APRIL 1978 PPPL-1435
UC-20g

A PROGRAM GENERATOR FOR THE INCOMPLETE CHOLESKY CONJUGATE GRADIENT (ICCG) METHOD

BY

G. KUO-PETRAVIC AND M. PETRAVIC

PLASMA PHYSICS LABORATORY

PRINCETON UNIVERSITY
PRINCETON, NEW JERSEY

This work was supported by the U. S. Department of Energy Contract No. EY-76-C-02-3073. Reproduction, translation, publication, use and disposal, in whole or in part, by or for the United States Government is permitted.

ABSTRACT

The Incomplete Cholesky Conjugate Gradient (ICCG) method has been found very effective for the solution of sparse systems of linear equations. Its implementation on a computer, however, requires a considerable amount of careful coding to achieve good machine efficiency. Furthermore, the resulting code is necessarily inflexible and cannot be easily adapted to different problems. We present in this paper a code generator GENIC which, given a small amount of information concerning the sparsity pattern and size of the system of equations, generates a solver package. This package, called SOLIC, is tailor made for a particular problem and can be easily incorporated into any user program.

PROGRAM SUMMARY

Title of Program: GENIC

Catalogue number:

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland, GB

Computer: PDP 10 Installation: Computer Centre, Plasma Physics Laboratory, Princeton University, James Forrestal Campus, P.O. Box 451, Princeton, NJ 08540 USA

Operating system: TOPS-10, version 6.03

Programming language used: Fortran IV

High speed storage required: 11,280 words for source module
14,463 words for relocatable module

Number of bits in a word: 36 bits/word

Overlay structure: None

Number of magnetic tapes required: None

Other peripherals used: None

Number of cards in combined program and test deck: approximately 1,800 cards

Card punching code: ASCII

Keywords: Linear Sparse System, Incomplete Cholesky Decomposition, Conjugate Gradient, Code Generator

Nature of physical problem

The program generator GENIC and the resultant Incomplete-Cholesky Conjugate Gradient (ICCG) Solver SOLIC are applicable to a wide range of physical problems, in particular these modelled by partial differential equations. Elliptic, parabolic and hyperbolic types of equations are all covered since the method can

be used to solve any sparse system of linear equations with a positive definite matrix of coefficients.

Method of solution

The Incomplete Cholesky Conjugate Gradient method, known as the ICCG method, based on the work of Meijerink and VanderVorst [1] and Hestenes and Stiefel [2] is used.

Restrictions on the complexity of the problem

The method will work for systems with any degree of sparsity. However, the particular generator is most efficient for sparse matrices with a definite band structure with only a few zeros in the sub-diagonals containing non-zero coefficients. If these conditions are not satisfied an unnecessarily large amount of redundant expressions will be computed. The generator GENIC also assumes a repetitive block structure, discussed in detail in the long write up. At present the dimensions of arrays which describe the beginnings and ends of diagonal bands in the problem matrix are set at 20, which means not more than 40 bands can appear in the problem matrix. This number, however, can easily be varied by re-dimensioning the Common list in GENIC.

Typical running time: For GENIC to produce the test solver SOLIC shown in this paper the typical running time is around 1.5 secs.

References: J.A. Meijerink and H. A. Van Der Vorst, "An Iterative Solution Method for Linear Systems of which the Coefficient Matrix is a Symmetric M-matrix," Technical report TR-1, Academic Computer Centre, Budapestlaan 6, de Uithof-UTRECHT, The Netherlands (1976).

*This work was supported by United States Department of Energy Contract no. EY-76-C-02-3073.

*Also Lawrence Livermore Laboratory, Livermore, CA 94550 Contract no. W-7405-ENG-48.

LONG WRITE UP

1. Introduction

Many different problems in physics and engineering often result in a common type of a mathematical equation. If the solution domains are such that the common equation can also be solved by the same method, a fixed solver program can usually be provided thereby avoiding a lot of program duplication. The core of a solver usually contains a computer programmed form of a particular numerical algorithm and usually requires only the actual equation coefficients to be input to produce a solution. A solver, being a fixed program, usually has to balance generality against efficiency. If many of the coefficients can be zero in some problems, a fixed program may find itself doing unnecessary work. Solving large sparse systems of linear equations resulting from differencing partial differential equations on a point grid is one typical example.

The awkwardness of a fixed program becomes especially obvious when a number of variants of the same numerical method are available but the choice between them cannot be made until the numerical values of, say, matrix coefficients are known. This type of problem can often be best solved by writing a program generator rather than a fixed program. Since the actual solver program produced by a generator usually gets executed many times during a computer run, the overheads of program generators are quite small. The advantages of the program generator approach are therefore two-fold: Variation and experimentation with a numerical algorithm becomes virtually

effortless, and the tailor-made solver program is generally very efficient. The improvement in speed comes from both being able to choose the best variant of a numerical algorithm and from moving work from the execution stage into the generation and compilation stages. The actual gains in execution speed depend strongly on a particular computer-compiler combination and even more on the numerical algorithm. In many instances a carefully Assembler coded fixed program will turn out to be the best answer to a particular need but there are also substantial areas of numerical analysis where coding by means of code generators has distinct advantages. Most likely candidates for code generation are problems for which not only variations of the solution method are possible but which also require elaborate coding involving a lot of hand calculation by a programmer.

2. Application to the ICCG Method

One case where conditions for code generation seem well satisfied is the Incomplete Cholesky Conjugate Gradient (ICCG) method as applied to solving partial differential equations using implicit finite difference schemes. The ICCG method is a much improved version of the conjugate gradient method developed by Hestenes and Stiefel [2] in the early 50's, the improvement resulting from replacing the iterations with the original problem matrix A in $A\underset{\sim}{x} = \underset{\sim}{b}$ by iteration with an approximate inverse of A. In the ICCG method the approximate inverse is obtained by imcomplete Cholesky L-U decomposition as proposed by Meijerink and Van Der Vorst [1]. The usual Cholesky version of Gaussian elimination is used but a pre-selected

sparsity pattern is forced upon the L and U matrices. The pattern is usually that of the original problem matrix. The freedom of choosing this sparsity pattern in different ways and the dependence of convergence rate on it, make for a type of coding very suitable for code generators. In the first code generator we make available, the allowed sparsity patterns are rather restricted since they consist only of variable location and width digonal and sub-diagonal bands. Even so, this modest amount of extra flexibility allows not only for instantaneous coding of many finite difference schemes in two or three dimensions but it also makes tuning practicable resulting in improvements in speed of more than 40% over the standard sparsity pattern in the test case shown in this paper.

3. The Numerical Algorithm

Extensive literature exists on both the original Conjugate Gradient method [2,3,4] and its extension the ICCG method [1,5,6], so we present only the actual algorithm coded by the generator. The algorithm solves a linear system of the form: $A\underset{\sim}{x} = \underset{\sim}{b}$, where $\underset{\sim}{x}$ and $\underset{\sim}{b}$ are column vectors and A is a non-singular positive definite square matrix. Since A need not be symmetric, this algorithm is somewhat more elaborate then the original Merjerink and Van Der Vorst algorithm. It can, however, be easily derived from the conjugate gradient algorithm of Hestenes [3], p. 93 and is essentially the same as that proposed by Kershaw [6]. The algorithm consists of successively approximating the solution $\underset{\sim}{x}$ by a series of direction vectors $\underset{\sim}{p}_i$ mutually conjugate with respect to the matrix $N = A^T HA$ and also orthogonal to a sequence

of gradient vectors $\underset{\sim}{g}_i$. The gradient vectors are in turn conjugate with respect to matrix K. The metrics H and K are arbitrary positive Hermitian matrics, which also makes N a positive Hermitian; A^T is a transpose of A.

The recursive procedure for constructing the $\underset{\sim}{p}_i$'s involves two other vector sequences, that of the residual vectors $\underset{\sim}{r}_i = \underset{\sim}{b} - A\underset{\sim}{x}_i$ and subsidiary vectors $\underset{\sim}{s}_i = H\underset{\sim}{r}_i$. The code generator assumes all the matrices to be real and uses the choice $H = (LL^T)^{-1}$ and $K = (U^TU)^{-1}$, where LU is an approximate of A obtained by the already described imcomplete Cholesky decomposition process and L^T and U^T are the transposes of L and U respectively. Our particular algorithum for solving $A\underset{\sim}{x} = \underset{\sim}{b}$ (1) then reads as follows:

$$\underset{\sim}{r}_o = \underset{\sim}{b} - A\underset{\sim}{x}_o \qquad (2:1) \qquad \underset{\sim}{s}_o = (LL^T)^{-1}\underset{\sim}{r}_o \qquad (2:2)$$

$$\underset{\sim}{g}_o = (A^T)\underset{\sim}{s}_o \qquad (2:3) \qquad \underset{\sim}{p}_o = (U^TU)^{-1}\underset{\sim}{g}_o \qquad (2:4)$$

$$\alpha_i = \frac{(\underset{\sim}{r}_i , \underset{\sim}{s}_i)}{(\underset{\sim}{p}_i , \underset{\sim}{g}_i)} \qquad (2:5) \qquad \underset{\sim}{x}_{i+1} = \underset{\sim}{x}_i + \alpha_i\underset{\sim}{p}_i \qquad (2:6)$$

$$\underset{\sim}{r}_{i+1} = \underset{\sim}{r}_i - \alpha A\underset{\sim}{p}_i \qquad (2:7) \qquad \underset{\sim}{s}_{i+1} = (LL^T)^{-1}\underset{\sim}{r}_{i+1} \qquad (2:8)$$

$$\beta_i = \frac{(\underset{\sim}{r}_{i+1} , \underset{\sim}{s}_{i+1})}{(\underset{\sim}{r}_i , \underset{\sim}{s}_i)} \qquad (2:9)$$

$$\underset{\sim}{g}_{i+1} = A^T\underset{\sim}{s}_{i+1} + \beta_i\underset{\sim}{g}_i \qquad (2:10) \qquad \underset{\sim}{p}_{i+1} = (U^TU)^{-1}\underset{\sim}{g}_{i+1} \qquad (2:11)$$

The following relationships hold among the direction and gradient vectors:

$$(\underset{\sim}{g}_i , K\underset{\sim}{g}_j) = 0 \qquad (i \neq j) \qquad (3:1)$$

$$(\underset{\sim}{p}_i , N\underset{\sim}{p}_j) = 0 \qquad (i \neq j) \qquad (3:2)$$

$$(\underset{\sim}{p}_i , \underset{\sim}{g}_j) = C_i \qquad (j = o, 1, \ldots, i) \qquad (3:3)$$

$$(\underset{\sim}{p}_i , \underset{\sim}{g}_j) = 0 \qquad (i = o, 1, \ldots, j-1) \qquad (3:4)$$

Also, the residuals satisfy:

$$(\underset{\sim}{r}_i , H\underset{\sim}{r}_j) = (\underset{\sim}{r}_i , \underset{\sim}{s}_j) = 0 \quad (i \neq j) \qquad (3:5)$$

The conjugate gradient method ensures convergence to the exact solution within a maximum of N iterations, where N is the dimension of the system. This fact is only of theoretical significance since in practice sufficiently good approximations must be obtained in around $\sqrt{N}$ iterations or less.

The method minimizes $\|\underset{\sim}{x}_i - \underset{\sim}{x}\|_N$ for all i and for all algorithms of the form:

$$\underset{\sim}{x}_i = \underset{\sim}{x}_o + P_{i-1}(T)T(\underset{\sim}{x} - \underset{\sim}{x}_o) \quad , \qquad (4:1)$$

where P is a polynomial of degree i - 1 in T and

$$T = KN = (U^TU)^{-1}A^T(LL^T)^{-1}A \qquad (4:2)$$

The upper bound on the error is then [1]:

$$\| \underset{\sim}{x}_i - \underset{\sim}{x} \|_N^2 \leq \left(\frac{\sqrt{c} - 1}{\sqrt{c} + 1} \right)^{2i} \| \underset{\sim}{x}_o - \underset{\sim}{x} \|_N^2 \qquad (5)$$

where $c = \lambda_{max}(T)/\lambda_{min}(T)$.

Since T is constructed to approximate the unit matrix, the expectations are that the algorithm will converge much faster than the original CG method. The performance of the method in practice is discussed at length in [1] and [6].

4. The Generator Program

The ICCG method is most effective when A is sparse and in particular when all the non-zero elements are concentrated on the main diagonal and a small number of sub-diagonals. The 5,7,9, etc., point finite difference schemes result in precisely such matrices with the sub-diagonals usually clustering together to form bands. This structure permits easy compression of the full matrix A since only the diagonals containing non-zero elements need be stored. These sub-diagonals together with the main diagonal are stored in two 2D arrays ADU and ADL corresponding to the upper and lower triangular parts of A and both containing the main diagonal. ADU is of dimension N,KP where N equals the dimension of A and KP is the number of non-zero diagonals in the upper triangle of A. Obviously, the diagonals of A are then mapped onto columns of ADU while rows are mapped onto rows. The same applies to ADL and the lower triangle of A. It is up to the user to pack the problem matrix A into the form required by the generated solver program. The generator program

itself on the other hand does not work on A directly and requires only information about the band and block structure of A. The task of the generator is to write a complete solver package SOLIC in Fortran which contains the implementation of the ICCG algorithm (2).

The solver is aimed principally at physics problems solved by finite difference methods on rectangular meshes. Hence all the non-zero diagonals are assumed to be full. However, some boundary conditions can produce non-zero diagonals which are very sparse but with regular patterns of non-zero elements, resulting in a block structure within A. While more general ways of specifying sparsity patterns will be provided in the future code generator a very simple optional device is provided in GENIC. Assume that all the elements of A are zero: beyond the column JLIM for the first IR rows, beyond the column JLIM + IR for the next IR rows and so on throughout A, and that this pattern is symmetric. That can be communicated to GENIC by simply giving IR and JLIM actual values through the namelist DIMS. If such a structure does not exist JLIM must be set equal to N.

The generator program performs the following sub-tasks:

1. Computation of the width of matrix bands and the actual array indices from the input information.
2. Output of the contents of the main solver subroutine SOLIC.
3. Output of subroutine SETLU which copies the contents of the two parts of A, ADL and ADU, into the workspace for decomposition and iteration: LD and UD.

4. Generation and output of LUDCMP which performs incomplete L-U decomposition.
5. Generation and output of ADOTV and ATDOTV which performs matrix-vector multiplication.
6. Generation and output of LLTINV and UTUINV which solve two subsidiary triangular systems.

The first of the sub-tasks is performed by the main program while 2-6 are carried out by the subroutines OUMAIN, SETLU, GENLU, OUTAV, OUTATV, OUTLLT, and OUTUTU used in an uninterruped sequence of calls in the order listed. A call to OUTUTU ends the program.

5. Description of Data Input to the Generator Program GENIC

When inputing data for GENIC, the user has to concern himself only with the structure of the full system of N equations which in vector form reads $\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{b}$. Take first the full matrix A and number its rows and columns 1 to N in the usual way starting from the upper left hand corner. Next imagine a splitting of A into a lower triangular matrix ADL and the upper triangular ADU, with both ADL and ADU containing the main diagonal, and number the diagonals of ADL by the row number of their first column elements. Also number the diagonals of ADU by the column number of their first row elements. The numbering is therefore in both cases from the main diagonal which bears the number 1. One is now ready to define the non-zero sub-diagonals. (A non-zero sub-diagonal must contain at least one non-zero element.) It is quite possible to provide GENIC with the numbers of all the non-zero elements but since

they are usually clustered in bands, it is sufficient to provide only the starts and finishes of the bands by giving the ordinal numbers of sub-diagonals at each side of every band.

Data input is through two namelists:

NAMELIST/DIMS/ NOUT, NOUTPP, N, JLIM, IR

NAMELIST/ABANDS/ IDFAUT, AUS, AUF, ALS, ALF, US, UF, LS, LF.

The user has to do the following:

1. Set the input channel number for reading the namelists by changing the first executable statement of GENIC which defines NIN.
2. Read via DIMS:
   a. the output channel numbers NOUT for GENIC and NOUTPP for SOLIC, and
   b. the system dimension N and the already described block structure parameters JLIM and IR. If there is no block structure JLIM and IR should be made equal to N.
3. Use namelist ABANDS to define the end of the first non-zero band of ADU starting from the main diagonal by setting AUF (1). AUS (1) being the start of the first band is automatically assumed equal to 1. Then define the limits of the second band through AUS (2) and AUF (2) and likewise for any additional bands. Note that one must always have

AUS(L) $\leq$ AUF(L), the equality sign corresponding to a degenerate band consisting of a single sub-diagonal.

The bands of ADL, UD and LD have to be defined only if they are different from those of ADU. The following rules must be observed:

4. Band pattern of A symmetric:
   a. pattern of LU the same as that of A: set IDFAUT = 1 , define elements of AUS and AUF only.
   b. pattern of LU different from A but symmetric: set IDFAUT = 3, and define elements of AUS, AUF, US and UF only.
5. Band pattern of A non-symmetric:
   a. pattern of LU the same as that of A: set IDFAUT = 2, and set elements of ALS and ALF as well as AUS and AUF.
   b. pattern of LU different from A: set IDFAUT = 4, and set elements of AUS, AUF, ALS, ALF, US, UF, LS, LF.

6. Output of the Generator Program GENIC

The output of GENIC is the solver program package SOLIC, in Fortran, which consists of the principal subroutine of the same name and six subsidiary subroutines: SETLU, LUDCMP, ADOTV, ATDOTV, LLTINV and UTUINV. Their combined purpose is to provide a linear system solver based on the CG algorithm (2), the

individual subroutines performing the following tasks: SOLIC contains the main body of the algorithm which it carries out with the help of subsidiary subroutines all of which are called only from SOLIC.

Having received the problem matrix A in the compacted form ADU and ADL, the right hand side $\underset{\sim}{b}$ and the trial vector $\underset{\sim}{x}_o$ for $A\underset{\sim}{x} = \underset{\sim}{b}$, SOLIC first copies the contents of ADU and ADL into UD and LD by calling SETLU. It then calls LUDCMP which performs the incomplete decomposition within UD and LD after which the algorithm (2) is applied. Several specialized tasks with the algorithm are performed by the subroutines: Operations by A and A^T (transpose of A) on any vector Y are performed by calling ADOTV (result, Y) and ATDOTV (result, Y) respectively. They correspond to sections (2:3) and (2:7) of the algorithm. Similarly the triangular systems (2:8) and (2:11) are solved with the help of LLTINV and UTUINV. The value of NOUT, which is input through the namelist DIMS into GENIC, determines the medium on which SOLIC is to appear.

7. Data Input for the Solver Program SOLIC

The user interface is quite a simple one and consists of two parameters of subroutine SOLIC (ISTOP, CONVEG) and a labelled common block: COMMON/IC/ADU (D1, D2), ADL (D1, D2), XICCG (D1), RHS (D1). The meaning of the two parameters are:

1. Integar ISTOP = the maximum number of iterations to be performed before passing control from SOLIC to the calling sub-program.
2. Real CONVEG = specification of the convergence

criterion. SOLIC returns control to the calling sub-program when $\| \underset{\sim}{r}_i \| / \| \underset{\sim}{x}_i \|$, where $\underset{\sim}{r}_i = b - A\underset{\sim}{x}_i$, becomes less or equal to CONVEG.

The labelled common block COMMON/IC/ should be included in the calling program and all elements of the four arrays in it are to be set. RHS must equal the right hand side vector $\underset{\sim}{b}$ of (1) and XICCG the trial vector $\underset{\sim}{x}_o$ of (2:1). ADU and ADL must contain the problem matrix A of (1) in a compacted form described earlier. It is important to note that A must be normalized so that all main diagonal elements are equal to 1. The symbolic dimension D1 has the meaning of row and D2 the diagonal of the A matrix. D1 and D2 will appear as absolute constants in SOLIC. The user must set ADU (I,KP) = A (I,KP) where ADU contains the upper triangular half of A; here I is the row index of A and KP is the packed diagonal index K of A. KP is determined from K simply by counting only the non-zero sub-diagonals sequentially starting from the main diagonal which is counted as 1. Similarly ADL (I,KP) = A (I,KP) where ADL contains the lower triangular portion of A. Again KP is an index of the non-zero sub-diagonals starting with the main diagonal which is counted as 1. Since A is assumed normalized, there is no need to copy into ADU (I,1) and ADL (I,1). The space taken up by these locations are later equivalenced to working arrays in SOLIC.

8. Output of the Solver Program SOLIC

SOLIC outputs only a small amount of diagonstic messages on an output channel defined by NOUTPP, a parameter read in through the namelist DIMS. The diagonstic output contains the following information:

1. ITSTEP = the actual number of iterations performed in the present call to SOLIC.
2. XNORM = $\| \underset{\sim}{x}_i \|$
3. RNORM = $\| \underset{\sim}{r}_i \|$
4. SIRI = $(\underset{\sim}{s}_i , \underset{\sim}{r}_i)$. This is a measure of orthogonality of $\underset{\sim}{s}_i$ and $\underset{\sim}{r}_i$. Though neither vector is normalized, this is still a good indication of the proper functioning of the code. SIRI should be normally around 10^{-10} .

The solution of the system (1) is returned in XICCG and the vector $\underset{\sim}{b}$ is overwritten by the residue $\underset{\sim}{r}_i$.

9. Test Case Input and Output

The sample output of GENIC as shown in this paper has been produced with an input as follows:

```
$DIMS  NOUT=3,NOUTPP=6,N=600,JLIM=30,IR=15$
$ABANDS IDFAUT=3, AUS=1,15, AUF=2,17, US=1,13, UF=2,17 $ END
```

This input corresponds to a linear-system arising from a 9 point discretization of a partial differential equation on a rectangular domain with aperiodic boundary conditions. After considerable optimization on the time required to reach a certain convergence ratio, it was found that adding 2 more sub-diagonals on the inside

of the side bands in UD and LD reduced the number of iterations required from 28 to 18 with a consequent saving in time of 40%. This gain is the result of a more complete L-U decomposition and naturally varies with the actual coefficients of the problem matrix A. The program stands to gain a great deal from the vector processing capabilities of modern computers. Subroutines SOLIC, ADOTV and ATDOTV can be completely vectored while subroutines LUDCMP, LLTINV, UTUINV, because of the recursive nature of the expressions, cannot. Although the vector capabilities of the CDC 7600 are limited, reprogramming the subroutines SOLIC, ADOTV and ATDOTV to make use of it [7] has yielded a further gain in speed of a factor of 2.

ACKNOWLEGMENTS

We would like to thank Dr. B. McNamara for useful discussions.

REFERENCES

[1] J. A. Meijerink and H. A. Van Der Vorst, "An Iterative Solution Method for Linear Systems of which the Coefficient Matrix is a Symmetric M-matrix," Technical report TR-1, Academic Computer Centre, Budapestlaan 6, de Uithof-UTRECHT, The Netherlands (1976).

[2] M. R. Hestenes and E. Stiefel, Journal of Research of National Bureau of Standards, 49 (1952) 409.

[3] M. R. Hestenes, "Process of Symposia on Applied Math, VI Numerical Analysis," (McGraw-Hill, New York 1956).

[4] J. K. Reid, "On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations," Proceeds of Conference, "Large Sparse Systems of Linear Equations," Academic Press, New York (1971).

[5] P. Concus, G. H. Golub, D. P. O'Leary, "A Generalized Conjugate Gradient Mehtod for the Numerical Solution of Elliptic Partial Differential Equations," Lawrence Berkeley Laboratory, Publication LBL-4604, Berkeley, CA, (1975).

[6] D. S. Kershaw, Journal of Computational Physics, in press, (1978) or UCRL, preprint - 78333.

[7] F. H. McMahon, L. J. Sloan and G. A. Long, "Stacklib - A Vector Function Library of Optimum Stack-loops for the DCD 7600," Lawrence Livermore Laboratory, Publication UCID - 30083, Livermore, CA, (1976).

```
C  THIS PROGRAM PACKAGE SOLIC USES THE FOLLOWING
C  PARAMETERS WHICH HAVE BEEN INPUT:
C  IDFAUT= 3  NOUT= 3    NOUTPP= 6     N=  600    JLIM=   30     IR=   15
C  AUS(I)=     1    15      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
C  AUF(I)=     2    17      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
C  ALS(I)=     1    15      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
C  ALF(I)=     2    17      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
C  US(I)=      1    13      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
C  UF(I)=      2    17      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
C  LS(I)=      1    13      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
C  LF(I)=      2    17      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
C-------------------------------------------------------------
C-------------------------------------------------------------
      SUBOUTINE SOLIC(ISTOP,CONVEG)
      COMMON/IC/ADU( 600, 5),ADL( 600, 5),XICCG( 600),RHS( 600)
      COMMON/UU/UD( 600, 7)
      COMMON/LL/LD( 600, 7)
      REAL LD
C  THIS SUBROUTINE IS CALLED BY THE PROBLEM PROGRAM. IT
C  CONTAINS THE MAIN CONJUGATE GRADIENT LOOP
      DIMENSION RI( 600),GI( 600),VTEMP( 600),SI( 600),PI( 600)
      EQUIVALENCE (VTEMP(1),ADL(1,1)),(SI(1),ADU(1,1)),(GI(1),LD(1,1))
     . ,(RI(1),RHS(1))
      CALL SETLU
      ITSTEP=1
      CALL LUDCMP
C  FORM RESIDUAL VECTOR AND STORE IN RI
      CALL ADOTV(VTEMP,XICCG)
      DO 10 M=1, 600
   10 RI(M)=RHS(M)-VTEMP(M)
C
C
C  FORM GRADIENT VECTOR
C  SI=(L.LT)**-1.RI
      CALL LLTINV(SI,RI,VTEMP)
C  GI=AT.SI
      CALL ATDOTV(GI,SI)
C  PI=(UT.U)**-1.GI
      CALL UTUINV(PI,GI,VTEMP)
C
C
C  CALCULATE ALFAI
      ANUM=0
      DO 20 M=1, 600
   20 ANUM=ANUM+RI(M)*SI(M)
  100 CONTINUE
      ADENOM=0
      DO 30 M=1, 600
   30 ADENOM=ADENOM+PI(M)*GI(M)
      ALFAI=ANUM/ADENOM
C
C
      DO 40 M=1, 600
   40 XICCG(M)=XICCG(M)+ALFAI*PI(M)
C  FIND NEW RESIDUE
      CALL ADOTV(VTEMP,PI)
      DO 50 M=1, 600
   50 RI(M)=RI(M)-ALFAI*VTEMP(M)
C
C
```

```
C  CHECK FOR CONVERGENCE
      RNORM=0.0
      XNORM=0.0
      DO 60 M=1, 600
      RNORM=RNORM+RHS(M)*RHS(M)
      XNORM=XNORM+XICCG(M)*XICCG(M)
   60 CONTINUE
      RATIO=SQRT(RNORM/XNORM)
C**********************
C  THIS SECTION DEMARCATED BY ********SERVE FOR DIAGNOSTIC
C  PURPOSES ONLY AND CAN BE DELETED FOR PRODUCTION RUNS
      XNORM1=SQRT(XNORM)
      RNORM1=SQRT(RNORM)
      SIRI=0.0
      DO 70 M=1, 600
   70 SIRI=SIRI+RI(M)*SI(M)
      WRITE(NOUTPP,101) ITSTEP,XNORM1,RNORM1,RATIO,SIRI
C**********************************************
      IF (RATIO.LE.CONVEG) GO TO 1000
      IF (ITSTEP.EQ.ISTOP) GO TO 1000
      ITSTEP=ITSTEP+1
C
C
C  CALCULATE SI
      CALL LLTINV(SI,RI,VTEMP)
      BNUM=0
      DO 80 M=1, 600
   80 BNUM=BNUM+RI(M)*RI(M)*SI(M)
      BETAI=BNUM/ANUM
      ANUM=BNUM
C
C
      CALL ATDOTV(VTEMP,SI)
      DO 90 M=1, 600
   90 GI(M)=VTEMP(M)+BETAI*GI(M)
      CALL UTUINV(PI,GI,VTEMP)
      GO TO 100
  101 FORMAT(1H ,"ITSTEP=",I3," XNORM=",E8.2,
     . " RNORM=",E8.2," RATIO=",E9.2," SIRI=",E9.2
 1000 RETURN
      END
C-------------------------------------------------------
C-------------------------------------------------------
      SUBROUTINE SETLU
      COMMON/IC/ADU( 600, 5),ADL( 600, 5),XICCG( 600),RHS( 600)
      COMMON/UU/UD( 600, 7)
      COMMON/LL/LD( 600, 7)
      REAL LD
C  THIS SUBROUTINE SETS UD AND LD TO THE VALUES GIVEN
C  IN ADU AND ADL BY THE USER
      DO 10 J=1, 7
      DO 10 M=1, 600
      UD(M,J)=0.0
   10 CONTINUE
      DO 20 J=1, 7
      DO 20 M=1, 600
      LD(M,J)=0.0
   20 CONTINUE
      DO 30 M=1, 600
      UD(M,1)=1.0
```

```
      UD(M, 2)=ADU(M, 2)
   30 CONTINUE
C
C
      DO 40 M=1, 600
      LD(M, 2)=ADL(M, 2)
   40 CONTINUE
C
C
      DO 50 J= 3, 5
      DO 50 M=1, 600
      UD(M,J+ 2)=ADU(M,J)
   50 CONTINUE
C
C
      DO 60 J= 3, 5
      DO 60 M=1, 600
      LD(M,J+ 2)=ADL(M,J)
   60 CONTINUE
       RETURN
       END
C-------------------------------------------------
C-------------------------------------------------
      SUBROUTINE LUDCMP
      COMMON/IC/ADU( 600, 5),ADL( 600, 5),XICCG( 600),RHS( 600)
      COMMON/UU/UD( 600, 7)
      COMMON/LL/LD( 600, 7)
      REAL LD
C  THIS SUBROUTINE PERFORMS THE INCOMPLETE L-U DECOMP-
C  OSITION OF THE ORIGINAL MATRIX A(ADU AND ADL)
C  RESULTS OF THE DECOMPOSITION ARE STORED IN UD AND LD
      PIVOTL=1.0E-10
      QPIVOT=SQRT(PIVOTL)
      DO  1 JMIN=   1, 571, 15
      J1=JMIN+    0
      J2=JMIN+   13
      DO 10 J=J1,J2
      IF(ABS(UD(J,1)).LT.PIVOTL
     . UD(J,1)=SIGN(QPIVOT,UD(J,1))
      RUDJJ=1.0/UD(J,1)
      LD(J+  1, 2)=LD(J+  1, 2)*RUDJJ
      R=-LD(J+  1, 2)
      UD(J+  1, 1)=UD(J+  1, 1)+R*UD(J, 2)
      UD(J+  1, 3)=UD(J+  1, 3)+R*UD(J, 4)
      UD(J+  1, 4)=UD(J+  1, 4)+R*UD(J, 5)
      UD(J+  1, 5)=UD(J+  1, 5)+R*UD(J, 6)
      UD(J+  1, 6)=UD(J+  1, 6)+R*UD(J, 7)
      LD(J+ 12, 3)=LD(J+ 12, 3)*RUDJJ
      R=-LD(J+ 12, 3)
      UD(J+ 12, 1)=UD(J+ 12, 1)+R*UD(J, 3)
      UD(J+ 12, 2)=UD(J+ 12, 2)+R*UD(J, 4)
      LD(J+ 13, 4)=LD(J+ 13, 4)*RUDJJ
      R=-LD(J+ 13, 4)
      LD(J+ 13, 3)=LD(J+ 13, 3)+R*UD(J, 2)
      LD(J+ 13, 2)=LD(J+ 13, 2)+R*UD(J, 3)
      UD(J+ 13, 1)=UD(J+ 13, 1)+R*UD(J, 4)
      UD(J+ 13, 2)=UD(J+ 13, 2)+R*UD(J, 5)
      LD(J+ 14, 5)=LD(J+ 14, 5)*RUDJJ
      R=-LD(J+ 14, 5)
      LD(J+ 14, 4)=LD(J+ 14, 4)+R*UD(J, 2)
```

```
      LD(J+ 14, 2)=LD(J+ 14, 2)+R*UD(J, 4)
      UD(J+ 14, 1)=UD(J+ 14, 1)+R*UD(J, 5)
      UD(J+ 14, 2)=UD(J+ 14, 2)+R*UD(J, 6)
      LD(J+ 15, 6)=LD(J+ 15, 6)*RUDJJ
      R=-LD(J+ 15, 6)
      LD(J+ 15, 5)=LD(J+ 15, 5)+R*UD(J, 2)
      LD(J+ 15, 2)=LD(J+ 15, 2)+R*UD(J, 5)
      UD(J+ 15, 1)=UD(J+ 15, 1)+R*UD(J, 6)
      UD(J+ 15, 2)=UD(J+ 15, 2)+R*UD(J, 7)
      LD(J+ 16, 7)=LD(J+ 16, 7)*RUDJJ
      R=-LD(J+ 16, 7)
      LD(J+ 16, 6)=LD(J+ 16, 6)+R*UD(J, 2)
      LD(J+ 16, 2)=LD(J+ 16, 2)+R*UD(J, 6)
      UD(J+ 16, 1)=UD(J+ 16, 1)+R*UD(J, 7)
   10 CONTINUE
      J=JMIN+   14
      IF(ABS(UD(J,1)).LT.PIVOTL
     . UD(J,1)=SIGN(QPIVOT,UD(J,1))
      RUDJJ=1.0/UD(J,1)
      LD(J+  1, 2)=LD(J+  1, 2)*RUDJJ
      R=-LD(J+  1, 2)
      UD(J+  1, 1)=UD(J+  1, 1)+R*UD(J, 2)
      UD(J+  1, 3)=UD(J+  1, 3)+R*UD(J, 4)
      UD(J+  1, 4)=UD(J+  1, 4)+R*UD(J, 5)
      UD(J+  1, 5)=UD(J+  1, 5)+R*UD(J, 6)
      LD(J+ 12, 3)=LD(J+ 12, 3)*RUDJJ
      R=-LD(J+ 12, 3)
      UD(J+ 12, 1)=UD(J+ 12, 1)+R*UD(J, 3)
      UD(J+ 12, 2)=UD(J+ 12, 2)+R*UD(J, 4)
      LD(J+ 13, 4)=LD(J+ 13, 4)*RUDJJ
      R=-LD(J+ 13, 4)
      LD(J+ 13, 3)=LD(J+ 13, 3)+R*UD(J, 2)
      LD(J+ 13, 2)=LD(J+ 13, 2)+R*UD(J, 3)
      UD(J+ 13, 1)=UD(J+ 13, 1)+R*UD(J, 4)
      UD(J+ 13, 2)=UD(J+ 13, 2)+R*UD(J, 5)
      LD(J+ 14, 5)=LD(J+ 14, 5)*RUDJJ
      R=-LD(J+ 14, 5)
      LD(J+ 14, 4)=LD(J+ 14, 4)+R*UD(J, 2)
      LD(J+ 14, 2)=LD(J+ 14, 2)+R*UD(J, 4)
      UD(J+ 14, 1)=UD(J+ 14, 1)+R*UD(J, 5)
      UD(J+ 14, 2)=UD(J+ 14, 2)+R*UD(J, 6)
      LD(J+ 15, 6)=LD(J+ 15, 6)*RUDJJ
      R=-LD(J+ 15, 6)
      LD(J+ 15, 5)=LD(J+ 15, 5)+R*UD(J, 2)
      LD(J+ 15, 2)=LD(J+ 15, 2)+R*UD(J, 5)
      UD(J+ 15, 1)=UD(J+ 15, 1)+R*UD(J, 6)
    1 CONTINUE
C  GENERATION COMPLETED FOR BLOCK NO  1
      JMIN=  566
      J=JMIN+    0
      IF(ABS(UD(J,1)).LT.PIVOTL
     . UD(J,1)=SIGN(QPIVOT,UD(J,1))
      RUDJJ=1.0/UD(J,1)
      LD(J+  1, 2)=LD(J+  1, 2)*RUDJJ
      R=-LD(J+  1, 2)
      UD(J+  1, 1)=UD(J+  1, 1)+R*UD(J, 2)
      UD(J+  1, 3)=UD(J+  1, 3)+R*UD(J, 4)
      UD(J+  1, 4)=UD(J+  1, 4)+R*UD(J, 5)
      LD(J+ 12, 3)=LD(J+ 12, 3)*RUDJJ
      R=-LD(J+ 12, 3)
```

```
      UD(J+ 12, 1)=UD(J+ 12, 1)+R*UD(J, 3)
      UD(J+ 12, 2)=UD(J+ 12, 2)+R*UD(J, 4)
      LD(J+ 13, 4)=LD(J+ 13, 4)*RUDJJ
      R=-LD(J+ 13, 4)
      LD(J+ 13, 3)=LD(J+ 13, 3)+R*UD(J, 2)
      LD(J+ 13, 2)=LD(J+ 13, 2)+R*UD(J, 3)
      UD(J+ 13, 1)=UD(J+ 13, 1)+R*UD(J, 4)
      UD(J+ 13, 2)=UD(J+ 13, 2)+R*UD(J, 5)
      LD(J+ 14, 5)=LD(J+ 14, 5)*RUDJJ
      R=-LD(J+ 14, 5)
      LD(J+ 14, 4)=LD(J+ 14, 4)+R*UD(J, 2)
      LD(J+ 14, 2)=LD(J+ 14, 2)+R*UD(J, 4)
      UD(J+ 14, 1)=UD(J+ 14, 1)+R*UD(J, 5)
      J=JMIN+    1
      IF(ABS(UD(J,1)).LT.PIVOTL
     . UD(J,1)=SIGN(QPIVOT,UD(J,1))
      RUDJJ=1.0/UD(J,1)
      LD(J+  1, 2)=LD(J+  1, 2)*RUDJJ
      R=-LD(J+  1, 2)
      UD(J+  1, 1)=UD(J+  1, 1)+R*UD(J, 2)
      UD(J+  1, 3)=UD(J+  1, 3)+R*UD(J, 4)
      LD(J+ 12, 3)=LD(J+ 12, 3)*RUDJJ
      R=-LD(J+ 12, 3)
      UD(J+ 12, 1)=UD(J+ 12, 1)+R*UD(J, 3)
      UD(J+ 12, 2)=UD(J+ 12, 2)+R*UD(J, 4)
      LD(J+ 13, 4)=LD(J+ 13, 4)*RUDJJ
      R=-LD(J+ 13, 4)
      LD(J+ 13, 3)=LD(J+ 13, 3)+R*UD(J, 2)
      LD(J+ 13, 2)=LD(J+ 13, 2)+R*UD(J, 3)
      UD(J+ 13, 1)=UD(J+ 13, 1)+R*UD(J, 4)
      J=JMIN+    2
      IF(ABS(UD(J,1)).LT.PIVOTL
     . UD(J,1)=SIGN(QPIVOT,UD(J,1))
      RUDJJ=1.0/UD(J,1)
      LD(J+  1, 2)=LD(J+  1, 2)*RUDJJ
      R=-LD(J+  1, 2)
      UD(J+  1, 1)=UD(J+  1, 1)+R*UD(J, 2)
      LD(J+ 12, 3)=LD(J+ 12, 3)*RUDJJ
      R=-LD(J+ 12, 3)
      UD(J+ 12, 1)=UD(J+ 12, 1)+R*UD(J, 3)
      J1=JMIN+    3
      J2=JMIN+   13
      DO 20 J=J1,J2
      IF(ABS(UD(J,1)).LT.PIVOTL
     . UD(J,1)=SIGN(QPIVOT,UD(J,1))
      RUDJJ=1.0/UD(J,1)
      LD(J+  1, 2)=LD(J+  1, 2)*RUDJJ
      R=-LD(J+  1, 2)
      UD(J+  1, 1)=UD(J+  1, 1)+R*UD(J, 2)
   20 CONTINUE
    2 CONTINUE
C  GENERATION COMPLETED FOR BLOCK NO  2
      RETURN
       END
C-------------------------------------------------------
C-------------------------------------------------------
      SUBROUTINE ADOTV(RESULT,COLVEC)
      COMMON/IC/ADU( 600, 5),ADL( 600, 5),XICCG( 600),RHS( 600)
      COMMON/UU/UD( 600, 7)
      COMMON/LL/LD( 600, 7)
```

```
      REAL LD
       DIMENSION RESULT(1),COLVEC(1)
C  THIS SUBROUTINE PERFORMS A DOT PRODUCT OF MATRIX A
C  WITH A COLUMN VECTOR
C  COMPUTE RESULT=A.COLVEC
      DO  9 M=1,600
    9 RESULT(M)=COLVEC(M)
C
C
      K= 2
      IULIM=601-K
      KP=K
      DO 10 M=1,IULIM
      RESULT(M)=RESULT(M)+ADU(M,KP)*COLVEC(K-1+M)
   10 CONTINUE
C
C
      DO 20 K= 15, 17
      IULIM=601-K
      KP=KP+1
      DO 20 M=1,IULIM
      RESULT(M)=RESULT(M)+ADU(M,KP)*COLVEC(K-1+M)
   20 CONTINUE
C
C
      K= 2
      ILLIM=K
      KP=K
      DO 30 M=ILLIM,600
      RESULT(M)=RESULT(M)+ADL(M,KP)*COLVEC(M-K+1)
   30 CONTINUE
C
C
      DO 40 K= 15, 17
      ILLIM=K
      KP=KP+1
      DO 40 M=ILLIM,600
      RESULT(M)=RESULT(M)+ADL(M,KP)*COLVEC(M-K+1)
   40 CONTINUE
       RETURN
       END
C-------------------------------------------------------
C-------------------------------------------------------
      SUBROUTINE ATDOTV(RESULT,COLVEC)
      COMMON/IC/ADU( 600, 5),ADL( 600, 5),XICCG( 600),RHS( 600)
      COMMON/UU/UD( 600, 7)
      COMMON/LL/LD( 600, 7)
      REAL LD
       DIMENSION RESULT(1),COLVEC(1)
C  THIS SUBROUTINE PERFORMS A DOT PRODUCT OF THE TRANSPOSE OF
C  THE MATRIX A WITH A COLUMN VECTOR
C  COMPUTE RESULT=AT.COLVEC
      DO  9 M=1,600
    9 RESULT(M)=COLVEC(M)
      K= 2
      ILLIM=K
      KP=K
      DO 10 M=ILLIM,600
      RESULT(M)=RESULT(M)+ADU(M-K+1,KP)*COLVEC(M-K+1)
   10 CONTINUE
```

```
C
C
      DO 20 K= 15, 17
      ILLIM=K
      KP=KP+1
      DO 20 M=ILLIM,600
      RESULT(M)=RESULT(M)+ADU(M-K+1,KP)*COLVEC(M-K+1)
   20 CONTINUE
C
C
      K= 2
      IULIM=601-K
      KP=K
      DO 30 M=1,IULIM
      RESULT(M)=RESULT(M)+ADL(M+K-1,KP)*COLVEC(M+K-1)
   30 CONTINUE
C
C
      DO 40 K= 15, 17
      IULIM=601-K
      KP=KP+1
      DO 40 M=1,IULIM
      RESULT(M)=RESULT(M)+ADL(M+K-1,KP)*COLVEC(M+K-1)
   40 CONTINUE
       RETURN
       END
C-------------------------------------------------
C-------------------------------------------------
      SUBROUTINE LLTINV(RESULT,COLVEC,AUX)
      COMMON/IC/ADU( 600, 5),ADL( 600, 5),XICCG( 600),RHS( 600)
      COMMON/UU/UD( 600, 7)
      COMMON/LL/LD( 600, 7)
      REAL LD
       DIMENSION RESULT(1),COLVEC(1),AUX(1)
C  THIS SUBROUTINE PERFORMS THE OPERATION (L.L(TRANSPOSED))**-1*C  A COLUMN VECTOR
C  COMPUTE RESULT=(L.LT)**-1*COLVEC
C  FIRST AUX=L**-1*COLVEC
C  THEN  RESULT=LT**-1*AUX
C  SECTION A
      AUX(  1)=COLVEC(  1)
C  SECTION B
      DO 10 M=  2, 12
   10 AUX(M)=COLVEC(M)
     .              -LD(M, 2)*AUX(M-  1)
C  SECTION C:  BAND NUMBER IBAND= 2
      AUX( 13)=COLVEC( 13)
     .              -LD( 13, 2)*AUX( 12)
     .              -LD( 13, 3)*AUX(  1)
      AUX( 14)=COLVEC( 14)
     .              -LD( 14, 2)*AUX( 13)
     .              -LD( 14, 3)*AUX(  2)
     .              -LD( 14, 4)*AUX(  1)
      AUX( 15)=COLVEC( 15)
     .              -LD( 15, 2)*AUX( 14)
     .              -LD( 15, 3)*AUX(  3)
     .              -LD( 15, 4)*AUX(  2)
     .              -LD( 15, 5)*AUX(  1)
      AUX( 16)=COLVEC( 16)
     .              -LD( 16, 2)*AUX( 15)
     .              -LD( 16, 3)*AUX(  4)
```

```
     .            -LD( 16, 4)*AUX(  3)
     .            -LD( 16, 5)*AUX(  2)
     .            -LD( 16, 6)*AUX(  1)
C  SECTION D:  BAND NUMBER IBAND= 2
      DO 20 M= 17,600
   20 AUX(M)=COLVEC(M)
     .            -LD(M, 2)*AUX(M-  1)
     .            -LD(M, 3)*AUX(M- 12)
     .            -LD(M, 4)*AUX(M- 13)
     .            -LD(M, 5)*AUX(M- 14)
     .            -LD(M, 6)*AUX(M- 15)
     .            -LD(M, 7)*AUX(M- 16)
C
C
C  NOW GENERATE BACKWARD SWEEP
C  SECTION E
      RESULT(600)=AUX(600)
C  SECTION F
      DO 40 M=  2, 12
      MR=601-M
   40 RESULT(MR)=AUX(MR)
     .            -LD(MR+  1, 2)*RESULT(MR+  1)
C  SECTION G:  BAND NUMBER IBAND= 2
      RESULT(588)=AUX(588)
     .            -LD(589, 2)*RESULT(589)
     .            -LD(600, 3)*RESULT(600)
      RESULT(587)=AUX(587)
     .            -LD(588, 2)*RESULT(588)
     .            -LD(599, 3)*RESULT(599)
     .            -LD(600, 4)*RESULT(600)
      RESULT(586)=AUX(586)
     .            -LD(587, 2)*RESULT(587)
     .            -LD(598, 3)*RESULT(598)
     .            -LD(599, 4)*RESULT(599)
     .            -LD(600, 5)*RESULT(600)
      RESULT(585)=AUX(585)
     .            -LD(586, 2)*RESULT(586)
     .            -LD(597, 3)*RESULT(597)
     .            -LD(598, 4)*RESULT(598)
     .            -LD(599, 5)*RESULT(599)
     .            -LD(600, 6)*RESULT(600)
C  SECTION H:  BAND NUMBER IBAND= 2
      DO 50 M= 17,600
      MR=601-M
   50 RESULT(MR)=AUX(MR)
     .            -LD(MR+  1, 2)*RESULT(MR+  1)
     .            -LD(MR+ 12, 3)*RESULT(MR+ 12)
     .            -LD(MR+ 13, 4)*RESULT(MR+ 13)
     .            -LD(MR+ 14, 5)*RESULT(MR+ 14)
     .            -LD(MR+ 15, 6)*RESULT(MR+ 15)
     .            -LD(MR+ 16, 7)*RESULT(MR+ 16)
      RETURN
      END
C-------------------------------------------------------
C-------------------------------------------------------
      SUBROUTINE UTUINV(RESULT,COLVEC,AUX)
      COMMON/IC/ADU( 600, 5),ADL( 600, 5),XICCG( 600),RHS( 600)
      COMMON/UU/UD( 500, 7)
      COMMON/LL/LD( 600, 7)
      REAL LD
```

```
      DIMENSION RESULT(1),COLVEC(1),AUX(1)
C  THIS SUBROUTINE PERFORMS THE OPERATION (U(TRANSPOSED)*U)**-1*C  A COLUMN VECTOR
C  COMPUTE RESULT=(UT.U)**-1*COLVEC
C  FIRST AUX=UT**-1*COLVEC
C  THEN  RESULT=U**-1*AUX
C  SECTION A
      AUX(  1)=(COLVEC(  1)
     .                )/UD(  1,1)
C  SECTION B
      DO 10 M=  2, 12
   10 AUX(M)=(COLVEC(M)
     .                -UD(M-  1, 2)*AUX(M-  1)
     .                )/UD(M,1)
C  SECTION C:  BAND NUMBER IBAND= 2
      AUX( 13)=(COLVEC( 13)
     .                -UD( 12, 2)*AUX( 12)
     .                -UD(  1, 3)*AUX(  1)
     .                )/UD( 13,1)
      AUX( 14)=(COLVEC( 14)
     .                -UD( 13, 2)*AUX( 13)
     .                -UD(  2, 3)*AUX(  2)
     .                -UD(  1, 4)*AUX(  1)
     .                )/UD( 14,1)
      AUX( 15)=(COLVEC( 15)
     .                -UD( 14, 2)*AUX( 14)
     .                -UD(  3, 3)*AUX(  3)
     .                -UD(  2, 4)*AUX(  2)
     .                -UD(  1, 5)*AUX(  1)
     .                )/UD( 15,1)
      AUX( 16)=(COLVEC( 16)
     .                -UD( 15, 2)*AUX( 15)
     .                -UD(  4, 3)*AUX(  4)
     .                -UD(  3, 4)*AUX(  3)
     .                -UD(  2, 5)*AUX(  2)
     .                -UD(  1, 6)*AUX(  1)
     .                )/UD( 16,1)
C  SECTION D:  BAND NUMBER IBAND= 2
      DO 20 M= 17,600
   20 AUX(M)=(COLVEC(M)
     .                -UD(M-  1, 2)*AUX(M-  1)
     .                -UD(M- 12, 3)*AUX(M- 12)
     .                -UD(M- 13, 4)*AUX(M- 13)
     .                -UD(M- 14, 5)*AUX(M- 14)
     .                -UD(M- 15, 6)*AUX(M- 15)
     .                -UD(M- 16, 7)*AUX(M- 16)
     .                )/UD(M,1)
C
C
C  NOW GENERATE BACKWARD SWEEP
C  SECTION E
      RESULT(600)=AUX(600)/UD(600,1)
C  SECTION F
      DO 40 M=  2, 12
      MR=601-M
   40 RESULT(MR)=(AUX(MR)
     .                -UD(MR, 2)*RESULT(MR+  1)
     .                )/UD(MR,1)
C  SECTION G:  BAND NUMBER IBAND= 2
      RESULT(588)=(AUX(588)
     .                -UD(588, 2)*RESULT(589)
```

```
     .              -UD(588, 3)*RESULT(600)
     .              )/UD(588,1)
      RESULT(587)=(AUX(587)
     .              -UD(587, 2)*RESULT(588)
     .              -UD(587, 3)*RESULT(599)
     .              -UD(587, 4)*RESULT(600)
     .              )/UD(587,1)
      RESULT(586)=(AUX(586)
     .              -UD(586, 2)*RESULT(587)
     .              -UD(586, 3)*RESULT(598)
     .              -UD(586, 4)*RESULT(599)
     .              -UD(586, 5)*RESULT(600)
     .              )/UD(586,1)
      RESULT(585)=(AUX(585)
     .              -UD(585, 2)*RESULT(586)
     .              -UD(585, 3)*RESULT(597)
     .              -UD(585, 4)*RESULT(596)
     .              -UD(585, 5)*RESULT(599)
     .              -UD(585, 6)*RESULT(600)
     .              )/UD(585,1)
C  SECTION H:  BAND NUMBER IBAND= 2
       DO 50 M= 17,600
       MR=601-M
   50  RESULT(MR)=(AUX(MR)
     .              -UD(MR, 2)*RESULT(MR+  1)
     .              -UD(MR, 3)*RESULT(MR+ 12)
     .              -UD(MR, 4)*RESULT(MR+ 13)
     .              -UD(MR, 5)*RESULT(MR+ 14)
     .              -UD(MR, 6)*RESULT(MR+ 15)
     .              -UD(MR, 7)*RESULT(MR+ 16)
     .              )/UD(MR,1)
       RETURN
       END
```