Inversion Techniques for Several Banded Matrices

J.W. Johnson

January 1985

UWFDM-612

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

Inversion Techniques for Several Banded Matrices

J.W. Johnson

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

January 1985

UWFDM-612

Inversion Techniques for Several Banded Matrices

J. W. Johnson

Fusion Technology Institute
Nuclear Engineering Department
University of Wisconsin–Madison
Madison, Wisconsin 53706

January, 1985

UWFDM-612

Abstract

Algorithms for inverting banded matrices which are approximately tridiagonal or pentadiagonal are given. Applications include solving the linear system of equations $Mx = f$ and computing M^{-1}.

Contents

1 Introduction

In this report, matrix inversion techniques for several types of banded matrices will be presented. The matrices in question will be approximately tridiagonal or pentadiagonal; such matrices frequently arise when solving boundary value problems or when computing spline interpolation coefficients. For the matrices discussed here, algorithms will be presented which can solve a simple matrix equation in $O(n)$ operations (n is the size of the matrix), or return the inverse of the matrix in $O(n^2)$ operations. This represents a substantial savings over the $O(n^3)$ operations needed for either task when a full matrix inversion algorithm, such as Gaussian elimination, is used.

The decision to use a sparse matrix algorithm should not be made lightly. If the matrix is small or the matrix inversion represents a small fraction of the total number of operations needed to solve the problem, use of a full matrix inversion routine will not result in a substantial performance penalty. If the inversion algorithm must be used repeatedly, use of a sparse matrix inversion routine will result in a large savings of operations. An example of such a case would be solving a parabolic partial differential equation. The finite difference operator gives a banded matrix which must be inverted at every time step.

The matrices which we will be considering are asymmetric banded matrices. Two of the matrices are nearly tridiagonal; each has two "extra" bands which contain only one element. The remaining two matrices are nearly pentadiagonal with four extra bands; each band contains one or two additional elements.

For the approximately tridiagonal matrices, there are $n - 3$ possible locations for the "extra" bands. The two matrices which are discussed in this report represent the limiting cases: the first has the extra bands located adjacent to the tridiagonal structure, the second has the bands located as far away from the main diagonal as possible, in the "corners" of the matrix. The two approximately pentadiagonal matrices have the extra bands in the analogous locations.

The starting point for each matrix inversion algorithm is the LU decomposition algorithm, by which a matrix can be expressed as the product of a lower triangular matrix and an upper triangular matrix. Two applications will be presented: solving a simple matrix equation and computing the inverse of a matrix. Listings of FORTRAN implementations of the algorithms presented here can be found in the Appendices.

For many cases of interest, each element of the $n \times n$ matrix is a scalar. This need not

always be the case: each matrix element could be an $m \times m$ matrix. Such "block tridiagonal" or "block pentadiagonal" systems often occur when solving coupled differential equations with finite difference methods. To preserve generality, and because the set of $m \times m$ matrices is a non-commutative ring, the algorithms will contain multiplication by the inverses of the elements instead of division.

2 The Decomposition Algorithms

Two of the matrices which we will be considering are approximately tridiagonal. They are defined by:

$$A = \begin{pmatrix} b_1 & c_1 & a_1 & & & & \\ a_2 & b_2 & c_2 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & c_n & a_n & b_n \end{pmatrix} \tag{1}$$

$$B = \begin{pmatrix} b_1 & c_1 & & & & & a_1 \\ a_2 & b_2 & c_2 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ c_n & & & & & a_n & b_n \end{pmatrix} \tag{2}$$

Note that all elements not shown or implied by the dots are zero. These matrices are just tridiagonal matrices with two additional elements. If the two elements were zero, Thomas's algorithm[1] could be used. The "extra" elements arise from the boundary conditions a spline or finite difference problem imposes. The corner elements in the second matrix come from periodic boundary conditions, while the extra elements in the first matrix come from fixing a function value (or derivative) at the boundary.

Matrix A can be inverted in several ways – it is pentadiagonal so the pentadiagonal form of Thomas's algorithm could be used. With two row operations it can be put in a

purely tridiagonal form, and Thomas's algorithm can be used. The algorithm which will be presented to invert A is equivalent to performing the two row operations and using Thomas's algorithm. We are not so fortunate with matrix B; $O(n)$ row operations would be needed to put it into tridiagonal form.

We will also be considering two pentadiagonal matrices. These are defined by:

$$C = \begin{pmatrix} c_1 & d_1 & e_1 & a_1 & b_1 & & & & \\ b_2 & c_2 & d_2 & e_2 & a_2 & & & & \\ a_3 & b_3 & c_3 & d_3 & e_3 & & & & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \cdot & \cdot & \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} & d_{n-2} & e_{n-2} \\ & & & & e_{n-1} & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ & & & & d_n & e_n & a_n & b_n & c_n \end{pmatrix} \tag{3}$$

$$D = \begin{pmatrix} c_1 & d_1 & e_1 & & & & & a_1 & b_1 \\ b_2 & c_2 & d_2 & e_2 & & & & & a_2 \\ a_3 & b_3 & c_3 & d_3 & e_3 & & & & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \cdot & \cdot & \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} & d_{n-2} & e_{n-2} \\ e_{n-1} & & & & & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ d_n & e_n & & & & & a_n & b_n & c_n \end{pmatrix} \tag{4}$$

C and D are similar to A and B; the main difference is that there are six "extra" elements for the pentadiagonal matrices as opposed to two extra elements for the tridiagonal case. As in the case of A, we see that we can coerce C into a purely pentadiagonal form by using several row operations.

The elements of the triangular matrices L_A and U_A are specified by the condition $L_A U_A = A$. In fact, this condition is not quite sufficient to uniquely define the matrices L_A and U_A; we have the freedom to choose the values for the main diagonal of either L_A or U_A. The product $L_A U_A$ is of the form:

$$\begin{pmatrix} m_1 & & & & & \\ l_2 & m_2 & & & & \\ & \cdot & \cdot & & & \\ & & \cdot & \cdot & & \\ & & & l_{n-1} & m_{n-1} & \\ & & & u_n & l_n & m_n \end{pmatrix} \begin{pmatrix} 1 & u_1 & l_1 & & & \\ & 1 & u_2 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & 1 & u_{n-1} \\ & & & & & 1 \end{pmatrix} \tag{5}$$

where the main diagonal elements of U_A have been arbitrarily chosen to be 1 (note: following standard notation, "1" is the multiplicative identity of the ring of the matrix elements; this is not necessarily the integer 1). If we consider the elements of A row by row, we can solve for the unknown quantities in Equation 5.

row 1

$$\begin{aligned} m_1 &= b_1 \\ u_1 &= m_1^{-1} c_1 \\ l_1 &= m_1^{-1} a_1 \end{aligned}$$

row 2

$$\begin{aligned} l_2 &= a_2 \\ m_2 &= b_2 - l_2 u_1 \\ u_2 &= m_2^{-1}(c_2 - l_2 l_1) \end{aligned}$$

row j: $2 < j < n$

$$\begin{aligned} l_j &= a_j \\ m_j &= b_j - l_j u_{j-1} \\ u_j &= m_j^{-1} c_j \end{aligned}$$

row n

$$\begin{aligned} u_n &= c_n \\ l_n &= a_n - u_n u_{n-2} \\ m_n &= b_n - l_n u_{n-1} \end{aligned}$$

These relations allow the computation of each element of L_A and U_A in terms of the elements of A and previously computed elements of L_A and U_A. The total number of operations needed to obtain the complete LU decomposition is proportional to n, the size of the matrix.

For matrix B, the tridiagonal matrix with corner elements, the matrices L_B and U_B are of the form:

$$\begin{pmatrix} m_1 & & & & & & \\ a_2 & m_2 & & & & & \\ & \ddots & \ddots & & & & \\ & & \ddots & \ddots & & & \\ & & & a_{n-1} & m_{n-1} & & \\ q_1 & q_2 & \cdots & \cdots & q_{n-1} & m_n \end{pmatrix} \begin{pmatrix} 1 & u_1 & & & & r_1 \\ & 1 & u_2 & & & r_2 \\ & & \ddots & \ddots & & \vdots \\ & & & \ddots & \ddots & \vdots \\ & & & & u_{n-2} & r_{n-2} \\ & & & & 1 & r_{n-1} \\ & & & & & 1 \end{pmatrix} \tag{6}$$

where the a_j which appear in L_B are the same as those given in B. Solving for the unknown elements of L_B and U_B we have:

row 1

$$\begin{aligned} m_1 &= b_1 \\ u_1 &= m_1^{-1} c_1 \\ q_1 &= c_n \\ r_1 &= m_1^{-1} a_1 \end{aligned}$$

row j: $1 < j < n-1$

$$\begin{aligned} m_j &= b_j - a_j u_{j-1} \\ u_j &= m_j^{-1} c_j \\ q_j &= -q_{j-1} u_{j-1} \\ r_j &= -m_j^{-1} a_j r_{j-1} \end{aligned}$$

rows $n-1$ *and* n

$$\begin{aligned} m_{n-1} &= b_{n-1} - a_{n-1} u_{n-2} \\ q_{n-1} &= a_n - q_{n-2} u_{n-2} \\ r_{n-1} &= m_{n-1}^{-1} (c_{n-1} - a_{n-1} r_{n-2}) \\ m_n &= b_n - \sum_{i=1}^{n-1} q_i r_i \end{aligned}$$

The LU decomposition algorithm for matrix C is similar to the algorithm for matrix A. The L_C and U_C matrices can be written as:

$$L_C = \begin{pmatrix} m_1 & & & & & & & \\ l_2 & m_2 & & & & & & \\ t_3 & l_3 & m_3 & & & & & \\ & \cdot & \cdot & \cdot & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & t_{n-2} & l_{n-2} & m_{n-2} & & \\ & & & w_{n-1} & t_{n-1} & l_{n-1} & m_{n-1} & \\ & & & u_n & w_n & t_n & l_n & m_n \end{pmatrix} \tag{7}$$

$$U_C = \begin{pmatrix} 1 & u_1 & w_1 & t_1 & l_1 & & & \\ & 1 & u_2 & w_2 & t_2 & & & \\ & & 1 & u_3 & w_3 & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & \cdot & \cdot & \\ & & & & & 1 & u_{n-2} & w_{n-2} \\ & & & & & & 1 & u_{n-1} \\ & & & & & & & 1 \end{pmatrix} \tag{8}$$

Solving for the unknown quantities, we have:

row 1

$$\begin{aligned} m_1 &= c_1 \\ u_1 &= m_1^{-1} d_1 \\ w_1 &= m_1^{-1} e_1 \\ t_1 &= m_1^{-1} a_1 \\ l_1 &= m_1^{-1} b_1 \end{aligned}$$

row 2

$$\begin{aligned} l_2 &= b_2 \\ m_2 &= c_2 - l_2 u_1 \\ u_2 &= m_2^{-1}(d_2 - l_2 w_1) \\ w_2 &= m_2^{-1}(e_2 - l_2 t_1) \\ t_2 &= m_2^{-1}(a_2 - l_2 l_1) \end{aligned}$$

row 3

$$\begin{aligned} t_3 &= a_3 \\ l_3 &= b_3 - t_3 u_1 \\ m_3 &= c_3 - t_3 w_1 - l_3 u_2 \\ u_3 &= m_3^{-1}(d_3 - t_3 t_1 - l_3 w_2) \\ w_3 &= m_3^{-1}(e_3 - t_3 l_1 - l_3 t_2) \end{aligned}$$

row 4

$$\begin{aligned} t_4 &= a_4 \\ l_4 &= b_4 - t_4 u_2 \\ m_4 &= c_4 - t_4 w_2 - l_4 u_3 \\ u_4 &= m_4^{-1}(d_4 - t_4 t_2 - l_4 w_3) \\ w_4 &= m_4^{-1} e_4 \end{aligned}$$

row j: $4 < j < n-1$

$$\begin{aligned} t_j &= a_j \\ l_j &= b_j - t_j u_{j-2} \\ m_j &= c_j - t_j w_{j-2} - l_j u_{j-1} \\ u_j &= m_j^{-1}(d_j - l_j w_{j-1}) \\ w_j &= m_j^{-1} e_j \end{aligned}$$

row $n-1$

$$\begin{aligned} w_{n-1} &= e_{n-1} \\ t_{n-1} &= a_{n-1} - w_{n-1} u_{n-4} \\ l_{n-1} &= b_{n-1} - w_{n-1} w_{n-4} - t_{n-1} u_{n-3} \\ m_{n-1} &= c_{n-1} - t_{n-1} w_{n-3} - l_{n-1} u_{n-2} \\ u_{n-1} &= m_{n-1}^{-1}(d_{n-1} - l_{n-1} w_{n-2}) \end{aligned}$$

row n

$$
\begin{aligned}
u_n &= d_n \\
w_n &= e_n - u_n u_{n-4} \\
t_n &= a_n - u_n w_{n-4} - w_n u_{n-3} \\
l_n &= b_n - w_n w_{n-3} - t_n u_{n-2} \\
m_n &= c_n - t_n w_{n-2} - l_n u_{n-1}
\end{aligned}
$$

The LU decomposition of matrix D can be expressed as:

$$
L_D = \begin{pmatrix}
m_1 & & & & & & & & \\
l_2 & m_2 & & & & & & & \\
t_3 & l_3 & m_3 & & & & & & \\
 & \cdot & \cdot & \cdot & & & & & \\
 & & \cdot & \cdot & \cdot & & & & \\
 & & & \cdot & \cdot & \cdot & & & \\
 & & & & t_{n-2} & l_{n-2} & m_{n-2} & & \\
p_1 & p_2 & \cdot & \cdot & \cdot & p_{n-3} & p_{n-2} & m_{n-1} & \\
q_1 & q_2 & \cdot & \cdot & \cdot & q_{n-3} & q_{n-2} & q_{n-1} & m_n
\end{pmatrix} \tag{9}
$$

$$
U_D = \begin{pmatrix}
1 & u_1 & w_1 & & & & & s_1 & r_1 \\
 & 1 & u_2 & w_2 & & & & s_2 & r_2 \\
 & & \cdot & \cdot & \cdot & & & \cdot & \cdot \\
 & & & \cdot & \cdot & \cdot & & \cdot & \cdot \\
 & & & & \cdot & \cdot & w_{n-4} & \cdot & \cdot \\
 & & & & & 1 & u_{n-3} & s_{n-3} & r_{n-3} \\
 & & & & & & 1 & s_{n-2} & r_{n-2} \\
 & & & & & & & 1 & r_{n-1} \\
 & & & & & & & & 1
\end{pmatrix} \tag{10}
$$

As before, we can solve for the unknown elements of L_D and U_D.

row 1

$$\begin{aligned} m_1 &= c_1 \\ u_1 &= m_1^{-1} d_1 \\ w_1 &= m_1^{-1} e_1 \\ s_1 &= m_1^{-1} a_1 \\ r_1 &= m_1^{-1} b_1 \\ p_1 &= e_{n-1} \\ q_1 &= d_n \end{aligned}$$

row 2

$$\begin{aligned} l_2 &= b_2 \\ m_2 &= c_2 - l_2 u_1 \\ u_2 &= m_2^{-1}(d_2 - l_2 w_1) \\ w_2 &= m_2^{-1} e_2 \\ s_2 &= -m_2^{-1} l_2 s_1 \\ r_2 &= m_2^{-1}(a_2 - l_2 r_1) \\ p_2 &= -p_1 u_1 \\ q_2 &= e_n - q_1 u_1 \end{aligned}$$

row j: $2 < j < n-3$

$$\begin{aligned} l_j &= b_j - a_j u_{j-2} \\ m_j &= c_j - a_j w_{j-2} - l_j u_{j-1} \\ u_j &= m_j^{-1}(d_j - l_j w_{j-1}) \\ w_j &= m_j^{-1} e_j \\ s_j &= -m_j^{-1}(a_j s_{j-2} + l_j s_{j-1}) \\ r_j &= -m_j^{-1}(a_j r_{j-2} + l_j r_{j-1}) \\ p_j &= -(p_{j-2} w_{j-2} + p_{j-1} u_{j-1}) \\ q_j &= -(q_{j-2} w_{j-2} + q_{j-1} u_{j-1}) \end{aligned}$$

row $n-3$

$$\begin{aligned}
l_{n-3} &= b_{n-3} - a_{n-3}u_{n-5} \\
m_{n-3} &= c_{n-3} - l_{n-3}u_{n-4} - a_{n-3}w_{n-5} \\
u_{n-3} &= m_{n-3}^{-1}(d_{n-3} - l_{n-3}w_{n-4}) \\
s_{n-3} &= m_{n-3}^{-1}(e_{n-3} - l_{n-3}s_{n-4} - a_{n-3}s_{n-5}) \\
r_{n-3} &= -m_{n-3}^{-1}(l_{n-3}r_{n-4} + a_{n-3}r_{n-5}) \\
p_{n-3} &= a_{n-1} - p_{n-5}w_{n-5} - p_{n-4}u_{n-4} \\
q_{n-3} &= -(q_{n-5}w_{n-5} + q_{n-4}u_{n-4})
\end{aligned}$$

row $n-2$

$$\begin{aligned}
l_{n-2} &= b_{n-2} - a_{n-2}u_{n-4} \\
m_{n-2} &= c_{n-2} - a_{n-2}w_{n-4} - l_{n-2}u_{n-3} \\
s_{n-2} &= m_{n-2}^{-1}(d_{n-2} - l_{n-2}s_{n-3} - a_{n-2}s_{n-4}) \\
r_{n-2} &= m_{n-2}^{-1}(e_{n-2} - l_{n-2}r_{n-3} - a_{n-2}r_{n-4}) \\
p_{n-2} &= b_{n-1} - p_{n-4}w_{n-4} - p_{n-3}u_{n-3} \\
q_{n-2} &= a_{n} - q_{n-4}w_{n-4} - q_{n-3}u_{n-3}
\end{aligned}$$

row $n-1$

$$\begin{aligned}
m_{n-1} &= c_{n-1} - \sum_{i=1}^{n-2} p_i s_i \\
r_{n-1} &= m_{n-1}^{-1}\left(d_{n-1} - \sum_{i=1}^{n-2} p_i r_i\right) \\
q_{n-1} &= b_{n} - \sum_{i=1}^{n-2} q_i s_i
\end{aligned}$$

row n

$$m_n = c_n - \sum_{i=1}^{n-1} q_i r_i$$

3 Solution of a Matrix Equation

One of the most common applications of matrix inversion techniques is the solution of a system of the equations which can be expressed in the form:

$$Mx = f \tag{11}$$

where M is an $n \times n$ matrix, x is the solution vector, and d is a vector of known values. If M is of the form of matrix A, B, C or matrix D, this equation can be simply solved using the LU decomposition scheme discussed in the previous section. Substituting the product LU for M, we have:

$$LUx = f \tag{12}$$

If we let y represent the product Ux, we have two very simple matrix equations to solve:

$$Ly = f \tag{13}$$

$$Ux = y \tag{14}$$

Given the matrices L and U, all that remains is to work out the relations for the vectors x and y. These are easy to establish, because in each case we are working with a triangular matrix. First, the expressions for matrix A will be derived.

for vector y:

$$\begin{aligned} y_1 &= m_1^{-1} f_1 \\ y_j &= m_j^{-1}(f_j - l_j y_{j-1}) \\ y_n &= m_n^{-1}(f_n - l_n y_{n-1} - u_n y_{n-2}) \end{aligned}$$

for vector x:

$$\begin{aligned} x_n &= y_n \\ x_j &= y_j - u_j x_{j+1} \\ x_1 &= y_1 - u_1 x_2 - l_1 x_3 \end{aligned}$$

Now, the expressions for the matrix B:

for vector y:

$$\begin{aligned} y_1 &= m_1^{-1} f_1 \\ y_j &= m_j^{-1}(f_j - a_j y_{j-1}) \\ y_n &= m_n^{-1}\left(f_n - \sum_{i=1}^{n-1} q_i y_i\right) \end{aligned}$$

for vector x:

$$\begin{aligned} x_n &= y_n \\ x_{n-1} &= y_{n-1} - r_{n-1} x_n \\ x_j &= y_j - u_j x_{j+1} - r_j x_n \end{aligned}$$

The algorithms for matrices C and D are similar to those for A and B. For matrix C:

for vector y:

$$\begin{aligned} y_1 &= m_1^{-1} f_1 \\ y_2 &= m_2^{-1}(f_1 - l_2 y_1) \\ y_j &= m_j^{-1}(f_j - t_j y_{j-2} - l_j y_{j-1}) \\ y_{n-1} &= m_{n-1}^{-1}(f_{n-1} - w_{n-1} y_{n-4} - t_{n-1} y_{n-3} - l_{n-1} y_{n-2}) \\ y_n &= m_n^{-1}(f_n - u_n y_{n-4} - w_n y_{n-3} - t_n y_{n-2} - l_n y_{n-1}) \end{aligned}$$

for vector x:

$$\begin{aligned} x_n &= y_n \\ x_{n-1} &= y_{n-1} - u_{n-1} x_n \\ x_j &= y_j - u_j x_{j+1} - w_j x_{j+2} \\ x_2 &= y_2 - u_2 x_3 - w_2 x_4 - t_2 x_5 \\ x_1 &= y_1 - u_1 x_2 - w_1 x_3 - t_1 x_4 - l_1 x_5 \end{aligned}$$

Finally, the expressions for the matrix D:

for vector y:

$$\begin{aligned} y_1 &= m_1^{-1} f_1 \\ y_2 &= m_2^{-1}(f_2 - l_2 y_1) \\ y_j &= m_j^{-1}(f_j - a_j y_{j-2} - l_j y_{j-1}) \\ y_{n-1} &= m_{n-1}^{-1}\left(f_{n-1} - \sum_{i=1}^{n-2} p_i y_i\right) \\ y_n &= m_n^{-1}\left(f_n - \sum_{i=1}^{n-1} q_i y_i\right) \end{aligned}$$

for vector x:

$$\begin{aligned} x_n &= y_n \\ x_{n-1} &= y_{n-1} - r_{n-1} x_n \\ x_{n-2} &= y_{n-2} - s_{n-2} x_{n-1} - r_{n-2} x_n \\ x_{n-3} &= y_{n-3} - u_{n-3} x_{n-2} - s_{n-3} x_{n-1} - r_{n-3} x_n \\ x_j &= y_j - u_j x_{j+1} - w_j x_{j+2} - s_j x_{n-1} - r_j x_n \end{aligned}$$

When implementing these equations, it is important to note that the vector y can be computed at the same time as the elements of L and U. Solving for x, which is a special case of the back–substitution algorithm common in Gaussian elimination routines, must be done in a separate loop after y has been calculated. The earlier decision to let the main diagonal of the U matrix be filled with unity is justified because it simplifies solving for x; also note that m_j and m_j^{-1} do *not* have to be saved after y_j is calculated.

This solution algorithm can be done in $O(nm^3)$ operations (m is the size of the elements, $m = 1$ for scalars) compared to $O(n^3m^3)$ operations if we were to use a standard full matrix inversion routine.

4 Inverting a Matrix

For some applications the inverse of the matrix is needed. An example of this occurs when calculating the B spline coefficients on a multidimensional interpolation grid. One way to compute the inverse is to rely on the LU decomposition of the matrix M:

$$M^{-1} = (LU)^{-1} = U^{-1}L^{-1} \tag{15}$$

It is a simple matter to invert triangular matrices. For a full triangular matrix $O(n^3)$ operations are needed; however, the simple form of both the L and U matrices (see Section 2) reduces the number of operations to $O(n^2)$. The problem lies in the multiplication of U^{-1} and L^{-1}; this requires $O(n^3)$ operations.

A more efficient way to invert M can be found by considering the method of solving a matrix equation given in the previous section. Our problem is to find M^{-1} such that:

$$MM^{-1} = 1 \tag{16}$$

where "1" is multiplicative identity, which if we restrict the elements of M to be scalars, is the identity matrix. If we compare this to Equation 11, we see that if we allow f to be a column of the multiplicative identity, we can solve for the corresponding column of M^{-1}. To solve for one column takes $O(n)$ operations; to find all n columns will take $O(n^2)$ operations.

The naive thing to do would be to use the methods of Section 3 on each column of the identity; while this would certainly work, it is not very efficient because we would have to

do the LU decomposition for each column. Instead, we do the LU decomposition once and then compute the vectors y and x for each column. An implementation of this algorithm for matrix B is given in Appendix B.

Acknowledgment

This work has been supported in part by the U. S. Department of Energy.

A Subroutines for Solving Matrix Equations

In this Appendix, four subroutines which solve matrix equations will be presented. Subroutine TRIDX solves equations of the form $Ax = f$ where matrix A has been defined in Section 2. Similarly, TRIDC solves $Bx = f$, PENTX solves $Cx = f$ and PENTC solves $Dx = f$. For the sake of simplicity, it is assumed that the matrix elements are scalars, thus each subroutine contains division by m_j instead of multiplication by m_j^{-1}.

A.1 Subroutine TRIDX

```
      subroutine tridx(a,b,c,f,x,n)

c     this subroutine solves the matrix equation Ax=f where
c     A is a tridiagonal matrix with 2 extra blocks. an example
c     of such a matrix arises in cubic B-spline interpolation
c     when derivative boundary conditions are used.

c     a is the subdiagonal, c is the superdiagonal, b is the
c     main diagonal. the two extra elements are in the A(1,3)
c     and A(n,n-2) positions.

      parameter (msize = 64)

      real a(msize), b(msize), c(msize), f(msize), x(msize)
      real y(msize), m(msize), u(msize), l(msize)

      m(1)=b(1)
      u(1)=c(1)/m(1)
      l(1)=a(1)/m(1)
      y(1)=f(1)/m(1)

      l(2)=a(2)
      m(2)=b(2)-l(2)*u(1)
      u(2)=(c(2)-l(2)*l(1))/m(2)
      y(2)=(f(2)-l(2)*y(1))/m(2)

      do 100 j=3,n-1
         l(j)=a(j)
         m(j)=b(j)-l(j)*u(j-1)
         u(j)=c(j)/m(j)
         y(j)=(f(j)-l(j)*y(j-1))/m(j)
 100     continue
```

```
      u(n)=c(n)
      l(n)=a(n)-u(n)*u(n-2)
      m(n)=b(n)-l(n)*u(n-1)

      x(n)=(f(n)-l(n)*y(n-1)-u(n)*y(n-2))/m(n)

      do 200 j=n-1,2,-1
  200    x(j)=y(j)-u(j)*x(j+1)
      x(1)=y(1)-u(1)*x(2)-l(1)*x(3)

      return
      end
```

A.2 Subroutine TRIDC

```
      subroutine tridc(a,b,c,f,x,n)

c     this subroutine solves the matrix equation Bx=d where
c     B is a tridiagonal matrix with 2 corner blocks. an example
c     of such a matrix arises in cubic B-spline interpolation
c     when periodic boundary conditions are used.

c     a is the subdiagonal, c is the superdiagonal, b is the
c     main diagonal. the two extra elements are in the B(1,n)
c     and B(n,1) positions.

      parameter (msize = 64)

      real a(msize), b(msize), c(msize), f(msize), x(msize)
      real y(msize), m(msize), u(msize), q(msize), r(msize)

      m(1)=b(1)
      u(1)=c(1)/m(1)
      q(1)=c(n)
      r(1)=a(1)/m(1)
      y(1)=f(1)/m(1)

      qr=q(1)*r(1)
      qy=q(1)*y(1)

      do 100 j=2,n-2
         m(j)=b(j)-a(j)*u(j-1)
```

```
        u(j)=c(j)/m(j)
        q(j)=-q(j-1)*u(j-1)
        r(j)=-a(j)*r(j-1)/m(j)
        y(j)=(f(j)-a(j)*y(j-1))/m(j)
        qr=qr+q(j)*r(j)
        qy=qy+q(j)*y(j)
100     continue

      nm1=n-1
      m(nm1)=b(nm1)-a(nm1)*u(n-2)
      q(nm1)=a(n)-q(n-2)*u(n-2)
      r(nm1)=(c(nm1)-a(nm1)*r(n-2))/m(nm1)
      y(nm1)=(f(nm1)-a(nm1)*y(n-2))/m(nm1)
      qr=qr+q(nm1)*r(nm1)
      qy=qy+q(nm1)*y(nm1)

      m(n)=b(n)-qr

      x(n)=(f(n)-qy)/m(n)
      x(nm1)=y(nm1)-r(nm1)*x(n)

      do 200 j=n-2,1,-1
200     x(j)=y(j)-u(j)*x(j+1)-r(j)*x(n)

      return
      end
```

A.3 Subroutine PENTX

```
      subroutine pentx(a,b,c,d,e,f,x,n)

c     this subroutine solves the matrix equation Cx=f where
c     C is a pentadiagonal matrix with 6 extra blocks. an example
c     of such a matrix arises in pentic B-spline interpolation
c     when derivative boundary conditions are used.

c     c is the main diagonal, a and b are the subdiagonals,
c     d and e are the superdiagonals. the 6 extra elements are
c     in the C(1,4), C(1,5), C(2,5), C(n-1,n-4), C(n,n-4) and
c     C(n,n-3) positions.

      parameter (msize = 64)
```

```
      real a(msize), b(msize), c(msize), d(msize), e(msize), x(msize)
      real y(msize), m(msize), u(msize), l(msize), t(msize), w(msize)
      real f(msize)

      m(1)=c(1)
      u(1)=d(1)/m(1)
      w(1)=e(1)/m(1)
      t(1)=a(1)/m(1)
      l(1)=b(1)/m(1)
      y(1)=f(1)/m(1)

      l(2)=b(2)
      m(2)=c(2)-l(2)*u(1)
      u(2)=(d(2)-l(2)*w(1))/m(2)
      w(2)=(e(2)-l(2)*t(1))/m(2)
      t(2)=(a(2)-l(2)*l(1))/m(2)
      y(2)=(f(2)-l(2)*y(1))/m(2)

      t(3)=a(3)
      l(3)=b(3)-t(3)*u(1)
      m(3)=c(3)-t(3)*w(1)-l(3)*u(2)
      u(3)=(d(3)-t(3)*t(1)-l(3)*w(2))/m(3)
      w(3)=(e(3)-t(3)*l(1)-l(3)*t(2))/m(3)
      y(3)=(f(3)-t(3)*y(1)-l(3)*y(2))/m(3)

      t(4)=a(4)
      l(4)=b(4)-t(4)*u(2)
      m(4)=c(4)-t(4)*w(2)-l(4)*u(3)
      u(4)=(d(4)-t(4)*t(2)-l(4)*w(3))/m(4)
      w(4)=e(4)/m(4)
      y(4)=(f(4)-t(4)*y(2)-l(4)*y(3))/m(4)

      do 100 j=5,n-2
         t(j)=a(j)
         l(j)=b(j)-t(j)*u(j-2)
         m(j)=c(j)-t(j)*w(j-2)-l(j)*u(j-1)
         u(j)=(d(j)-l(j)*w(j-1))/m(j)
         w(j)=e(j)/m(j)
         y(j)=(f(j)-t(j)*y(j-2)-l(j)*y(j-1))/m(j)
100      continue

      nm1=n-1
      w(nm1)=e(nm1)
      t(nm1)=a(nm1)-w(nm1)*u(n-4)
```

```
      l(nm1)=b(nm1)-w(nm1)*w(n-4)-t(nm1)*u(n-3)
      m(nm1)=c(nm1)-t(nm1)*w(n-3)-l(nm1)*u(n-2)
      u(nm1)=(d(nm1)-l(nm1)*w(n-2))/m(nm1)
      y(nm1)=(f(nm1)-w(nm1)*y(n-4)-t(nm1)*y(n-3)-l(nm1)*y(n-2))/m(nm1)

      u(n)=d(n)
      w(n)=e(n)-u(n)*u(n-4)
      t(n)=a(n)-u(n)*w(n-4)-w(n)*u(n-3)
      l(n)=b(n)-w(n)*w(n-3)-t(n)*u(n-2)
      m(n)=c(n)-t(n)*w(n-2)-l(n)*u(nm1)

      x(n)=(f(n)-u(n)*y(n-4)-w(n)*y(n-3)-t(n)*y(n-2)-l(n)*y(nm1))/m(n)
      x(nm1)=y(nm1)-u(nm1)*x(n)

      do 200 j=n-2,3,-1
  200    x(j)=y(j)-u(j)*x(j+1)-w(j)*x(j+2)

      x(2)=y(2)-u(2)*x(3)-w(2)*x(4)-t(2)*x(5)
      x(1)=y(1)-u(1)*x(2)-w(1)*x(3)-t(1)*x(4)-l(1)*x(5)

      return
      end
```

A.4 Subroutine PENTC

```
      subroutine pentc(a,b,c,d,e,f,x,n)

c     this subroutine solves the matrix equation Dx=f where
c     D is a pentadiagonal matrix with 6 extra blocks. an example
c     of such a matrix arises in pentic B-spline interpolation
c     when periodic boundary conditions are used.

c     c is the main diagonal, a and b are the subdiagonals,
c     d and e are the superdiagonals. the 6 extra elements are
c     in the D(1,n-1), D(1,n), D(2,n), D(n-1,1), D(n,1) and
c     D(n,2) positions.

      parameter (msize = 64)

      real a(msize), b(msize), c(msize), d(msize), e(msize), x(msize)
      real y(msize), m(msize), u(msize), l(msize), w(msize)
      real p(msize), q(msize), s(msize), r(msize), f(msize)
```

```
m(1)=c(1)
u(1)=d(1)/m(1)
w(1)=e(1)/m(1)
s(1)=a(1)/m(1)
r(1)=b(1)/m(1)
p(1)=e(n-1)
q(1)=d(n)
y(1)=f(1)/m(1)

ps=p(1)*s(1)
pr=p(1)*r(1)
py=p(1)*y(1)
qs=q(1)*s(1)
qr=q(1)*r(1)
qy=q(1)*y(1)

l(2)=b(2)
m(2)=c(2)-l(2)*u(1)
u(2)=(d(2)-l(2)*w(1))/m(2)
w(2)=e(2)/m(2)
s(2)=-l(2)*s(1)/m(2)
r(2)=(a(2)-l(2)*r(1))/m(2)
p(2)=-p(1)*u(1)
q(2)=e(n)-q(1)*u(1)
y(2)=(f(2)-l(2)*y(1))/m(2)

ps=ps+p(2)*s(2)
pr=pr+p(2)*r(2)
py=py+p(2)*y(2)
qs=qs+q(2)*s(2)
qr=qr+q(2)*r(2)
qy=qy+q(2)*y(2)

do 100 j=3,n-4
   l(j)=b(j)-a(j)*u(j-2)
   m(j)=c(j)-a(j)*w(j-2)-l(j)*u(j-1)
   u(j)=(d(j)-l(j)*w(j-1))/m(j)
   w(j)=e(j)/m(j)
   s(j)=-(a(j)*s(j-2)+l(j)*s(j-1))/m(j)
   r(j)=-(a(j)*r(j-2)+l(j)*r(j-1))/m(j)
   p(j)=-(p(j-2)*w(j-2)+p(j-1)*u(j-1))
   q(j)=-(q(j-2)*w(j-2)+q(j-1)*u(j-1))
   y(j)=(f(j)-a(j)*y(j-2)-l(j)*y(j-1))/m(j)
```

```
      ps=ps+p(j)*s(j)
      pr=pr+p(j)*r(j)
      py=py+p(j)*y(j)
      qs=qs+q(j)*s(j)
      qr=qr+q(j)*r(j)
      qy=qy+q(j)*y(j)

100   continue

      nm3=n-3
      l(nm3)=b(nm3)-a(nm3)*u(n-5)
      m(nm3)=c(nm3)-l(nm3)*u(n-4)-a(nm3)*w(n-5)
      u(nm3)=(d(nm3)-l(nm3)*w(n-4))/m(nm3)
      s(nm3)=(e(nm3)-l(nm3)*s(n-4)-a(nm3)*s(n-5))/m(nm3)
      r(nm3)=-(l(nm3)*r(n-4)+a(nm3)*r(n-5))/m(nm3)
      p(nm3)=a(n-1)-p(n-5)*w(n-5)-p(n-4)*u(n-4)
      q(nm3)=-(q(n-5)*w(n-5)+q(n-4)*u(n-4))
      y(nm3)=(f(nm3)-a(nm3)*y(n-5)-l(nm3)*y(n-4))/m(nm3)

      ps=ps+p(nm3)*s(nm3)
      pr=pr+p(nm3)*r(nm3)
      py=py+p(nm3)*y(nm3)
      qs=qs+q(nm3)*s(nm3)
      qr=qr+q(nm3)*r(nm3)
      qy=qy+q(nm3)*y(nm3)

      nm2=n-2
      l(nm2)=b(nm2)-a(nm2)*u(n-4)
      m(nm2)=c(nm2)-a(nm2)*w(n-4)-l(nm2)*u(nm3)
      s(nm2)=(d(nm2)-l(nm2)*s(nm3)-a(nm2)*s(n-4))/m(nm2)
      r(nm2)=(e(nm2)-l(nm2)*r(nm3)-a(nm2)*r(n-4))/m(nm2)
      p(nm2)=b(n-1)-p(n-4)*w(n-4)-p(nm3)*u(nm3)
      q(nm2)=a(n)-q(n-4)*w(n-4)-q(nm3)*u(nm3)
      y(nm2)=(f(nm2)-a(nm2)*y(n-4)-l(nm2)*y(nm3))/m(nm2)

      ps=ps+p(nm2)*s(nm2)
      pr=pr+p(nm2)*r(nm2)
      py=py+p(nm2)*y(nm2)
      qs=qs+q(nm2)*s(nm2)
      qr=qr+q(nm2)*r(nm2)
      qy=qy+q(nm2)*y(nm2)

      nm1=n-1
      m(nm1)=c(nm1)-ps
```

```
      r(nm1)=(d(nm1)-pr)/m(nm1)
      q(nm1)=b(n)-qs
      y(nm1)=(f(nm1)-py)/m(nm1)

      qr=qr+q(nm1)*r(nm1)
      qy=qy+q(nm1)*y(nm1)

      m(n)=c(n)-qr

      x(n)=(f(n)-qy)/m(n)
      x(nm1)=y(nm1)-r(nm1)*x(n)
      x(nm2)=y(nm2)-s(nm2)*x(nm1)-r(nm2)*x(n)
      x(nm3)=y(nm3)-u(nm3)*x(nm2)-s(nm3)*x(nm1)-r(nm3)*x(n)

      do 200 j=n-4,1,-1
  200    x(j)=y(j)-u(j)*x(j+1)-w(j)*x(j+2)-s(j)*x(nm1)-r(j)*x(n)

      return
      end
```

B A Matrix Inversion Subroutine

In this Appendix a subroutine which uses the method of Section 4 to invert a matrix will be given. We will consider only matrix B here; the emphasis will be on converting the routine given in the previous Appendix to solve the problem of inverting the matrix. Thus the inversion subroutines for matrices A, C, and D are left as an excercise for the reader.

The subroutine `trinvc` given below is a modified version of the subroutine `tridc` from Appendix A.2. Because we will not need the "right hand side" of a matrix equation, the array `f` has been eliminated. We will need a representation of the identity matrix; this could be either a two dimensional array or an implementation of the Kronecker delta function. In `trinvc` I have chosen the latter course; it is assumed that there exists a function `delta` which has a value of `1.0` when its two integer arguments are equal and is `0.0` otherwise.

The conversion is straightforward; references to the array `y` and variable `qy` are removed from the forward loop. The forward loop now contains only the LU decomposition algorithm. After the LU decomposition is complete, we can solve for `xinv` column by column. The variable `jcol` is the column index. As in routine `tridc`, we compute the values of `y` and `qy`, except that `delta(i,jcol)` is used in place of `f(i)`. Now we can solve for the

column vector of the inverse matrix using back substitution. This loop is identical to that in tridc except that xinv(i,jcol) is used in place of x(i). The process repeats for the next value of jcol.

```
      subroutine trinvc(a,b,c,xinv,n)

c     this subroutine inverts the matrix B where
c     B is a tridiagonal matrix with 2 corner blocks. an example
c     of such a matrix arises in cubic b-spline interpolation
c     when periodic boundary conditions are used.

c     a is the subdiagonal, c is the superdiagonal, b is the
c     main diagonal. the two extra elements are in the B(1,n)
c     and B(n,1) positions.

      parameter (msize = 64)

      real a(msize), b(msize), c(msize), xinv(msize,msize)
      real y(msize), m(msize), u(msize), q(msize), r(msize)

      m(1)=b(1)
      u(1)=c(1)/m(1)
      q(1)=c(n)
      r(1)=a(1)/m(1)

      qr=q(1)*r(1)

      do 100 j=2,n-2
         m(j)=b(j)-a(j)*u(j-1)
         u(j)=c(j)/m(j)
         q(j)=-q(j-1)*u(j-1)
         r(j)=-a(j)*r(j-1)/m(j)
         qr=qr+q(j)*r(j)
 100     continue

      nm1=n-1
      m(nm1)=b(nm1)-a(nm1)*u(n-2)
      q(nm1)=a(n)-q(n-2)*u(n-2)
      r(nm1)=(c(nm1)-a(nm1)*r(n-2))/m(nm1)
      qr=qr+q(nm1)*r(nm1)

      m(n)=b(n)-qr

      do 300 jcol=1,n
```

```
      y(1)=delta(1,jcol)/m(1)
      qy=q(1)*y(1)
      do 200 i=2,n-1
         y(i)=(delta(i,jcol)-a(i)*y(i-1))/m(i)
         qy=qy+q(i)*y(i)
200      continue
      xinv(n,jcol)=(delta(n,jcol)-qy)/m(n)
      xinv(nm1,jcol)=y(nm1)-r(nm1)*xinv(n,jcol)
      do 250 i=n-2,1,-1
250      xinv(i,jcol)=y(i)-u(i)*xinv(i+1,jcol)-r(i)*xinv(n,jcol)
300   continue

   return
   end
```

Bibliography

[1] D.U. von Rosenberg, *Methods for the Numerical Solution of Partial Differential Equations*, Appendix B, American Elsevier Publishing, New York, (1969).

[2] G. Dahlquist, Å. Björck and N. Anderson, *Numerical Methods*, Prentice–Hall, Englewood Cliffs, NJ, (1974).