NSPIV: A FORTRAN Subroutine for Gaussian Elimination with Partial Pivoting

by

Andrew H. Sherman

January, 1977

TR-65
CNA-118

This work was supported in part by the National Science Foundation under grant NSF DCR 73-07998 and by the Energy Research and Development Administration under grant US ERDA E911-1) 2383. It is being released jointly with the Center for Numerical Analysis at The University of Texas at Austin under the number CNA-118.

Department of Computer Sciences
The University of Texas at Austin

NSPIV: A FORTRAN Subroutine for Sparse Gaussian Elimination with Partial Pivoting*

Andrew H. Sherman
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

Keywords and phrases: sparse Gaussian elimination, sparse linear systems, linear equations, partial pivoting algorithms

CR Categories: 5.14

Language: FORTRAN

* This work was supported in part by the National Science Foundation under grant NSF DCR 73-07998 and by the Energy Research and Development Administration under grant US ERDA E(11-1) 2383.

Description

1. Introduction

NSPIV is a FORTRAN subroutine which solves a sparse system of linear equations

$$Ax = b$$

by sparse Gaussian elimination with partial pivoting. More precisely, it performs Gaussian elimination with column interchanges on the nonsingular N x N matrix A to effectively obtain a factorization of the form

$$AQ = LU, \qquad (1)$$

where L is lower triangular, U is unit upper triangular, and Q is a permutation matrix corresponding to the column interchanges. To conserve storage, only the factor U is retained, so during elimination, operations are performed on the righthand side to obtain the solution y of the system

$$Ly = b.$$

Once U has been obtained, x is computed by solving the upper triangular system

$$UQ^T x = y.$$

The following sections of the algorithm description discuss the computational method and usage of NSPIV. No test results have been included because they already appear in [4]. Those test results show NSPIV to be somewhat more efficient than other currently available software for sparse Gaussian elimination with pivoting.

2. Method

The algorithm used by NSPIV is a row-oriented version of Gaussian elimination with column interchanges. It consists of N steps, during each of which one row of A is processed. When processing the k-th row at the k-th step, a list I_k is used to hold the indices of all columns containing nonzeroes in the k-th row. Then, in increasing order, for each index $m < k$ in I_k, a multiple of row m of U is subtracted from row k to annihilate the corresponding nonzero. This may cause fill-in, i.e., the introduction of new nonzero elements in the k-th row, so the list I_k must be updated. Finally, when all $m < k$ have been processed, I_k contains the indices of columns which contain nonzeroes in the portion of the k-th row lying in the upper triangle of U. Since A is nonsingular, I_k will not be empty, and the algorithm selects a remaining nonzero element with maximum modulus and interchanges its column with the k-th column.

That the NSPIV algorithm is numerically stable can be shown quite easily by relating it to the application of standard Gaussian elimination with row interchanges to A^T. In fact, assume that such a procedure produces a factorization of A^T as

$$PA^T = \tilde{L}\,\tilde{U}, \qquad (2)$$

where $\tilde{L}$ is unit lower triangular, $\tilde{U}$ is upper triangular, and P is a permutation matrix corresponding to the row interchanges. Then we can show that the NSPIV algorithm produces the factorization (1) of A with $L = \tilde{U}^T$, $U = \tilde{L}^T$, and $Q = P^T$. Since the computation of (2) is numerically stable (cf. [2]), that of (1) by the NSPIV algorithm is also.

The keys to efficiency in NSPIV lie in the methods used to store and update the list I_k. In [4] several different methods were implemented and compared, and the best one, "run insertion," is used in NSPIV. The list I_k is stored as a linked list in increasing order relative to the current column ordering at the k-th step. Similarly, the columns of each previous row of U are arranged in increasing order relative to the column ordering at the time the row was computed. (It is important to note that subsequent interchanges may mean that these columns are not in increasing order relative to the current column ordering at the k-th step.)

To update I_k for column index m, the list of nonzero columns for row m of U must be merged with I_k. In NSPIV, this is done as a linear merge, except that each out-of-current-order column in the list for row m causes a return to the beginning of I_k. This is equivalent to splitting the list of indices for row m into its component increasing runs (cf. [3], p. 34) and merging each run separately with I_k using a linear merge. (Hence the name "run insertion.")

3. Matrix Storage

The matrix A is stored in sparse form using the three arrays IA, JA, and A. The array A contains the nonzeroes of the matrix row-by-row, not necessarily in increasing column order. The array JA contains the column numbers corresponding to the nonzeroes in the array A (i.e., if A(K) = a_{IJ}, then JA(K) = J). Finally the array IA contains pointers to the rows of nonzeroes and column numbers in the arrays A and JA (i.e., the I-th row occupies positions IA(I) through IA(I + 1) - 1 of the arrays A and JA, with IA(N + 1) set so that this holds for row N.)

4. Usage

The calling sequence for NSPIV is

```
CALL NSPIV (N,IA,JA,A,B,MAX,R,C,IC,X,ITEMP,RTEMP,IERR)
```

with (arguments preceded by an asterisk are altered by the subroutine):

N — An integer specifying the number of equations and unknowns.

IA — An integer array of N + 1 entries containing row pointers to A.

JA — An integer array with one entry per nonzero in A, containing the column numbers of the nonzeroes of A.

A — A real array with one entry per nonzero in A, containing the actual nonzeroes.

B A real array of N entries containing the righthand side data.

MAX An integer specifying the maximum number of off-diagonal nonzeroes of U which may be stored.

R An integer array of N entries specifying the order of the rows of A (i.e., the elimination order of the equations).

*C An integer array of N entries. On input, C specifies the order of the columns of A. On output, C specifies the order of the columns of U.

*IC An integer array of N entries which is the inverse permutation of C (i.e., IC(C(I)) = I).

*X A real array of N entries which contains the solution on output.

*ITEMP An integer array of 2N + MAX + 2 entries which is used for temporary storage by NSPIV.

*RTEMP A real array of N + MAX entries which is used for temporary storage by NSPIV.

*IERR An integer which indicates error conditions or (on successful termination) the number of off-diagonal entries in U. The comments in the code describe the values assigned to IERR.

The actual numerical computations are performed in an internal subroutine NSPIV1, which is written to perform all computations in single precision. Conversion to double precision may be accomplished simply by changing REAL declarations to DOUBLE PRECISION declarations in both NSPIV and NSPIV1, and by changing the calls to ABS into calls to DABS.

In practice, the efficiency of NSPIV may be affected by the initial ordering of the rows of A. (cf. [1,4]). One strategy which has been found to be helpful is to order the rows of A by increasing numbers of nonzeroes. Providing such an initial ordering to NSPIV is accomplished by setting the array R appropriately; no actual changes in the arrays IA, JA, and A are required. In this case, row R(1) will be the row with the fewest nonzeros, and row R(N) will be the row with the most nonzeroes.

References

[1] A. R. Curtis and J. K. Reid. The Solution of Large Sparse Unsymmetric Systems of Linear Equations. Information Processing 71, pp. 1240-45, 1971.

[2] G. E. Forsythe and C. B. Moler. Computer Solution of Linear Algebraic Equations. Prentice-Hall, 1967.

[3] D. E. Knuth. The Art of Computer Programming, Volume 3: Searching and Sorting. Addison-Wesley, 1973.

[4] A. H. Sherman. Algorithms for Sparse Gaussian Elimination with Partial Pivoting. University of Illinois Department of Computer Science report UIUCDCS-R-76-817, 1976.

```
      SUBROUTINE NSPIV (N,IA,JA,A,B,MAX,R,C,IC,X,ITEMP,RTEMP,IERR)
C
C
C   NSPIV CALLS NSPIV1 WHICH USES SPARSE GAUSSIAN ELIMINATION WITH
C   COLUMN INTERCHANGES TO SOLVE THE LINEAR SYSTEM A X = B.  THE
C   ELIMINATION PHASE PERFORMS ROW OPERATIONS ON A AND B TO OBTAIN
C   A UNIT UPPER TRIANGULAR MATRIX U AND A VECTOR Y.  THE SOLUTION
C   PHASE SOLVES U X = Y.
C
C
C   INPUT ARGUMENTS---
C
C   N       INTEGER NUMBER OF EQUATIONS AND UNKNOWNS
C
C   IA      INTEGER ARRAY OF N+1 ENTRIES CONTAINING ROW POINTERS TO A
C           (SEE MATRIX STORAGE DESCRIPTION BELOW)
C
C   JA      INTEGER ARRAY WITH ONE ENTRY PER NONZERO IN A, CONTAINING
C           COLUMN NUMBERS OF THE NONZEROES OF A.  (SEE MATRIX STORAGE
C           DESCRIPTION BELOW)
C
C   A       REAL ARRAY WITH ONE ENTRY PER NONZERO IN A, CONTAINING THE
C           ACTUAL NONZEROES.  (SEE MATRIX STORAGE DESCRIPTION BELOW)
C
C   B       REAL ARRAY OF N ENTRIES CONTAINING RIGHT HAND SIDE DATA
C
C   MAX     INTEGER NUMBER SPECIFYING MAXIMUM NUMBER OF OFF-DIAGONAL
C           NONZERO ENTRIES OF U WHICH MAY BE STORED
C
C   R       INTEGER ARRAY OF N ENTRIES SPECIFYING THE ORDER OF THE
C           ROWS OF A (I.E., THE ELIMINATION ORDER FOR THE EQUATIONS)
C
C   C       INTEGER ARRAY OF N ENTRIES SPECIFYING THE ORDER OF THE
C           COLUMNS OF A.  C IS ALSO AN OUTPUT ARGUMENT
C
C   IC      INTEGER ARRAY OF N ENTRIES WHICH IS THE INVERSE OF C
C           (I.E., IC(C(I)) = I).  IC IS ALSO AN OUTPUT ARGUMENT
C
C   ITEMP   INTEGER ARRAY OF 2*N + MAX + 2 ENTRIES, FOR INTERNAL USE
C
C   RTEMP   REAL ARRAY OF N + MAX ENTRIES FOR INTERNAL USE
C
C
C   OUTPUT ARGUMENTS---
C
C   C       INTEGER ARRAY OF N ENTRIES SPECIFYING THE ORDER OF THE
C           COLUMNS OF U.  C IS ALSO AN INPUT ARGUMENT
C
C   IC      INTEGER ARRAY OF N ENTRIES WHICH IS THE INVERSE OF C
C           (I.E., IC(C(I)) = I).  IC IS ALSO AN INPUT ARGUMENT
C
C   X       REAL ARRAY OF N ENTRIES CONTAINING THE SOLUTION VECTOR
C
C   IERR    INTEGER NUMBER WHICH INDICATES ERROR CONDITIONS OR
C           THE ACTUAL NUMBER OF OFF-DIAGONAL ENTRIES IN U (FOR
C           SUCCESSFUL COMPLETION)
C
C           IERR VALUES ARE---
C
C           0 LT IERR                  SUCCESSFUL COMPLETION.  U HAS IERR
C                                      OFF-DIAGONAL NONZERO ENTRIES
```

```
C
C         IERR = 0                   ERROR.  N = 0
C
C         -N LE IERR LT 0            ERROR.  ROW NUMBER IABS(IERR) OF A IS
C                                    IS NULL
C
C         -2*N LE IERR LT -N         ERROR.  ROW NUMBER IABS(IERR+N) HAS A
C                                    DUPLICATE ENTRY
C
C         -3*N LE IERR LT -2*N       ERROR.  ROW NUMBER IABS(IERR+2*N)
C                                    HAS A ZERO PIVOT
C
C         -4*N LE IERR LT -3*N       ERROR.  ROW NUMBER IABS(IERR+3*N)
C                                    EXCEEDS STORAGE
C
C
C   STORAGE OF SPARSE MATRICES---
C
C   THE SPARSE MATRIX A IS STORED USING THREE ARRAYS IA, JA, AND A.
C   THE ARRAY A CONTAINS THE NONZEROES OF THE MATRIX ROW-BY-ROW, NOT
C   NECESSARILY IN ORDER OF INCREASING COLUMN NUMBER.  THE ARRAY JA
C   CONTAINS THE COLUMN NUMBERS CORRESPONDING TO THE NONZEROES STORED
C   IN THE ARRAY A (I.E., IF THE NONZERO STORED IN A(K) IS IN
C   COLUMN J, THEN JA(K) = J).  THE ARRAY IA CONTAINS POINTERS TO THE
C   ROWS OF NONZEROES/COLUMN INDICES IN THE ARRAY A/JA (I.E.,
C   A(IA(I))/JA(IA(I)) IS THE FIRST ENTRY FOR ROW I IN THE ARRAY A/JA).
C   IA(N+1) IS SET SO THAT IA(N+1) - IA(1) = THE NUMBER OF NONZEROES IN A
C
C
      REAL A(1),B(1),X(1),RTEMP(1)
      INTEGER IA(1),JA(1),R(1),C(1),IC(1),ITEMP(1)
      INTEGER IU,JU,U,Y,P
C
C   SET INDICES TO DIVIDE TEMPORARY STORAGE FOR NSPIV1
C
      Y = 1
      U = Y + N
      P = 1
      IU = P + N + 1
      JU = IU + N + 1
C
C   CALL NSPIV1 TO PERFORM COMPUTATIONS
C
      CALL NSPIV1 (N,IA,JA,A,B,MAX,R,C,IC,X,RTEMP(Y),ITEMP(P),
     C             ITEMP(IU),ITEMP(JU),RTEMP(U),IERR)
      RETURN
      END
      SUBROUTINE NSPIV1 (N,IA,JA,A,B,MAX,R,C,IC,X,Y,P,IU,JU,U,IERR)
C
C
C   NSPIV1 USES SPARSE GAUSSIAN ELIMINATION WITH
C   COLUMN INTERCHANGES TO SOLVE THE LINEAR SYSTEM A X = B.  THE
C   ELIMINATION PHASE PERFORMS ROW OPERATIONS ON A AND B TO OBTAIN
C   A UNIT UPPER TRIANGULAR MATRIX U AND A VECTOR Y.  THE SOLUTION
C   PHASE SOLVES U X = Y.
C
C
C   SEE NSPIV FOR DESCRIPTIONS OF ALL INPUT AND OUTPUT ARGUMENTS
C   OTHER THAN THOSE DESCRIBED BELOW
C
C   INPUT ARGUMENTS (USED INTERNALLY ONLY)---
C
```

```
C   Y    REAL ARRAY OF N ENTRIES USED TO COMPUTE THE UPDATED
C        RIGHT HAND SIDE
C
C   P    INTEGER ARRAY OF N+1 ENTRIES USED FOR A LINKED LIST.
C        P(N+1) IS THE LIST HEADER, AND THE ENTRY FOLLOWING
C        P(K) IS IN P(P(K)).  THUS, P(N+1) IS THE FIRST DATA
C        ITEM, P(P(N+1)) IS THE SECOND, ETC.  A POINTER OF
C        N+1 MARKS THE END OF THE LIST
C
C   IU   INTEGER ARRAY OF N+1 ENTRIES USED FOR ROW POINTERS TO U
C        (SEE MATRIX STORAGE DESCRIPTION BELOW)
C
C   JU   INTEGER ARRAY OF MAX ENTRIES USED FOR COLUMN NUMBERS OF
C        THE NONZEROES IN THE STRICT UPPER TRIANGLE OF U.  (SEE
C        MATRIX STORAGE DESCRIPTION BELOW)
C
C   U    REAL ARRAY OF MAX ENTRIES USED FOR THE ACTUAL NONZEROES IN
C        THE STRICT UPPER TRIANGLE OF U.  (SEE MATRIX STORAGE
C        DESCRIPTION BELOW)
C
C
C  STORAGE OF SPARSE MATRICES---
C
C  THE SPARSE MATRIX A IS STORED USING THREE ARRAYS IA, JA, AND A.
C  THE ARRAY A CONTAINS THE NONZEROES OF THE MATRIX ROW-BY-ROW, NOT
C  NECESSARILY IN ORDER OF INCREASING COLUMN NUMBER.  THE ARRAY JA
C  CONTAINS THE COLUMN NUMBERS CORRESPONDING TO THE NONZEROES STORED
C  IN THE ARRAY A (I.E., IF THE NONZERO STORED IN A(K) IS IN
C  COLUMN J, THEN JA(K) = J).  THE ARRAY IA CONTAINS POINTERS TO THE
C  ROWS OF NONZEROES/COLUMN INDICES IN THE ARRAY A/JA (I.E.,
C  A(IA(I))/JA(IA(I)) IS THE FIRST ENTRY FOR ROW I IN THE ARRAY A/JA).
C  IA(N+1) IS SET SO THAT IA(N+1) - IA(1) = THE NUMBER OF NONZEROES IN
C  A.  IU, JU, AND U ARE USED IN A SIMILAR WAY TO STORE THE STRICT UPPER
C  TRIANGLE OF U, EXCEPT THAT JU ACTUALLY CONTAINS C(J) INSTEAD OF J
C
C
      REAL A(1),B(1),U(1),X(1),Y(1)
      REAL DK,LKI,ONE,XPV,XPVMAX,YK,ZERO
      INTEGER C(1),IA(1),IC(1),IU(1),JA(1),JU(1),P(1),R(1)
      INTEGER CK,PK,PPK,PV,V,VI,VJ,VK
C
C
      IF (N .EQ. 0) GO TO 1001
C
      ONE = 1.0
      ZERO = 0.0
C
C  INITIALIZE WORK STORAGE AND POINTERS TO JU
C
      DO 10 J=1,N
        X(J) = ZERO
 10     CONTINUE
      IU(1) = 1
      JUPTR = 0
C
C  PERFORM SYMBOLIC AND NUMERIC FACTORIZATION ROW BY ROW
C  VK (VI,VJ) IS THE GRAPH VERTEX FOR ROW K (I,J) OF U
C
      DO 170 K=1,N
C
C  INITIALIZE LINKED LIST AND FREE STORAGE FOR THIS ROW
C  THE R(K)-TH ROW OF A BECOMES THE K-TH ROW OF U.
```

```
C
         P(N+1) = N+1
         VK = R(K)
C
C  SET UP ADJACENCY LIST FOR VK, ORDERED IN
C  CURRENT COLUMN ORDER OF U.  THE LOOP INDEX
C  GOES DOWNWARD TO EXPLOIT ANY COLUMNS
C  FROM A IN CORRECT RELATIVE ORDER
C
         JMIN = IA(VK)
         JMAX = IA(VK+1) - 1
         IF (JMIN .GT. JMAX)  GO TO 1002
         J = JMAX
 20        JAJ = JA(J)
           VJ = IC(JAJ)
C
C  STORE A(K,J) IN WORK VECTOR
C
           X(VJ) = A(J)
C  THIS CODE INSERTS VJ INTO ADJACENCY LIST OF VK
           PPK = N+1
 30        PK = PPK
           PPK = P(PK)
           IF (PPK - VJ)  30,1003,40
 40        P(VJ) = PPK
           P(PK) = VJ
           J = J - 1
           IF (J .GE. JMIN) GO TO 20
C
C  THE FOLLOWING CODE COMPUTES THE K-TH ROW OF U
C
         VI = N+1
         YK = B(VK)
 50      VI = P(VI)
         IF (VI .GE. K) GO TO 110
C
C  VI LT VK -- PROCESS THE L(K,I) ELEMENT AND MERGE THE
C  ADJACENCY OF VI WITH THE ORDERED ADJACENCY OF VK
C
         LKI = - X(VI)
         X(VI) = ZERO
C
C  ADJUST RIGHT HAND SIDE TO REFLECT ELIMINATION
C
         YK = YK + LKI * Y(VI)
         PPK = VI
         JMIN = IU(VI)
         JMAX = IU(VI+1) - 1
         IF (JMIN .GT. JMAX)  GO TO 50
         DO 100 J=JMIN,JMAX
           JUJ = JU(J)
           VJ = IC(JUJ)
C
C  IF VJ IS ALREADY IN THE ADJACENCY OF VK,
C  SKIP THE INSERTION
C
           IF (X(VJ) .NE. ZERO)  GO TO 90
C
C  INSERT VJ IN ADJACENCY LIST OF VK.
C  RESET PPK TO VI IF WE HAVE PASSED THE CORRECT
C  INSERTION SPOT.  (THIS HAPPENS WHEN THE ADJACENCY OF
C  VI IS NOT IN CURRENT COLUMN ORDER DUE TO PIVOTING.)
```

```
C
          IF (VJ - PPK) 60,90,70
 60       PPK = VI
 70       PK = PPK
          PPK = P(PK)
          IF (PPK - VJ)  70,90,80
 80       P(VJ) = PPK
          P(PK) = VJ
          PPK = VJ
C
C  COMPUTE L(K,J) = L(K,J) - L(K,I)*U(I,J) FOR L(K,I) NONZERO
C  COMPUTE U*(K,J) = U*(K,J) - L(K,I)*U(I,J) FOR U(K,J) NONZERO
C  (U*(K,J) = U(K,J)*D(K,K))
C
 90       X(VJ) = X(VJ) + LKI * U(J)
 100      CONTINUE
        GO TO 50
C
C  PIVOT--INTERCHANGE LARGEST ENTRY OF K-TH ROW OF U WITH
C  THE DIAGONAL ENTRY.
C
C  FIND LARGEST ENTRY, COUNTING OFF-DIAGONAL NONZEROES
C
 110    IF (VI .GT. N) GO TO 1004
        XPVMAX = ABS(X(VI))
        MAXC = VI
        NZCNT = 0
        PV = VI
 120      V = PV
          PV = P(PV)
          IF (PV .GT. N) GO TO 130
          NZCNT = NZCNT + 1
          XPV = ABS(X(PV))
          IF (XPV .LE. XPVMAX) GO TO 120
          XPVMAX = XPV
          MAXC = PV
          MAXCL = V
          GO TO 120
 130    IF (XPVMAX .EQ. ZERO) GO TO 1004
C
C  IF VI = K, THEN THERE IS AN ENTRY FOR DIAGONAL
C  WHICH MUST BE DELETED.  OTHERWISE, DELETE THE
C  ENTRY WHICH WILL BECOME THE DIAGONAL ENTRY
C
        IF (VI .EQ. K) GO TO 140
        IF (VI .EQ. MAXC) GO TO 140
        P(MAXCL) = P(MAXC)
        GO TO 150
 140    VI = P(VI)
C
C  COMPUTE D(K) = 1/L(K,K) AND PERFORM INTERCHANGE.
C
 150    DK = ONE / X(MAXC)
        X(MAXC) = X(K)
        I = C(K)
        C(K) = C(MAXC)
        C(MAXC) = I
        CK = C(K)
        IC(CK) = K
        IC(I) = MAXC
        X(K) = ZERO
C
```

```
C  UPDATE RIGHT HAND SIDE.
C
         Y(K) = YK * DK
C
C  COMPUTE VALUE FOR IU(K+1) AND CHECK FOR STORAGE OVERFLOW
C
         IU(K+1) = IU(K) + NZCNT
         IF (IU(K+1) .GT. MAX+1) GO TO 1005
C
C  MOVE COLUMN INDICES FROM LINKED LIST TO JU.
C  COLUMNS ARE STORED IN CURRENT ORDER WITH ORIGINAL
C  COLUMN NUMBER (C(J)) STORED FOR CURRENT COLUMN J
C
         IF (VI .GT. N)  GO TO 170
         J = VI
 160       JUPTR = JUPTR + 1
           JU(JUPTR) = C(J)
           U(JUPTR) = X(J) * DK
           X(J) = ZERO
           J = P(J)
           IF (J .LE. N) GO TO 160
 170     CONTINUE
C
C  BACKSOLVE U X = Y, AND REORDER X TO CORRESPOND WITH A
C
       K = N
       DO 200 I=1,N
         YK = Y(K)
         JMIN = IU(K)
         JMAX = IU(K+1) - 1
         IF (JMIN .GT. JMAX)  GO TO 190
         DO 180 J=JMIN,JMAX
           JUJ = JU(J)
           JUJ = IC(JUJ)
           YK = YK - U(J) * Y(JUJ)
 180       CONTINUE
 190     Y(K) = YK
         CK = C(K)
         X(CK) = YK
         K = K-1
 200     CONTINUE
C
C  RETURN WITH IERR = NUMBER OF OFF-DIAGONAL NONZEROES IN U
C
       IERR = IU(N+1) - IU(1)
       RETURN
C
C  ERROR RETURNS
C
C  N = 0
C
 1001 IERR = 0
       RETURN
C
C  ROW K OF A IS NULL
C
 1002 IERR = -K
       RETURN
C
C  ROW K OF A HAS A DUPLICATE ENTRY
C
 1003 IERR = -(N+K)
```

```
      RETURN
C
C  ZERO PIVOT IN ROW K
C
 1004 IERR = -(2*N+K)
      RETURN
C
C  STORAGE FOR U EXCEEDED ON ROW K
C
 1005 IERR = -(3*N+K)
      RETURN
      END
```

Appendix A

The code in this appendix illustrates the use of NSPIV. The linear system is of the form

$$Ax = b$$

where A is an 10x10 block tridiagonal matrix with 10x10 blocks. Specifically,

$$A = \begin{bmatrix} C & D & & & 0 \\ B & C & D & & \\ & B & \ddots & \ddots & \\ & & \ddots & \ddots & D \\ 0 & & & B & C \end{bmatrix}$$

with

$$B = \begin{bmatrix} -1 & & & 0 \\ & -1 & & \\ & & \ddots & \\ 0 & & & -1 \end{bmatrix},$$

$$C = \begin{bmatrix} 4 & & & 0 \\ -1 & 4 & & \\ & -1 & \ddots & \\ & & \ddots & \ddots & \\ 0 & & & -1 & 4 \end{bmatrix},$$

and

$$D = \begin{bmatrix} -1.5 & & & 0 \\ & -1.5 & & \\ & & \ddots & \\ 0 & & & -1.5 \end{bmatrix}.$$

This example is chosen for its simplicity; it does not exercise the algorithm, since A is a strictly diagonally dominant matrix.

```
C
C  THIS PROGRAM ILLUSTRATES THE USE OF NSPIV BY SOLVING THE
C  SYSTEM OF LINEAR EQUATIONS
C
C       A X = B
C
C  WITH A AN NG X NG BLOCK TRIDIAGONAL MATRIX, WITH NG X NG BLOCKS.
C  THE DIAGONAL BLOCKS OF A ARE LOWER BI-DIAGONAL (ENTRIES ARE 4.0
C  ON THE DIAGONAL, -1.0 ON THE SUBDIAGONAL), AND THE OFF-DIAGONAL
C  BLOCKS OF A ARE DIAGONAL (ENTRIES ARE -1.0 IN THE LOWER TRIANGLE,
C  -1.5 IN THE UPPER TRIANGLE.)  X IS CHOSEN TO BE A VECTOR
C  OF ALL ONES, AND B IS COMPUTED ACCORDINGLY.
C
C
      INTEGER IA(101),JA(400),R(100),C(100),IC(100),ITEMP(597)
      REAL A(400),B(100),X(100),RTEMP(495)
      DATA MAX/395/,NG/10/,N/100/
C
C  SET UP PROBLEM
C
      K = 1
      IA(1) = 1
      IAPTR = 1
      DO 5 I=1,NG
        DO 5 J=1,NG
          BK = 0.
          IF (I .EQ. 1) GO TO 1
          JA(IAPTR) = K - NG
          A(IAPTR) = -1.
          BK = BK - 1.
          IAPTR = IAPTR + 1
 1        IF (J .EQ. 1) GO TO 2
          JA(IAPTR) = K - 1
          A(IAPTR) = -1.
          BK = BK - 1.
          IAPTR = IAPTR + 1
 2        JA(IAPTR) = K
          A(IAPTR) = 4.
          BK = BK + 4.
          IAPTR = IAPTR + 1
          IF (I .EQ. NG) GO TO 4
          JA(IAPTR) = K + NG
          A(IAPTR) = -1.5
          BK = BK - 1.5
          IAPTR = IAPTR + 1
 4        B(K) = BK
          K = K + 1
          IA(K) = IAPTR
 5        CONTINUE
C
C  CALL PREORD TO ORDER ROWS OF A BY INCREASING NUMBERS OF NONZEROES
C
      CALL PREORD(N,IA,R,C,IC)
C
C  CALL NSPIV TO SOLVE SYSTEM
C
      CALL NSPIV(N,IA,JA,A,B,MAX,R,C,IC,X,ITEMP,RTEMP,IERR)
      WRITE (6,101) IERR
 101  FORMAT (8H IERR = ,I10)
C
```

```
C   CALL RESCHK TO COMPUTE MAX-NORM AND 2-NORM OF RESIDUAL
C
      CALL RESCHK(N,IA,JA,A,B,X)
C
      STOP
      END
      SUBROUTINE PREORD(N,IA,R,C,IC)
C
C   PREORD ORDERS THE ROWS OF A BY INCREASING NUMBER OF NONZEROES.
C   THE ROW PERMUTATION IS RETURNED IN R.  C IS SET TO THE IDENTITY.
C
      INTEGER IA(1),R(1),C(1),IC(1)
C
      DO 1 I=1,N
        R(I) = I
        C(I) = I
        IC(I) = I
1       CONTINUE
      DO 5 I = 1,N
5       C(I) = 0
      DO 10 K = 1,N
        KDEG = IA(K+1) - IA(K)
        IF (KDEG .EQ. 0) KDEG = KDEG + 1
        IC(K) = C(KDEG)
        C(KDEG) = K
10      CONTINUE
      I = 0
      DO 30 J = 1,N
        IF (C(J) .EQ. 0) GO TO 30
        K = C(J)
20      I = I + 1
        R(I) = K
        K = IC(K)
        IF (K .GT. 0) GO TO 20
30      CONTINUE
      DO 40 I = 1,N
        C(I) = I
        IC(I) = I
40      CONTINUE
      RETURN
      END
      SUBROUTINE RESCHK(N,IA,JA,A,B,X)
C
C   RESCHK COMPUTES THE MAX-NORM AND 2-NORM OF THE RESIDUAL.
C   DOUBLE PRECISION IS USED FOR THE COMPUTATION.
C
      INTEGER IA(1),JA(1)
      REAL A(1),B(1),X(1)
      DOUBLE PRECISION RESID,RESIDM,ROWSUM
      RESID = 0.
      RESIDM = 0.
      DO 20 I=1,N
        ROWSUM = DBLE(B(I))
        JMIN = IA(I)
        JMAX = IA(I+1) - 1
        DO 10 J=JMIN,JMAX
          JAJ = JA(J)
          ROWSUM = ROWSUM - DBLE(A(J)) * DBLE(X(JAJ))
10        CONTINUE
        IF (DABS(ROWSUM) .GT. RESIDM) RESIDM = DABS(ROWSUM)
        RESID = RESID + ROWSUM**2
20      CONTINUE
```

```
      RESID = DSQRT(RESID)
      WRITE (6,25) RESID
25    FORMAT (22H 2-NORM OF RESIDUAL = ,D14.7)
      WRITE (6,30) RESIDM
30    FORMAT(24H MAX NORM OF RESIDUAL = ,D14.7)
      RETURN
      END
```