On a recursive Schur preconditioner for iterative solution of a class of dense matrix problems

Judith M. Ford[*][†], Ke Chen[‡], David Evans[§]

December 31, 2001

Abstract

There are currently several distinct preconditioning methods for dense matrices based on applying a wavelet transform to obtain a matrix with a large number of small entries. A sparse preconditioner for this transformed matrix can be formed by setting to zero entries that are assumed to be unimportant. The effectiveness of the preconditioner depends on retaining the most important entries and on ensuring that they are positioned conveniently within the transformed matrix. In this paper we present a new, recursive preconditioning strategy that takes into account more of the significant entries without greatly increasing cost and outperforms existing methods in certain cases.

Keywords: Schur complement, GMRES, DWT, dense matrices

AMS subject classification: 65F10, 65N22, 65T60

1 Introduction

We are interested in linear systems

$$Ax = b, \quad x, b \in \mathbb{C}^n, \ A \in \mathbb{C}^{n\times n} \tag{1}$$

where A is a large, dense matrix. Such matrices arise in a number of applications including acoustic scattering [1], elastohydrodynamic lubrication [17] and electromagnetic

[*]Judith M. Ford was supported in this work by the Engineering and Physical Sciences Research Council, UK

[†]Department of Mathematical Sciences, University of Liverpool, Liverpool L69 7ZL, UK. judyford@liv.ac.uk

[‡]Department of Mathematical Sciences, University of Liverpool, Liverpool L69 7ZL, UK. k.chen@liv.ac.uk //www.liv.ac.uk/maths/applied

[§]Department of Computing, Nottingham Trent University, Burton Street, Nottingham NG1 4BU, UK

simulations [15]. Krylov subspace iterative methods, such as GMRES (described in [19]), can be used to solve (1), but in most cases preconditioning is required in order to obtain good convergence. One method of preconditioning is to seek a matrix $M \approx A$ such that

1. the iteration converges quickly when applied to AM^{-1}, and

2. $M^{-1}v$ can be calculated cheaply for any vector v.

In many cases the task of finding such an M is made easier by transforming (1) into a wavelet basis. (See, for example, [4, 5, 6, 7, 10, 13, 14, 15].) When a discrete wavelet transform (DWT, see, for example, [20]) is applied to a smooth matrix A, the result is a matrix $\tilde{A}$ with many small elements. A sparse approximation $\bar{A}$ to the transformed matrix can be obtained by setting to zero those elements that are sufficiently small to be considered unimportant. This is the main idea underlying most wavelet-based preconditioners.

Typically, $\bar{A}$ has a "finger pattern" such as that shown in figure 1(b). Although $\bar{A}$ is sparse, this sparsity pattern means that $\bar{A}^{-1}v$ cannot be computed efficiently, which makes it unsuitable as a preconditioner because, although the first of the above criteria for a good preconditioner is likely to be satisfied, the second is not. This issue is not widely addressed because most authors in wavelets literature are concerned with operator compression (i.e. obtaining $\bar{A}$) and not with iterative solution. To appreciate the nature of the problem, consider again figure 1, which shows the distribution of larger elements within the transformed matrix when a Daubechies 4 DWT is applied to the Calderón-Zygmund matrix

$$A_{ij} = \begin{cases} 2, & i = j, \\ 1/|i-j|, & i \neq j, \end{cases} \tag{2}$$

where $i, j = 1, \ldots n$. When a level 1 DWT is applied, the resulting matrix (figure 1(a)) is insufficiently sparse to be of much use as a preconditioner. The level 4 transformed matrix (figure 1(b)) is sparser, but has the problematic finger pattern, which causes a large amount of fill-in when $\bar{A}$ is decomposed into LU factors. To obtain a preconditioner that is cheaper to apply, we can form a banded M by setting to zero those elements of $\bar{A}$ lying outside of a chosen diagonal band, but this may be at the cost of impaired convergence. For some matrices this type of banded preconditioner works well, but for others neglect of the the off-diagonal elements results in poor convergence (see [5]).

In this paper, we propose a recursive band preconditioner based on exploring how the finger pattern arises and developing ways of avoiding its adverse consequences. We remark that a different idea of treating the finger pattern matrix in [8] uses the sparse inverse approximation in the space of wavelets for the inverse operator. To introduce our idea, we first adopt the following notation, based on that used in [14] to describe the NS-form of an operator.

$\langle \alpha \rangle_m, \quad \alpha \in \mathbb{Z}, \quad m \in \mathbb{N}$ denotes the value of α modulo m.
$[a], \quad a \in \mathbb{R}$ denotes the integer part of a.

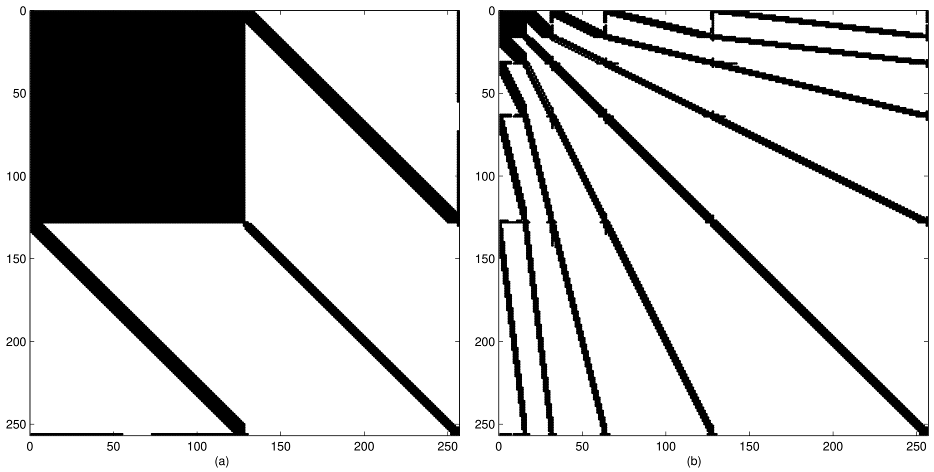

Figure 1: Standard DWT of 256 × 256 Calderón-Zygmund matrix, showing elements > 10^{-3}. Left: level 1, right: level 4.

Let $c_0, c_1, \ldots c_{m-1}$, $d_0, d_1, \ldots d_{m-1}$ be the wavelet filter coefficients. Define P_i, Q_i to be the $n/2^i \times n/2^{i-1}$ matrices

$$P_i = \begin{pmatrix} c_0 & c_1 & \cdots & c_{m-1} & 0 & \cdots & \cdots & 0 \\ 0 & 0 & c_0 & c_1 & \cdots & c_{m-1} & \cdots & \vdots \\ \vdots & & & \ddots & \ddots & & & \vdots \\ & & & & & & & \\ \vdots & & & & & \ddots & \ddots & \vdots \\ c_2 & c_3 & \cdots & c_{m-1} & \cdots & \cdots & c_0 & c_1 \end{pmatrix}, \tag{3}$$

$$Q_i = \begin{pmatrix} d_0 & d_1 & \cdots & d_{m-1} & 0 & \cdots & \cdots & 0 \\ 0 & 0 & d_0 & d_1 & \cdots & d_{m-1} & \cdots & \vdots \\ \vdots & & & \ddots & \ddots & & & \vdots \\ & & & & & & & \\ \vdots & & & & & \ddots & \ddots & \vdots \\ d_2 & d_3 & \cdots & d_{m-1} & \cdots & \cdots & d_0 & d_1 \end{pmatrix}. \tag{4}$$

Given an n-vector v, define $s^{(0)} = v$, $s^{(i)} = P_i s^{(i-1)}$, $d^{(i)} = Q_i s^{(i-1)}$. Equivalently,

$$s_j^{(i)} = \sum_{l=0}^{m-1} c_l s^{(i-1)}_{\langle l+2j \rangle_{n/2^{i-1}}}, \quad d_j^{(i)} = \sum_{l=0}^{m-1} d_l s^{(i-1)}_{\langle l+2j \rangle_{n/2^{i-1}}}, \quad j = 0, 1, \ldots, n/2^i - 1. \tag{5}$$

Given an $n \times n$ matrix, A, define $T_0 = A$,

$$T_i = P_i T_{i-1} P_i^T, \tag{6}$$

$$A_i = Q_i T_{i-1} Q_i^T, \tag{7}$$

$$B_i = Q_i T_{i-1} P_i^T, \tag{8}$$

$$C_i = P_i T_{i-1} Q_i^T, \tag{9}$$

$$\tilde{T}_i = \begin{pmatrix} A_{i+1} & B_{i+1} \\ C_{i+1} & T_{i+1} \end{pmatrix}. \tag{10}$$

Applying a 1-D level 1 standard DWT to the columns of T_{i-1} is equivalent to pre-multiplying by $W_i = \begin{pmatrix} P_i \\ Q_i \end{pmatrix}$. In the same way a 1-D DWT of the rows can be obtained by post-multplying by W_i^T. A 2-D DWT of T_{i-1} is given by $W_i T_{i-1} W_i^T$. Thus $\tilde{T}_i$ is just a re-arrangement of the 2-D level 1 standard DWT of T_i.

The finger pattern we saw in figure 1 arises in the following way. Consider an $n \times n$ diagonal band matrix D. To apply a level 1 DWT we first transform the columns. The resultant matrix $\hat{D}_1$ has two blocks each of size $n/2 \times n$ with a wrap-round band (see figure 2(a)). Because the submatrices are rectangular, rather than square, the bands are no longer parallel to the diagonal of the matrix. When the level 1 DWT is completed by transforming the rows of $\hat{D}_1$ to give $\tilde{D}_1$ (figure 2(b)) each of the submatrices is transformed into two $n/2 \times n/2$ blocks each with a wrap-round band. At the next level we apply a level 1 column transform *to the upper half only* of $\tilde{D}_1$ to give $\hat{D}_2$, which has four rectangular blocks in its upper half (figure 2(c)). We then apply a row transform *to the leftmost half only* of $\hat{D}_2$ to give $\tilde{D}_2$ (figure 2(d)). The fingers are the result of transforming the two off-diagonal submatrices of $\tilde{D}_1$ in one direction only.

The finger pattern can be avoided by permuting the rows and columns of $\tilde{A}$ to centre the fingers as shown in figure 3(a). A preconditioner can now be formed by setting to zero elements lying outside a diagonal band. This method (DWTPer), described in [7], has been shown to be effective in a number of applications (see, for example, [7, 12, 13]).

Alternatively NS-forms (see [3, 14]) can be used to represent A in terms of the blocks of a larger matrix. The non-zero submatrices of the level k NS-form of A are the A_i, B_i, C_i, $i = 1, \ldots k$ and T_k defined above. The level 8 NS form of a 256×256 Calderón-Zygmund matrix is shown in figure 3(b). Notice that each of the submatrices has a diagonal structure. The finger pattern has been avoided by transforming only the T_i submatrix at each level. This means that the mapping from A to its NS-form is not a straightforward matrix operation, as is the standard DWT. The NS-form is a new type of matrix with a new multiplication operator. Using NS-forms, an approximate solution to (1) can be obtained at low cost by employing a non-standard LU-factorization procedure. This approach has been used in [14] to solve linear equations directly, but here we consider the use of NS-forms to precondition an iterative solution method.

Instead of attempting to solve the NS-form equations directly, in this paper we propose a recursive algorithm to improve the approximation of the finger pattern matrices by

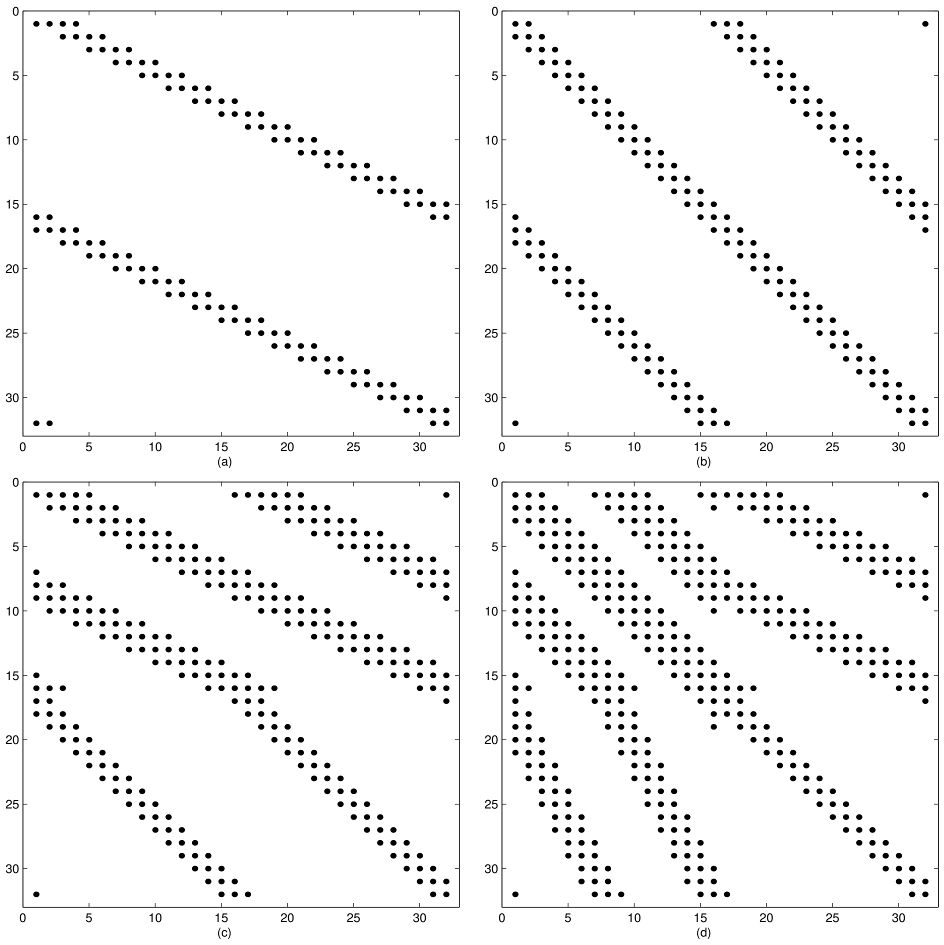

Figure 2: DWT of a diagonal matrix. (a) level 1 column DWT, (b) level 1 column and row DWT, (c) level 2 column and level 1 row DWT, (d) level 2 column and row DWT.

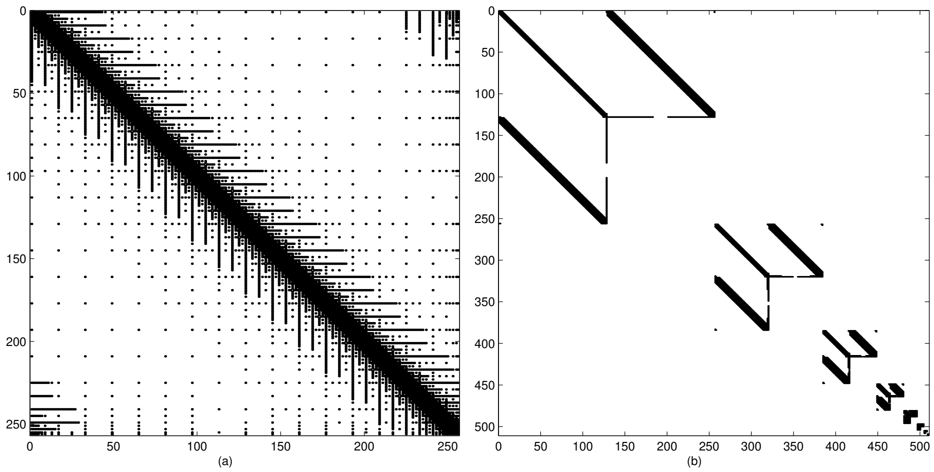

Figure 3: Alternative representations of DWT of 256×256 Calderón-Zygmund matrix, showing elements $> 10^{-3}$. Left: level 4 DWTPer, right: level 8 NS-form .

working with the submatrices of the NS-form (acting on vectors in $\mathbb{C}^{n/2^i}$ and using the usual arithmetic) to produce a new preconditioning method for dense matrices with non-smooth diagonal features. Our method exploits the diagonal band structure of these submatrices, but is fundamentally different from the ideas presented in [3, 14] where NS-forms are studied as operators on spaces other than $\mathbb{C}^j$, $j \in \mathbb{N}$.

2 A recursive preconditioning method

When a level 1 DWT is applied to a smooth matrix with a diagonal singularity the result is a block matrix with three banded blocks and one full block (see figure 1). This structure has been exploited in [3, 14] in the development of NS-forms to represent, by sparse matrices, a class of operators whose representation in the standard basis is dense. For suitably smooth A, NS-forms can be used to solve (1) to any required degree of accuracy by means of multiresolution LU factorization. We propose to use the blocks of the NS-form to solve (1) in a rather different way. The main features of our approach are the use of

1. flexible GMRES, which allows a different preconditioning matrix to be used at each iteration;

2. Schur complement to exploit the banded block structure of the level 1 transformed matrix;

3. recursion to reduce successively the dimension of the dense block.

If $A = T_0$ is smooth (apart from a diagonal band) then, in each $\tilde{T}_i$, A_{i+1}, B_{i+1}, C_{i+1} will have small entries except for a wrap-round diagonal band (see [14]). So we can approximate them by $\bar{A}_{i+1}$, $\bar{B}_{i+1}$, $\bar{C}_{i+1}$, formed by cutting to a band, giving an approximation $\bar{T}_i$ to $\tilde{T}_i$:

$$\bar{T}_i = \begin{pmatrix} \bar{A}_{i+1} & \bar{B}_{i+1} \\ \bar{C}_{i+1} & T_{i+1} \end{pmatrix}. \tag{11}$$

$\bar{T}_i$ is a potential preconditioner for $\tilde{T}_i$, but, unless the dimension of the dense block T_{i+1} is small, it will be expensive to apply directly. We now show how the $\bar{T}_i$ can be used in the solution of (1) without forming the $\bar{T}_i$ explicitly. We set $T_0 = A$ and use a flexible GMRES iteration (see §2.2) preconditioned by approximate solution of an equation of the form

$$T_0 y = v \tag{12}$$

at each step. To do this we first apply a level 1 DWT to give

$$\begin{pmatrix} A_1 & B_1 \\ C_1 & T_1 \end{pmatrix} \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix} = \begin{pmatrix} \tilde{v}_1 \\ \tilde{v}_2 \end{pmatrix}, \tag{13}$$

where $\tilde{y}_1 = Q_1 y$, $\tilde{y}_2 = P_1 y$, $\tilde{v}_1 = Q_1 v$, $\tilde{v}_2 = P_1 v$. We approximate this equation by

$$\begin{pmatrix} \bar{A}_1 & \bar{B}_1 \\ \bar{C}_1 & T_1 \end{pmatrix} \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix} = \begin{pmatrix} \tilde{v}_1 \\ \tilde{v}_2 \end{pmatrix}, \tag{14}$$

and solve using the Schur complement $S_1 = T_1 - \bar{C}_1 \bar{A}_1^{-1} \bar{B}_1$ (see [2, 9]). This requires us to solve an equation of the form

$$S_1 z = w, \quad z, w \in \mathbb{C}^{n/2} \tag{15}$$

(see §2.1), which we do by a further flexible GMRES iteration. We expect that T_1 will be an effective preconditioner for S_1 (see [2, §9.3]), but, because T_1 is dense, it will be expensive to apply directly. At each iteration during the solution of (15) we must solve an equation of the form

$$T_1 y = v, \quad y, v \in \mathbb{C}^{n/2}. \tag{16}$$

To do this cheaply we repeat the process of applying a level 1 DWT and using the Schur complement and obtain an equation of the form

$$S_2 z = w, \quad z, w \in \mathbb{C}^{n/4}. \tag{17}$$

This in turn can be solved using flexible GMRES preconditioned by T_2. We continue recursively, solving equations of the form

$$S_i z = w, \quad z, w \in \mathbb{C}^{n/2^i}. \tag{18}$$

iteratively, preconditioning by solving equations of the form

$$T_i y = v, \quad y, v \in \mathbb{C}^{n/2^i}, \tag{19}$$

until the matrix T_i is small enough that T_i^{-1} can be applied directly by means of LU factorization at low cost. Thus we have many layers of GMRES iterations with all but the innermost layer calling on GMRES at each iteration to apply the preconditioning step. In order to do this $T_1, \ldots, T_k$ must be computed, where k is the highest level of transform required. (That is, $\frac{n}{2^k}$ is sufficently small that $T_k x = b$ can be solved by a direct method.) This need only be done once, and the submatrices stored for future use. The following sections describe the method in more detail.

2.1 The Schur complement method

Suppose that we wish to solve

$$T_i y = v. \tag{20}$$

We apply a level 1 DWT to obtain

$$\tilde{T}_i \tilde{y} = \tilde{v}, \tag{21}$$

where $\tilde{y} = \begin{pmatrix} Qy \\ Py \end{pmatrix} = \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix}$, $\tilde{v} = \begin{pmatrix} Qv \\ Pv \end{pmatrix} = \begin{pmatrix} \tilde{v}_1 \\ \tilde{v}_2 \end{pmatrix}$, and $\tilde{T}_i$ is a given by (10). Equation (21) can be approximated by

$$\bar{T}_i \tilde{y} = \tilde{v}, \tag{22}$$

which can then be solved by a partial LU decomposition of $\bar{T}_i$ to give

$$\begin{pmatrix} \bar{A}_{i+1} & 0 \\ \bar{C}_{i+1} & I \end{pmatrix} \begin{pmatrix} I & \bar{A}_{i+1}^{-1} \bar{B}_{i+1} \\ 0 & T_{i+1} - \bar{C}_{i+1} \bar{A}_{i+1}^{-1} \bar{B}_{i+1} \end{pmatrix} \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix} = \begin{pmatrix} \tilde{v}_1 \\ \tilde{v}_2 \end{pmatrix}, \tag{23}$$

yielding

$$\bar{A}_{i+1} \tilde{w}_1 = \tilde{v}_1, \tag{24}$$

$$\bar{C}_{i+1} \tilde{w}_1 + \tilde{w}_2 = \tilde{v}_2, \tag{25}$$

$$\tilde{y}_1 + \bar{A}_{i+1}^{-1} \bar{B}_{i+1} \tilde{y}_2 = \tilde{w}_1, \tag{26}$$

$$S_{i+1} \tilde{y}_2 = \tilde{w}_2. \tag{27}$$

Since $\bar{A}_i$, $\bar{B}_i$, $\bar{C}_i$ are sparse, equations (24), (25) and (26) can be solved directly at low cost. Equation (27) can be solved by a GMRES iteration, preconditioned by T_{i+1}.

The fundamental building block of our method is Algorithm 1 below. This provides a way of finding an approximate solution of (20) by transforming T_i to $\bar{T}_i$ and then solving $\bar{T}_i \tilde{y} = \tilde{v}$, using the Schur complement in conjunction with flexible GMRES. The GMRES iteration is preconditioned either by a further application of Algorithm 1 or, at the highest level, by the LU factors L_{i+1}, U_{i+1} of T_{i+1}.

Algorithm 1 *Approximate solution of* $T_i y = v$

1. *Compute* $\tilde{v}_1 = Q_i v$, $\tilde{v}_2 = P_i v$.
2. *Solve* $\bar{A}_{i+1}\tilde{w}_1 = \tilde{v}_1$ *for* $\tilde{w}_1$.
3. *Solve* $\bar{C}_{i+1}\tilde{w}_1 + \tilde{w}_2 = \tilde{v}_2$ *for* $\tilde{w}_2$.
4. *Define* $S_{i+1} = T_{i+1} - \bar{C}_{i+1}\bar{A}_{i+1}^{-1}\bar{B}_{i+1}$
5. *Solve* $S_{i+1}\tilde{y}_2 = \tilde{w}_2$ *for* $\tilde{y}_2$ *by flexible GMRES iteration, preconditioning with* T_{i+1}, *using Algorithm 1 to apply the preconditioner if* $i+1 \leq k$ *and using LU matrices* L_{i+1}, U_{i+1} *otherwise.*
6. *Solve* $\tilde{y}_1 + \bar{A}_{i+1}^{-1}\bar{B}_{i+1}\tilde{y}_2 = \tilde{w}_1$ *for* $\tilde{y}_1$.
7. *Set* $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} Q_i^T \tilde{y}_1 \\ P_i^T \tilde{y}_2 \end{pmatrix}$

2.2 Flexible GMRES

GMRES may be preconditioned by multiplying on the left or right by a matrix M^{-1}, where M is chosen to ensure that $M^{-1}A$ (for *left* preconditioning) or AM^{-1} (for *right* preconditioning) give better convergence of GMRES than A itself. Effectively this type of preconditioning changes the equation that we are solving from $Ax = b$ to $M^{-1}Ax = M^{-1}b$ or $AM^{-1}y = b$, $Mx = y$. We wish to use a different way of preconditioning: approximate solution of $Ax = v$. To do this we use a variant of right preconditioned GMRES called 'flexible GMRES', presented in [18]. This is set out below as Algorithm 2. It allows us to use a different M at each iteration, or, as in the current case, not to use a matrix M at all. The computational cost of flexible GMRES is the same as that of standard right-preconditioned GMRES, but, because it is necessary to store the m preconditioned vectors $M_j^{-1}v_j$ as well as the Krylov subspace vectors v_j, the storage requirement is increased by a small amount.

Algorithm 2 *Flexible GMRES*

1. **Start:** Choose x_0 and m. Define an $(m+1)\times m$ matrix $\bar{H}_m$ and initialise its entries $h_{i,j}$ to zero.
2. **Arnoldi process**
   (a) Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$.
   (b) For $j = 1, \ldots, m$, do
   - Compute $z_j = M_j^{-1}v_j$;
   - Compute $w = Az_j$;

- For $i = 1, \ldots, j$, do $\begin{cases} h_{i,j} = \langle w, v_i \rangle, \\ w = w - h_{i,j} vi; \end{cases}$
- Compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$.

(c) Define $Z_m = [z_1, \ldots, z_m]$.

3. **Form the approximate solution:**
   Compute $x_m = x_0 + Z_m y_y$ where $y_m = \text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ and $e_1 = [1, 0, \ldots, 0]^T$.

4. **Restart:** If satisfied stop, else set $x_0 \leftarrow a_m$ and go to 2.

2.3 Calculating a banded approximation to A_{i+1}, B_{i+1}, C_{i+1}

In the case where T_i is smooth apart from a diagonal band, we can approximate A_{i+1}, B_{i+1}, C_{i+1} by cutting to a band. This has two advantages over thresholding:

- The band structure reduces the amount of fill-in that occurs under LU factorization.
- $\bar{T}_i$ can be calculated cheaply by computing only those elements that will be included in the band.

In this section we consider two issues connected with this band approximation. Firstly we establish a criterion for the choice of bandwidth, based on the width of the non-smooth band in T_i. Secondly, we look at a way of reducing the work involved in performing the DWT of T_i to obtain $\bar{T}_i$. A similar analysis for DWTPer may be found in [7] and [11].

We first consider the effect of DWT on a band matrix. Figure 2 illustrates how applying level 1 DWT to a band matrix, first in the column direction and then in the row direction produces new matrices with a predictable band structure. A diagonal matrix is transformed by an order 4 column DWT to give the double band matrix shown top left. When the rows are also transformed, the result is a matrix consisting of four submatrices, each with a band structure (top right).

Because we are using periodic DWT it is convenient to assume that the matrix A has a wrap-round band and to work in modulo n arithmetic. To predict the positions of the non-zero entries after a 2-D transform of a band matrix we must consider the effect on both the columns and the rows.

Theorem 1 *After a 2-D level 1, order m, DWT has been applied to an $n \times n$ wrap-round band matrix, A, with lower bandwidth λ and upper band width μ, the resultant matrix $\tilde{A}$ comprises four $n/2 \times n/2$ wrap-round band submatrices each with lower bandwidth $\lambda_1 \leq [(\lambda + m))/2]$ and upper band width $\mu_1 \leq [(\mu + m))/2]$.*

Proof

We first transform the columns of A to give $\hat{A}$. Consider column j:

$$a_{ij} = 0 \quad \text{for } i < j - \mu \text{ or } i > j + \lambda.$$

From the recurrence relation (5) it is clear that, for $i < n/2$,

$$\hat{a}_{ij} = 0 \quad \text{for } m - 1 + 2i < j - \mu \text{ or } 2i > j + \lambda. \tag{28}$$

(A similar result holds for $n/2 \leq i < n$.) Now we apply a row DWT to $\hat{A}$, giving $\tilde{A}$. Consider row i, $i < n/2$:

$$\hat{a}_{ij} = 0 \quad \text{for } j > 2i + \mu + m - 1 \text{ or } j < 2i - \lambda. \tag{29}$$

From (5), for $j < n/2$,

$$\tilde{a}_{ij} = 0 \quad \text{for } 2j > 2i + \mu + m - 1 \text{ or } m - 1 + 2j < 2i - \lambda. \tag{30}$$

(Again a similar result holds for $n/2 \leq j < n$.) Hence the bandwidths of the submatrices are given by $(\lambda + m - 1)/2$ and $(\mu + m - 1)/2$. If these values are not integers, we round up to give the following upper bounds on the band widths: $\lambda_1 \leq [(\lambda + m))/2]$, $\mu_1 \leq [(\mu + m))/2]$. □

This means that, if T_i has a non-smooth band with bandwiths λ_i, μ_i, then in approximating A_{i+1}, B_{i+1}, C_{i+1} we should cut the submatrices to bands *at least* as wide as $[(\lambda + m))/2]$, $[(\mu + m))/2]$. In addition, the non-smooth band within T_{i+1} should also be assumed to have these bandwidths. For example, if T_i has a tridiagonal non-smooth band (see figure 4) and a Daubechies 4 DWT is used, then the bandwidths of $\bar{A}_{i+1}$, $\bar{B}_{i+1}$, $\bar{C}_{i+1}$ are each chosen to be $[(1 + 4)/2] = 2$ and T_{i+1} is assumed to have a non-smooth band with half band width 2.

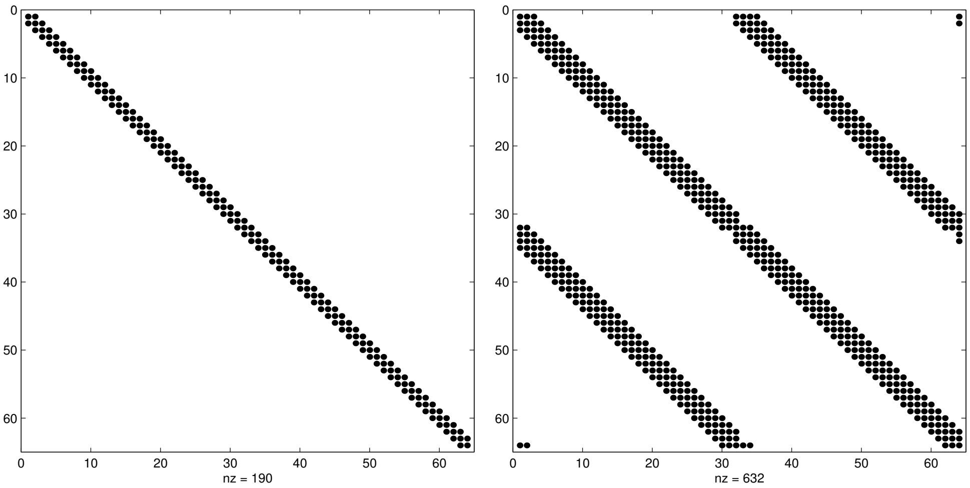

Figure 4: Effect of level 1 standard DWT on a 64×64 tridiagonal matrix.

The DWT is done in two stages: 1-D column DWT followed by 1-D row DWT. At the second stage we can reduce the cost by computing only those elements that we are going to keep in our final approximation. At the column DWT stage we do not need to compute every element, but we must take care to ensure that we *do* calculate every element that can contribute to the final band. We can deduce which these elements are by considering the effect of the DWT on diagonal bands outside the bands of $\tilde{A}$ given in theorem 1.

Definition 1 *The p^{th} wrap-round diagonal of an $n \times n$ matrix A is the set of entries a_{ij} such that $\langle j - i \rangle_n = p$.*

Definition 2 *The p^{th} wrap-round diagonal of an $n/2 \times n$ matrix A is the set of entries a_{ij} such that $\langle j - 2i \rangle_n = p$.*

In what follows we work in modulo n or modulo $n/2$ arithmetic as appropriate.

Theorem 2 *After a level 1, order m, DWT has been applied to the rows of an $n/2 \times n$ matrix, $\hat{A}^{(p)}$ that is zero apart from the p^{th} wrap-round diagonal, the resultant matrix $\tilde{A}^{(p)}$ is a block matrix consisting of two $n/2 \times n/2$ matrices, each of which are zero apart from the q^{th} wrap-round diagonals, where $p - (m-1) \leq 2q \leq p$.*

Proof
$\hat{a}_{ij}^{(p)} = 0$ whenever $j < 2i+p$ or $j > 2i+p$. From (5), $\tilde{a}_{ij}^{(p)} = 0$ whenever $2j+m-1 < 2i+p$ or $2j > 2i+p$, for $j < n/2$, with a similar results for $n/2 \leq, j < n$. $\square$

Using theorem 2 together with theorem 1 and working backwards, we can see that it is sufficient to compute, on the l^{th} row of $\hat{T}_i$, the elements lying within a band given by

$$\alpha_l = \begin{cases} 2l - \lambda_i - (m-1), & l = 0 \ldots n/2 - 1 \\ 2(l - n/2) - \lambda_i - (m-1), & l = n/2 \ldots n - 1 \end{cases}$$

$$\beta_l = \begin{cases} 2l + \mu_i + 2(m-1) & l = 0 \ldots n/2 - 1, \\ 2(l - n/2) + \mu_i + 2(m-1) & l = n/2, \ldots n - 1. \end{cases}$$

For our purposes here we need to compute *the whole* of T_{i+1}, so at the column transform stage we must compute *all* the $s_l^{(1)}$, but we can reduced the cost by computing only those $d_l^{(1)}$ that will contribute to A_{i+1}, B_{i+1}. At the row transform stage we must calculate T_{i+1} in its entirety, but we need only compute the entries in A_{i+1},B_{i+1}, C_{i+1} that we are going to retain. This process is described in Algorithm 3 below and illustrated in figure 5.

Algorithm 3 *Banded approximation to level 1 DWT of T_i*

1. Input bandwidths λ_i, μ_i of non-smooth band of T_i and dimension n_i of T_i.

2. **For** $k = 1 \ldots n_i/2$ compute column transform:
   whole of row k
   elements j of row $k + n/2$, where $k - (\lambda_i + (m-1)) \leq j \leq k + \mu_i + 2(m-1)$.

3. **For** $k = 1 \ldots n_i/2$ compute row transform:
   elements j of row k, where $j \leq n/2$ or $k - [(\lambda_i + m)/2] \leq j \leq k + [(\mu_i + m)/2]$.
   elements j of row $k + n/2$, where $k - [(\lambda_i + m)/2] \leq j \leq k + [(\mu_i + m)/2]$, or $k + n/2 - [(\lambda_i + m)/2] \leq j \leq k + n/2 + [(\mu_i + m)/2]$.

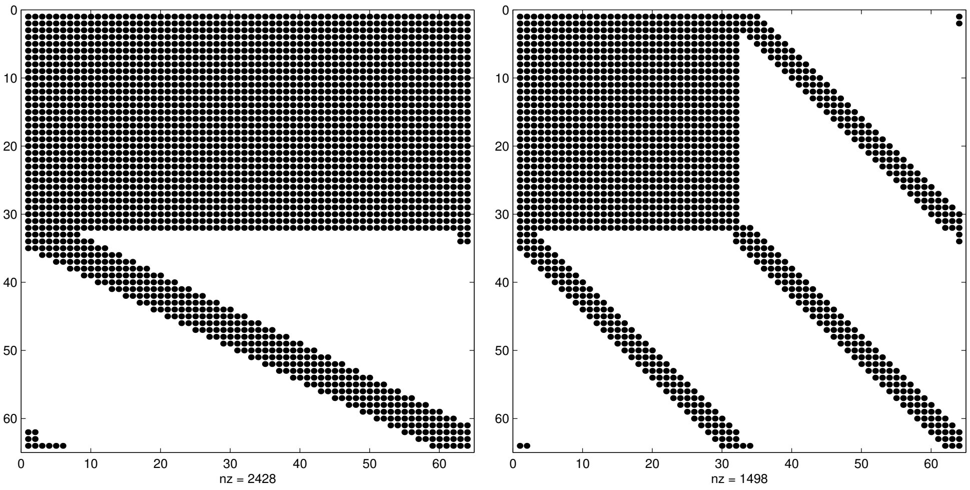

Figure 5: Entries that must be computed for 64×64 matrix with tridiagonal non-smooth band. Left: stage 1; right: stage 2.

2.4 Summary of the recursive Schur complement method

The overall method is summarised in Algorithm 4 below. Figure 6 gives a representation of a single step of the outer GMRES iteration. Each node represents an approximate solution to the linear system at the level indicated on the left-hand side. In the example illustrated, three levels of DWT are used; two GMRES iterations are required for convergence to the chosen inner tolerance at each level.

Algorithm 4 *Solution of $Ax = b$ by recursively preconditioned flexible GMRES.*

1. **Set up**

   (a) *Input matrix A, RHS b, tolerance t_{outer}.*

   (b) *Decide on a maximum wavelet level, k, tolerance t_{inner} for inner iterations and bandwidth for approximating the submatrices.*

   (c) *Set $T_0 = A$ and $i = 0$.*

   (d) *Recursively, using Algorithm 3, for $i = 1 \ldots k+1$, find T_i, $\bar{A}_i$, $\bar{B}_i$, $\bar{C}_i$, and factorize $\bar{A}_i$.*

   (e) *Factorize T_{k+1} into L_{k+1}, U_{k+1}.*

2. **Solve** *$T_0 x = b$ by flexible GMRES iteration preconditioned using Algorithm 1 recursively.*

Note In our numerical experiments we used a fixed value for t_{inner}, but it would be of interest to find an optimal value (see [16]).

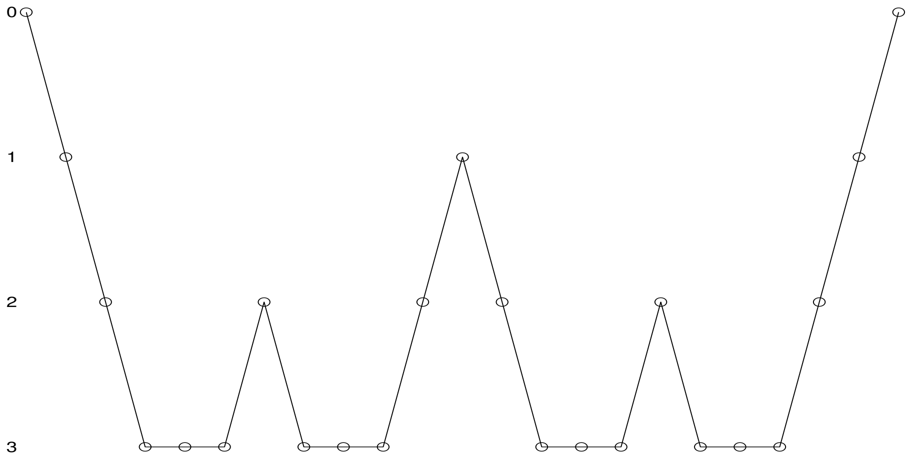

Figure 6: The recursive Schur complement W-cycle

3 Comparison of preconditioning methods

The large elements that we seek to retain in our wavelet-based preconditioner are of two types: elements corresponding to the weighted averages (the $s_j^{(i)}$ of (5)) and elements corresponding to differences between elements near singularities. We refer to these as a-type and b-type elements respectively. Ideally we would retain all of both a-type and b-type elements, but in practice this is difficult to attain without making the preconditioner expensive to apply. We now consider in detail the case of a smooth matrix with a diagonal singularity.

In the case of a level k standard DWT, the a-type elements are confined to an $n/2^k \times n/2^k$ block at the top left-hand corner of the transformed matrix (see figure 1 (b)). This means that a banded approximation can easily be made to include all of these elements provided that the half band width is at least as great as $n/2^k \times n/2^k$. However, the b-type elements are spread out in the characteristic finger pattern and can only be included at high cost. So we expect this type of preconditioner to perform well in cases where the a-type elements are of greater importance than the b-type. (See [5] for a discussion of ways of reducing the relative importance of b-type elements.)

DWTPer has the effect of confining the b-type elements to a diagonal band of known width, but in doing this the a-type elements are dispersed throughout the matrix. (In figure 3(a) they appear as regularly-spaced dots.) We expect this type of preconditioner to outperform standard DWT in cases where the singularity produces very large b-type elements, but to be less effective with smoother matrices.

In the NS-form, the b-type elements are confined to bands within the submatrices A_i, B_i, C_i and the a-type elements are contained in T_k. This enables accurate direct solution of linear systems using a multiresolution LU factorization procedure. We have chosen instead to use NS-forms as a preconditioning method based on thresholded approximation of the

A_i, B_i, C_i and retention of the whole of T_k.

Our recursive preconditioning strategy is an alternative way of exploiting the banded block structure of the NS-form. Again, both a-type and b-type elements are taken into account in the preconditioner, so we expect to be able to achieve better results than for standard DWT or DWTPer in cases where both types of element are important.

4 Numerical Results

Here we compare the performance of the four preconditioning methods previously discussed applied to matrices which are smooth apart from a singular diagonal feature. These methods are

P1 standard DWT with band cut [4, 6],

P2 DWTPer with band cut [7, 11, 13],

P3 NS-form with threshold [3, 14],

P4 Algorithm 4.

Each preconditioner was tested using GMRES restarted after 10 iterations. The purpose of this comparison is to illustrate the robustness of **P4** compared with **P1** and **P2** (our previous work). Parameter choices for these methods (for example width of band cut) were made in accordance with our previously established practice. We include **P3** for interest, because this method uses the NS-form submatrices, but in a different way from **P4**. We have not attempted to optimize the performance of **P3** beyond doing some preliminary experiments to find a threshold value that gave reasonable sparsity without the submatrices becoming singular. The inner tolerance t_{inner} for **P4** was chosen (after some preliminary experiments) to be 10^{-1} throughout and the bandwidth was chosen based on the assumption that non-smoothness is confined to the leading diagonal. All the preconditioners were based on a Daubechies 4 DWT.

We look at two examples. The first illustrates the effect of the relative significance of a-type and b-type elements on preconditioner performance. The second shows the effectiveness of our new method in a realistic situation.

Example 1

$$A_{ij} = \begin{cases} 2, & i = j, \\ 1/|i-j|^q, & i \neq j, \end{cases} \tag{31}$$

where $i, j = 1, \ldots n$, $q \in \mathbb{R}^+$.

This example illustrates how the relative effectiveness of the four preconditioners depends on the nature of the diagonal singularity. The smaller the value of q the less pronounced the singularity and hence the less important the b-type elements of the transformed matrix are compared with the a-type elements. By way of illustration, figure 7 shows the DWT for $q = 0.2$ and $q = 2$.

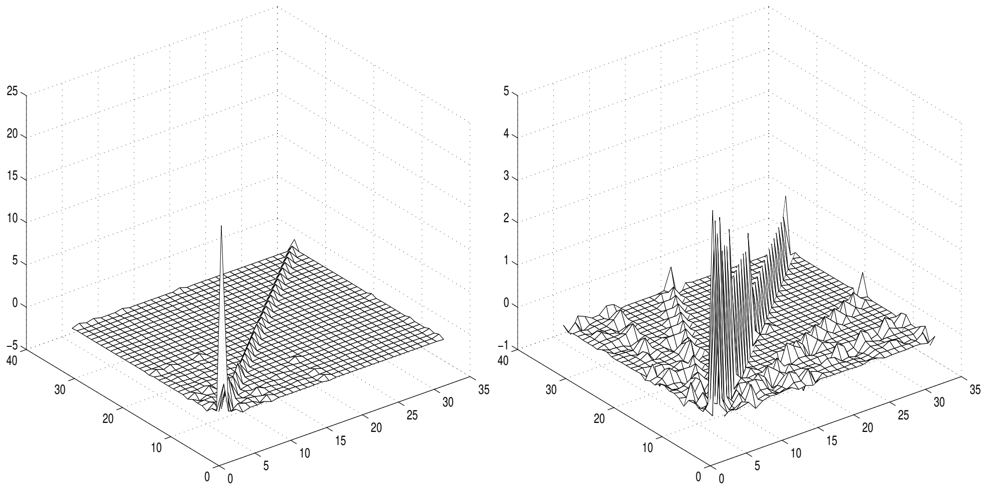

Figure 7: DWT of Example 1 matrices. Left: $q = 0.2$, right: $q = 2$.

When $q = 0.2$ the finger pattern is almost absent, whereas for $q = 2$ the magnitude of the entries in the fingers is of a similar order to that of the diagonal entries. We therefore expect P1 to perform well for small q, but P2 to provide a better preconditioner for large q. Since P4 aims to take into account both a-type and b-type elements, we anticipate that it will be competitive for all values of q. In experiments these predictions were confirmed.

Table 1 summarises the relative performance of the four methods for different values of q based on the number of flops required to achieve a tolerance of 10^{-6} by each method. The 'best' preconditioner is rated '1', the next best '2' and so on. From this we can see that, as predicted, P1 performs better than P2 for small q and that their positions become reversed as q increases. P4 gives good performance for all values of q, but is not as good as P2 for large q. P3 also performs well for small q, but its effectiveness rapidly diminishes as q increases. This may be because, for larger values of q, more entries exceed the threshold and so the preconditioner becomes more expensive to apply. It is possible that the performance of P3 could be improved if a better way of choosing the threshold could be devised or if a band cut were used instead of thresholding.

Figure 8 shows the relative residual norm plotted against the computation cost (measured as a percentage of the cost of direct solution of the equation) for some example values of q. Following the example of [18], we have chosen to plot the residual against the operation count because the number of iterations is not a good guide to the effectiveness of the preconditioners, since the cost of each iteration varies widely between the different methods. P2 is omitted from the graph for $q = 0.2$ because restarted GMRES failed to converge. From these graphs we can see that P4 performs consistently well for all values of q; P1 and P3 are satisfactory for small values of q, but become less effective as q increases; P2 is ineffective for small q, but performs increasingly well as q increases and is the best preconditioner for large q. These results suggest that our new preconditioner, P4, is less affected by the relative importance of the a-type and b-type elements than are the other

Table 1: Summary of the relative performance (based on the number of flops required for convergence to a tolerance of 10^{-6}) for Example 1 matrices with different values of q.

	P1	P2	P3	P4
$0 < q \leq 0.2$	3	4	2	1
$0.2 < q \leq 0.3$	2	4	2	1
$0.3 < q \leq 0.4$	2	4	3	1
$0.4 < q \leq 0.5$	4	2	3	1
$0.5 < q \leq 0.8$	3	1	4	1
$q > 0.8$	3	1	4	2

DWT-based preconditioners. This makes it particularly useful for applications where this information is not known.

For the case $q = 2$, where P2 outperformed P4, some further experiments were carried out to assess the usefulness of P4 in cases where the b-type elements dominate. P4 and P2 were compared using wavelet levels from 1 to 6. When a low level transform was used P2 was clearly superior. However, performance was critically dependent on choosing the right level: at level 3 P2 was little better than P4 and for higher levels P4 gave the better results. Importantly, the performance of P4 for levels 2 to 6 was almost identical, which means that satisfactory results can be obtained even when the "wrong" level is chosen.

Table 2: Iteration and flop counts for convergence to a tolerance of 10^{-6} with a 512×512 Example 1 matrix with $q = 2$.

level	P2		P4	
	iterations	flops	iterations	flops
1	6	7.7×10^6	5	1.9×10^7
2	6	8.4×10^6	5	1.4×10^7
3	7	1.1×10^7	5	1.3×10^7
4	8	1.7×10^7	5	1.3×10^7
5	8	3.2×10^7	5	1.3×10^7
6	8	6.6×10^7	5	1.3×10^7

By way of example, Table 2 shows the flop counts and number of iterations required for convergence to a tolerance of 10^{-6} with a 512×512 matrix. Notice that the number of iterations using P4 remains the same regardless of the transform level and that the number of iterations actually *increases* slightly as the level increases when P2 is used (because more of the b-type elements are in the wrap-round part of the band, which is not included in the preconditioner). This suggests that the diagonal singularity is so strong that wavelet compression may not be of great value for this matrix. This observation led us to make a comparison with banded preconditioners, to test whether the diagonal dominance is so

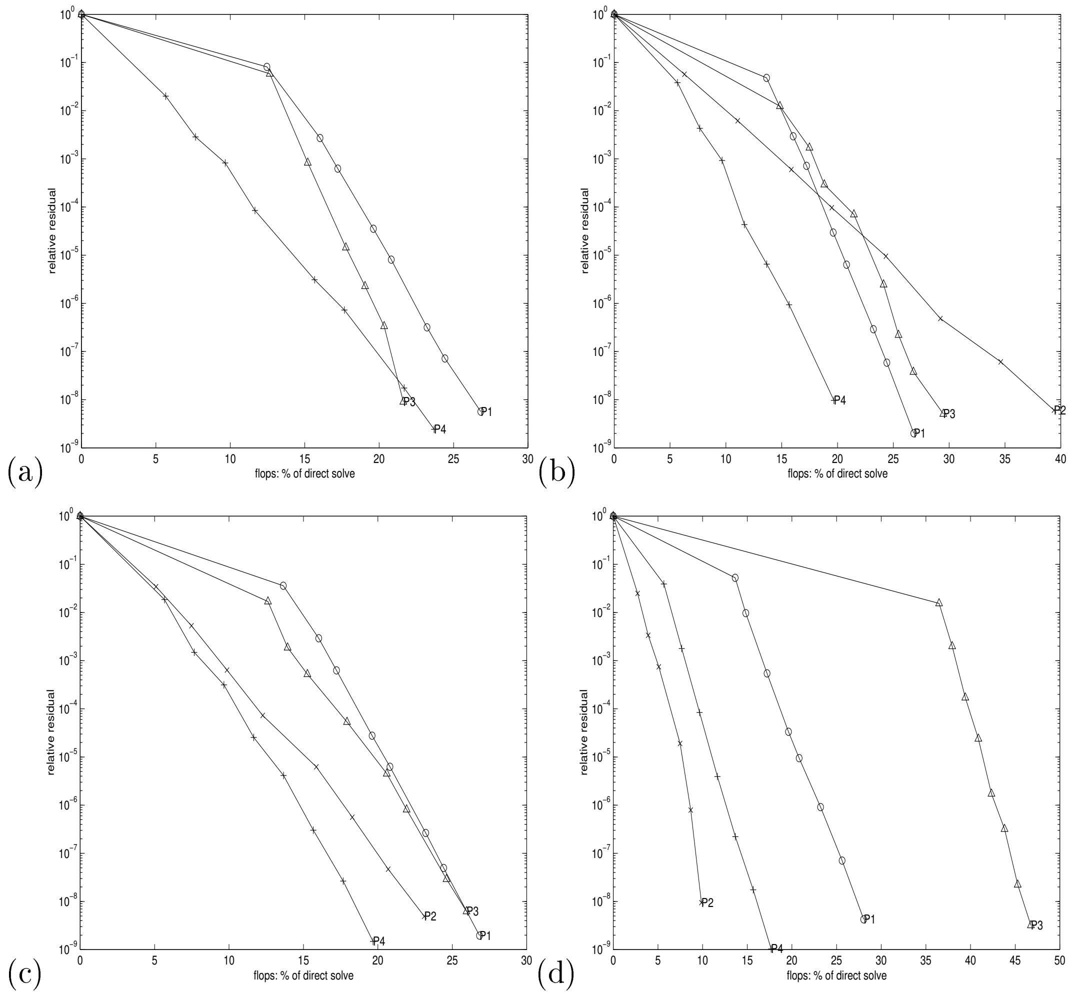

Figure 8: Comparison of preconditioners P1-P4 for 1024×1024 Example 1 matrix. (a) $q = 0.2$ (a-type elements dominate), (b) $q = 0.4$, (c) $q = 0.5$, (d) $q = 2$ (b-type elements dominate). P4 gives good results for all values of q. P1 and P3 are good for small q. P2 is best for large q, but no good for $q < 0.4$.

great that wavelet compression is unnecessary. We found that it was possible to produce a good preconditioner (better even than level 1 P2) by cutting the matrix to a band, but performance depended on the "correct" choice of bandwidth: a very narrow band gave poor convergence, while one that was too wide was expensive to apply.

The advantanges of P4 over other DWT-based preconditioners can be summarised as follows. Suppose that you have a dense matrix that is smooth apart from a diagonal singularity. If you happen to *know* that the singularity is much more important than the values of the off-diagonal elements then you could try a banded preconditioner, but it may be difficult to choose a good bandwidth. In this case P2 offers a more fool-proof alternative, because it combines banding with wavelet compression and reduces the chance of choosing an unsuitable bandwidth. On the other hand, if the singularity is known to be relatively unimportant, or if you *do not know* how important it is, then P4 is the most reliable DWT-based preconditioner, because *even if you do not choose the best level* it will perform nearly as well as the best out of P1-P4 and in most cases will actually *be* the best.

Example 2 Singular Boundary Integral Equation for Helmholtz Equation

DWTPer has been used successfully (see [7]) to generate preconditioners for the complex matrices that arise from solving the exterior Helmholtz equation

$$(\nabla^2 + \mathtt{k}^2)\phi(p) = 0 \tag{32}$$

in an elliptical domain (see [1]). We compared the performance of P1-P4 as preconditioners for such matrices and found that our new recursive technique enabled further cost savings to be made. Figure 9 illustrates this. In this example there is relatively little difference between the performance of P1 and P2, suggesting that a-type and b-type elements are of similar importance. By including both element types, P4 improves upon both methods. Again the performance of P3 is disappointing.

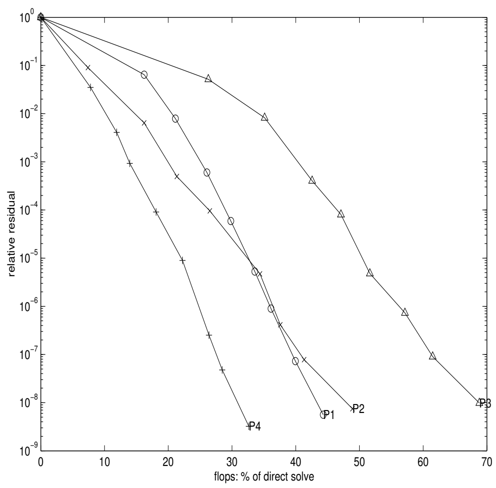

Figure 9: Comparison of preconditioners P1-P4 for 512×512 Example 2 matrix. P4 gives improved performance compared with both DWT and DWTPer based preconditioners.

5 Conclusion

An improved approximation for the finger pattern matrices that arise when DWT is applied to dense matrices with diagonal singularities can be obtained by using our recursive Schur complement idea. Numerical experiments have confirmed that the new recursive preconditioner is more robust than previously studied band preconditioners in speeding up iterative solution of dense matrix problems. Work on higher dimensional problems is in progress.

References

[1] S. Amini, P. J. Harris and D. T. Wilton, *Coupled boundary and finite element methods for the solution of the dynamic fluid-structure interaction problem* (Springer-Verlag, Berlin, 1992).

[2] Owe Axelsson, *Iterative solution methods* (Cambridge University Press, Cambridge, UK, 1996).

[3] G. Beylkin, R. R. Coifman and V. Rokhlin, 'Fast wavelet transforms and numerical algorithms I.' *Commun. Pure Appl. Math.* XLIV (1991) 141–183.

[4] T. F. Chan, W. P. Tang and W. L. Wan, 'Wavelet sparse approximate inverse preconditioners.' *BIT* 37 (1997) 644–660.

[5] Tony F. Chan and Ke Chen, 'Two-stage preconditioners using wavelet band splitting and sparse approximation.' Report CAM 00-26, UCLA, (2000).

[6] Ke Chen, 'On a class of preconditioning methods for dense linear systems from boundary elements.' *SIAM J. Sci. Comput.* 20 (1998) 684–698.

[7] Ke Chen, 'Discrete wavelet transforms accelerated sparse preconditioners for dense boundary element systems.' *Elec. Trans. Numer. Anal.* 8 (1999) 138–153.

[8] Albert Cohen, Wolfgang Dahmen and Ronald DeVore, 'Adaptive wavelet methods for elliptic operator equations - convergence rates.' *Math. Comput.* (to appear).

[9] James. W. Demmel, *Applied Numerical Linear Algebra* (SIAM, Philadelphia, 1997).

[10] Judith Ford and Ke Chen, 'Sparse preconditioners exploiting band structures in dense matrices arising from elastohydrodynamic lubrication problems.' 'Proceedings of 16th IMACS World Congress, Lausanne,' (eds Michael Deville and Robert Owens) (IMACS, New Brunswick, 2000) Paper 317-2.

[11] Judith Ford and Ke Chen, 'An algorithm for accelerated computation of DWTPer-based band preconditioners.' *Num. Alg.* 26 (2001) 167–172.

[12] Judith Ford and Ke Chen, 'Wavelet-based preconditioners for dense matrices with non-smooth local features.' *BIT* 41 (2001) 282–307.

[13] Judith Ford, Ke Chen and Laurence Scales, 'A new wavelet transform preconditioner for iterative solution of elastohydrodynamic lubrication problems.' *Int. J. Comput. Maths.* 75 (2000) 497–513.

[14] D. Gines, G. Beylkin and J. Dunn, 'LU factorization of non-standard forms and direct multiresolution solvers.' *Applied and Computational Harmonic Analysis* 5 (1998) 156–201.

[15] David Gines, 'Fast electromagnetic simulations using wavelets.' Ph.D. thesis, University of Colorado, (1997).

[16] Gene H. Golub and Michael L. Overton, 'The convergence of ineact Chebyshev and Richardson iterative methods for solving linear systems.' *Numerische Mathematik* 53 (1988) 571–593.

[17] Elyas Nurgat, Martin Berzins and Laurence Scales, 'Solving EHL problems using iterative, multigrid, and homotopy methods.' *Transactions of the ASME* 121 (1999) 28–34.

[18] Youcef Saad, 'A flexible inner-outer preconditioned GMRES algorithm.' *SIAM J. Sci. Comput.* 14 (1993) 461–469.

[19] Youcef Saad, *Iterative Methods for Sparse Linear Systems* (PWS, Boston, 1996).

[20] Gilbert Strang and Truong Nguyen, *Wavelets and Filter Banks* (Wellesley-Cambridge Press, USA, 1996).