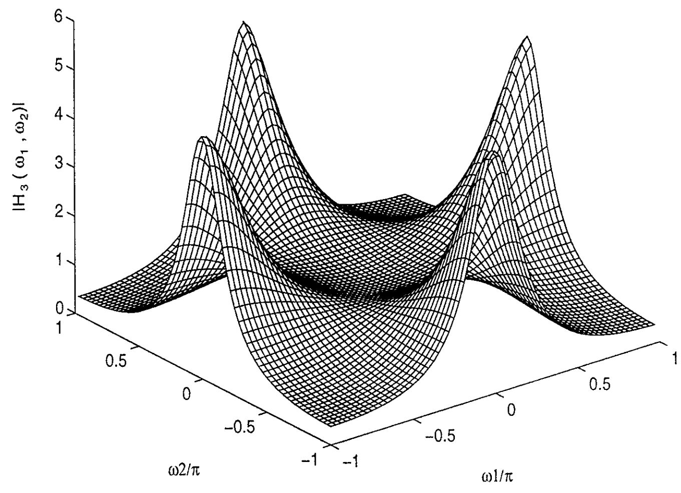
The coefficients of each filter are scaled such that $H(e^{j\omega_1}, e^{j\omega_2})|_{(\omega_1,\omega_2)=(0,0)} = 1$ — a desirable filter property which prevents the filter from biasing image intensity [13].



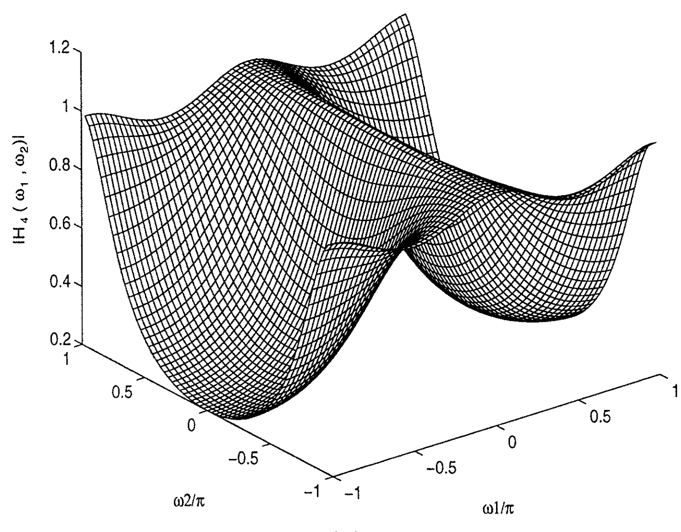
(a)



(b)



(c)



(d)

Figure 4: 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) edge enhancer H_3 given by J_3; (d) fan filter H_4 given by J_4.

The frequency responses of all four difference equations are illustrated in Figure 4. The frequency response H_1 is that of a low-pass filter, while H_2 corresponds to a high-pass filter. The difference equation for system H_2 is identical to the frequency response of the edge enhancer in [13]. The frequency selection of H_3 can also be used to enhance edges aligned with the j and i axes, while H_4 corresponds to the frequency response of a primitive (low-order) fan-filter. Fan filters are commonly employed in seismic signal processing to select wavefronts by their velocity of propagation [13]. As demonstrated by the examples given in [8,15], the frequency responses of each of the example filters could be improved, e.g., with sharper transition regions, by using higher-order ARMA difference equations. (Higher-order filters can be implemented, exactly or approximately, with simple extensions of the algorithms prescribed in Sections 2 through 4.)

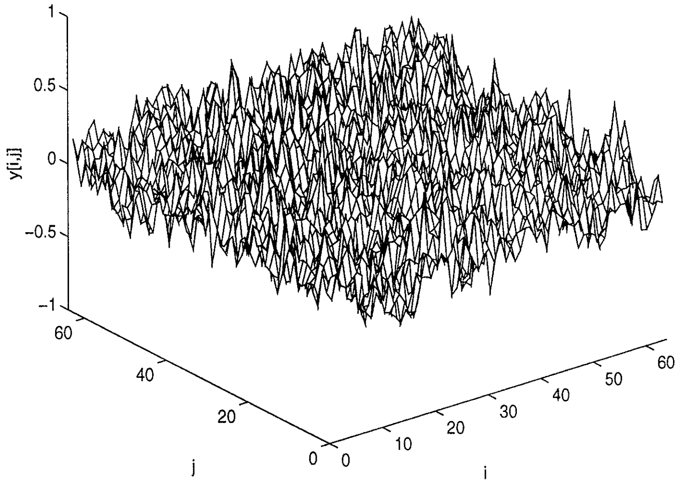
5.1 LSI Filtering with 2DNC-IIR Filters: the Effects of BC's

As mentioned previously, boundary conditions must be specified for each of the four difference equations corresponding to (33) with $J = J_1, J_2, J_3, J_4$. In addition, since these filters will be applied to images defined over finite $N \times N$ domains, we expect that the responses of these filters will exhibit transient behavior near the boundary of the domain. To examine and illustrate the choice of boundary conditions and the transient behavior near the boundaries, we first implement the low-pass filter corresponding to J_1 assuming homogeneous Dirichlet conditions. Since low-pass filters are often used as shaping filters, the filter input is chosen to be White Gaussian Noise (WGN) with zero mean and unit variance.

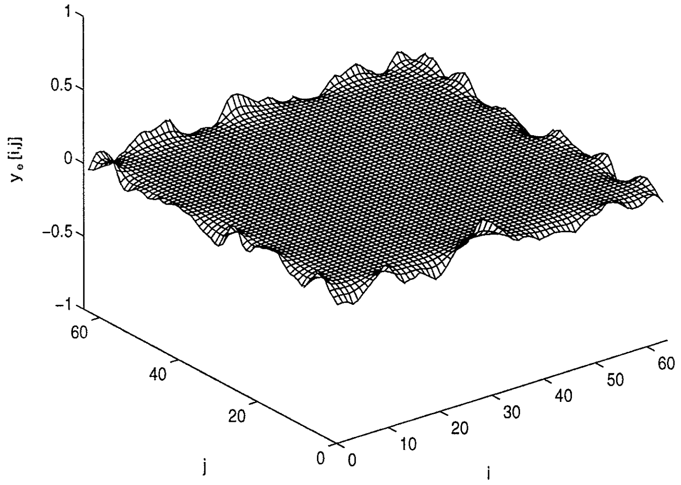
The low-pass filter system, in the form of Equation (11) with $\underline{r} = 0$, is solved directly and without approximation in MATLAB using sparse matrix data types and the minimum degree ordering algorithm. A sample output $y[i,j]$ for $N = 64$ is illustrated in Figure 5(a). Ideally, this output signal will be identical to the signal obtained by frequency-selective filtering according to the frequency response given in Figure 4(a). However, transient effects are introduced by the BC's. The transient signal y_e is defined by

$$y[i,j] = y_{lsi}[i,j] + y_e[i,j]\,, \qquad (i,j) \in \Omega_N$$

where y_{lsi} is the output of an FFT filter obtained by sampling $H_1(e^{j\omega_1}, e^{j\omega_2})$. For the output signal $y[i,j]$ in



(a)



(b)

Figure 5: The 2DNC-IIR low-pass filter (a) response to WGN, and (b) the corresponding transient signal induced by the boundary conditions.

Figure 5(a), the corresponding transient signal is illustrated in Figure 5(b). Note that the energy in $y_e[i, j]$ is concentrated at the boundary of Ω_N, and has relatively insignificant values away from $\partial\Omega_N$. Nearly identical results are obtained for other realizations of the WGN input signal or if Neumann conditions are imposed in place of Dirichlet conditions.

The magnitude of the transient signal $y_e[i, j]$ relative to that of the filter output is a function of both the difference equation coefficients and the choice of boundary conditions. Fortunately, for the examples of the following section, boundary conditions are easily prescribed which lead to relatively insignificant transient signals. In fact, for the examples in which the energy of the filter response $y[i, j]$ is small near $\partial\Omega_N$, the transient effects are barely distinguishable from finite-precision arithmetic error.

5.2 The Utility of the Approximate Block LU Factorization

In this section, each of the four example filters described at the beginning of Section 5 is implemented with the approximate block LU algorithm of Section 4.1. One can check that, for each of the example difference equations, the coefficients satisfy $|n| + |s| \leq |c|/2$, a condition argued in Section 4.2 to allow efficient implementation of such filters by the approximation algorithm.

For analyzing the approximation errors, two error measures are given by

$$\epsilon = \frac{||\underline{e}||_2}{||\underline{y}||_2}. \qquad \eta = \frac{||\underline{e}||_1}{||\underline{y}||_1}, \tag{37}$$

where ϵ is the relative energy of the approximation error. The vector $\underline{e}$ contains the approximation errors at each point in Ω_N and $||\cdot||_p$, $p = 1, 2$, is the standard l_p norm.

For simplicity, homogeneous Dirichlet boundary conditions are assumed in all of the examples which follow. For the first three examples, assume $N = 64$. In the fourth example, the effect of variations in N upon the approximation accuracy is considered.

Example 1: The Frequency Selection of Low-pass and Fan Filters

In this example, we consider the response of the low-pass and fan filters to single frequency sinusoids of the form

$$x[i,j] = \frac{1}{N^2} \cos\left(\frac{2\pi k_1}{N} i\right) \cos\left(\frac{2\pi k_2}{N} j\right). \qquad (i,j) \in \Omega_N \tag{38}$$

The approximation errors are then analyzed for various values of β. Analysis for this example is thus most easily done in the frequency domain.

For each filter, two different input signals, corresponding to two different choices for the pair (k_1, k_2) in (38), are chosen such that one input lies in the filter pass-band and the other in the stop-band. For the low-pass filter, Equation (38) with $(k_1, k_2) = (3, 2)$ lies in the pass-band, while $(k_1, k_2) = (25, 20)$ is clearly in the stop-band (see Figure 4(a)). The exact response of the low-pass filter to the pass-band input is illustrated at the top of Figure 6, while the exact response to the stop-band input is illustrated at the top of Figure 7. The relative magnitudes of the two outputs illustrate the 9-to-1 selection ratio of the low-pass filter. Also, the transient signals — manifested by the ridges in the DFT's — are relatively small; in fact, these ridges are barely noticeable for the response to the stop-band sinusoid.

The effectiveness of the algorithm of Section 4.1 is demonstrated in Figures 6 and 7 for $\beta = 2$ and 4. For each of the two low-pass filter inputs, the approximate filter solution $y_a[i,j]$ and the approximation error $e[i,j] = y[i,j] - y_a[i,j]$ are illustrated. For both inputs, the approximation errors are small even when $\beta = 2$, and the errors decrease by an order of magnitude when β increases from 2 to 4. (The approximation error will be shown to decrease geometrically as a function β in Example 3.)

Similar results are obtained for the fan filter. For a sinusoidal input in the pass-band with $(k_1, k_2) = (0, 20)$, the exact filter response, the approximate responses for $\beta = 2$ and 4, and the corresponding approximation errors are illustrated in Figure 8. Figure 9 contains the analogous plots for the stop-band input given by $(k_1, k_2) = (25, 0)$. For both figures, the exact responses exhibit the frequency-selectivity given by H_4 in Figure 4(d), and the transient signals are relatively small. Also, as for the low-pass filter, the approximation errors are small even when $\beta = 2$ and decrease an order of magnitude when β increases to 4.

Example 2: Edge Enhancement of a Square Pulse

For this example, the response of the two edge-enhancing filters (J_2 and J_3 in Equation (35)) to a square pulse is approximated with $\beta = 2$ and $\beta = 4$. The square pulse input and the exact response of filter J_2 are illustrated at the top of Figure 10. The exact response of filter J_3 is given in Figure 11. Ignoring machine error, both of the exact responses are identical to those dictated by the frequency response of the filter difference equations, i.e., there is **no** transient signal induced by the boundary conditions. The absence of the transient is due to the fact that the filter responses determined directly from the frequency responses are zero on the boundary of Ω_N.

The approximation errors are unrecognizable in Figures 10 and 11, unless one closely examines the approximate solution for system J_3 when $\beta = 2$. In any case, for both filters the errors again decrease an order of magnitude as β increases from 2 to 4.

Example 3: Accuracy of the Approximation vs. β

To demonstrate the relationship between the approximation bandwidth β and the accuracy of the approximation, the relative energy of the approximation error, defined by ϵ in (37), is plotted versus β in Figure 12 for each of the four example filters. (For Figure 12, the input is WGN and the error ϵ is the sample average over 10 sample inputs.) For each filter, the errors decrease with β at a constant geometric rate, which is consistent with the analysis of Section 4.2. In Section 4.2, we argued that the approximation errors will be small, even for small β, when $|n| + |s| \leq |c|/2$. In these cases, the elements in the blocks $\overline{D}_i$ and $\overline{E}_i$ will typically decrease geometrically in magnitude with distance from the diagonal. Thus, for β-banded approximations of these blocks, one would expect the approximation errors to also decrease geometrically with increasing approximation bandwidth.

Example 4: The Independence of the Approximation Accuracy upon N

In Section 4.1, 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. However, if the computational and storage loads are to be truly constant **per pixel**, the approximation bandwidth needed for a desired approximation accuracy must not be an increasing function of N. Figure 13 shows that the approximation error remains constant over a wide range of N for a fixed value of $\beta = 4$. (As for Figure 12, the errors ϵ in Figure 13 are computed by averaging over ten WGN inputs.) 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 4.2, and we expect when $|n| + |s| \leq |c|/2$ that β will be independent of N for a desired level of approximation accuracy. Thus, the approximation algorithm has constant per pixel computational and storage load.

6 Conclusion

In this paper we describe an approach to the efficient implementations of 2-D noncausal IIR filters. In the past, 2-D IIR filters received comparatively limited attention due to the apparent difficulty of implementing these filters efficiently. In addition to efficiency, we were also motivated to consider filters specified by boundary conditions rather than initial conditions, as the former are frequently the natural choice and are required, for example, if zero-phase filtering is desired. Indeed, a number of methodologies now exist for designing 2-D difference equations to meet desired frequency-selective specifications [8], and we demonstrated that imposing boundary conditions, such as Dirichlet or Neumann, upon these difference equations leads to the desired frequency selectivity.

The approach we developed for efficiently implementing 2-D noncausal IIR filters involved 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 the algorithms' 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 just described is, in principle, applicable to a broad range of filtering problems, e.g., those that are higher order or have non-constant-coefficient difference equations. 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. These properties also suggest a theoretical investigation of general conditions on the difference equation coefficients under which our approach is guaranteed to provide accurate answers for small values of β. Finally, we also believe that there is much more that can be done in adapting other numerical methods, both direct and iterative, for solving PDE's in order to develop efficient procedures for 2-D IIR filtering.

A Computing Certain Elements of the Inverse of a Banded Matrix

A special application of the results in [5] is the ability to compute very efficiently 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

$$\begin{aligned} Z &= D^{-1}L^{-1} + (I-U)Z && (39) \\ Z &= D^{-1}U^{-1} + Z(I-L) && (40) \end{aligned}$$

From these relations, $[Z]_{lk}$ for $|l-k| \leq \beta$ is given by Equation (41), which **does not depend upon computing any elements in L^{-1} and U^{-1}.**

$$[Z]_{lk} = \begin{cases} ([D]_{ll})^{-1} - \sum_{m=l+1}^{l+\beta}[U]_{lm}[Z]_{ml}, & l=k \\ -\sum_{m=l+1}^{l+\beta}[U]_{lm}[Z]_{mk}, & m<k \\ -\sum_{m=l+1}^{l+\beta}[Z]_{lm}[L]_{mk}, & m>k \end{cases} \qquad (41)$$

For certain matrices, such as those discussed in Section 4.1, 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 $m \geq l$, $n \geq k$, and $|l-k| \leq \beta$ 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 closely by

$$\#\,\text{computations} < N[(\beta+1) + 2\beta^2]\,, \qquad (42)$$

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] T. M. Chin, W. C. Karl, and A. S. Willsky. Sequential filtering for multi-frame visual reconstruction. *Signal Processing*, August 1992.

[2] H. Derin and P. A. Kelly. Discrete-index Markov-type random processes. *IEEE Proceedings*, October 1989.

[3] D. E. Dudgeon and R. M. Mersereau. *Multidimensional digital signal processing.* Prentice Hall, Englewood Cliffs, NJ, 1984.

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

[5] A. M. Erisman and W. F. Tinney. On computing certain elements of the inverse of a sparse matrix. *Communications of the ACM*, 18(3), 1975.

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

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

[8] Q. Gu and M. N. S. Swamy. On the design of a broad class of 2-D recursive digitial filters with fan, diamond, and elliptically-symmetric responses. *IEEE Trans. Circuits Syst. II*, 41(9):603–614, September 1994.

[9] B. K. P. Horn. *Robot Vision.* MIT Press, Cambridge, MA, 1987.

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

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

[12] J. S. Lim. *Two-dimensional signal and image processing.* Prentice Hall, Englewood Cliffs, NJ, 1990.

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

[14] A. R. Mitchell and D. F. Griffiths. *The Finite Difference Method in Partial Differential Equations.* Wiley and Sons, 1980.

[15] N. A. Pendergrass, S. K. Mitra, and E. I. Jury. Spectral transformations for two-dimensional digital filters. *IEEE Transactions on Circuits and Systems*, 23(1):26–35, 1976.

[16] G. Sharma and R. Chellappa. Two-dimensional spectrum estimation using noncausal autoregressive models. *IEEE Transactions on Information Theory*, 32(2):268–275, March 1986.

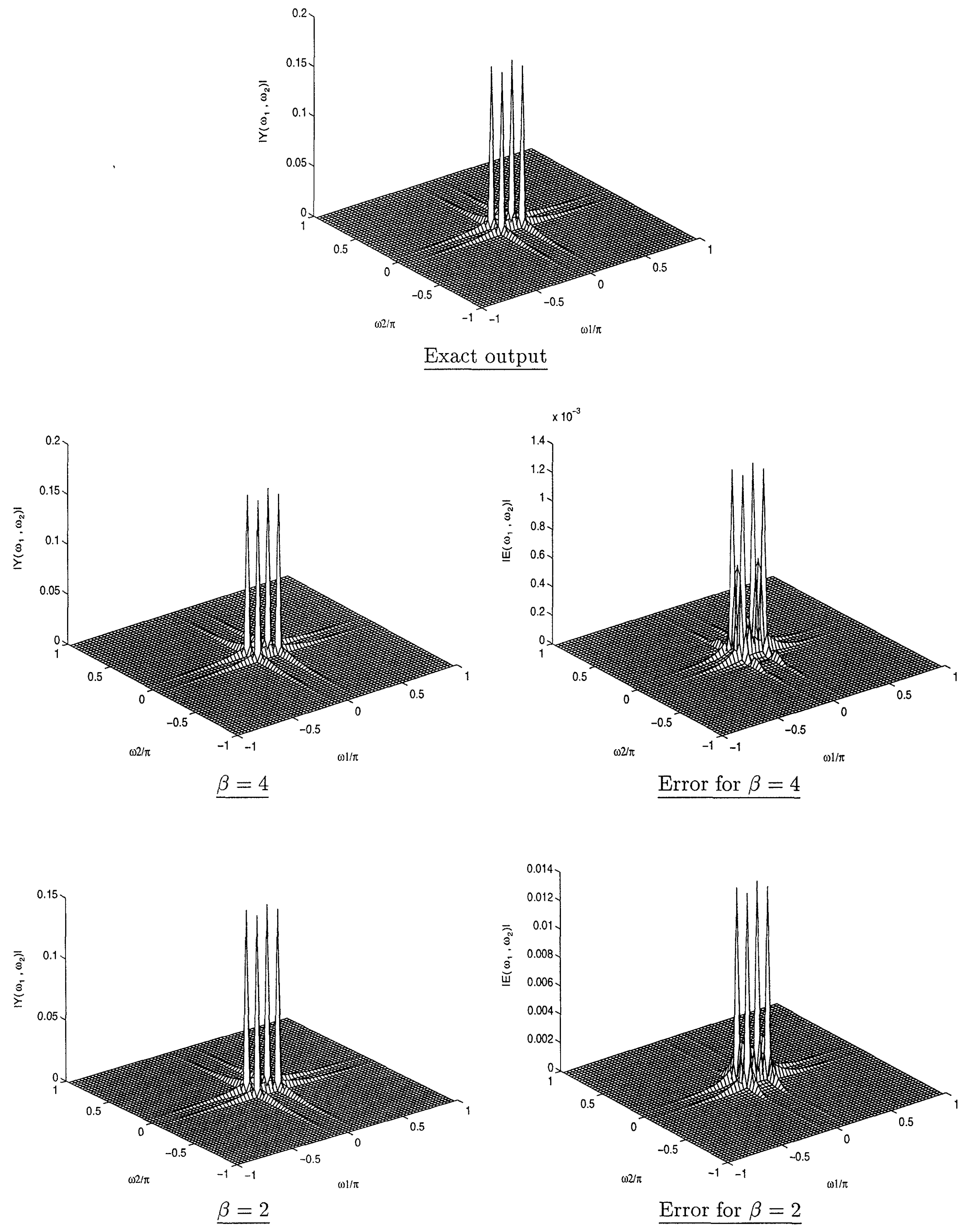


Figure 6: The DFT magnitude of the exact and approximate responses of the 2DNC-IIR **low-pass** filter to Equation (38) for $(k_1, k_2) = (3, 2)$. Note from the axes of the two error plots that the approximation error for $\beta = 4$ is an order of magnitude less than that for $\beta = 2$. For $\beta = 2$, $\epsilon = 9.1 \times 10^{-2}$ and $\eta = 8.6 \times 10^{-2}$. For $\beta = 4$, $\epsilon = 9.7 \times 10^{-3}$ and $\eta = 8.7 \times 10^{-3}$.

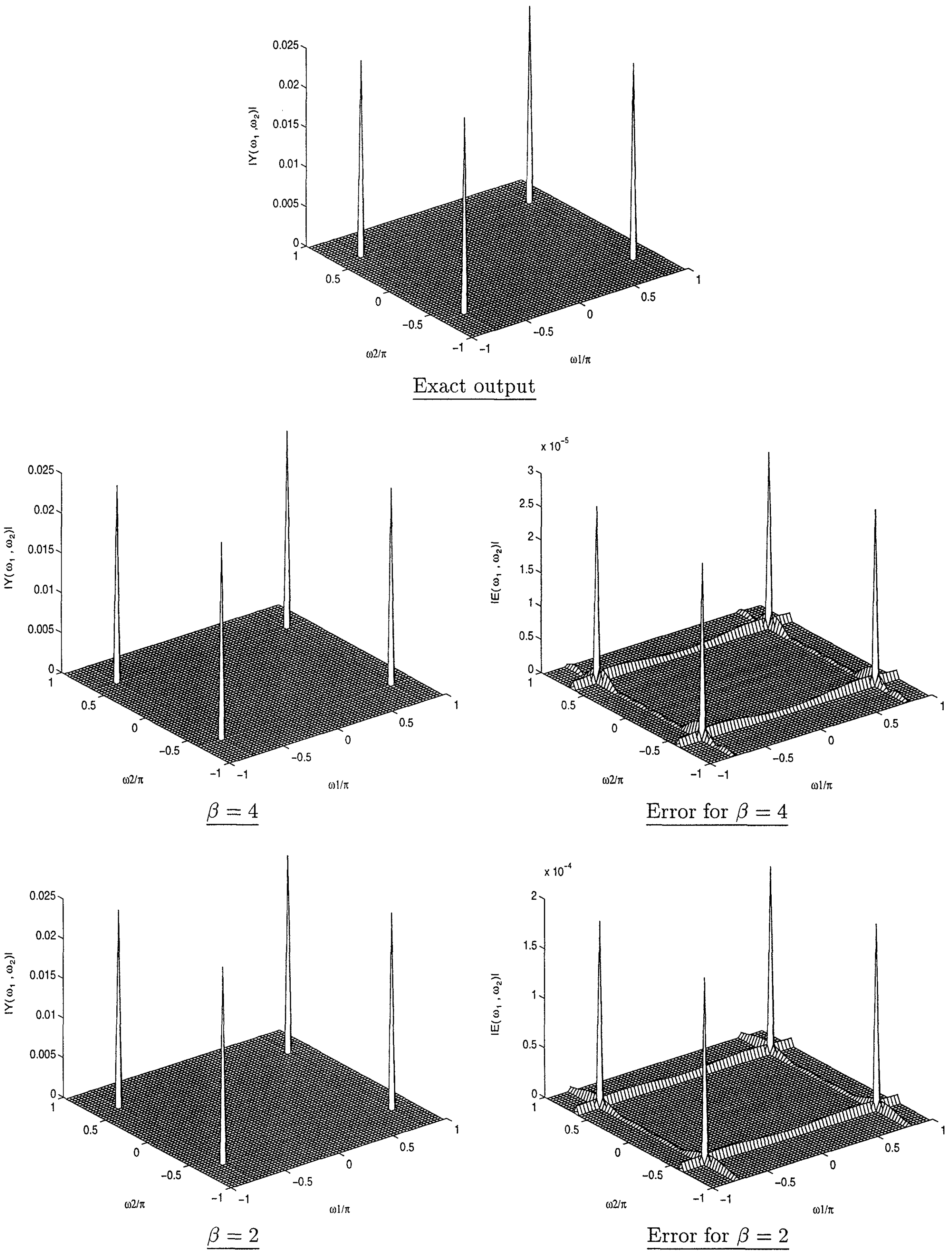
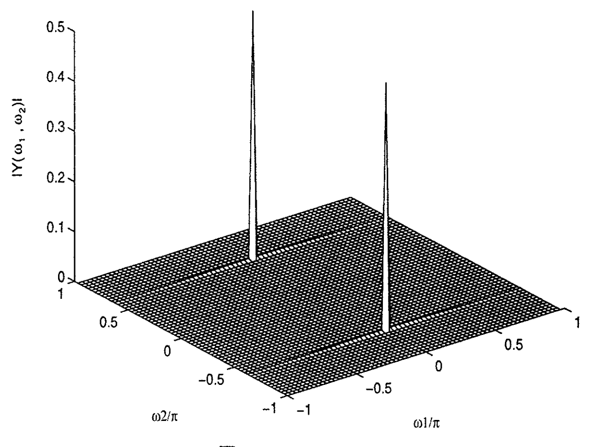
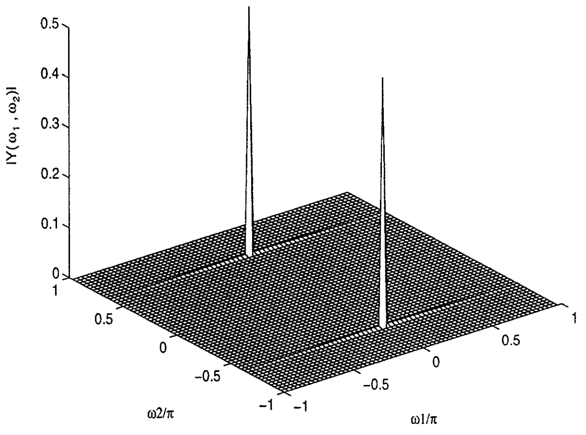


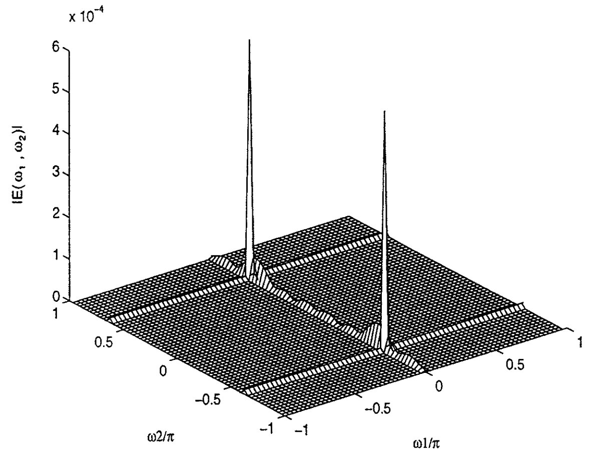
Figure 7: The DFT magnitude of the exact and approximate responses of the 2DNC-IIR **low-pass** filter to Equation (38) for $(k_1, k_2) = (25, 20)$. Note from the axes of the two error plots that the approximation error for $\beta = 4$ is an order of magnitude less than that for $\beta = 2$. For $\beta = 2$, $\epsilon = 8.0 \times 10^{-3}$ and $\eta = 7.8 \times 10^{-3}$. For $\beta = 4$, $\epsilon = 1.1 \times 10^{-3}$ and $\eta = 1.1 \times 10^{-3}$.



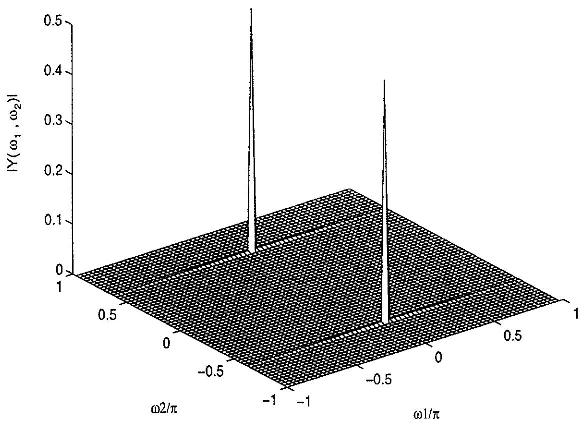
Exact output



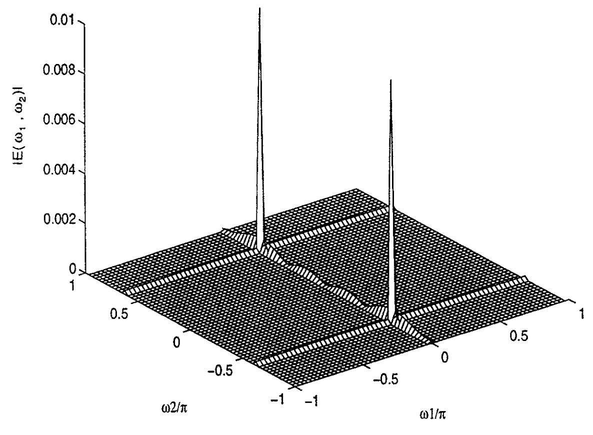
$\beta = 4$



Error for $\beta = 4$



$\beta = 2$



Error for $\beta = 2$

Figure 8: The DFT magnitude of the exact and approximate responses of the 2DNC-IIR **fan** filter to Equation (38) for $(k_1, k_2) = (0, 20)$. Note from the axes of the two error plots that the approximation error for $\beta = 4$ is an order of magnitude less than that for $\beta = 2$. For $\beta = 2$, $\epsilon = 2.0 \times 10^{-2}$ and $\eta = 2.0 \times 10^{-2}$. For $\beta = 4$, $\epsilon = 1.2 \times 10^{-3}$ and $\eta = 1.2 \times 10^{-3}$.

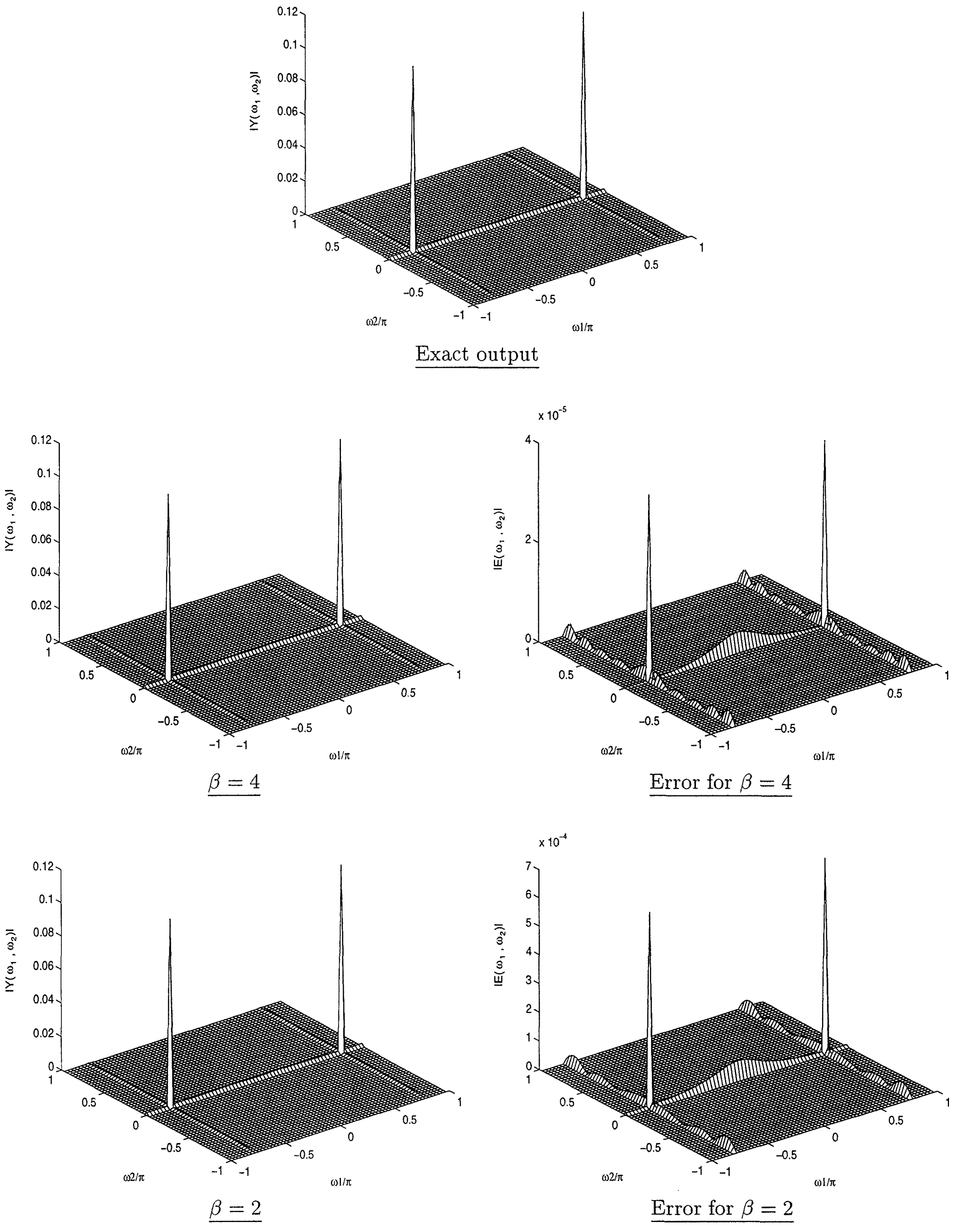


Figure 9: The DFT magnitude of the exact and approximate responses of the 2DNC-IIR **fan** filter to Equation (38) for $(k_1, k_2) = (25, 0)$. Note from the axes of the two error plots that the approximation error for $\beta = 4$ is an order of magnitude less than that for $\beta = 2$. For $\beta = 2$, $\epsilon = 6.4 \times 10^{-3}$ and $\eta = 6.3 \times 10^{-3}$. For $\beta = 4$, $\epsilon = 3.6 \times 10^{-4}$ and $\eta = 3.4 \times 10^{-4}$.

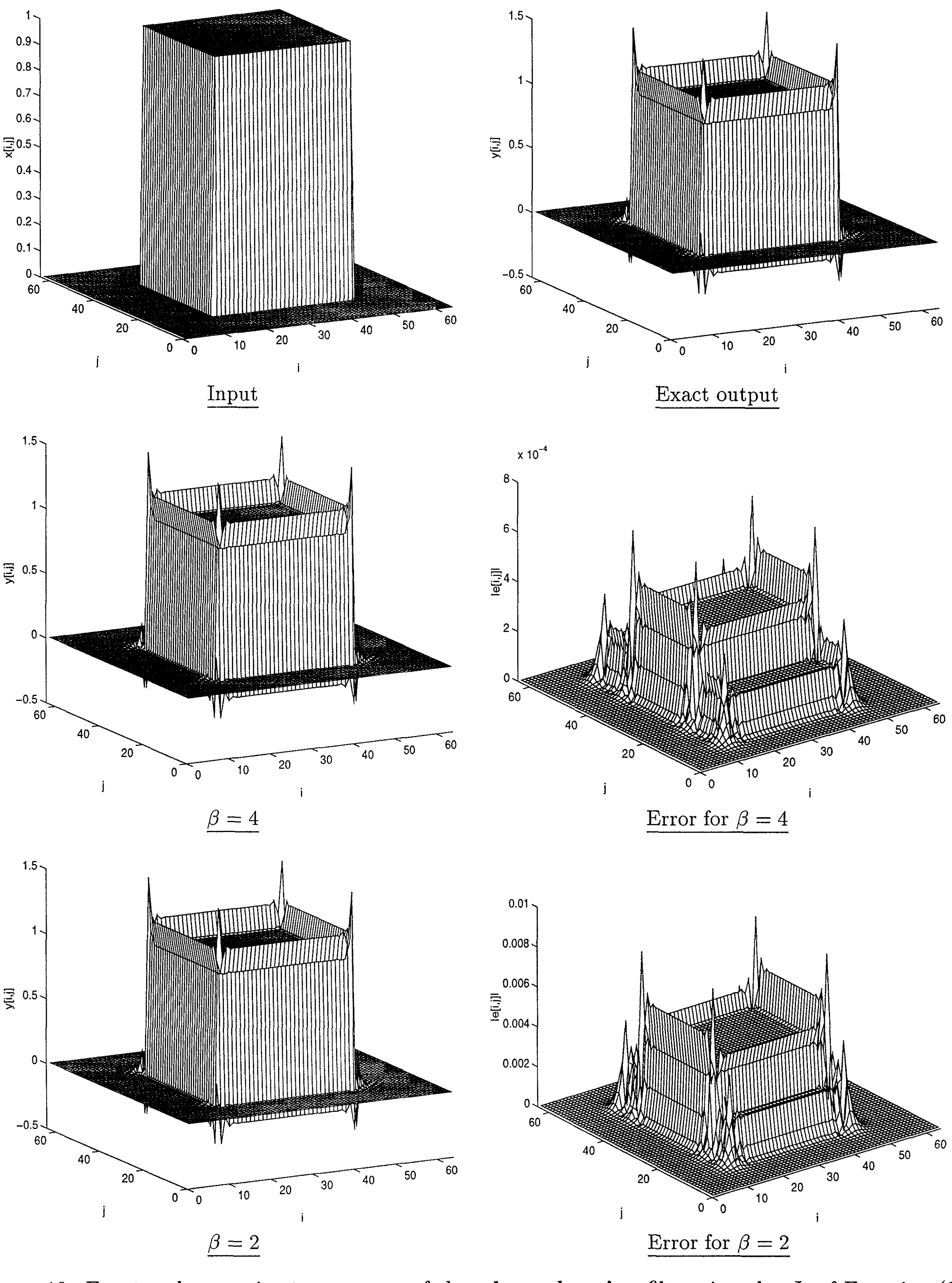


Figure 10: Exact and approximate responses of the **edge-enhancing** filter given by J_2 of Equation (35) to a square pulse. Note from the axes of the two error plots that the approximation error for $\beta = 4$ is an order of magnitude less than that for $\beta = 2$. For $\beta = 2$, $\epsilon = 4.0 \times 10^{-3}$ and $\eta = 4.3 \times 10^{-3}$. For $\beta = 4$, $\epsilon = 3.0 \times 10^{-4}$ and $\eta = 3.3 \times 10^{-4}$.

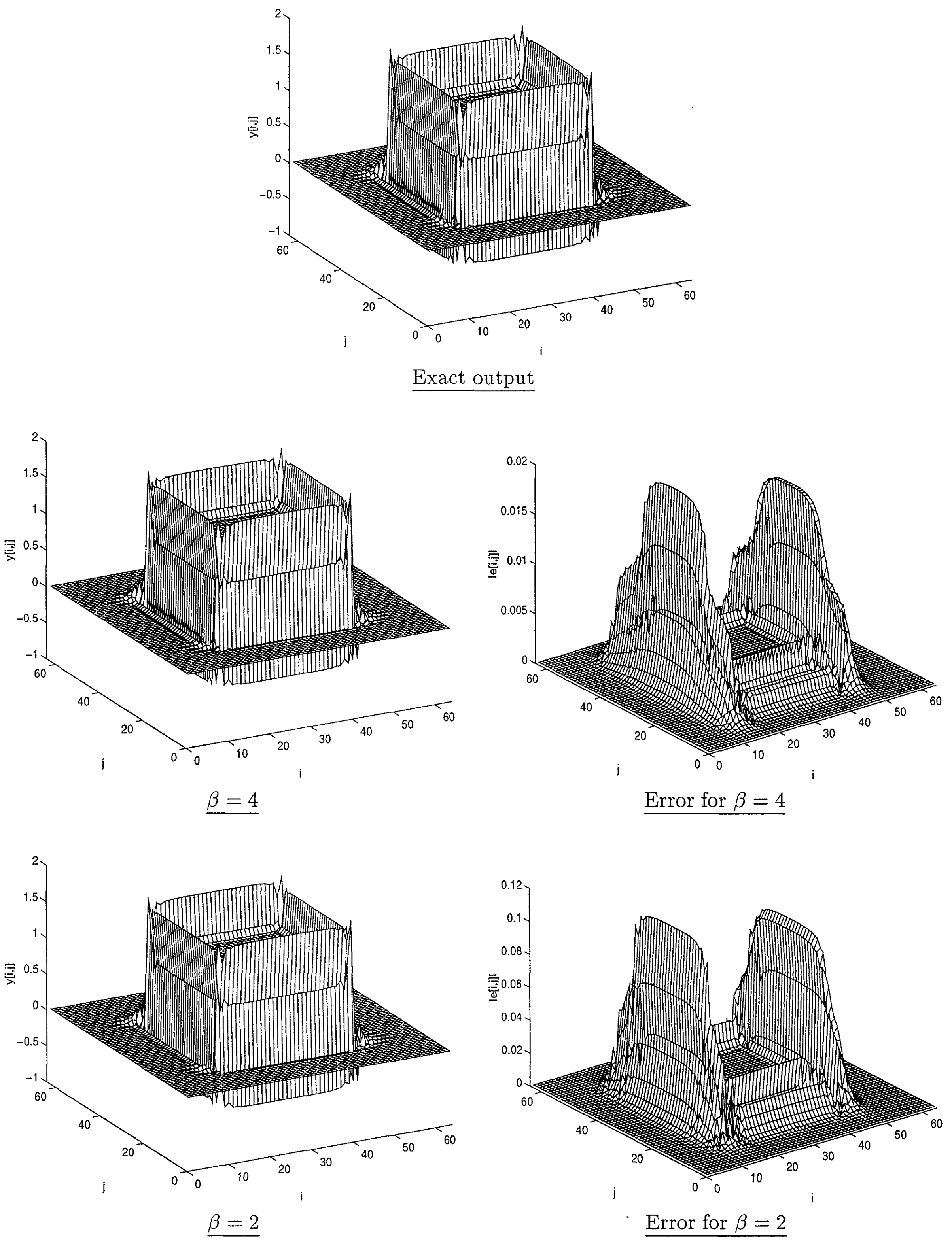


Figure 11: Exact and approximate responses of the *edge-enhancing* filter given by J_3 of Equation (35) to a square pulse. Note from the axes of the two error plots that the approximation error for $\beta = 4$ is an order of magnitude less than that for $\beta = 2$. For $\beta = 2$, $\epsilon = 3.6 \times 10^{-2}$ and $\eta = 3.2 \times 10^{-2}$. For $\beta = 4$, $\epsilon = 6.3 \times 10^{-3}$ and $\eta = 5.7 \times 10^{-3}$.

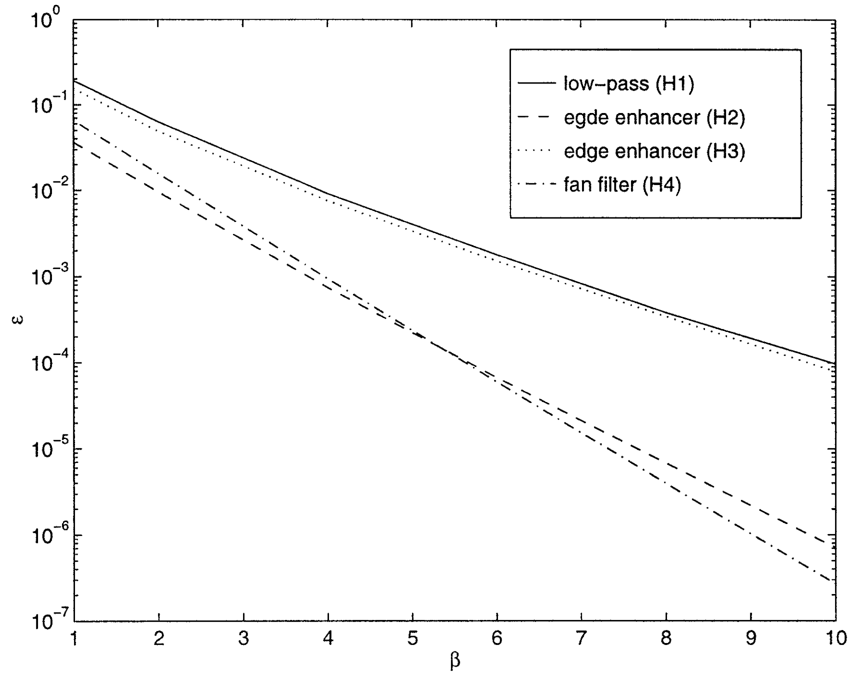


Figure 12: Approximation errors vs. β when $N = 64$. The four example filters H_i are given by the difference equation coefficients in Equations (35) and (36).

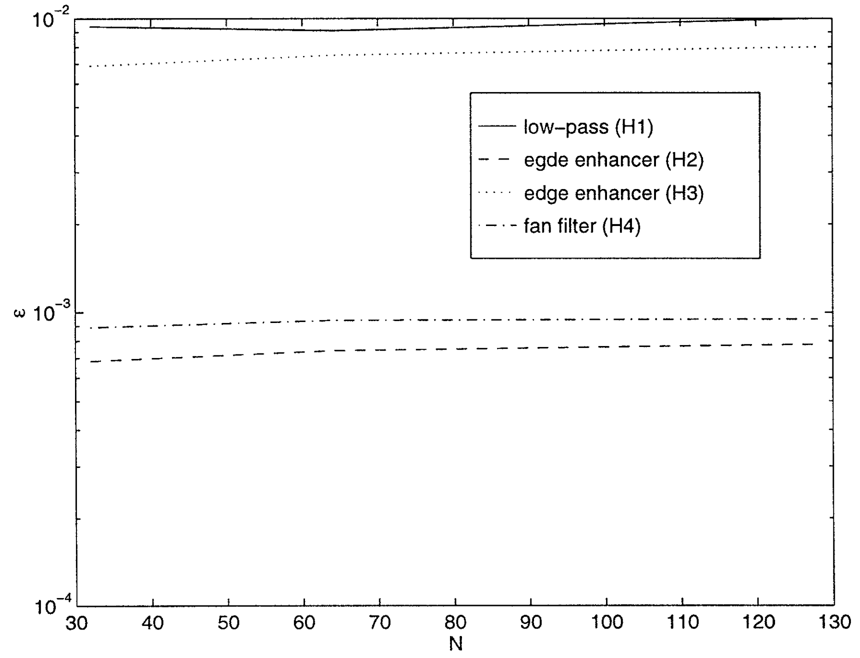


Figure 13: Approximation errors for $\beta = 8$. The four example filters H_i are given by the difference equation coefficients in Equations (35) and (36).