APMATH64 MANUAL

VOLUME 2 OF 4

MODELS M64/40,
M64/50, M64/60

860-7482-001C

FLOATING POINT SYSTEMS, INC.

APMATH64 MANUAL

VOLUME 2 OF 4

MODELS M64/40,
M64/50, M64/60

860-7482-001C

by FPS Technical Publications Staff

Publication No. 860-7482-001C
December, 1987

NOTICE

The information in this publication is subject to change without notice.

Floating Point Systems, Inc. accepts no liability for any loss, expense, or damage resulting from the use of any information appearing in this publication.

Printed in USA

The postpaid Reader's Comment Form on the last page of this document requests the user's critical evaluation to assist in preparing and revising future documents.

REVISION HISTORY

This manual is the *APMATH64 Manual*, Volume 2, 860-7482-001. The letter shown under the revision number column indicates the portion of the part number that changes for each revision. The last entry is the latest revision to this manual.

REV. NO.	DESCRIPTION	DATE
-001A	The revision history begins with this manual.	8/86
-001B	Deleted Utilities Library, deleted the LPSPFI subroutine, added internal subroutine information, and added 16 new routines.	1/87
-001C	Added new routines to Basic Math Library, Double Precision Library, and Matrix Algebra Accelerated Math Library.	12/87

NOTE: For revised manuals, a vertical line "|" outside the left margin of the text signifies where changes have been made.

NOTE TO READER

This is the second volume of the APMATH64 Manual. Volume 2 is comprised of part 2 of Appendix A. Note that Appendix A continues through Volumes 1, 2, and 3. The page numbers are listed consecutively through the volumes.

The APMATH64 Manual has three indices located at the end of Volume 3 and two at the end of Volume 4. The first index (Appendix I) is a list of the APMATH64 routines in page order by type. The second index (Appendix J) is an alphabetical list of all the APMATH64 routines. The third index is a key word index of the APMATH64 routines. The fourth index (Appendix L) is an alphabetical list of the APMATH64/MAX routines. The fifth index is a key word index of the APMATH64/MAX routines.

CONTENTS (VOLUME 2)

ILLUSTRATIONS

APMATH64 ROUTINES (VOLUME 2)
ADVANCED MATH LIBRARY

DESCRIPTION: This routine first calls HTRIDI to reduce A to a real symmetric tridiagonal matrix using unitary similarity transformations. IMTQL2 is then called to determine the eigenvalues and eigenvectors of the real tridiagonal matrix. IMTQL2 uses the implicit QL method to compute the eigenvalues and accumulates the QL transformations to compute the eigenvectors. Finally, HTRIBK is called to backtransform the eigenvectors to those of the original matrix.

If N is less than or equal to zero, then IERR is set to 999999. If N is greater than NM, then IERR is set to 10*N. If more than 30 iterations are required to determine an eigenvalue, the subroutine terminates with IERR set equal to the index of the eigenvalue for which the failure occurs. In this case, the eigenvalues in W should be correct for indices 1, 2,..., IERR-1, but no eigenvectors are computed. If all of the eigenvalues are determined within 30 iterations, then IERR is set to zero.

The function selector, MATZ, may be made functional in a future release as follows: If MATZ = 0, then only the eigenvalues will be determined; otherwise, both the eigenvalues and eigenvectors will be determined.

With the exception of error code 999999 and the nonfunctionality of the selector flag, this routine is functionally the same as the FORTRAN routine of the same name found in the "Matrix Eigensystem Routines - EISPACK Guide", 2nd edition, by B.T. Smith, et al., Springer-Verlag (1976). For further information, refer to pages 235-239 of the EISPACK Guide.

The execution time for this routine is highly data dependent.

EXAMPLE:

Input:

```
NM = 4
N  = 4

AR :   3.0     1.0     0.0     0.0
       1.0     3.0     0.0     0.0
       0.0     0.0     1.0     1.0
       0.0     0.0     1.0     1.0
```

```
**********                                                  **********
*        *                                                  *        *
* EIGRS  *  --- REAL SYMMETRIC EIGENSYSTEM SOLVER ---       *  EIGRS *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To determine eigenvalues and eigenvectors of a real symmetric matrix.

CALL FORMAT: CALL EIGRS(NM,N,A,D,E,Z,IERR)

PARAMETERS:
- NM = Integer row dimension of matrices A and Z
- N = Integer order of matrix (N .LE. NM)
- A = Floating-point input matrix
- D = Floating-point output vector (eigenvalues)
- E = Floating-point scratch vector
- Z = Floating-point output matrix (eigenvectors)
- IERR = Integer error flag set if routine does not converge within 30 iterations (refer to IMTQL2).

NOTE: The dimension of matrices A and Z is NM*N.
The dimension of matrices D and E is N.

DESCRIPTION: EIGRS first reduces the full matrix to tridiagonal form by Householder's method, diagonalizing the resulting matrix by the QL algorithm (using implicit origin shifts). The APAL subroutines used to accomplish this, TRED2 and IMTQL2, are based on the FORTRAN programs of the same name found in the "Matrix Eigensystem Routines - EISPACK Guide" by B.T. Smith et al., Springer-Verlag (1976).

EXAMPLE:

```
NM = 5
N  = 4

A : 5.Ø  4.Ø  1.Ø   1.Ø
    4.Ø  5.Ø  1.Ø   1.Ø
    1.Ø  1.Ø  4.Ø   2.Ø
    1.Ø  1.Ø  2.Ø   4.Ø
    Ø.Ø  Ø.Ø  Ø.Ø   Ø.Ø

D :  1.Ø  2.Ø  5.Ø  1Ø.Ø
```

```
**********                                                  **********
*        *                                                  *        *
* HTRIBK *      --- COMPLEX HERMITIAN EIGENVECTORS ---      * HTRIBK *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To form the eigenvectors of a complex Hermitian matrix, A, by back transforming those of the corresponding real symmetric tridiagonal matrix determined by the routine HTRIDI.

CALL FORMAT: CALL HTRIBK(NM, N, AR, AI, TAU, M, ZR, ZI)

PARAMETERS:

NM = Integer input scalar
Row dimension of the matrices

N = Integer input scalar
Order of matrix A and column dimension of the matrices. N must be less than or equal to NM.

AR = Floating-point NM by N input matrix
The strict lower triangle of the first N rows contains information about the unitary transformations used in the reduction by HTRIDI. The remaining elements are ignored.

AI = Floating-point NM by N input matrix
The full lower triangle of the first N rows contains information about the unitary transformations used in the reduction by HTRIDI. The remaining elements are ignored.

TAU = Floating-point 2 by N input matrix
Contains the remaining information about the unitary transformations.

M = Integer input scalar
Number of eigenvectors to be back transformed.

ZR = Floating-point NM by N input/output matrix
On input, the columns of ZR contain the eigenvectors to be back transformed in their first N elements. On output, the first M columns and N rows contain the real parts of the transformed eigenvectors.

ZI = Floating-point NM by N output matrix
The first M columns and N rows contain the imaginary parts of the transformed eigenvectors.

```
**********                                                  **********
*        *                                                  *        *
* HTRIDI * --- COMPLEX HERMITIAN TRIDIAGONALIZATION --- * HTRIDI *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To reduce a complex Hermitian matrix, A, to a real symmetric tridiagonal matrix using unitary similarity transformations.

CALL FORMAT: CALL HTRIDI(NM, N, AR, AI, D, E, E2, TAU)

PARAMETERS:

NM = Integer input scalar
Row dimension of the matrices

N = Integer input scalar
Order of matrix A and column dimension of the matrices . N must be less than or equal to NM.

AR = Floating-point NM by N input/output matrix
On input, the first N rows of AR contain the real parts of the elements of A. The last NM - N rows are ignored. Only the full lower triangle of AR need be supplied. On output, the strict lower triangle of AR contains information about the unitary transformations used in the reduction. The full upper triangle of AR is unaltered.

AI = Floating-point NM by N input/output matrix
On input, the first N rows of AI contain the imaginary parts of the elements of A. The last NM - N rows are ignored. Only the strict lower triangle of AI need be supplied. On output, the full lower triangle of AI contains information about the unitary transformations used in the reduction. The strict upper triangle of AI is unaltered.

D = Floating-point output vector of length N
Contains the diagonal elements of the tridiagonal matrix.

E = Floating-point output vector of length N
Contains the subdiagonal elements of the tridiagonal matrix in its last N-1 elements. The element E(1) is set to zero.

E2 = Floating-point output vector of length N
Contains the squares of the corresponding elements of E.

TAU = Floating-point 2 by N output matrix
Contains the remaining information about the unitary transformations.

********** **********
* IMTQL1 * --- DIAGONALIZE TRIDIAGONAL MATRIX --- * IMTQL1 *
********** **********

PURPOSE: To determine the eigenvalues of an N by N real symmetric tridiagonal matrix using the implicit QL method.

CALL FORMAT: CALL IMTQL1 (N, D, E, IERR)

PARAMETERS:
- N = Integer input order of the matrix
- D = Floating-point input/output vector
  Vector of length N containing the diagonal elements of the symmetric matrix on input; vector of length N containing the eigenvalues on output.
- E = Floating-point input vector
  Vector of length N containing the subdiagonal elements of the symmetric matrix. The subdiagonal is contained in elements E(2) through E(N); E(1) is arbitrary.
- IERR = Integer output error status
  - IERR = Ø: No errors encountered, normal completion.
  - IERR = -1: The routine received an invalid input argument, N < 1.
  - IERR > Ø: The routine was unable to finish because more than 3Ø iterations were required to determine an eigenvalue. IERR is set to the index of the offending eigenvalue. The eigenvalues in D are correct for all preceding indices, but are unordered.

DESCRIPTION: IMTQL1 determines the eigenvalues of a symmetric tridiagonal matrix using the QL algorithm with implicit origin shifts at each iteration.

Upon convergence, the eigenvalues are ordered in ascending order.

The vector E is destroyed by this routine.

IMTQL1 is based on the FORTRAN program found in the EISPACK GUIDE, 2nd ed., B.T. Smith, et al., Springer-Verlag, 1976. That program in turn is based on an Algol procedure discussed by Martin and Wilkinson, NUM. MATH.,12, 1968, pg. 377.

```
**********                                                **********
*        *                                                *        *
* IMTQL2 *   --- DIAGONALIZE A TRIDIAGONAL MATRIX ---     * IMTQL2 *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To determine eigenvalues and eigenvectors of a real symmetric tridiagonal matrix.

CALL FORMAT: CALL IMTQL2(NM,N,D,E,Z,IERR)

PARAMETERS:

NM = Integer row dimension of matrices A and Z

N = Integer order of matrix (N .LE. NM)

D = Floating-point input/output vector
Diagonal elements on input;
Eigenvalues in ascending order on output

E = Floating-point input vector
Codiagonal elements

Z = Floating-point input/output matrix
For eigenvectors of sym.tridiag. matrix:
Nth-order identity matrix on input;
Eigenvectors on output
For eigenvectors of full sys. matrix:
Trans.matrix from TRED2 on input;
Eigenvectors on output

IERR = Integer index of eigenvalue if convergence not obtained by 3Ø iterations, else Ø

NOTE: The dimension of arrays A and Z is NM*N.
The dimension of arrays D and E is N.

DESCRIPTION: IMTQL2 diagonalizes an N-by-N tridiagonal matrix using the implicit QL algorithm (Martin and Wilkinson, Num. Math. 12, 377(1968); Dubrulle, Num. Math. 15, 45Ø (197Ø)). The initial diagonal begins at D(1), and the codiagonal at E(2). At each iteration, a new tridiagonal matrix is formed, is stored by overwriting the previous result, and continues until convergence, or 3Ø iterations have passed. If convergence does not occur by 3Ø iterations, IERR is set equal to the index of the sought eigenvalue, and the routine is exited. Previously calculated results are valid. The transformation matrices are accumulated and the results stored in column order in matrix Z.

```
**********                                                **********
*        *                                                *        *
*   RS   *   --- REAL SYMMETRIC EIGENSYSTEM SOLVER ---    *   RS   *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To determine the eigenvalues and eigenvector of a real symmetric matrix, A.

CALL FORMAT: CALL RS(NM, N, A, W, MATZ, Z, FV1, FV2, IERR)

PARAMETERS:

NM = Integer input scalar
Number of rows of matrices A and Z

N = Integer input scalar
Order of matrix A and column dimension of matrices A and Z. N must be less than or equal to NM.

A = Floating-point NM by N input matrix
The first N rows contain the matrix and the last NM - N rows are ignored. Only the full lower triangle of the matrix need be supplied.

W = Floating-point output vector of length N
Contains the eigenvalues of A in ascending order.

MATZ = Integer input scalar
MATZ is not currently used.

Z = Floating-point NM by N output matrix
The first N elements of the j-th column of Z is the eigenvector that corresponds to the j-th eigenvalue in W. The last NM - N elements in each column are not altered.

FV1 = Floating-point work area vector of length N

FV2 = Floating-point work area vector of length N

IERR = Integer output scalar
Error code as described below.

DESCRIPTION: This routine first calls TRED2 to reduce A to a symmetric tridiagonal matrix using and accumulating orthogonal similarity transformations. IMTQL2 is then called to determine the eigenvalues and eigenvectors of the original matrix from the symmetric tridiagonal matrix. IMTQL2 uses the implicit QL method to compute the eigenvalues and accumulates the QL transformations to compute the eigenvectors.

If N is less than or equal to zero, then IERR is set to 999999. If N is greater than NM, then IERR is set to 10*N. If more than 30 iterations are required to determine an eigenvalue, the subroutine terminates with IERR set equal to the index of the eigenvalue for which the failure occurs.

```
**********                                              **********
*        *                                              *        *
* SIMPLE *          --- REVISED SIMPLEX ---             * SIMPLE *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To solve a linear programming problem that is in the standard form:

maximize Z = C'* X

subject to A * X = B
and X(j) >= Ø, for j = 1 to N

where B(i) >= Ø, for i = 1 to M

CALL FORMAT:
```
CALL SIMPLE(M,N,MP2,NP1,KI,NS,S,IRN,ICP,B,C,WRK,
            X,Y,Z,IB,KO)
```

PARAMETERS:

M = Integer input scalar
Number of constraints (rows in A).

N = Integer input scalar
Number of variables (columns in A).

MP2 = Integer input scalar
MP2 = M + 2

NP1 = Integer input scalar
NP1 = N + 1

KI = Integer input vector of length 1Ø
Contains the program control parameters. If any of these parameters is less than or equal to zero, then a default value is supplied for that parameter. The parameters are:

- KI(1) = Input basis flag. KI(1) > Ø indicates that an initial basis is supplied in IB. Default = No initial basis.
- KI(2) = Iteration limit. Default = 4 * N + 1Ø
- KI(3) = Inversion interval. Default = M/2 + 5
- KI(4) = Zero tolerance exponent. The zero tolerance value = Ø.5 ** KI(4). Default = 2Ø.
- KI(5) = Partial pricing step size. Default = min (N, max(2Ø,N/2Ø)). NOTE: The default value is also used if KI(5) > N and a value of 2Ø is used if Ø < KI(5) < 2Ø.
- KI(6) to KI(1Ø) are reserved for future use.

NS = Integer input scalar
Number of nonzero elements in A.

S = Floating-point input array of length NS
Contains the nonzero elements of A stored by columns.

IRN = Integer input array of length NS
Contains the row numbers (in A) that correspond to the nonzero elements in S.

The problem must be stated in the standard form:

```
maximize Z = C'* X

subject to   A * X = B
and          X(j) >= Ø, for j = 1 to N

where        B(i) >= Ø, for i = 1 to M
```

Therefore, it is the responsibility of the user to:

(a) Convert a minimization problem to a maximization problem by replacing C with -C.
(b) Convert inequality constraints to equality constraints by adding a slack variable or subtracting a surplus variable.
(c) Ensure that B(i) >= Ø by multiply the i-th constraint by -1.Ø if B(i) < Ø.
(d) Ensure that the decision variables are constrained to be nonnegative. If X(j) is unconstrained in sign then replace it by the difference of two new nonnegative variables.

In this variation of the two phase, revised simplex method, a composite problem is formed (virtually) in SIMPLE that includes both the actual (phase 2) objective equation and the artificial (phase 1) objective equation as constraints making a total of MP2 constraints. The variables for the internal composite problem are:

X(Ø) - The actual objective; i.e., Z
X(1) to X(N) - The actual decision variables
X(N+1) - The artificial objective
X(N+2) to X(N+M+1) - The artificial variables
where X(N+1+i) is the artificial variable for the i-th constraint.

The variables X(Ø) and X(N+1) to X(N+M+1) are virtual variables and, thus, do not use any storage space.

X(Ø) must always be a basic variable and IB(1) must always be zero. X(N+1) must be a basic variable during phase one and IB(2) must equal N+1 whenever X(N+1) is basic. At least one artificial variable (including X(N+1)) must always be basic. During phase two, any artificial variables in the basis will have a value of zero. Generally, during phase two, only one artificial variable will be basic and it will be X(N+1); however, this need not be the case.

EXAMPLE: Given a problem in standard form where

```
A: 1.  2.  3.  Ø.  Ø.  Ø.  Ø.  Ø.  1.  Ø.  Ø.  Ø.  Ø.
   Ø.  Ø.  Ø.  3.  1.  2.  Ø.  Ø.  Ø.  1.  Ø.  Ø.  Ø.
   2.  3.  Ø.  2.  Ø.  Ø.  2.  Ø.  Ø.  Ø.  1.  Ø.  Ø.
   Ø.  Ø.  3.  Ø.  5.  Ø.  2.  3.  Ø.  Ø.  Ø.  1.  Ø.
   3.  Ø.  Ø.  Ø.  Ø.  3.  Ø.  1.  Ø.  Ø.  Ø.  Ø.  1.
```

the inputs are:

```
M   = 5
N   = 13
MP2 = 7
NP1 = 14

KI  : Ø,  Ø,  Ø,  Ø

NS  = 22

S   : 1., 2., 3., 2., 3., 3., 3., 3., 2., 1., 5., 2.,
      3., 2., 2., 3., 1., 1., 1., 1., 1., 1.

IRN :  1,  3,  5,  1,  3,  1,  4,  2,  3,  2,  4,  2,
       5,  3,  4,  4,  5,  1,  2,  3,  4,  5

ICP :  1,  4,  6,  8, 1Ø, 12, 14, 16, 18, 19, 2Ø, 21,
      22, 23

B   : 14., 25., 21., 3Ø., 34.

C   :  9.,  9.,  4.,  8.,  7.,  6.,  8.,  6.,  Ø.,
       Ø.,  Ø.,  Ø.,  Ø.

IB  :  Don't care since KI(1) = Ø
```

The outputs are:

```
X   : Ø., 5., Ø., 3., Ø., 8., Ø., 1Ø., 4., Ø., Ø.,
      Ø., Ø.

Y   : Ø.ØØØØ, Ø.6667, 3.ØØØØ, 1.4815, 1.5556, y6, y7
      where y6 and y7 not of interest (scratch)

Z   = 177.Ø

IB  : Ø, 14,  9,  2,  8,  4,  6

KO  : Ø,  7,  7,  Ø,  7,  4
```

EXAMPLE:

```
A(INPUT)  =  13.ØØØ   4.ØØØ   1.ØØØ   9.ØØØ   7.ØØØ
             Ø.ØØØ  -3.ØØØ   8.ØØØ  -2.ØØØ  -7.ØØØ

A(OUTPUT) =  13.ØØØ   3.923   Ø.Ø77   9.ØØØ   4.7Ø6
             Ø.ØØØ  -Ø.765   1.7Ø6  -Ø.425  -Ø.778

V(INPUT)  =   Ø.ØØØ   1.ØØØ   Ø.ØØØ   Ø.5ØØ   Ø.ØØØ
MAXA      =   1       2       4       5       8       11
NN        =   5
MA        =   3
NWA       =  1Ø
KKK       =   3

V(OUTPUT) = -Ø.Ø43   Ø.563    Ø.245  Ø.4Ø3   Ø.315
```

EXAMPLE:

Input:

```
N  = 5
NM = 5

A :  -1.Ø   Ø.Ø   2.Ø   4.Ø   Ø.Ø
      Ø.Ø   2.Ø   3.Ø   Ø.Ø  -1.Ø
      2.Ø   3.Ø   5.Ø   Ø.Ø   Ø.Ø
      4.Ø   Ø.Ø   Ø.Ø  -2.Ø   Ø.Ø
      Ø.Ø  -1.Ø   Ø.Ø   Ø.Ø   1.Ø
```

Output:

```
D :  -2.Ø  -1.Ø   5.Ø   2.Ø   1.Ø

E :   Ø.Ø  -4.Ø  -2.Ø  -3.Ø  -1.Ø

E2:   Ø.Ø  16.Ø   4.Ø   9.Ø   1.Ø
```

```
**********                                                **********
*        *                                                *        *
* VASORT *      --- VECTOR SORT ALGEBRAIC VALUES ---      * VASORT *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To sort a vector into an ascending vector of algebraic values using Quicksort.

CALL FORMAT: CALL VASORT(A,I,N,W)

PARAMETERS:
A = Floating-point vector to be sorted in place
I = Integer element step for A
N = Integer element count
W = Floating-point vector of at most 2*log2(N) words of contiguous space for working stack of pointers

DESCRIPTION: VASORT sorts elements of a vector into an ascending vector of algebraic values by the method of Quicksort (Hoare's partition-exchange sort) in place. The procedure iteratively partitions the vector creating two subvectors, one whose values are less than or equal to the value initially at the middle location, and the other with elements greater than or equal to that value. This chosen value ends up in its true (post-sorted) position between the two subvectors. The half-way location was chosen for initial trial comparison in order to speed the sort when the original vector is already partly ordered.

After each partition, first and last locations of the larger subvector are stored in a pointer stack, which can accumulate no more than log2(N) pairs, and the process of partitioning is continued on the smaller subvector. The process of comparison and partitioning is continued until no subvectors remain. The vector is then completely sorted.

EXAMPLE:

N = 5

```
A(input)  :  5.Ø  4.Ø -Ø.5 -1.Ø  8.Ø
A(output) : -1.Ø -Ø.5  4.Ø  5.Ø  8.Ø
```

```
**********                                                **********
*        *                                                *        *
* VSORT  *        --- VECTOR SORT WITH INDICES ---        * VSORT  *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To sort a vector into an ascending vector of algebraic values using Quicksort. When the elements of the A vector are swapped, corresponding elements of the P vector are also swapped. Typical use of the P vector is to record the original indices of the sorted vector.

CALL FORMAT: CALL VSORT(A,I,P,J,N)

PARAMETERS:
A = Floating-point vector to be sorted in place
I = Integer element step for A
P = Integer or real vector of starting indices
J = Integer element step for P
N = Integer element count

DESCRIPTION: VSORT sorts elements of a vector into an ascending vector of algebraic values by the method of Quicksort (Hoare's partition-exchange sort) in place. The procedure iteratively partitions the vector creating two subvectors, one whose values are less than or equal to the value initially at the middle location, and the other with elements greater than or equal to that value. This chosen value ends up in its true (post-sorted) position between the two subvectors. The half-way location was chosen for initial trial comparison in order to speed the sort when the original vector is already partly ordered.

After each partition, first and last locations of the larger subvector are stored in a pointer stack, which can accumulate no more than log2(N) pairs, and the process of partitioning is continued on the smaller subvector. The process of comparison and partitioning is continued until no subvectors remain. The vector is then completely sorted.

SIGNAL PROCESSING LIBRARY

```
**********                                                  **********
*        *                                                  *        *
* ACORT  *      --- AUTO-CORRELATION (TIME-DOMAIN) ---      * ACORT  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To perform an auto-correlation operation on a vector using time-domain techniques.

CALL FORMAT: CALL ACORT(A,C,N,M)

PARAMETERS:
A = Floating-point input vector
C = Floating-point output vector
N = Integer element count for C
(Number of lags)
M = Integer element count for A
(Note vector elements occupy consecutive addresses.)

DESCRIPTION: $C(m)=\sum_{q=1}^{M-m+1} A(m+q-1)*A(q)$,
for q=1 to M-m+1
m=1 to N

ACORT uses time-domain techniques (compare with ACORF) to compute the auto-correlation function. This routine needs less storage than ACORF, and runs faster when N and/or M is small. The resultant vector C must not overlay the source vector A.

EXAMPLE:

```
N = 3
M = 5

A :  1.Ø   2.Ø   3.Ø  4.Ø  5.Ø
C : 55.Ø  4Ø.Ø  26.Ø
```

```
**********                                                **********
*        *                                                *        *
* BLKMAN *      --- BLACKMAN WINDOW MULTIPLY ---          * BLKMAN *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To multiply a vector by a Blackman window.

CALL FORMAT: CALL BLKMAN(A,I,C,K,N)

PARAMETERS:
A = Floating-point input vector
I = Integer element step for A
C = Floating-point output vector
K = Integer element step for C
N = Integer element count (a power of 2)

DESCRIPTION:
```
C(m) = A(m)*(Ø.42-Ø.5Ø*COS((m-1)*(2*PI/N))
                    +Ø.Ø8*COS((m-1)*(4*PI/N)))
        for m=1 to N
```

Multiplies the elements of the vector A by an N element Blackman window function, and stores the results in the vector C. N must be a power of 2.

EXAMPLE:

```
I = 1
K = 1
N = 8

A :  1.Ø     1.Ø     1.Ø     1.Ø
     1.Ø     1.Ø     1.Ø     1.Ø

C :  Ø.ØØØ   Ø.Ø66   Ø.34Ø   Ø.774
     1.ØØØ   Ø.774   Ø.34Ø   Ø.Ø66
```

THLINC = Floating-point input scalar containing the phase increment threshold (used to obtain more confident phase estimates near sharp zeros)

THLCON = Floating-point input scalar containing the phase consistency threshold

WMD = Integer work area vector of length 39 used for various software stacks during phase unwrapping

IXCXST = Integer input scalar X and CX input status:
Ø if X is provided as input and CX is not provided as input
1 if X is not provided as input and CX is provided as input
2 if both X and CX are provided as input

IAUXST = Integer input scalar AUX input status:
Ø if AUX is not provided as input
1 if AUX is provided as input

IPHWST = Integer input scalar phase unwrapping only status:
Ø if complex cepstrum is desired
1 if phase unwrapping only is desired

NOTE: For APFTN64 calls to CCEPS, the dimension of arrays X, CX, and AUX must be greater than or equal to NFFT2 and the dimension of array WMD must be greater than or equal 39.

DESCRIPTION: See "Programs for Digital Signal Processing", IEEE Press, 1979.

1) Input parameters are checked for out of range conditions. If any errors are detected, then SSUC gets the appropriate error code (2.Ø - 9.Ø) and CCEPS returns.
2) If IXCXST=Ø then X is used to compute CX.
3) If IXCXST=1 then CX is used to compute X. Note that in this case the vector X will occupy NFFT2 words in Main Memory but only the first NX elements of X will be used in further calculations.
4) If IAUXST=Ø then X is used to compute AUX.
5) Each of the NFFT2 elements of CX and AUX are divided by 2.Ø to match IEEE formulation.
6) If the first element of CX is less than Ø.Ø
then SNX = -1.Ø
else SNX = +1.Ø .
7) The magnitude of the spectrum is computed and stored in the real positions of AUX;
the phase derivative of the spectrum is computed and stored in the imaginary positions of AUX;
and twice the linear phase estimate (mean of the phase derivative) is computed for use in the phase unwrapping computation.

```
CX(OUT) :  -1.6639    Ø.ØØØØ   -5.9134    Ø.7447
             Ø.9543    1.4Ø85    3.Ø149    Ø.7278
             3.5771    Ø.ØØØØ

AUX(OUT):    Ø.Ø359   -2.614Ø    Ø.ØØØØ    5.5523
             6.7434   -2.7728  415.6262   -2.9137
          1279.4929   -2.9325

SNX     =  1.ØØØØ
SFX     = -2.1ØØØ
SSUC    =  Ø.ØØØØ
```

```
**********                                                  **********
*        *                                                  *        *
* CCORT  *      --- CROSS-CORRELATION (TIME-DOMAIN) ---     * CCORT  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To perform a cross-correlation operation on two vectors using time-domain techniques.

CALL FORMAT: CALL CCORT(A,B,C,N,M)

PARAMETERS:
A = Floating-point input vector (operand)
B = Floating-point input vector (operator)
C = Floating-point output vector
N = Integer element count for C (number of lags)
M = Integer element count for A and B
(Note vector elements occupy consecutive addresses.)

DESCRIPTION: C(m)=SUM(A(m+q-1)*B(q));
for q=1 to M-m+1
and m=1 to N

CCORT uses time-domain techniques (compare with CCORF) to compute the cross-correlation function. This routine needs less storage than CCORF, and runs faster when N and/or M is small.

EXAMPLE:

```
N = 3
M = 4

A :   1.Ø    2.Ø   3.Ø   4.Ø
B :  1Ø.Ø   2Ø.Ø  3Ø.Ø  4Ø.Ø
C : 3ØØ.Ø  2ØØ.Ø 11Ø.Ø
```

```
**********                                              **********
*        *                                              *        *
* COHER  *          --- COHERENCE FUNCTION ---          * COHER  *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To compute the coherence function, given the auto-spectra of two signals and the cross-spectrum between them.

CALL FORMAT: CALL COHER(A,B,C,D,N)

PARAMETERS:
- A = Floating-point input vector (Auto-spectrum)
- B = Floating-point input vector (Auto-spectrum)
- C = Complex-floating-point input vector (Cross-spectrum)
- D = Floating-point output vector (Coherence function)
- N = Integer element count (Note vector elements occupy consecutive addresses.)

DESCRIPTION: D(m)= (R(C(m))**2+I(C(m))**2)/(A(m)*B(m)); for m=1 to N

EXAMPLE:

```
N = 3

A :  1.Ø       2.Ø       3.Ø
B :  4.Ø       5.Ø       6.Ø
C : (1.Ø,2.Ø) (3.Ø,4.Ø) (5.Ø,6.Ø)
D :  1.25      2.5       3.39
```

```
**********                                          **********
*        *                                          *        *
* DECFIR *          --- DECIMATION ---              * DECFIR *
*        *                                          *        *
**********                                          **********
```

PURPOSE: To FIR filter an input vector using a convolution technique incorporating decimation by a factor D. Typically, the input vector is a digital signal requiring low pass filtering and the operator vector is the array of pre-determined filter coefficients.

CALL FORMAT: CALL DECFIR(A,B,C,D,N,M)

PARAMETERS:
A = Floating-point input (undecimated) vector
B = Floating-point input operator vector
C = Floating-point output vector
D = Integer input decimation factor (D > Ø)
N = Integer input element count expected for C
when convolving without decimation
(NOTE: the actual size of the output vector
C will be [(N-1)/D]+1)
M = Integer input element count for B
(NOTE: element count for A must be N+M-1)

DESCRIPTION: C(m) = SUM (A(D*(m-1)+q) * B(q) for q=1 to M
and m=1 to [(N-1)/D]+1
(NOTE: This assumes that the operator array B
is loaded with the elements arranged in
reverse order. Thus:
B(1) = Mth operator point
B(2) = (M-1)th operator point
.
.
.
B(M) = 1st operator point)

For references see:
(1) A. Peled and B. Liu, "Digital Signal Processing: Theory, Design and Implementation." John Wiley, 1976.
(2) R. E. Crochiere and L. R. Rabiner, "Optimum FIR digital filter implementation for decimation, interpolation, and narrow band filtering," IEEE Trans. Acoust. Speech Signal Processing, vol ASSP-23 pp 444-456, Oct. 1975.

This routine performs a convolution on the decimated operand A with the operator B. The results are stored in [(N-1)/D]+1 elements of vector C.

```
**********                                              **********
*        *                                              *        *
*  ENVEL *          --- ENVELOPE DETECTOR ---           *  ENVEL *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To obtain the envelope of a vector X(t).

CALL FORMAT: CALL ENVEL(X,E,N)

PARAMETERS: X = Floating-point input vector
E = Floating-point output envelope vector
N = Integer element count (a power of 2)

DESCRIPTION: E(t) = SQRT { X(t)**2 + H{X(t)}**2 } for t=1 to N
where: H{X(t)} = Hilbert transform of X(t).

For references see any standard text on communication theory, viz. "Communications Systems and Techniques," M.Schwartz, W.Bennet, & Stein, McGraw Hill.

This routine starts by obtaining the Hilbert transform of the input vector. The formula shown above is then used to extract the envelope.

EXAMPLE:

N = 8

X :	2.7	1.6	8.3	4.2	9.7	14.1	3.6	Ø.5
E :	2.72	4.32	8.82	4.3Ø	11.33	14.73	9.21	Ø.85

```
**********                                                **********
*        *                                                *        *
*  HANN  *        --- HANNING WINDOW MULTIPLY ---         *  HANN  *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To multiply a vector by a Hanning window.

CALL FORMAT: CALL HANN(A,I,C,K,N,F)

PARAMETERS:
A = Floating-point input vector
I = Integer element step for A
C = Floating-point output vector
K = Integer element step for C
N = Integer element count (a power of 2)
F = Integer normalization flag
Ø for unnormalized Hanning window
(peak window value=1.Ø)
1 for normalized Hanning window
(peak window value=1.63)

DESCRIPTION: N should be a power of 2. If not, HANN sets N to the next lower power of 2. For further information see Digital Time Series Analyses, Otnes and Enochsen, John Wiley '72, page 294.

C(m)=W*A(m)*(1.Ø-COS(2*PI*(m-1)/N); for m=1 to N

where:

W = Ø.5 for F=Ø
W = Ø.8165 for F=1

EXAMPLE:

```
N = 4
F = Ø

A : 1.Ø  1.Ø  1.Ø  1.Ø
C : Ø.Ø  Ø.5  1.Ø  Ø.5

N = 4
F = 1

A : 1.ØØ  1.ØØ   1.ØØ  1.ØØ
C : Ø.ØØ  Ø.82   1.63  Ø.82
```

```
**********                                                **********
*        *                                                *        *
*  HLBRT *          --- HILBERT TRANSFORMER ---           *  HLBRT *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To obtain the Hilbert transform of an analytic signal.

CALL FORMAT: CALL HLBRT(X,H,N)

PARAMETERS:
X = Floating-point input vector
H = Floating-point output Hilbert transformed vector
N = Integer element count (a power of 2)

DESCRIPTION: F{H{X(t) = -J * F{X(t)} for t=1 to N
where: F{X(t)} = Fourier transform of X(t).
H{X(t)} = Hilbert transform of X(t).
J = SQRT(-1)

(1) A real to complex FFT of X(t) is obtained.
(2) Real components of the result are multiplied by -1.
(3) Positions of the real and imaginary components are switched.
(4) A complex to real inverse FFT is performed on the results of step 3.

EXAMPLE:

N = 8

X : 2.7 1.6 8.3 4.2 9.7 14.1 3.6 Ø.5

H : Ø.4 -4.Ø -3.Ø -Ø.9 -5.9 4.3 8.5 Ø.7

EXAMPLE:

```
N  = 128
M  = 1Ø

X(i) : 1Ø * SIN(i * 1Ø * 2*PI/128) +
       2Ø * SIN(i * 2Ø * 2*PI/128) +
       3Ø * SIN(i * 3Ø * 2*PI/128)
        for i = 1 to 128
```

j	RC(j)	A(j)	AL(j)	R(j)
1	-Ø.2847	1.ØØØØ	89599.9	89599.9
2	Ø.8183	-Ø.8897	82335.4	25512.8
3	-Ø.52ØØ	1.Ø4Ø4	27198.6	-6Ø112.8
4	Ø.54Ø3	-Ø.25Ø9	19844.6	-37858.5
5	Ø.1863	Ø.1Ø24	14Ø51.1	322Ø8.9
6	-Ø.2451	Ø.2Ø85	13563.2	24552.1
7	-Ø.Ø955	Ø.1145	12748.7	-3ØØ17.1
8	Ø.3127	Ø.Ø661	12632.4	-189Ø9.6
9	Ø.4627	Ø.1Ø81	11397.1	35262.2
1Ø	Ø.3Ø54	Ø.1478	8957.3	18797.9
11		Ø.3Ø54	8121.7	-52577.9

```
ER = Ø
```

EXAMPLE:

```
N     = 8
NP    = 5
R     = 4.Ø
MODE = 1

A :  Ø.Ø  1Ø.Ø  2Ø.Ø   3.Ø   4.Ø  5Ø.Ø   6.Ø  7Ø.Ø

B : 2Ø.Ø   3.Ø  5Ø.Ø   6.Ø
C :  3.Ø   4.Ø   6.Ø   7.Ø

R     = 4.Ø
```

```
Inverse transform:
x(k) = SUM { X((r-1)*df-F1*df) *
             EXP(j*2*pi*(r-1)*df*(k-1)*dt) }
       for r = 1 to NF
   where  dt = 1/XM.
```

Thus the same formula used for the forward transform may be used for the inverse transform if here W = -2*pi*(k-1)/XM and F1 and NF replace T1 and NT respectively. If the r=1 component X(1) is input, it must have an imaginary part equal to Ø.

The DFT is produced by the modified Goertzel algorithm as described in

(1) A.V. Oppenheim and Schafer, "Digital Signal Processing," Prentice Hall, 1975

and

(2) F. Bonzanigo, "An improvement of Tribolet's phase unwrapping algorithm," IEEE Trans. ASSP, Feb. 1978, pp. 1Ø4-1Ø5

Additionally, an exponential factor has been used to account for any offset of the input values from zero (T1 or F1).

Inverse times are approximately double for forward times after the NT and NF values are interchanged.

EXAMPLE:

```
F1 = Ø.Ø
T1 = 1.Ø
NT = 8
NF = 4
XM = 8.Ø
I  = 1

A(INPUT)  :  1.Ø  Ø.Ø -1.Ø  Ø.Ø  1.Ø  Ø.Ø -1.Ø  Ø.Ø

B(OUTPUT) : (Ø.Ø, Ø.Ø)(Ø.Ø, Ø.Ø)(Ø.Ø,-4.Ø)(Ø.Ø, Ø.Ø)


F1 = 2.Ø
T1 = Ø.Ø
NT = 8
NF = 2
XM = 8.Ø
I  = -1

B(INPUT)  : (4.Ø, Ø.Ø)(Ø.Ø, Ø.Ø)

A(OUTPUT) :  8.Ø  Ø.Ø -8.Ø  Ø.Ø  8.Ø  Ø.Ø -8.Ø  Ø.Ø
```

```
**********                                                  **********
*        *                                                  *        *
* RFTII  *  --- REAL FFT WITH QUARTER INTERPOLATION ---     * RFTII  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To perform an in-place real-to-complex forward or a complex-to-real inverse fast Fourier transform (FFT) including the case of N=64K via quarter interpolation in the 4K cosine table.

CALL FORMAT: CALL RFTII(C,N,F)

PARAMETERS:
C = Floating-point input/output vector
N = Integer input element count (power of 2)
F = Integer input direction flag:
+1 for forward
-1 for inverse

DESCRIPTION: See RFFT.

EXAMPLE:

```
N = 4
F = 1 (Forward)

C(IN)  :   1Ø.Ø       1Ø.Ø       1Ø.Ø       1Ø.Ø
C(OUT) :  (8Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)

N = 4
F = -1 (Inverse)

C(IN)  :  (8Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)
C(OUT) :   8Ø.Ø       8Ø.Ø       8Ø.Ø       8Ø.Ø
```

```
**********                                                  **********
*        *                                                  *        *
* TCONV  *  --- POST-TAPERED CONVOLUTION (CORRELATION) ---  * TCONV  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To perform a post-tapered convolution or correlation operation on two vectors.

CALL FORMAT: CALL TCONV(A,I,B,J,C,K,N,M,L) for correlation
CALL TCONV(A,I,B(N),J,C,K,N,M,L) for convolution

PARAMETERS:
A = Floating-point input vector (operand)
I = Integer element step for A (>Ø)
B = Floating-point input vector (operator)
J = Integer element step for B (<Ø => Convolution)
C = Floating-point output vector
K = Integer element step for C
N = Integer element count for C
M = Integer element count for B
L = Integer element count for A

FORMULA:
C(m)=SUM(A(m+q-1)*B(q));
for q=1 to R
and m=1 to N

where:

R=MIN(M,L-M+1)

DESCRIPTION: TCONV performs either a correlation (I and J positive) or a convolution (I positive and J negative) operation between the L-element operand (trace) vector A and the M-element operator (kernel) vector B. The N-element result vector is stored in C. TCONV automatically inserts zeros into the calculation if N+M-1 exceeds the operand length L, thus saving storage and zeroing of N+M-1-L extra operand elements. (Compare with CONV.)

EXAMPLE:

N = 4
M = 2
L = 4

CORRELATION:

A : Ø.Ø 1.Ø 3.Ø 5.Ø
B : 2.Ø 1.Ø
C : 1.Ø 5.Ø 11.Ø 1Ø.Ø

```
**********                                              **********
*        *                                              *        *
* TRANS  *          --- TRANSFER FUNCTION ---           * TRANS  *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To perform a complex transfer function calculation by dividing the cross-spectrum by the auto-spectrum.

CALL FORMAT: CALL TRANS(A,B,C,N)

PARAMETERS:
- A = Floating-point input vector (Auto-spectrum)
- B = Complex-floating-point input vector (Cross-spectrum)
- C = Complex-floating-point output vector (Transfer function)
- N = Integer element count (Note vector elements occupy consecutive addresses.)

DESCRIPTION: R(C(m))+I(C(m))=(R(B(m))+I(B(m)))/A(m); for m=1 to N

EXAMPLE:

```
N = 3

A :  1.Ø       2.Ø       3.Ø
B : (1.Ø,2.Ø) (3.Ø,4.Ø) (5.Ø,6.Ø)
C : (1.Ø,2.Ø) (1.5,2.Ø) (1.67,2.Ø)
```

```
**********                                                **********
*        *                                                *        *
* VAVLIN *        --- VECTOR LINEAR AVERAGING ---         * VAVLIN *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To update the linear average of a sequence of vectors to include a new vector.

CALL FORMAT: CALL VAVLIN(A,I,B,C,K,N)

PARAMETERS:
A = Floating-point input vector
I = Integer element step for A
B = Floating-point input scalar
(Number of vectors included in current average)
C = Floating-point input/output vector
K = Integer element step for C
N = Integer element count

DESCRIPTION: C(m)=C(m)*B/(B+1.Ø) + A(m)/(B+1.Ø); for m=1 to N

EXAMPLE:

N = 5

A	: 5.ØØØ	1Ø.ØØØ	2Ø.ØØØ	25.ØØØ	3Ø.ØØØ
B	: 5.ØØØ				
C(INPUT)	:1Ø.ØØØ	1Ø.ØØØ	1Ø.ØØØ	1Ø.ØØØ	1Ø.ØØØ
C(OUTPUT)	: 9.167	1Ø.ØØØ	11.667	12.5ØØ	13.333

```
**********                                                **********
*        *                                                *        *
*  VXCS  *     --- VECTOR MULTIPLIED BY SIN AND COS ---   *  VXCS  *
*        *                  (TABLE LOOKUP)                *        *
**********                                                **********
```

PURPOSE: To multiply a vector with the sine and cosine of a linearly increasing argument with a given initial phase.

CALL FORMAT: CALL VXCS(A,C,K,F,P,N)

PARAMETERS:
- A = Floating-point input vector to be multiplied by the sine and cosine functions
- C = Complex floating-point output vector
- K = Integer input element step for C (K >= 2)
- F = Floating-point input scalar frequency
- P = Floating-point input scalar phase at t=Ø
  = Floating-point output scalar initial phase value for next frame
- N = Integer element count

DESCRIPTION:

$$Re(C(m)) = A(m) * COS((m-1)*F+P)$$
$$Im(C(m)) = A(m) * SIN((m-1)*F+P)$$
for m = 1 to N

NOTE: The arguments for COS and SIN are expected to be in radians.

This routine multiplies vector A with a sine and cosine function defined by frequency F and initial phase P. Straight ROM table lookup is used for generating the sine and cosine values and thus this routine has limited precision. The initial phase value for the next frame is returned in P.
NOTE: K should be greater than or equal to 2 so as not to destroy part of the resultant vector C as it is generated.

EXAMPLE:

```
K = 2
F = Ø.5
P = 3.1415927
N = 8

A :  Ø.Ø   1.Ø   2.Ø   3.Ø   4.Ø   5.Ø   6.Ø   7.Ø
```

```
**********                                                **********
*        *                                                *        *
* WIENER *        --- WIENER LEVINSON ALGORITHM ---       * WIENER *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To solve a system of single channel normal equations which arise in least squares filtering and prediction problems.

CALL FORMAT: CALL WIENER(LR,R,G,F,A,ISW,IERR)

PARAMETERS:
- LR = Integer filter length
- R = Floating-point input vector (Auto-correlation coefficients)
- G = Floating-point input vector (Cross correlation)
- F = Floating-point output vector (Filter weighting coefficients)
- A = Floating-point output vector (Prediction error operator)
- ISW = Integer input (algorithm switch)
  - Ø = spike deconvolution
  - 1 = general deconvolution
- IERR = Integer output scalar (failure flag)

DESCRIPTION: WIENER solves:

1. The following set of LR equations for F;

   SUM [F(p)*R(m-p+1)=G(m);
   for p=1 to LR and m=1 to LR

2. The following set of LR equations for A;

   SUM [A(p)*R(m-p+1)=V*D;
   for p=1 to LR and m=1 to LR

where, A(1)=1.Ø
D=1.Ø when m=1
D=Ø.Ø when m not = 1
V=A(1)*R(1)+...+A(LR)*R(LR)
R(-i)=R(i)

If the algorithm is successful IERR is set to Ø; else it is set to the pass number at which the failure occurred.

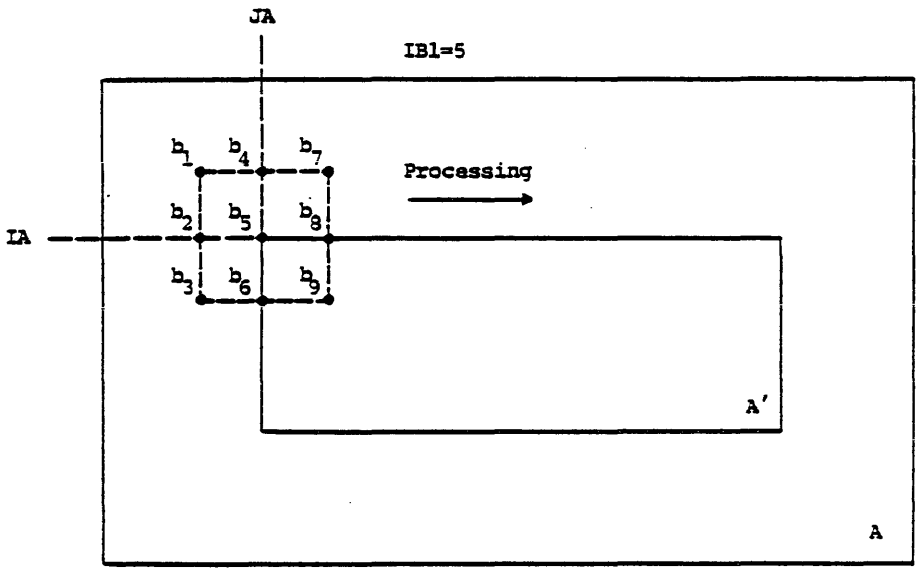
IMAGE PROCESSING LIBRARY

```
N1 = 4
N2 = 4
F  = -1 (Inverse)

C(IN)  : ( 4.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)
         ( 4.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)
         ( 4.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)
         ( 4.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)

C(OUT) : (16.Ø,Ø.Ø) (16.Ø,Ø.Ø) (16.Ø,Ø.Ø) (16.Ø,Ø.Ø)
         ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)
         ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)
         ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø) ( Ø.Ø,Ø.Ø)
```

```
            IR  = Integer input scalar flag:
                  non-zero for correlation
                  Ø for convolution

DESCRIPTION:  C((i+IC-1),(j+JC-1))=
              SUM(A((i+IA+k-2-irbias),(j+JA+1-2-icbias))*B(k,l))
                where  i=1 to M
                       j=1 to N
                  for  k=1 to MB
                       l=1 to NB
                and    IBl=MB*NB-IBl+1 for convolution
                       icbias=(IBl-1)/MB
                       irbias=(IBl-1)-MB*icbias
                       (row and column biases are from the
                       initial B(1,1) position.

              CONV2D correlates or convolves a two-dimensional
              operand submatrix A' of A with a two-dimensional
              operator matrix B, and stores the result in
              submatrix C' of C.  A one-to-one correspondence
              exists between the elements of A' and C'.

              This routine does not do boundary testing.
              Therefore care must be taken when choosing values
              for IA, JA, and IBl for given values of M, N, MB,
              NB, and IR to avoid using data outside of A when
              computing C'.
```

EXAMPLE:

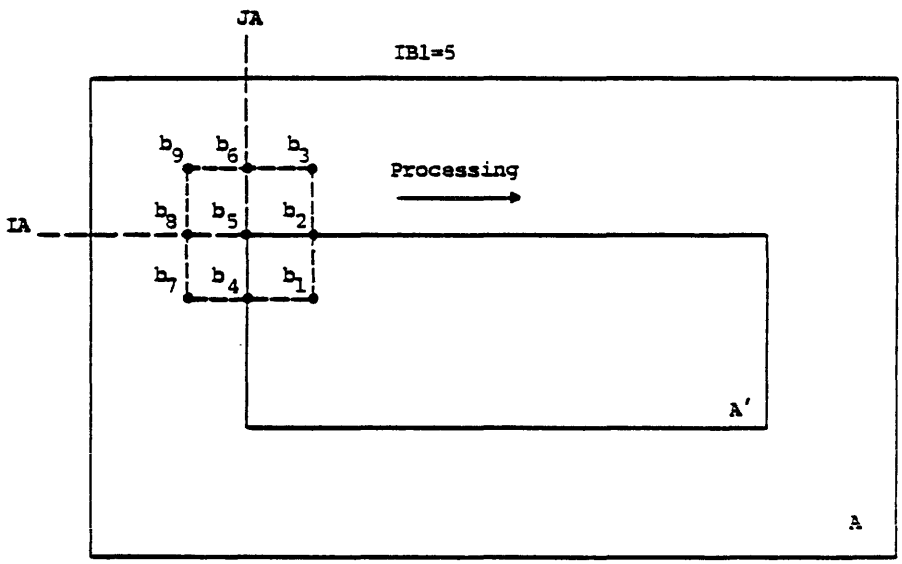
```
MA  = 9
IA  = 1
JA  = 1
M   = 7
N   = 7
MB  = 3
NB  = 3
MC  = 9
IC  = 1
JC  = 1

A :  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  4.Ø  4.Ø  8.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  4.Ø  4.Ø  8.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  4.Ø  4.Ø  8.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  4.Ø  4.Ø  8.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
```



Here the operator, B, is positioned for processing the initial point in A'.

-5455-

Figure A-1 Correlation



Here the operator, B, is positioned for processing the initial point in A'.

-5456-

Figure A-2 Convolution

EXAMPLE:

```
MA = 8
IA = 2
JA = 2
MC = 8
IC = 2
JC = 2
M  = 6
N  = 6

A :  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  2.Ø  2.Ø  1.Ø  1.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  2.Ø  2.Ø  1.Ø  1.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  Ø.Ø
     Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø

C :   U    U    U    U    U    U    U    U
      U   2.Ø  3.Ø  3.Ø  3.Ø  3.Ø  2.Ø   U
      U   3.Ø  1.Ø  2.Ø  2.Ø  1.Ø  3.Ø   U
      U   3.Ø  2.Ø  2.Ø  2.Ø  2.Ø  3.Ø   U
      U   3.Ø  2.Ø  2.Ø  2.Ø  2.Ø  3.Ø   U
      U   3.Ø  1.Ø  2.Ø  2.Ø  1.Ø  3.Ø   U
      U   2.Ø  3.Ø  3.Ø  3.Ø  3.Ø  2.Ø   U
      U    U    U    U    U    U    U    U
```

(U indicates unchanged elements of C)

This routine differs from GRAD2D in that it can perform testing for image boundaries, substituting zeros for values that are needed outside the boundary. The routine runs somewhat more slowly than GRAD2D.

If testing is employed, zeros are substituted for those elements in the formula which fall outside of A. This is useful in preventing wrap-around and incorrect processing of the columns and rows on the borders of A. However, the testing adds processing time and is unnecessary when there is a border of width one around A' which lies totally within A.

If boundary testing is not employed (i.e. B = Ø) and if a boundary of A' coincides with all or part of a boundary of A, then boundary effects will be observed in the computation of C'. In the cases of JA=1 or JA+N-1=NA these boundary effects may not be predictable since data stored adjacent to A may not be predictable.

EXAMPLE:

```
MA = 8
NA = 8
IA = 1
JA = 1
C  = 64
MC = 8
NC = 8
IC = 1
JC = 1
M  = 8
N  = 8
B  = 1

A :  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø
     1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø
     1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø
     1.Ø  1.Ø  1.Ø  2.Ø  2.Ø  1.Ø  1.Ø  1.Ø
     1.Ø  1.Ø  1.Ø  2.Ø  2.Ø  1.Ø  1.Ø  1.Ø
     1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø
     1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø
     1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø
```

```
**********                                              **********
*        *                                              *        *
* LAPL2D *          --- LAPLACIAN FILTER ---            * LAPL2D *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To filter images for edge enhancement by applying a two-dimensional Laplacian operator.

CALL FORMAT: CALL LAPL2D(A,MA,NA,IA,JA,C,MC,NC,IC,JC,M,N,IX)

PARAMETERS:

A = Floating-point input matrix (column ordered)
MA = Integer number of rows of A
NA = Integer number of columns of A
IA = Integer initial row of the submatrix A' of A to be processed (1 < or = IA < or = MA)
JA = Integer initial column of the submatrix A' of A to be processed (1 < or = JA < or = NA)
C = Floating-point output matrix (column ordered)
MC = Integer number of rows of C
NC = Integer number of columns of C
IC = Integer initial row of C which locates the submatrix C', where C' will be the processed A' (1 < or = IC < or = MC)
JC = Integer initial column of C which locates the submatrix C' (1 < or = JC < or = NC)
M = Integer number of rows in A' (1 < or = M < or = MA)
N = Integer number of columns in A' (1 < or = N < or = NA)
IX = Integer distance to filter side from center of square: side S=2*(IX+1); filter area = S**2

DESCRIPTION:

$$C'(p,q) = 128 - 4*A'(p,q) + A'(p-IX,q) + A'(p+IX,q) + A'(p,q-IX) + A'(p,q+IX)$$

Each of the elements in C' is calculated according to the above formula, which adds to a bias of 128 a weighted combination of each pixel and its 4 horizontal and vertical neighbors at distance IX.

If a boundary of A' coincides with all or part of a boundary of A, then boundary effects will be observed in the computation of C'. In the cases of JA<=IX or JA+N-IX>=NA these boundary effects may not be predictable since data stored adjacent to A may not be predictable. Boundary effects will be predictable if A' is initially ringed with a known constant, such as zero.

```
**********                                                  **********
*        *                                                  *        *
* LPL2DB *  --- LAPLACIAN FILTER WITH BOUNDARY TEST ---     * LPL2DB *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To filter images for edge enhancement by applying a two-dimensional Laplacian operator. This routine does special boundary testing.

CALL FORMAT: `CALL LPL2DB(A,MA,NA,IA,JA,C,MC,NC,IC,JC,M,N,IX,B)`

PARAMETERS:

- A = Floating-point input matrix (column ordered)
- MA = Integer number of rows of A
- NA = Integer number of columns of A
- IA = Integer initial row of the submatrix A' of A to be processed (1 < or = IA < or = MA)
- JA = Integer initial column of the submatrix A' of A to be processed (1 < or = JA < or = NA)
- C = Floating-point output matrix (column ordered)
- MC = Integer number of rows of C
- NC = Integer number of columns of C
- IC = Integer initial row of C which locates the submatrix C', where C' will be the processed A' (1 < or = IC < or = MC)
- JC = Integer initial column of C which locates the submatrix C' (1 < or = JC < or = NC)
- M = Integer number of rows in A' (1 < or = M < or = MA)
- N = Integer number of columns in A' (1 < or = N < or = NA)
- IX = Integer distance to filter side from center of square: side S=2*(IX+1); filter area = S**2
- B = Integer input scalar which is Ø if no boundary testing is desired; if not = Ø, values needed outside of A are evaluated as zeros

DESCRIPTION:

```
C'(p,q)= 128 -4*A'(p,q)+A'(p-IX,q)+A'(p+IX,q)
                       +A'(p,q-IX)+A'(p,q+IX)
```

Each of the elements in C' is calculated according to the above formula, which adds to a bias of 128 a weighted combination of each pixel and its 4 horizontal and vertical neighbors at distance IX.

This routine differs from LAPL2D in that it can perform testing for image boundaries, substituting zeros for values that are needed outside the boundary. The routine runs somewhat more slowly than LAPL2D.

```
**********                                              **********
*        *                                              *        *
* MED2D  *              --- MEDIAN FILTER ---           * MED2D  *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To filter out noise in images by replacing each pixel with the median value of the pixels in a square window centered around the pixel.

CALL FORMAT: CALL MED2D(A,MA,IA,JA,C,MC,IC,JC,M,N,IX,H,L)

PARAMETERS:

A = Floating-point input matrix (column ordered)
MA = Integer number of rows of A
(NA = Number of columns of A)
IA = Integer initial row of the submatrix A' of A to be processed (1 < or = IA < or = MA)
JA = Integer initial column of the submatrix A' of A to be processed (1 < or = JA < or = NA)
C = Floating-point output matrix (column ordered)
MC = Integer number of rows of C
(NC = Number of columns of C)
IC = Integer initial row of C which locates the submatrix C', where C' will be the processed A' (1 < or = IC < or = MC)
JC = Integer initial column of C which locates the submatrix C' (1 < or = JC < or = NC)
M = Integer number of rows in A' (1 < or = M < or = MA)
N = Integer number of columns in A' (1 < or = N < or = NA)
IX = Integer distance to median filter side from center of square: side S=(2*IX)+1; filter area = S**2; IX>Ø
H = Floating-point vector histogram used as a work area
L = Integer input scalar length of H = 2**(number of bits per pixel)

DESCRIPTION: C'(p,q)=median of all elements A'(t,u),
p-IX<=t<=p+IX, q-IX<=u<=q+IX

For each of the elements in A' a histogram is formed from the median of the elements within + or - IX row and column distance from the element. The median is found via a fast algorithm published in:

```
**********                                                  **********
*        *                                                  *        *
* MOVREP *    --- SUB-IMAGE MOVE AND LEVEL REPLACE ---      * MOVREP *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To simply move a sub-image A' of an image A and/or to replace each pixel value with another value as specified in the lookup table, vector T, whose elements are the new values and whose subscripts are the original pixel values + 1.

CALL FORMAT: CALL MOVREP(A,MA,IA,JA,C,MC,IC,JC,M,N,T,NT)

PARAMETERS:

A = Floating-point input matrix (column ordered)
MA = Integer number of rows of A
(NA = Number of columns of A)
IA = Integer initial row of the submatrix A' of A to be processed (1 < or= IA < or = MA)
JA = Integer initial column of the submatrix A' of A to be processed (1 < or= JA < or = NA)
C = Floating-point output matrix (column ordered)
MC = Integer number of rows of C
(NC = Number of columns of C)
IC = Integer initial row of C which locates the submatrix C' of C; C' will be the processed A' (1 < or= IC < or = MC)
JC = Integer initial column of C which locates the submatrix C' of C (1 < or= JC < or = NC)
M = Integer number of rows in A' (1 < or = M < or = MA)
N = Integer number of columns in A' (1 < or = N < or = NA)
T = Floating-point input vector pixel replacement table
NT = Integer input scalar length of vector T = 2**(# of bits per pixel) (NT = Ø indicates only submatrix move is desired)

DESCRIPTION: For pixel replacement,
C'(p,q)=T(FIX(A'(p,q))+1)

For submàtrix move,
C'(p,q)=A'(p,q)

```
**********                                                  **********
*        *                                                  *        *
* RFFT2D *    --- REAL TO COMPLEX 2-DIMENSIONAL FFT ---     * RFFT2D *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To perform an in-place two-dimensional real-to-complex forward or a complex-to-real inverse fast Fourier transform (FFT).

CALL FORMAT: CALL RFFT2D(C,N1,N2,F)

PARAMETERS:
- C = Floating-point input/output matrix (column ordered)
- N1 = Integer number of rows = number of real elements per column (power of 2 < or = 16384)
- N2 = Integer number of columns = number of real elements per row (power of 2 < or = 16384)
  NOTE: N1*N2 must be < or = available main data
- F = Integer direction flag:
  +1 for forward
  -1 for inverse

DESCRIPTION: Forward: RFFT2D performs a two-dimensional real to complex forward FFT on the N1 by N2 real array C, storing the (N1/2 + 1) by (N2/2 + 1) complex array result in a special packed complex array form occupying the same N1 by N2 locations of array C:

Let E1 = N1/2 and E2 = N2/2

```
R(1,1)     R(1,E2+1)    R(1,2)     I(1,2)  ... R(1,E2)   I(1,E2)
R(E1+1,1)  R(E1+1,E2+1) R(E1+1,2)  I(E1+1,2).. R(E1+1,E2)I(E1+1,E2)
R(2,1)     R(2,2)       R(2,3)     R(2,4)  ... R(2,N2-1) R(2,N2)
I(2,1)     I(2,2)       I(2,3)     R(2,4)  ... I(2,N2-1) I(2,N2)
  .            .            .        .            .          .
  .            .            .        .            .          .
  .            .            .        .            .          .
R(E1,1)    R(E1,2)      R(E1,3)    R(E1,4) ... R(E1,N2-1)I(E1,N2)
I(E1,1)    I(E1,2)      I(E1,3)    I(E1,4) ... I(E1,N2-1)I(E1,N2)
```

The results of a two-dimensional real-to-complex forward FFT should be multiplied by 1/(2*N1*N2) for proper scaling.

LINPACK BLAS LIBRARY

```
**********                                                  **********
*        *                                                  *        *
* CAXPYN *      --- NESTED COMPLEX  A * X + Y ---           * CAXPYN *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To add a scalar multiple of one complex floating-point vector to another complex floating-point vector N times, each time for a different pair of vectors and a different complex floating-point scalar. The first vector is a subset of the vector X, and the second is a subset of the vector Y. The scalar is an element of the vector A.

CALL FORMAT: CALL CAXPYN(ISW,N,M,A,IAO,X,IXI,IXO,Y,IYI,IYO)

PARAMETERS:

ISW = Integer input scalar. ISW is a function selector switch and is treated as a bit string with the bits numbered from the least significant bit (bit Ø). If a given bit is set (equal to 1), then the function option that corresponds to that bit is selected. All options are independent of each other and are summarized below.

- Bit Ø: Negate A * X.
- Bit 1: Not used.
- Bit 2: Use conjugate of A.
- Bit 3: Use conjugate of X.

All other bits are ignored.

N = Integer input scalar. Number of A * X + Y operations, i.e., outer loop count.

M = Integer input scalar. Number of elements in each A * X + Y operation, i.e., inner loop count.

A = Complex floating point input vector. Array of scalars.

IAO = Integer input scalar. Outer loop element increment for A.

X = Complex floating point input vector. First input vector.

IXI = Integer input scalar. Inner loop element increment for X.

IXO = Integer input scalar. Outer loop element increment for X.

Y = Complex floating point input/output vector. Second input vector on input. Output vector on output.

IYI = Integer input scalar. Inner loop element increment for Y.

IYO = Integer input scalar. Outer loop element increment for Y.

EXAMPLE:

```
Input:   ISW = Ø
         N   = 2
         M   = 3
         IAO = 1
         IXI = 1
         IXO = Ø
         IYI = 1
         IYO = 3

         A  :  ( 3.Ø,-1.Ø)  ( 2.Ø, Ø.Ø)

         X  :  ( Ø.Ø, 1.Ø)  ( 2.Ø, 1.Ø)  (-1.Ø, Ø.Ø)

         Y  :  (-1.Ø, 2.Ø)  ( Ø.Ø, Ø.Ø)  ( 2.Ø, Ø.Ø)
               ( 1.Ø,-3.Ø)  (-2.Ø,-1.Ø)  ( Ø.Ø,-2.Ø)

Output:  Y  :  ( Ø.Ø, 5.Ø)  ( 7.Ø, 1.Ø)  (-1.Ø, 1.Ø)
               ( 1.Ø,-1.Ø)  ( 2.Ø, 1.Ø)  (-2.Ø,-2.Ø)
```

```
**********                                                  **********
*        *                                                  *        *
* CDOTC  *          --- COMPLEX INNER PRODUCT ---           * CDOTC  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To sum conjugates of first complex vector times elements of second complex vector.

CALL FORMAT: CW = CDOTC(N,CX,I,CY,J)

PARAMETERS:
- N = Integer element count
- CX = First complex floating-point input vector
- I = Integer element step for CX
- CY = Second complex floating-point input vector
- J = Integer element step for CY
- CW = Complex floating-point output value

DESCRIPTION: CW = SUM((R(CX(m))-I(CX(m)))*(R(CY(m))+I(CY(m))));
for m=1 to N

CW = (Ø.Ø,Ø.Ø) if N<1.

EXAMPLE:

```
N = 2
I = 1
J = 1

CX : (Ø.3Ø,Ø.4Ø) (Ø.ØØ,1.ØØ)
CY : (Ø.3Ø,Ø.4Ø) (8.ØØ,9.ØØ)
CW : (9.25,-8.Ø)
```

Z = Complex floating point input/output vector.
An input only if bit 1 of ISW is set.
IZO = Integer input scalar. Element increment
for Z.

DESCRIPTION: $Z(jz) = r * Z(jz) + s * SUM[X(ix) * Y(iy), i=1,M] \quad j=1,N$

where: ix = (j-1) * IXO + (i-1) * IXI + 1
iy = (j-1) * IYO + (i-1) * IYI + 1
jz = (j-1) * IZO + 1

s = 1.Ø, if ISW[Ø] = Ø
= -1.Ø, if ISW[Ø] = 1

r = Ø.Ø, if ISW[1] = Ø
= 1.Ø, if ISW[1] = 1

X = X , if ISW[2] = Ø
= Conjg(X), if ISW[2] = 1

Y = Y , if ISW[3] = Ø
= Conjg(Y), if ISW[3] = 1

Z = Z , if ISW[4] = Ø
= Conjg(Z), if ISW[4] = 1

and ISW[k] = bit k of ISW.

NOTES: If IZO = Ø, then CDOTN will set Z(1) equal to the accumulated sum of all N dot products. If ISW[1] = 1 also, then input Z(1) will be added to this sum.

Memory words occupied by X may intersect those occupied by Y. In fact, X and Y may coincide. However, memory occupied by Z should not, in general, intersect that occupied by X or Y.

If N < 1, CDOTN returns with no action taken.

If M < 1 and ISW[1] = 1, CDOTN returns with no action taken.

If M < 1 and ISW[1] = Ø, CDOTN returns with Z(j) = Ø.Ø for j = 1 to N.

In general, M < 1 implies a zero sum of products.

```
**********                                                  **********
*        *                                                  *        *
* CDOTU  *          --- COMPLEX DOT PRODUCT ---             * CDOTU  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To compute the inner (unconjugated) product of two complex vectors.

CALL FORMAT: CW = CDOTU(N,CX,I,CY,J)

PARAMETERS:
- N = Integer element count
- CX = First complex floating-point input vector
- I = Integer step for CX
- CY = Second complex floating-point input vector
- J = Integer step for CY
- CW = Complex floating-point scalar output result

DESCRIPTION: CW = SUM(CX(m)*CY(m)); for m=1 to N

CW = (Ø.Ø,Ø.Ø) if N<1.

EXAMPLE:

```
N = 2
I = 1
J = 1

CX : (Ø.3Ø,Ø.4Ø) (Ø.ØØ,1.ØØ)
CY : (Ø.3Ø,-.4Ø) (8.ØØ,9.ØØ)
CW : (-8.75,8.ØØ)
```

```
**********                                              **********
*        *                                              *        *
* CSCAL  *            --- COMPLEX SCALING ---           * CSCAL  *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To multiply each component of a vector by a complex scalar.

CALL FORMAT: CALL CSCAL(N,CA,CX,I)

PARAMETERS:
N = Integer element count
CA = Complex floating-point scalar multiple
CX = Complex floating-point input/output vector
I = Integer step increment for CX

DESCRIPTION: CX(m) = CA*CX(m); for m=1 to N

EXAMPLE:

```
N = 3
I = 1

CA          : ( Ø.Ø, 1.Ø)
CX(INPUT)   : ( 1.Ø, 2.Ø) ( 3.Ø, 4.Ø) ( 5.Ø, 6.Ø)
CX(OUTPUT)  : (-2.Ø, 1.Ø) (-4.Ø, 3.Ø) (-6.Ø, 5.Ø)
```

```
**********                                              **********
*        *                                              *        *
* CSSCAL *          --- REAL TIMES COMPLEXES ---        * CSSCAL *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To multiply the elements of a complex vector by a real scalar.

CALL FORMAT: CALL CSSCAL(N,SA,CX,I)

PARAMETERS:
N = Integer element count for CX
SA = Floating-point input scalar multiple
CX = Complex floating-point input/output vector
I = Integer element step increment for CX

DESCRIPTION: CX(m) = SA*CX(m); for m=1 to N

EXAMPLE:

```
N = 3
I = 1

SA          : Ø.5
CX(INPUT)   : (2.Ø,4.Ø) (6.Ø,8.Ø) (Ø.Ø,1.Ø)
CX(OUTPUT)  : (1.Ø,2.Ø) (3.Ø,4.Ø) (Ø.Ø,Ø.5)
```

```
**********                                                **********
*        *                                                *        *
* ICAMAX *    --- INDEX OF LARGEST COMPLEX ELEMENT ---    * ICAMAX *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To calculate the index of the complex element of largest real plus imaginary magnitude.

CALL FORMAT: IMAX = ICAMAX(N,CX,I)

PARAMETERS:
- N = Integer element count
- CX = Complex floating-point input vector
- I = Integer step increment for CX
- IMAX = Integer value of index with largest components

DESCRIPTION: cmag(CX(IMAX)) = MAX(cmag(CX(m)); m=1 for N where cmag(C) = ABS(R(C))+ABS(I(C)), with 1 < = IMAX < = N. If N < 1, IMAX = Ø.

EXAMPLE:

```
N = 3
I = 1

CX   : ( 3.Ø, 3.Ø) ( 5.Ø,-9.Ø) ( Ø.Ø,13.Ø)
IMAX : 2
```

```
**********                                              **********
*        *                                              *        *
* SASUM  *          --- SUM OF MAGNITUDES ---           * SASUM  *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To sum magnitudes of elements of a real vector.

CALL FORMAT: SW = SASUM(N,SX,I)

PARAMETERS:
N = Integer element count
SX = Floating-point source vector
I = Integer incremental step for SX
SW = Floating-point scalar result

DESCRIPTION: SW = SUM(ABS(SX(m))); for m=1 to N

EXAMPLE:

```
N = 3

SX : -1.Ø  Ø.Ø  5.Ø
SW :  6.Ø
```

```
**********                                              **********
*        *                                              *        *
* SAXPYN *        --- NESTED REAL  A * X + Y ---        * SAXPYN *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To add a scalar multiple of one floating-point vector to another floating-point vector N times, each time for a different pair of vectors and a different scalar. The first vector is a subset of the vector X, and the second vector is a subset of the vector Y. The scalar is an element of the vector A.

CALL FORMAT: CALL SAXPYN(ISW,N,M,A,IAO,X,IXI,IXO,Y,IYI,IYO)

PARAMETERS:

ISW = Integer input scalar. ISW is a function selector switch and is treated as a bit string with the bits numbered from the least significant bit (bit Ø). If a given bit is set (equal to 1), then the function option that corresponds to that bit is selected. Only bit Ø is used in SAXPYN.

Bit Ø: Negate the product term A * X before adding to Y. That is, compute - A * X + Y instead of A * X + Y.

All other bits are ignored.

N = Integer input scalar. Number of A * X + Y operations, i.e., outer loop count.

M = Integer input scalar. Number of elements in each A * X + Y operation, i.e., inner loop count.

A = Floating point input vector. Array of scalars.

IAO = Integer input scalar. Outer loop element increment for A.

X = Floating point input vector. First input vector.

IXI = Integer input scalar. Inner loop element increment for X.

IXO = Integer input scalar. Outer loop element increment for X.

Y = Floating point input/output vector. Second input vector on input. Output vector on output.

IYI = Integer input scalar. Inner loop element increment for Y.

IYO = Integer input scalar. Outer loop element increment for Y.

```
          A  :  3.Ø  -1.Ø   2.Ø

          X  :  2.Ø   3.Ø

          Y  :  7.Ø   6.Ø         2.Ø   3.Ø         5.Ø   6.Ø

Output:   Y  : 13.Ø  15.Ø         Ø.Ø   Ø.Ø         9.Ø  12.Ø
```

```
**********                                                **********
*        *                                                *        *
* SCNRM2 *          --- COMPLEX EUCLIDEAN NORM ---        * SCNRM2 *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To compute the square root of sum of squares of elements of a complex floating-point vector.

CALL FORMAT: SW = SCNRM2(N,CX,I)

PARAMETERS:
N = Integer element count
CX = Complex floating-point input vector
I = Integer step increment
SW = Floating-point scalar output result

DESCRIPTION: SW = SQRT(SUM(R(CX(m))**2 + I(CX(m))**2));
for m=1 to N

EXAMPLE:

```
N = 2
I = 1

CX : (Ø.Ø,3.Ø) (4.Ø,Ø.Ø)
SW : 5.Ø
```

```
**********                                                  **********
*        *                                                  *        *
*  SDOT  *      --- DOT PRODUCT OF REAL VECTORS ---         *  SDOT  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To compute the inner (dot) product of two vectors.

CALL FORMAT: SW = SDOT(N,SX,I,SY,J)

PARAMETERS:
N = Integer element count for SX and SY
SX = Floating-point input vector
I = Integer element step for SX
SY = Floating-point input vector
J = Integer element step for SY
SW = Floating-point output value

DESCRIPTION: SW=SUM(SX(m)*SY(m)); for m=1 to N

SW=Ø.Ø if N<1.

EXAMPLE:

N = 3

SX : 1.Ø 2.Ø 3.Ø
SY : 4.Ø Ø.5 Ø.Ø
SW : 5.Ø

DESCRIPTION: Z(jz) = r * Z(jz) + s * SUM[X(ix) * Y(iy), i=1,M] j=1,N

where: ix = (j-1) * IXO + (i-1) * IXI + 1
iy = (j-1) * IYO + (i-1) * IYI + 1
jz = (j-1) * IZO + 1

s = 1.Ø, if ISW[Ø] = Ø
= -1.Ø, if ISW[Ø] = 1

r = Ø.Ø, if ISW[1] = Ø
= 1.Ø, if ISW[1] = 1

and ISW[k] = bit k of ISW.

NOTES: If IZO = Ø, then SDOTN will set Z(1) equal to the accumulated sum of all N dot products. If ISW[1] = 1 also, then input Z(1) will be added to this sum.

Memory words occupied by X may intersect those occupied by Y. In fact, X and Y may coincide. However, memory occupied by Z should not, in general, intersect that occupied by X or Y. For sample applications, see Sections D.4.9 and D.4.11.

If N < 1, SDOTN returns with no action taken.

If M < 1 and ISW[1] = 1, SDOTN returns with no action taken.

If M < 1 and ISW[1] = Ø, SDOTN returns with Z(j) = Ø.Ø for j = 1 to N.

In general, M < 1 implies a zero sum of products.

EXAMPLE:

Input: ISW = Ø
N = 2
M = 3
IXI = 2
IXO = 1
IYI = 1
IYO = Ø
IZO = 1

X : 3.Ø 2.Ø -1.Ø 1.Ø Ø.Ø -2.Ø

Y : 1.Ø 2.Ø 3.Ø

Output: Z : 1.Ø -2.Ø

```
**********                                              **********
*        *                                              *        *
*  SROT  *          --- PLANE ROTATION ---              *  SROT  *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To perform two dimensional rotations.

CALL FORMAT: CALL SROT(N,SX,I,SY,J,C,S)

PARAMETERS:
- N = Integer count of elements in SX and SY
- SX = Floating-point input vector of first components
  = (On output) first components of rotated vector
- I = Integer step increment for SX
- SY = Floating-point input vector of second components
  = (On output) second components of rotated vector
- J = Integer step increment for SY
- C = Floating-point input scalar cosine
- S = Floating-point input scalar sine

DESCRIPTION:
SX(m) = C*SX(m)+S*SY(m)
SY(m) =-S*SX(m)+C*SY(m); for m=1 to N

EXAMPLE:

N = 3

C	:	Ø.3		
S	:	Ø.4		
SX(INPUT)	:	1.Ø	2.Ø	3.Ø
SY(INPUT)	:	Ø.Ø	1.Ø	2.Ø
SX(OUTPUT)	:	Ø.3	1.Ø	17.Ø
SY(OUTPUT)	:	-Ø.4	-5.Ø	-6.Ø

```
**********                                                **********
*        *                                                *        *
* SROTM  *        --- MODIFIED GIVENS ROTATIONS ---       * SROTM  *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To perform two-dimensional rotations using the rotation matrix constructed from a parameter vector according to the modified Givens scheme.

CALL FORMAT: CALL SROTM(N,SX,INCX,SY,INCY,PARAM)

PARAMETERS:
- N = Integer element count
- SX = Floating-point input/output vector of first components
- INCX = Integer element step for SX
- SY = Floating-point input/output vector of second components
- INCY = Integer element step for SY
- PARAM = Five element floating-point input vector used to construct the rotation matrix
  H = H11 H12
      H21 H22.

DESCRIPTION: SX(m) = H11*SX(m) + H12*SY(m)
SY(m) = H21*SX(m) + H22*SY(m), for m=1 to N, where

H11,	H12,	H21,	H22	=
PARAM(2),	1.Ø,	-1.Ø,	PARAM(5)	or
1.Ø,	PARAM(4),	PARAM(3),	1.Ø	or
PARAM(2),	PARAM(4),	PARAM(3),	PARAM(5)	according to

whether PARAM(1) = 1.Ø or Ø.Ø or -1.Ø, respectively.

If PARAM(1) is not equal to zero, one, or minus one, the routine returns with no action performed. This is equivalent to having the identity matrix as the rotation matrix.

EXAMPLE:

N = 5

SX(input)	:	Ø.Ø	1.Ø	-2.Ø	2.Ø	-4.Ø
SY(input)	:	Ø.Ø	Ø.Ø	2.Ø	-2.Ø	-2.Ø
PARAM	:	-1.Ø	1.Ø	-1.Ø	1.Ø	1.Ø
SX(output)	:	Ø.Ø	1.Ø	Ø.Ø	Ø.Ø	-6.Ø
SY(output)	:	Ø.Ø	-1.Ø	4.Ø	-4.Ø	2.Ø

Rescaling continues until D1 and D2 are within the window.

Output parameters PARAM(1,2,3,4,5) = (-1.Ø,H11,H21,H12,H22) and D1,D2,B1 are updated according to the scaling factors above.

EXAMPLE:

```
D1,D2,B1,B2 (input) :  4.ØØØ  3.ØØØ  2.ØØØ  1.ØØØ

D1,D2,B1    (output) :  3.368  2.526  2.375
PARAM       (output) :  Ø.ØØØ  Ø.ØØØ -Ø.5ØØ  Ø.375  Ø.Ø
```

```
**********                                              **********
*        *                                              *        *
* SSWAP  *          --- INTERCHANGES VECTORS ---        * SSWAP  *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To interchange elements of two real vectors.

CALL FORMAT: CALL SSWAP(N,SX,I,SY,J)

PARAMETERS:
N = Integer element count
SX = Floating-point first vector for swap
I = Integer element step for SX
SY = Floating-point second vector for swap
J = Integer element step for SY

DESCRIPTION: SX(m) :=: SY(m); for m=1 to N

EXAMPLE:

N = 3

```
SX(INPUT)  :  1.Ø  2.Ø  3.Ø
SY(INPUT)  :  9.Ø  8.Ø  7.Ø
SX(OUTPUT) :  9.Ø  8.Ø  7.Ø
SY(OUTPUT) :  1.Ø  2.Ø  3.Ø
```

```
**********                                                  **********
*        *                                                  *        *
*  ABP1  *  --- ADAMS-BASHFORTH PREDICTOR (ORDER 1) ---     *  ABP1  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To solve an initial value problem for a set of ordinary differential equations, using a first order predictor (Euler's) method.

CALL FORMAT: CALL ABP1(N,H,Y,F,YP)

PARAMETERS:
- N = Integer element count, number of equations
- H = Floating-point input scalar step size for t
- Y = Floating-point input vector of dependent variables Y(t)
- F = Floating-point input vector of derivative elements dY/dt =F(t,Y(t))
- YP = Floating-point output vector of predicted variables Y(t+H)

DESCRIPTION: For the system of equations dY/dt=F(t,Y(t)), the solution at t'=t+H is given by

YP(m) = Y(m) + H*F(m); for m=1 to N

This provides an explicit first order solution to the initial value problem for a given function at time t'=t+H, given the values of the function and its derivative at time t. The evaluation of the next derivative, corresponding to F(t+H,Y(t+H)) at the new time point, t'=t+2*H follows similarly.

EXAMPLE:

```
N = 3
H = Ø.1

Y   :   1.Ø   2.Ø   3.Ø
F   :   1.Ø   1.Ø   1.Ø

YP  :   1.1   2.1   3.1
```

```
**********                                                    **********
*        *                                                    *        *
*  ABP3  *  --- ADAMS-BASHFORTH PREDICTOR (ORDER 3) ---      *  ABP3  *
*        *                                                    *        *
**********                                                    **********
```

PURPOSE: To solve an initial value problem for a set of ordinary differential equations, using Adams' third order predictor method.

CALL FORMAT: CALL ABP3(N,H,Y,F,F1,F2,YP)

PARAMETERS:
- N = Integer element count, number of equations
- H = Floating-point input scalar step size for t
- Y = Floating-point input vector of dependent variables Y(t)
- F = Floating-point input vector of derivative elements dY/dt=F(t,Y(t))
- F1 = Floating-point input vector of derivative functions at preceeding time t1=t-H
- F2 = Floating-point input vector of derivative functions at preceeding time t2=t-2H
- YP = Floating-point output vector of predicted variables Y(t+H)

DESCRIPTION: For the system of equations dY/dt=F(t,Y(t)), the solution at t'=t+H is given by

```
YP(m) = Y(m) + (H/12)*(23*F(m)-16*F1(m)+5*F2(m));

     for m=1 to N
```

This provides an explicit third order solution to the initial value problem for a given function at time t'=t+H, given the values of the function and its derivative at t and its derivatives F1 and F2 at times t1=t-H and t2=t-2H, respectively. Evaluation of the next derivative, corresponding to F(t+H,Y(t+H)) at the new time point, t'=t+2*H follows similarly.

EXAMPLE:

```
N = 3
H = Ø.1

Y   :   1.Ø   2.Ø   3.Ø
F   :   3.Ø   3.Ø   3.Ø
F1  :   2.Ø   2.Ø   2.Ø
F2  :   1.Ø   1.Ø   1.Ø

YP  :   1.35  2.35  3.35
```

EXAMPLE:

```
N = 3
H = Ø.1

Y    :   1.Ø   2.Ø   3.Ø
F    :   3.Ø   3.Ø   3.Ø
F1   :   2.Ø   2.Ø   2.Ø
F2   :   1.Ø   1.Ø   1.Ø
F3   :   4.Ø   4.Ø   4.Ø

YP   :   1.2   2.2   3.2
```

DESCRIPTION: This routine integrates a set of N first order differential equations from t=A to t=B, given the initial values Y(t) and the values of the derivative functions dY/dt=F(t,Y(t)) calculated in the user supplied routine DFUNF(T,N,Y,F). The step size H is regulated to keep the maximum local error less than EPS. The maximum number of steps taken per call is limited by MAXIT. The maximum step size is limited by HMAX. Error return codes are provided to monitor the progress of the algorithm.

REFERENCE: Burden,R.L., Faires,J.D., and Reynolds,A.C., "Numerical Analysis", Prindle, Weber & Schmidt, Inc., Boston, 1978: "Adams Variable Step-size Predictor-Corrector" Algorithm 6.5

EXAMPLE:

DFUNF (user supplied APFTN64 subroutine):

```
      SUBROUTINE DFUNF(T,N,Y,F)
C
C   ***  DFUNF  *** SAMPLE APFTN64 ROUTINE ***
C
      DIMENSION Y(N), F(N)
C
      DO 1Ø I=1,N
         F(I) = -Y(I) + T + 1.Ø
   1Ø CONTINUE
C
C   CORRESPONDS TO SOLUTIONS OF THE FORM
C
C   Y(T) = YØ * EXP(-T) + T
C
      RETURN
      END
```

INPUT:

```
A     =   Ø.Ø
B     =   3.Ø
N     =   5
HMAX  =   Ø.2
MAXIT = 1ØØ
EPS   =   1.ØE-6

Y(1,1), ..., Y(5,1):
1.Ø  2.Ø  3.Ø  4.Ø  5.Ø
```

```
**********                                                  **********
*        *                                                  *        *
*  AMC1  *   --- ADAMS-MOULTON CORRECTOR (ORDER 1) ---      *  AMC1  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To solve an initial value problem for a set of ordinary differential equations, using a first order corrector (backward Euler) method.

CALL FORMAT: CALL AMC1(N,H,Y,FP,YP)

PARAMETERS:
- N = Integer element count, number of equations
- H = Floating-point input scalar step size for t
- Y = Floating-point input vector of dependent variables Y(t)
- FP = Floating-point input vector of derivative elements dY/dt=F(t+H,Y(t+H))
- YP = Floating-point output vector of predicted variables Y(t+H)

DESCRIPTION: For the system of equations dY/dt=F(t,Y(t)), the solution at t'=t+H is given by

YP(m) = Y(m) + H*FP(m); for m=1 to N

This provides an implicit first order solution to the initial value problem for a given function at time t'=t+H, given the values of the function and its derivative at time t. The evaluation of the next derivative, corresponding to F(t+H,Y(t+H)) at the new time point, t'=t+2*H follows similarly.

EXAMPLE:

```
N = 3
H = Ø.1

Y   :   1.Ø   2.Ø   3.Ø
FP  :   1.Ø   1.Ø   1.Ø

YP  :   1.1   2.1   3.1
```

```
**********                                                  **********
*        *                                                  *        *
*  AMC3  *   --- ADAMS-MOULTON CORRECTOR (ORDER 3) ---      *  AMC3  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To solve an initial value problem for a set of ordinary differential equations, using Adams' third order corrector method.

CALL FORMAT: CALL AMC3(N,H,Y,F,F1,FP,YP)

PARAMETERS:
- N = Integer element count, number of equations
- H = Floating-point input scalar step size for t
- Y = Floating-point input vector of dependent variables Y(t)
- F = Floating-point input vector of derivative elements dY/dt=F(t,Y(t))
- F1 = Floating-point input vector of derivative functions at preceeding time t1=t-H
- FP = Floating-point input vector of derivative functions estimated for t'=t+H
- YP = Floating-point output vector of predicted variables Y(t+H)

DESCRIPTION: For the system of equations dY/dt=F(t,Y(t)), the solution at t'=t+H is given by

```
YP(m) = Y(m) + (H/12)*(8*F(m)-F1(m)+5*FP(m));

     for m=1 to N
```

This provides an implicit third order solution to the initial value problem for a given function at time t'=t+H, given the values of the function and its derivative at t, as well as, its derivatives at times t1=t-H and t'=t+H, corresponding to F1 and FP. Evaluation of the next derivative, corresponding to F(t+H,Y(t+H)) at the new time point, t'=t+2*H follows similarly.

EXAMPLE:

```
N = 3
H = Ø.1

Y   :   1.Ø   2.Ø   3.Ø
F   :   2.Ø   2.Ø   2.Ø
F1  :   1.Ø   1.Ø   1.Ø
FP  :   3.Ø   3.Ø   3.Ø
YP  :   1.25  2.25  3.25
```

EXAMPLE:

```
N = 3
H = Ø.1

Y   :   1.Ø   2.Ø   3.Ø
F   :   3.Ø   3.Ø   3.Ø
F1  :   2.Ø   2.Ø   2.Ø
F2  :   1.Ø   1.Ø   1.Ø
FP  :   4.Ø   4.Ø   4.Ø

YP  :   1.35  3.35  3.35
```

BRK(N,2) = Ø.Ø
and an input coordinate value x, BIN uses a binary

1. The index IX that locates x within the coordinate value breakpoint table such that

   x(IX) <= x < x(IX+1)

2. The product DR = D(IX) * R(IX) where

   D(IX) = x(IX)-x

   R(IX) = 1/(x(IX+1)-x(IX))

When a program makes repeated calls to a breakpoint search routine (i.e., BIN or STEP), BIN should be used if it is suspected that the input coordinate x varies rapidly with respect to the values in the coordinate value breakpoint table. In this case, the binary (successive interval halving) search employed by BIN is more efficient than the step (nearest neighbor) search used by STEP.

Refer to the function generation in Appendix E for additional information.

EXAMPLE:

N = 3

BRK = 1.Ø 2.Ø 7.Ø 1.Ø Ø.2 Ø.Ø
X = 2.1
IX = 2
DR = -Ø.Ø2

NOTE

If x <= x(1) then IX = 1

If x >= x(N) then IX = N-1

DESCRIPTION: I(I+1), for I = Ø to N-1, is the value of the Ith modified Bessel functions of the first kind evaluated at the point X. Refer to equation 9.6.3 of Abramowitz and Stegun for the defining equation.

K(I+1), for I = Ø to N-1, is the value of the Ith modified Bessel functions of the second kind evaluated at the point X. Refer to equation 9.6.4 of Abramowitz and Stegun for the defining equation.

Warnings and errors are reported to the calling routine via IERR. If CBEIK completes normally, then IERR is set to zero.

Warning condition codes are all between 1 and 99 inclusive. The possible warning values and their meanings are as follows:

IERR = 1	N is too large for computation of outputs. In most instances, ABS(X) < 4ØØ.Ø; this means that the Nth order outputs exceed the dynamic range of the machine. A suitable N is calculated, the Bessel function values are computed up to this new N, and the new N value is returned.

Error condition codes are all greater than or equal to 1ØØ. The possible error values and their meanings are as follows:

IERR = 1ØØ	ISTEP and/or KSTEP are equal to -1,Ø, or 1.
IERR = 1Ø1	X does not lie within the boundary of (+/-6ØØ, +/-6ØØi).
IERR = 1Ø2	N is equal to 1. N must be greater than or equal to 2.

References: Abramowitz, M., and Stegun, I., "Handbook of Mathematical Functions", Ninth printing, pp.358-36Ø.

Mason, J.P., "Cylindrical Bessel Functions for a Large Range of Complex Arguments", Computer Physics Communications, 3Ø(1983), pp.1-11.

********** **********
* CBEJYH * --- COMPLEX BESSEL J, Y, AND H --- * CBEJYH *
********** **********

PURPOSE: To compute the complex Bessel functions of integer order of the first kind, second kind, and one of the Hankel functions at a point X.

CALL FORMAT: CALL CBEJYH (X, N, J, JSTEP, Y, YSTEP, H, HSTEP, IERR)

PARAMETERS:

X = Complex input scalar
The point at which to evaluate all functions. This is restricted to the portion of the complex plane bounded by (+/-6ØØ,+/-6ØØi). It can take on the values (+/-6ØØ, +/-6ØØi).

N = Integer input/output scalar
On input, the number of function values to evaluate. If N <= Ø, then this routine returns with no action. If N = 1, then an error is reported. Note that the zero order function values are stored in the first elements of the complex output vectors.
On output, the actual number of Bessel functions computed. The input value of N is modified only in the case where IERR = 1, if too many function values were requested. If IERR is not equal to 1, then N is not modified on return to the calling routine.

JSTEP = Integer input scalar
Element step for J. This can be any value except -1, Ø, or 1. This is the number of words to skip between complex elements.

YSTEP = Integer input scalar
Element step for Y. This can be any value except -1, Ø, or 1. This is the number of words to skip between complex elements.

HSTEP = Integer input scalar
Element step for H. This can be any value except -1, Ø, or 1. This is the number of words to skip between complex elements.

J = Complex output vector
The function values of functions Ø through N-1 for Bessel functions of the first kind.

Y = Complex output vector
The function values of functions Ø through N-1 for Bessel functions of the second kind.

H = Complex output vector
The function values of functions Ø through N-1 for one of the Hankel functions. If the sign of the imaginary part of X is positive, then the

Note: If the second Hankel function is desired when the imaginary part of X is nonnegative, it can be computed with the following equation:

H2 = J-iY

Similarly, the first Hankel function can be computed when the imaginary part of X is negative by the following equation:

H1 = J+iY

References: Abramowitz, M., and Stegun, I., "Handbook of Mathematical Functions", Ninth printing, pp.358-360.

Mason, J.P., "Cylindrical Bessel Functions for a Large Range of Complex Arguments", Computer Physics Communications, 30(1983), pp.1-11.

EXAMPLE:

```
N      =  3
JSTEP  =  2
YSTEP  =  2
HSTEP  =  2

J      :  ( Ø.937ØØ847680Ø6Ø30E+ØØØ, -Ø.4965299476ØØ9122E+ØØØ),
          ( Ø.6141ØØ33492290Ø4E+ØØØ,  Ø.365Ø28Ø28827Ø88E+ØØØ),
          ( Ø.415798869439622E-ØØ1,  Ø.247397641513306E+ØØØ)

Y      :  ( Ø.445474488934634E+ØØØ,  Ø.71Ø158582ØØ150Ø5E+ØØØ),
          (-Ø.657694535589279E+ØØØ,  Ø.6298Ø1ØØ399Ø9Ø7E+ØØØ),
          (-Ø.473368Ø2Ø533ØØ7E+ØØØ,  Ø.577336957578681E+ØØØ)

H      :  ( Ø.227449894804525E+ØØØ, -Ø.51Ø554586744886E-ØØ1),
          (-Ø.1564Ø669Ø68ØØ27E-ØØ1, -Ø.292666506762191E+ØØØ),
          (-Ø.535757Ø7Ø634719E+ØØØ, -Ø.2259703790197ØØE+ØØØ)

IERR   =  Ø
```

EXAMPLE:

See Appendix E for function generation.

```
F(x)=F(x(i))+(F(x(i+1))-F(x(i)))*(x-x(i))/(x(i+1)-x(i))
```

where

```
x(i)        = x-coordinate value at the i-th
              x-coordinate breakpoint
x(i+1)      = x-coordinate value at the (i+1)-th
              x-coordinate breakpoint
x           = Input x-coordinate value where the
              interpolated function value is desired
F(x(i))     = Function value at x(i)
F(x(i+1))   = Function value at x(i+1)
F(x)        = Interpolated function value at x
              and x(i) <= x < x(i+1)
```

EXAMPLE:

See Appendix E for function generation.

desired functions, storing them in FVAL. Refer to the function generation in Appendix E for additional information.

```
F(x)=F(x(i))+(F(x(i+1))-F(x(i)))*(x-x(i))/(x(i+1)-x(i))
```

where

x(i)	=	x-coordinate value at the i-th x-coordinate breakpoint
x(i+1)	=	x-coordinate value at the (i+1)-th x-coordinate breakpoint
x	=	Input x-coordinate value where the interpolated function value is desired
F(x(i))	=	Function value at x(i)
F(x(i+1))	=	Function value at x(i+1)
F(x)	=	Interpolated function value at x and x(i) <= x < x(i+1)

EXAMPLE:

See in Appendix E on function generation.

DESCRIPTION: FUN4 uses the indexes IX, IY, IZ and IW from the breakpoint searches and the values NX, NY, NZ, and NW to find the first function value pairs in the function value breakpoint table. It then performs a linear interpolation between them by applying the formula given below eight times over the x-axis, four times over the y-axis, twice over the z-axis, and once over the w-axis. FUN4 repeats the process for all the desired functions, storing the computed function values in FVAL. Refer to the function generation in Appendix E for additional information.

F(x)=F(x(i))+(F(x(i+1))-F(x(i)))*(x-x(i))/(x(i+1)-x(i))

where

x(i)	= x-coordinate value at the i-th x-coordinate breakpoint
x(i+1)	= x-coordinate value at the (i+1)-th x-coordinate breakpoint
x	= Input x-coordinate value where the interpolated function value is desired
F(x(i))	= Function value at x(i)
F(x(i+1))	= Function value at x(i+1)
F(x)	= Interpolated function value at x and x(i) <= x < x(i+1)

EXAMPLE:

See Appendix E for function generation.

C.W.Gear, "Numerical Initial Value Problem in Ordinary Differential Equations", Prentice-Hall, 1971.

RKGIL performs integration for given time, step size, and integration steps. The right-hand subroutine DFUN can be coded in either APFTN64 or APAL64. The parameter-passing method employed by RKGIL requires that DFUN be coded in APFTN64. As such, RKGIL relies on assumed procedure entry conventions, because APFTN64 automatically generates code using this convention. If DFUN is written in APAL64, the user must resolve the parameters correctly.

At output, vector V contains the numerical solutions while TØ contains the new value of the independent variable; i.e., TØ=TØ+M*H.

Repeated calls to RKGIL can cause stability problems. So the user must be on guard against instability and must take care specifying the H parameter.

EXAMPLE:

Solve the following second-order differential equation

```
Y'' = -4.Ø*Y
```

with initial conditions

```
Y(Ø.Ø)  = 1.Ø

Y'(Ø.Ø) = Ø.Ø
```

starting at TØ = Ø.Ø with H = Ø.1 for 32 iterations.

An equivalent system of first-order differential equations can be written in the form

```
DV(1) = V(2)

DV(2) = -4.Ø*V(1)
```

with initial conditions at the point Ø.Ø of

```
V(1) = 1.Ø

V(2) = Ø.Ø
```

```
**********                                                  **********
*        *                                                  *        *
* RKGTF  *   --- R-K-GILL-THOMPSON INTEG.(ORDER 4) ---      * RKGTF  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To solve an initial value problem for a set of ordinary differential equations, using the fourth order Runge-Kutta-Gill method as described by Thompson.

CALL FORMAT: CALL RKGTF(T,N,Y,F,Q,H,M)

PARAMETERS:
- T = Floating-point input scalar independent variable, initial value of t
- N = Integer input element count, number of equations, dimension of Y, F and Q
- Y = Floating-point input/output vector of dependent variables (Y(t))
- F = Floating-point working vector of derivative functions dY/dt=F(t,Y(t))
- Q = Floating-point working vector used for temporary storage (must have length N)
- H = Floating-point input scalar step size for t
- M = Integer input scalar number of integration steps to be performed

DESCRIPTION: For the system of equations dY/dt=F(t,Y(t)), the solution at each step is given by

```
Y(m) = Y(m)
     +(H/6)*(k1+(2-sqrt(2))*k2+(2+sqrt(2))*k3+k4)
```

for m=Ø to N-1, where

```
k1=F(T,Y)

k2=F(T+H/2,Y+Ø.5*H*k1)

k3 = F(T+H/2,Y+Ø.5*(-1+sqrt(2))*H*k1
                +Ø.5*( 2-sqrt(2))*H*k2)

k4 = F(T+H,Y-Ø.5*sqrt(2)*H*k2
           +Ø.5*(2+sqrt(2))*H*k3)
```

while the independent variable is advanced by H until T = T + M*H.

```
**********                                                **********
*        *                                                *        *
* ROT3   *      --- 3D ROTATION MATRIX, 3-ANGLE ---       * ROT3   *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To form a three-dimentional rotation matrix as a product of three successive rotations about any three orthogonal axes.

CALL FORMAT: CALL ROT3(I,A,J,B,K,C,R)

PARAMETERS:
- I = Integer input scalar axis indicator (plus or minus 1=x, 2=y, 3=z)
- A = Floating-point input scalar angle(radians) of rotation about axis I
- J = Integer input scalar axis indicator (plus or minus 1=x, 2=y, 3=z)
- B = Floating-point input scalar angle(radians) of rotation about axis J
- K = Integer input scalar axis indicator (plus or minus 1=x, 2=y, 3=z)
- C = Floating-point input scalar angle(radians) of rotation about axis K
- R = Floating-point output rotation matrix (3x3 matrix stored in column order)

DESCRIPTION: This routine calculates a 3x3 matrix as a product of three rotations about any three orthogonal axes:

R(matrix) = R(K,C)xR(J,B)xR(I,A)

where $R(1,w) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(w) & \sin(w) \\ 0 & -\sin(w) & \cos(w) \end{bmatrix}$

$R(2,w) = \begin{bmatrix} \cos(w) & 0 & -\sin(w) \\ 0 & 1 & 0 \\ \sin(w) & 0 & \cos(w) \end{bmatrix}$

and $R(3,w) = \begin{bmatrix} \cos(w) & \sin(w) & 0 \\ -\sin(w) & \cos(w) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

```
**********                                                  **********
*        *                                                  *        *
*  SCS1  *   --- SCALAR COS/SIN, TM INTERP.(ORD 1) ---      *  SCS1  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To rapidly calculate the cosine and sine of an angle(radians) using values stored in TMROM.

CALL FORMAT: CALL SCS1(A,CA,SA)

PARAMETERS:
A = Floating-point input scalar angle(radians)
CA = Floating-point output scalar cosine(A)
SA = Floating-point output scalar sine(A)

DESCRIPTION: CA = COS(A), SA = SIN(A)

by interpolation of values stored in TMROM using a first order Taylor's series approximation. The returned values are accurate to approximately seven decimal digits.

NOTE: For 15 decimal digits of accuracy at a slight decrease in speed, see the routine SINCOS.

EXAMPLE:

A = 1.Ø

CA = Ø.54Ø3Ø23
SA = Ø.8414710

An input coordinate value x, and the index IX from a previous call to STEP or BIN, STEP uses a step search to determine the following:

1. The index IX that locates x within the coordinate value breakpoint table such that

   x(IX) <= x < x(IX+1)

2. The product DR = D(IX) * R(IX) where

   D(IX) = x(IX)-x

   R(IX) = 1/(x(IX+1)-x(IX))

When a program makes repeated calls to a breakpoint search routine (i.e., BIN or STEP), STEP should be used if it is suspected that the input coordinate x varies slowly with respect to the values in the coordinate value breakpoint table. STEP's nearest neighbor searching is more efficient than the binary (successive interval halving) search used by BIN.

At the outset, if no a priori knowledge of the value of x is available, the first call to STEP should set IX = N/2. An alternative strategy is to make the first call to BIN, which initializes IX, and then make subsequent calls to STEP.

Refer to the function generation in Appendix E for additional information.

EXAMPLE:

```
N = 3

BRK  =  1.Ø  2.Ø  7.Ø  1.Ø  Ø.2  Ø.Ø
X    =  2.1
IX   =  2
DR   = -Ø.Ø2
```

NOTE

If x <= x(1) then IX = 1

If x >= x(N) then IX = N-1

```
**********                                                **********
*        *                                                *        *
* CONNMO *      --- NMO WITH CONSTANT VELOCITY ---        * CONNMO *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To apply normal moveout (NMO), with constant velocity, to a seismic trace.

CALL FORMAT: CALL CONNMO(D,N,X,V,SR,NNMO)

PARAMETERS:

- D = Floating-point output vector of trace sample times.
- N = Integer input scalar; element count for D.
- X = Floating-point input scalar; offset distance in feet.
- V = Floating-point input scalar; velocity in feet.
- SR = Floating-point input scalar; sample rate (ms).
- NNMO = Integer output scalar; index of initial sample of zero-fill in destination trace.

DESCRIPTION: The normal moveout computation is described in seismic signal processing references, such as:

"Introduction to Geophysical Prospecting"
Dobrin, M.B.,
McGraw-Hill, Inc.,
New York, N.Y., 1976,
pp. 201-254.

"Geophysical Signal Analysis"
Robinson, E.A and Treitel, S.,
Prentice-Hall, Inc.,
Englewood Cliffs, N.J., 1980,
pp. 1-35.

The square-root computation inherent in the process is accomplished with one iteration of the Newton-Raphson method.

Using a normal moveout process as defined by X, V, and SR, destination trace D is filled with the times from which to interpolate the adjusted trace values.

The initial sample value of zero-fill in the destination trace is returned in parameter NNMO. A value of N+1 for NNMO indicates no zero-fill.

```
**********                                          **********
*        *                                          *        *
*  IIR3Ø *          --- RECURSIVE FILTER ---        *  IIR3Ø *
*        *                                          *        *
**********                                          **********
```

PURPOSE: To perform a recursive digital filter with up to 3Ø poles and 3Ø zeros.

CALL FORMAT: CALL IIR3Ø(A,I,B,C,K,N,NZ,NP)

PARAMETERS:

A = Floating-point input vector of length N+NZ.
Contains the data to be filtered. It will be assumed that A is indexed from -NZ to N-1.

I = Integer input scalar.
Element step for vector A.

B = Floating-point input vector of length NZ+NP+1.
Contains the coefficients of the filter. It will be assumed that B is indexed from Ø to NZ+NP. B(Ø) contains the scalar multiple coefficient, B(1) to B(NZ) contain the coefficients of the zeros, and B(NZ+1) to B(NZ+NP) contain the coefficients of the poles.

C = Floating-point input/output vector of length N+NP.
Contains the filtered data. It will be assumed that C is indexed from -NP to N-1. On input, C(-NP) to C(-1) contain the initial values. On output, the computed values are contained in C(Ø) to C(N-1).

K = Integer input scalar.
Element step for vector C.

N = Integer input scalar.
Element count.

NZ = Integer input scalar.
Number of zeros.

NP = Integer input scalar.
Number of poles.

DESCRIPTION: Performs a recursive (IIR - Infinite Impulse Response) digital filtering difference equation as follows:

$$C(t) = \sum_{j=0}^{NZ} B(j) * A(t-j) - \sum_{m=1}^{NP} B(m+NZ) * C(t-m), \quad \text{for } t = 0 \text{ to } N-1$$

where the dimensions of the arrays are A(-NZ:N-1), B(Ø:NZ+NP), C(-NP:N-1). The second sum equals zero if NP = Ø.

```
**********                                                  **********
*        *                                                  *        *
* KSMLV  *    --- K-TH SMALLEST ELEMENT IN VECTOR ---       * KSMLV  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To find the k-th smallest element of a vector.

CALL FORMAT: CALL KSMLV(A,N,K,W,C)

PARAMETERS:
- A = Floating-point input vector
- N = Integer element count for A
- K = Order of the element to be selected; K=1 will select the smallest element; K=N will select the largest element; K=INT((N+1)/2) will select the median element.
- W = Work-space vector; the size of the work space must be equal to N
- C = Floating-point output scalar

DESCRIPTION: C = k-th smallest element of A(m), m1 to N.

The k-th smallest element of the vector stored in Main Memory starting at location A is found using an application of the divide and conquer strategy. The algorithm implemented is as described by Aho, Hopcroft, and Ullman: THE DESIGN AND ANALYSIS OF COMPUTER ALGORITHMS, Addison-Wesley, 1974, pp. 97-99. The resultant element is stored into Main Memory at location C. The original contents of the input vector are lost.

The speed of this routine is data dependent.

EXAMPLE:

```
N = 8
K = 3

A : 1.Ø  5.Ø  2.Ø  -1.Ø  3.Ø  -3Ø.6  1Ø.7  5.Ø
C : 1.Ø
```

EXAMPLE:

```
SR   =  2.Ø
N    = 2Ø
NNMO = 14

C:
 1.Ø  2.Ø  3.Ø  4.Ø  5.Ø  6.Ø  7.Ø  8.Ø  9.Ø 1Ø.Ø
11.Ø  8.Ø  7.Ø  6.Ø  5.Ø  4.Ø  3.Ø  2.Ø  1.Ø  Ø.Ø

D: (input)
 3.Ø  6.Ø  9.Ø 12.Ø 15.Ø 18.Ø 21.Ø 24.Ø 27.Ø 3Ø.Ø
33.Ø 36.Ø 39.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø

D: (output)
 2.5  4.Ø  5.5  7.Ø  8.5 1Ø.Ø  9.5  7.Ø  5.5  4.Ø
 2.5  1.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
```

EXAMPLE:

```
SR    =  2.Ø
N     = 2Ø
NNMO = 14

C:
 1.Ø  2.Ø  3.Ø  4.Ø  5.Ø  6.Ø  7.Ø  8.Ø  9.Ø 1Ø.Ø
11.Ø  8.Ø  7.Ø  6.Ø  5.Ø  4.Ø  3.Ø  2.Ø  1.Ø  Ø.Ø

D: (input)
 3.Ø  6.Ø  9.Ø 12.Ø 15.Ø 18.Ø 21.Ø 24.Ø 27.Ø 3Ø.Ø
33.Ø 36.Ø 39.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø

D: (output)
 2.5  4.Ø  5.5  7.Ø  8.5 1Ø.Ø 1Ø.Ø  7.Ø  5.5  4.Ø
 2.5  1.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
```

```
**********                                              **********
*        *                                              *        *
* RESNMO *        --- RESIDUAL NORMAL MOVEOUT ---       * RESNMO *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To stretch or squeeze a seismic trace via linear interpolation.

CALL FORMAT: CALL RESNMO(A, B, C, NI, SR, D, NO, NNMO)

PARAMETERS:

- A = Floating-point input vector; source trace of sample values.
- B = Floating-point input vector of input control times (ms).
- C = Floating-point input vector of output control times (ms).
- NI = Integer element count for B and C.
- SR = Floating-point input scalar; sample rate (ms).
- D = Floating-point output trace vector of sample values.
- NO = Integer element count for D.
- NNMO = Integer output scalar; index of initial sample of zero-fill in destination trace D.

DESCRIPTION: The normal moveout computation is described in seismic signal processing references, such as:

"Introduction to Geophysical Prospecting"
Dobrin, M.B.,
McGraw-Hill, Inc.,
New York, N.Y., 1976,
pp. 201-254.

"Geophysical Signal Analysis"
Robinson, E.A and Treitel, S.,
Prentice-Hall, Inc.,
Englewood Cliffs, N.J., 1980,
pp. 1-35.

Using a stretching/squeezing function as defined by B, C, and SR, source trace C is converted into destination trace D.

The initial sample value of zero-fill in the destination trace is returned in parameter NNMO. A value of N+1 for NNMO indicates no zero-fill.

The speed of this routine is data dependent.

```
**********                                                **********
*        *                                                *        *
* TMCONV *       --- CONVOLUTION (CORRELATION) ---        * TMCONV *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To perform a convolution or correlation operation on two vectors, with the operand in Main Memory and the operator in TM.

CALL FORMAT: CALL TMCONV (A, ITMB, C, N, M)

PARAMETERS:
A = Floating-point input vector (operand)
ITMB = Integer address of B in TM
C = Floating-point output vector
N = Integer element count for C
M = Integer element count for B
(Integer element count for A = N+M-1)

DESCRIPTION: C(m) = SUM(A(m+q-1)*B(q));
for q=1 to M and m=1 to N.

NOTE: For convolution, the elements of operator vector B must be stored in TM in reverse order.

TMCONV performs either a correlation or a convolution operation between the (N+M-1)-element operand (trace) vector A and the M-element operator (kernel) vector B. The N-element result vector is stored in C. The result vector C may overlay the operand A. Vectors A and C reside in main data; vector B is in TMRAM. B must be placed in TMRAM using MTMOV or another Table Memory Library routine before calling TMCONV.

NOTE: TMCONV is superior to CONV for M greater than or equal to 128; otherwise, CONV is superior.

```
**********                                              **********
*        *                                              *        *
*  VØ1   *          --- VECTOR ZERO TRENDS ---          *  VØ1   *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To produce an output vector of Ø's and 1's based on zero trends in the input vector.

CALL FORMAT: CALL VØ1(A,I,B,J,N,NPTS)

PARAMETERS:

```
A    = Floating-point input vector
I    = Integer element step for A
B    = Floating-point output vector
J    = Integer element step for B
N    = Integer element count for A and B
NPTS = Number of points of source to be
       considered in creating a
       destination point
```

DESCRIPTION:

```
B(m) = Ø.Ø if ( (A(m-NPTS+1) .EQ. Ø.Ø) .AND.
                (A(m-NPTS+2) .EQ. Ø.Ø) .AND.
                  . . . . . .
                (A(m) .EQ. Ø.Ø) )
B(m) = 1.Ø otherwise
                 for m = NPTS to N
(Note that B(1) = ... = B(NPTS-1) = 1.Ø)
```

The vector A is scanned. If the current point of A and the last NPTS-1 points of A are Ø, then the current point of B is set to zero. Otherwise the current point of B is set to 1.Ø. The resultant vector B is useful in stacking operations.

EXAMPLE:

```
N    = 16
NPTS =  3

A :  1.Ø  2.Ø  Ø.Ø  Ø.Ø  5.Ø  Ø.Ø  Ø.Ø  Ø.Ø
     Ø.Ø 1Ø.Ø 11.Ø 12.Ø 13.Ø  Ø.Ø  Ø.Ø  Ø.Ø

B :  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  Ø.Ø
     Ø.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  Ø.Ø
```

A(1) should be equal to Ø.Ø, and all other values of A(i) and B(i), for i = 1 to NC, should be greater than Ø.Ø.

The initial sample value of zero-fill in the destination trace is returned in parameter NNMO. A value of N+1 for NNMO indicates no zero-fill.

Routine NMOLI (linear interpolation) or NMOQI (quadratic interpolation) is generally called subsequent to routine VARNMO.

The speed of this routine is data dependent.

EXAMPLE:

```
NC =    4
N  =  1ØØ
SR =    3.Ø
X  =  1ØØ.Ø

A:    Ø.Ø   75.Ø  1ØØ.Ø  2ØØ.Ø
B: 5ØØØ.Ø 6ØØØ.Ø 7ØØØ.Ø 85ØØ.Ø

NNMO = 68

 D( 1)  D( 2)  D( 3)  D( 4)  D( 5)  D( 6)  D( 7)
 2Ø.ØØ  2Ø.Ø7  2Ø.59  21.53  22.83  24.44  26.3Ø

          ...

 D(65)  D(66)  D(67)  D(68)  D(69)        D(1ØØ)
192.39 195.38 198.37   Ø.ØØ   Ø.ØØ    ...   Ø.ØØ
```

```
**********                                              **********
*        *                                              *        *
* VSCANØ *       --- VECTOR SCAN FOR ZEROS ---          * VSCANØ *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To scan a source vector and record in a destination vector a running total of the number of zeros encountered.

CALL FORMAT: CALL VSCANØ(A,B,N)

PARAMETERS:
- A = Floating-point input vector
- B = Floating-point output vector
- N = Integer element count for A and B

DESCRIPTION: B(m) = number of Ø's in A(1) through A(m); for m = 1 to N

Scans the N values of the source vector A. Records the cumulative total of zero values in the N elements of vector B. The resultant vector B is useful as a mute finder.

EXAMPLE:

N = 2Ø

```
A :  1.Ø  1.Ø  Ø.Ø  Ø.Ø  1.Ø  Ø.Ø  Ø.Ø  Ø.Ø  1.Ø  1.Ø
     1.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  1.Ø  Ø.Ø  Ø.Ø  1.Ø

B :  Ø.Ø  Ø.Ø  1.Ø  2.Ø  2.Ø  3.Ø  4.Ø  5.Ø  5.Ø  5.Ø
     5.Ø  6.Ø  7.Ø  8.Ø  9.Ø 1Ø.Ø 1Ø.Ø 11.Ø 12.Ø 12.Ø
```

```
**********                                                  **********
*        *                                                  *        *
*  CSFR2 *     --- SPARSE COMPLEX SYMMETRIC FACTOR ---      *  CSFR2 *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To perform an LDL' factorization of a complex, symmetric matrix A, where A is sparse and is represented in packed form.

CALL FORMAT: CALL CSFR2(N,NS,S,ICP,IRN,ZTOL,WRK,IERR)

PARAMETERS:

N = Integer input scalar
Order of the matrix A (must be greater than 1)

NS = Integer input scalar
Number of sparse elements (i.e., nonzero and fill-in elements) in the lower triangle of A

S = Complex input/output array of length NS
On input, S contains the sparse elements of the lower triangle of A in column order. On output, S contains the superposition of L and D with the diagonal elements reciprocated.

ICP = Integer input array of length N+1
Contains pointers into S to the first sparse element of each column with ICP(N+1) = NS + 1

IRN = Integer input array of length NS
Contains the row numbers that correspond to the elements in S

ZTOL = Floating-point input scalar
Zero tolerance value

WRK = Complex scratch vector of length N

IERR = Integer output scalar
Error code whose values are:
- Ø - Normal termination
- 1 - Routine aborted because a diagonal element was computed to be zero (i.e., its absolute value squared was less than or equal to ZTOL)
- 2 - Routine aborted because N < 2

DESCRIPTION: This routine factors A into LDL' where L is a lower triangular matrix with ones on its diagonal, D is a diagonal matrix, and L' is the transpose of L. The factorization is performed without any row or column interchanges.
L and D are superpositioned by suppressing the ones on the diagonal of L; i.e., if the superposition of L and D is denoted by C, then C = L + D - I. The sparse elements of the superposition of L and D are stored in the corresponding

Thus the superposition of L and D with the diagonal elements of D replaced by their reciprocals is

```
(Ø.5,-Ø.5)
(Ø.Ø, Ø.Ø)  (Ø.5, Ø.5)
(2.Ø,-1.Ø)  (Ø.Ø, Ø.Ø)  (Ø.2,-Ø.4)
(Ø.Ø, Ø.Ø)  (1.Ø, 1.Ø)  (Ø.Ø, Ø.Ø) (-Ø.25,Ø.25)
(Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)  (Ø.Ø, 1.Ø)  (Ø.25,Ø.Ø)
```

DESCRIPTION: First CSFR2 is called to factor A into LDL' where L is a lower triangular matrix with ones on its diagonal, D is a diagonal matrix, and L' is the transpose of L. The factorization is performed without any row or column interchanges. L and D are superpositioned by suppressing the ones on the diagonal of L; i.e., if the superposition of L and D is denoted by C, then C = L + D - I. The sparse elements of the superposition of L and D are stored in the corresponding locations of S with the diagonal elements of D replaced by their reciprocals. L and D may contain nonzero elements where A contains zero elements. Collectively called "fill-in", these zeros must be included in S as input sparse elements of A. Failure to properly provide for fill-in results in undetermined action by this routine.

Next, CSSV2 is called to solve the system in three steps:

(1) Solve Lz=b for z (forward elimination)
(2) Solve Dy=z for y
(3) Solve L'x=y for x (backward substitution)

This routine supercedes CSFS and differs from it in two important respects. First, CSFS2 is much faster than CSFS. Second, CSFS2 does not check to ensure that fill-in has been provided for properly; whereas, CSFS does.

The scratch parameter WRK is not used in the current release of this routine; however, it has been retained for compatibility with CSFS. Thus, a scalar may be used in place for a vector for WRK.

For a more detailed discussion, refer to Appendix C.

The execution time for this routine is data dependent.

EXAMPLE: Let A be the complex, symmetric matrix

(1.Ø, 1.Ø)	(Ø.Ø, Ø.Ø)	(3.Ø, 1.Ø)	(Ø.Ø, Ø.Ø)	(Ø.Ø, Ø.Ø)
(Ø.Ø, Ø.Ø)	(1.Ø,-1.Ø)	(Ø.Ø, Ø.Ø)	(2.Ø, Ø.Ø)	(Ø.Ø, Ø.Ø)
(3.Ø, 1.Ø)	(Ø.Ø, Ø.Ø)	(8.Ø, 1.Ø)	(Ø.Ø, Ø.Ø)	(Ø.Ø, Ø.Ø)
(Ø.Ø, Ø.Ø)	(2.Ø, Ø.Ø)	(Ø.Ø, Ø.Ø)	(Ø.Ø, Ø.Ø)	(2.Ø,-2.Ø)
(Ø.Ø, Ø.Ø)	(Ø.Ø, Ø.Ø)	(Ø.Ø, Ø.Ø)	(2.Ø,-2.Ø)	(6.Ø, 2.Ø)

NOTE: It is known apriori that fill-in occurs in element (4,4).

```
**********                                                  **********
*        *                                                  *        *
*  CSSV2 *      --- SPARSE COMPLEX SYMMETRIC SOLVE ---      *  CSSV2 *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To find the solution to the system Ax = b, where A is a sparse, complex, symmetric matrix that is LDL' factored and is represented in packed form.

CALL FORMAT: CALL CSSV2(N,NS,S,ICP,IRN,BX)

PARAMETERS:

N = Integer input scalar
Order of the matrix A (must be greater than 1)

NS = Integer input scalar
Number of sparse elements (i.e., nonzero and fill-in elements) in A

S = Complex input array of length NS
Contains the sparse elements of the superposition of L and D with the diagonal elements reciprocated. The elements are stored in column order.

ICP = Integer input array of length N+1
Contains pointers into S to the first sparse element of each column with ICP(N+1) = NS + 1

IRN = Integer input array of length NS
Contains the row numbers that correspond to the elements in S

BX = Complex input/output vector of length N
On input, BX contains the right-hand side vector b. On output, BX contains the solution vector x.

DESCRIPTION: This routine solves the system Ax = b where A is a sparse, complex, symmetric matrix that is factored into LDL'. L is a lower triangular matrix with ones on its diagonal, D is a diagonal matrix, and L' is the transpose of L. L and D are superpositioned by suppressing the ones on the diagonal of L; i.e., if the superposition of L and D is denoted by C, then C = L + D - I.

The solution process consists of three steps:

(1) Solve Lz=b for z (forward elimination)
(2) Solve Dy=z for y
(3) Solve L'x=y for x (backward substitution)

This routine supercedes CSSV.

For a more detailed discussion, refer to Appendix C.

The execution time for this routine is data dependent.

```
**********                                                  **********
*        *                                                  *        *
* CUFR2  *   --- SPARSE COMPLEX UNSYMMETRIC FACTOR ---      * CUFR2  *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To perform an LU factorization of a complex, unsymmetric matrix A, where A is sparse and is represented in packed form.

CALL FORMAT: CALL CUFR2(N,NS,S,ICP,IRN,IDP,ZTOL,WRK,IERR)

PARAMETERS:

- N = Integer input scalar
  Order of the matrix A (must be greater than 1)
- NS = Integer input scalar
  Number of sparse elements (i.e., nonzero and fill-in elements) in A
- S = Complex input/output array of length NS
  On input, S contains the sparse elements of A in column order. On output, S contains the sparse elements of the superposition of L and U with the diagonal elements reciprocated.
- ICP = Integer input array of length N+1
  Contains pointers into S to the first sparse element of each column with ICP(N+1) = NS + 1
- IRN = Integer input array of length NS
  Contains the row numbers that correspond to the elements in S
- IDP = Integer input array of length N
  Contains pointers into S to the diagonal elements
- ZTOL = Floating-point input scalar
  Zero tolerance value
- WRK = Complex scratch vector of length N
- IERR = Integer output scalar
  Error code whose values are:
  - Ø - Normal termination
  - 1 - Routine aborted because a diagonal element was computed to be zero (i.e., its absolute value squared was less than or equal to ZTOL)
  - 2 - Routine aborted because N < 2

The output parameters are:

```
S    = Ø.5, -Ø.5, 3.Ø, 1.Ø, Ø.5, Ø.5, 2.Ø, Ø.Ø,
       2.Ø, -1.Ø, Ø.2, -Ø.4, 1.Ø, 1.Ø, -Ø.25, Ø.25,
       2.Ø, -2.Ø, Ø.Ø, 1.Ø, Ø.25, Ø.Ø
IERR = Ø
```

Thus the superposition of L and U with the diagonal elements of L replaced by their reciprocals is

```
(Ø.5,-Ø.5)  (Ø.Ø, Ø.Ø)  (2.Ø,-1.Ø)  (Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)
(Ø.Ø, Ø.Ø)  (Ø.5, Ø.5)  (Ø.Ø, Ø.Ø)  (1.Ø, 1.Ø)  (Ø.Ø, Ø.Ø)
(3.Ø, 1.Ø)  (Ø.Ø, Ø.Ø)  (Ø.2,-Ø.4)  (Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)
(Ø.Ø, Ø.Ø)  (2.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø) (-Ø.25,Ø.25) (Ø.Ø, 1.Ø)
(Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)  (2.Ø,-2.Ø)  (Ø.25,Ø.Ø)
```

DESCRIPTION: First CUFR2 is called to factor A into LU where L is a lower triangular matrix and U is an upper triangular matrix with ones on its diagonal. The factorization is performed without any row or column interchanges. L and U are superpositioned by suppressing the ones on the diagonal of U; i.e., if the superposition of L and U is denoted by C, then C = L + U - I. The sparse elements of the superposition of L and U are stored in the corresponding locations of S with the diagonal elements of L replaced by their reciprocals. L and U may contain nonzero elements where A contains zero elements. Collectively called "fill-in", these zeros must be included in S as input sparse elements of A. Failure to properly provide for fill-in results in undetermined action by this routine.

Next, CUSV2 is called to solve the system in two steps:

(1) Solve Ly=b for y (forward elimination)
(2) Solve Ux=y for x (backward substitution)

This routine supercedes CUFS and differs from it in two important respects. First, CUFS2 is much faster than CUFS. Second, CUFS2 does not check to ensure that fill-in has been provided for properly; whereas, CUFS does.

For a more detailed discussion, refer to Appendix C.

The execution time for this routine is data dependent.

EXAMPLE: Let A be the complex matrix

```
(1.Ø, 1.Ø)  (Ø.Ø, Ø.Ø)  (3.Ø, 1.Ø)  (Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)
(Ø.Ø, Ø.Ø)  (1.Ø,-1.Ø)  (Ø.Ø, Ø.Ø)  (2.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)
(3.Ø, 1.Ø)  (Ø.Ø, Ø.Ø)  (8.Ø, 1.Ø)  (Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)
(Ø.Ø, Ø.Ø)  (2.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)  (2.Ø,-2.Ø)
(Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)  (Ø.Ø, Ø.Ø)  (2.Ø,-2.Ø)  (6.Ø, 2.Ø)
```

NOTE: It is known apriori that fill-in occurs in element (4,4).

Let b be the complex vector

```
( Ø.Ø, Ø.Ø)
( 3.Ø, 3.Ø)
(-7.Ø,-9.Ø)
( 4.Ø, 2.Ø)
(12.Ø, 4.Ø)
```

```
**********                                                  **********
*        *                                                  *        *
*  CUSV2 *    --- SPARSE COMPLEX UNSYMMETRIC SOLVE ---      *  CUSV2 *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To find the solution to the system Ax = b, where A is a sparse, complex, unsymmetric matrix that is LU factored and is represented in packed form.

CALL FORMAT: CALL CUSV2(N,NS,S,ICP,IRN,IDP,BX)

PARAMETERS:

- N = Integer input scalar
  Order of the matrix A (must be greater than 1)
- NS = Integer input scalar
  Number of sparse elements (i.e., nonzero and fill-in elements) in A
- S = Complex input array of length NS
  Contains the sparse elements of the superposition of L and U with the diagonal elements reciprocated. The elements are stored in column order.
- ICP = Integer input array of length N+1
  Contains pointers into S to the first sparse element of each column with ICP(N+1) = NS + 1
- IRN = Integer input array of length NS
  Contains the row numbers that correspond to the elements in S
- IDP = Integer input array of length N
  Contains pointers into S to the diagonal elements
- BX = Complex input/output vector of length N
  On input, BX contains the right-hand side vector b. On output, BX contains the solution vector x.

DESCRIPTION: This routine solves the system Ax = b where A is a sparse, complex matrix that is factored into LU. L is a lower triangular matrix and U is an upper triangular matrix with ones on its diagonal. L and U are superpositioned by suppressing the ones on the diagonal of U; i.e., if the superposition of L and U is denoted by C, then C = L + U - I.

The solution process consists of two steps:

(1) Solve Ly=b for y (forward elimination)
(2) Solve Ux=y for x (backward substitution)

This routine supercedes CUSV.

For a more detailed discussion, refer to Appendix C.

```
**********                                                **********
*        *                                                *        *
*  RSFR2 *      --- SPARSE REAL SYMMETRIC FACTOR ---      *  RSFR2 *
*        *                                                *        *
**********                                                **********
```

PURPOSE: To perform an LDL' factorization of a real, symmetric matrix A, where A is sparse and is represented in packed form.

CALL FORMAT: CALL RSFR2(N,NS,S,ICP,IRN,ZTOL,WRK,IERR)

PARAMETERS:

N = Integer input scalar
Order of the matrix A (must be greater than 1)

NS = Integer input scalar
Number of sparse elements (i.e., nonzero and fill-in elements) in the lower triangle of A

S = Floating-point input/output array of length NS
On input, S contains the sparse elements of the lower triangle of A in column order. On output, S contains the superposition of L and D with the diagonal elements reciprocated.

ICP = Integer input array of length N+1
Contains pointers into S to the first sparse element of each column with ICP(N+1) = NS + 1

IRN = Integer input array of length NS
Contains the row numbers that correspond to the elements in S

ZTOL = Floating-point input scalar
Zero tolerance value

WRK = Floating-point scratch vector of length N

IERR = Integer output scalar
Error code whose values are:
- Ø - Normal termination
- 1 - Routine aborted because a diagonal element was computed to be zero (i.e., its absolute value was less than or equal to ZTOL)
- 2 - Routine aborted because N < 2

Then the input parameters are:

```
N    = 1Ø
NS   = 22
S    =  8.Ø,  8.Ø, 16.Ø, 16.Ø, 32.Ø, 8Ø.Ø, 16.Ø,
       24.Ø, 16.Ø,  8.Ø, 24.Ø,  8.Ø,  4.Ø, 16.Ø,
       32.Ø, 16.Ø, 8Ø.Ø, 4Ø.Ø,  8.Ø,  4.Ø,  Ø.Ø,
       -1.25
ICP  =  1,  2,  4,  6,  8, 11, 14, 17, 2Ø, 22, 23
IRN  =  1,  2, 1Ø,  3,  4,  4,  5,  5,  6,  8,  6,
        8, 1Ø,  7,  8,  9,  8,  9, 1Ø,  9, 1Ø, 1Ø
ZTOL = 1.ØE-6
```

The output parameters are:

```
S    = Ø.125, Ø.125, 2.Ø, Ø.Ø625, 2.Ø, Ø.Ø625, 1.Ø,
       Ø.125, 2.Ø, 1.Ø, -Ø.125, 1.Ø, -Ø.5, Ø.Ø625,
       2.Ø, 1.Ø, Ø.Ø625, Ø.5, Ø.25, -Ø.Ø625, Ø.125,
       -Ø.Ø3125
IERR = Ø
```

Thus the superposition of L and D with the diagonal elements of D replaced by their reciprocals is

```
Ø.125
Ø.Ø   Ø.125
Ø.Ø   Ø.Ø   Ø.Ø625
Ø.Ø   Ø.Ø   2.Ø    Ø.Ø625
Ø.Ø   Ø.Ø   Ø.Ø    1.Ø    Ø.125
Ø.Ø   Ø.Ø   Ø.Ø    Ø.Ø    2.Ø  -Ø.125
Ø.Ø   Ø.Ø   Ø.Ø    Ø.Ø    Ø.Ø   Ø.Ø   Ø.Ø625
Ø.Ø   Ø.Ø   Ø.Ø    Ø.Ø    1.Ø   1.Ø   2.Ø    Ø.Ø625
Ø.Ø   Ø.Ø   Ø.Ø    Ø.Ø    Ø.Ø   Ø.Ø   1.Ø    Ø.5   -Ø.Ø625
Ø.Ø   2.Ø   Ø.Ø    Ø.Ø    Ø.Ø  -Ø.5   Ø.Ø    Ø.25   Ø.125 -Ø.Ø3125
```

DESCRIPTION: First RSFR2 is called to factor A into LDL' where L is a lower triangular matrix with ones on its diagonal, D is a diagonal matrix, and L' is the transpose of L. The factorization is performed without any row or column interchanges.
L and D are superpositioned by suppressing the ones on the diagonal of L; i.e., if the superposition of L and D is denoted by C, then C = L + D - I. The sparse elements of the superposition of L and D are stored in the corresponding locations of S with the diagonal elements of D replaced by their reciprocals. L and D may contain nonzero elements where A contains zero elements. Collectively called "fill-in", these zeros must be included in S as input sparse elements of A. Failure to properly provide for fill-in results in undetermined action by this routine.

Next, RSSV2 is called to solve the system in three steps:

(1) Solve Lz=b for z (forward elimination)
(2) Solve Dy=z for y
(3) Solve L'x=y for x (backward substitution)

This routine supercedes RSFS and differs from it in two important respects. First, RSFS2 is much faster than RSFS. Second, RSFS2 does not check to ensure that fill-in has been provided for properly; whereas, RSFS does.

The scratch parameter WRK is not used in the current release of this routine; however, it has been retained for compatibility with RSFS. Thus, a scalar may be used in place for a vector for WRK.

For a more detailed discussion, refer to Appendix C.

The execution time for this routine is data dependent.

EXAMPLE: Let A be the symmetric matrix

8.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	8.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø
Ø.Ø	Ø.Ø	16.Ø	32.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	32.Ø	8Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	24.Ø	16.Ø	Ø.Ø	8.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	24.Ø	Ø.Ø	8.Ø	Ø.Ø	4.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	32.Ø	16.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	8.Ø	8.Ø	32.Ø	8Ø.Ø	4Ø.Ø	8.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	4Ø.Ø	4.Ø	Ø.Ø
Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	4.Ø	Ø.Ø	8.Ø	Ø.Ø	-1.25

Thus the superposition of L and D with the diagonal elements of D replaced by their reciprocals is

```
Ø.125
Ø.Ø    Ø.125
Ø.Ø    Ø.Ø    Ø.ØØ625
Ø.Ø    Ø.Ø    2.Ø     Ø.ØØ625
Ø.Ø    Ø.Ø    Ø.Ø     1.Ø     Ø.125
Ø.Ø    Ø.Ø    Ø.Ø     Ø.Ø     2.Ø   -Ø.125
Ø.Ø    Ø.Ø    Ø.Ø     Ø.Ø     Ø.Ø    Ø.Ø    Ø.ØØ625
Ø.Ø    Ø.Ø    Ø.Ø     Ø.Ø     1.Ø    1.Ø    2.Ø     Ø.ØØ625
Ø.Ø    Ø.Ø    Ø.Ø     Ø.Ø     Ø.Ø    Ø.Ø    1.Ø     Ø.5    -Ø.ØØ625
Ø.Ø    2.Ø    Ø.Ø     Ø.Ø     Ø.Ø   -Ø.5    Ø.Ø     Ø.25    Ø.125 -Ø.ØØ3125
```

and the solution vector, x, is

```
3.Ø
1.Ø
4.Ø
1.Ø
5.Ø
9.Ø
Ø.Ø
Ø.Ø
7.Ø
Ø.Ø
```

EXAMPLE: Let A be the symmetric matrix

8.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	8.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø
Ø.Ø	Ø.Ø	16.Ø	32.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	32.Ø	8Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	24.Ø	16.Ø	Ø.Ø	8.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	24.Ø	Ø.Ø	8.Ø	Ø.Ø	4.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	32.Ø	16.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	8.Ø	8.Ø	32.Ø	8Ø.Ø	4Ø.Ø	8.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	4Ø.Ø	4.Ø	Ø.Ø
Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	4.Ø	Ø.Ø	8.Ø	Ø.Ø	-1.25

Then the superposition of L and D with the diagonal elements of D replaced by their reciprocals is

Ø.125									
Ø.Ø	Ø.125								
Ø.Ø	Ø.Ø	Ø.Ø625							
Ø.Ø	Ø.Ø	2.Ø	Ø.Ø625						
Ø.Ø	Ø.Ø	Ø.Ø	1.Ø	Ø.125					
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	2.Ø	-Ø.125				
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø625			
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	1.Ø	1.Ø	2.Ø	Ø.Ø625		
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	1.Ø	Ø.5	-Ø.Ø625	
Ø.Ø	2.Ø	Ø.Ø	Ø.Ø	Ø.Ø	-Ø.5	Ø.Ø	Ø.25	Ø.125	-Ø.Ø3125

Let b be the vector

24.Ø
8.Ø
96.Ø
288.Ø
28Ø.Ø
296.Ø
112.Ø
392.Ø
28.Ø
52.Ø

Then the input parameters are:

```
N    = 1Ø
NS   = 22
S    = Ø.125, Ø.125, 2.Ø, Ø.Ø625, 2.Ø, Ø.Ø625, 1.Ø,
       Ø.125, 2.Ø, 1.Ø, -Ø.125, 1.Ø, -Ø.5, Ø.Ø625,
       2.Ø, 1.Ø, Ø.Ø625, Ø.5, Ø.25, -Ø.Ø625, Ø.125,
       -Ø.Ø3125
ICP  =  1,  2,  4,  6,  8, 11, 14, 17, 2Ø, 22, 23
IRN  =  1,  2, 1Ø,  3,  4,  4,  5,  5,  6,  8,  6,
        8, 1Ø,  7,  8,  9,  8,  9, 1Ø,  9, 1Ø, 1Ø
BX   = 24.Ø, 8.Ø, 96.Ø, 288.Ø, 28Ø.Ø, 296.Ø, 112.Ø,
       392.Ø, 28.Ø, 52.Ø
```

```
**********                                                  **********
*        *                                                  *        *
*  RUFR2 *     --- SPARSE REAL UNSYMMETRIC FACTOR ---       *  RUFR2 *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To perform an LU factorization of a real, unsymmetric matrix A, where A is sparse and is represented in packed form.

CALL FORMAT: CALL RUFR2(N,NS,S,ICP,IRN,IDP,ZTOL,WRK,IERR)

PARAMETERS:

- N = Integer input scalar
  Order of the matrix A (must be greater than 1)
- NS = Integer input scalar
  Number of sparse elements (i.e., nonzero and fill-in elements) in A
- S = Floating-point input/output array of length NS
  On input, S contains the sparse elements of A in column order. On output, S contains the sparse elements of the superposition of L and U with the diagonal elements reciprocated.
- ICP = Integer input array of length N+1
  Contains pointers into S to the first sparse element of each column with ICP(N+1) = NS + 1
- IRN = Integer input array of length NS
  Contains the row numbers that correspond to the elements in S
- IDP = Integer input array of length N
  Contains pointers into S to the diagonal elements
- ZTOL = Floating-point input scalar
  Zero tolerance value
- WRK = Floating-point scratch vector of length N
- IERR = Integer output scalar
  Error code whose values are:
  - Ø - Normal termination
  - 1 - Routine aborted because a diagonal element was computed to be zero (i.e., its absolute value was less than or equal to ZTOL)
  - 2 - Routine aborted because N < 2

Then the input parameters are:

```
N    = 1Ø
NS   = 34
S    =  8.Ø,  8.Ø, 16.Ø, 16.Ø, 32.Ø, 32.Ø, 8Ø.Ø,
       16.Ø, 16.Ø, 24.Ø, 16.Ø,  8.Ø, 16.Ø, 24.Ø,
        8.Ø,  4.Ø, 16.Ø, 32.Ø, 16.Ø,  8.Ø,  8.Ø,
       32.Ø, 8Ø.Ø, 4Ø.Ø,  8.Ø, 16.Ø, 4Ø.Ø,  4.Ø,
        Ø.Ø, 16.Ø,  4.Ø,  8.Ø,  Ø.Ø, -1.25
ICP  =  1,  2,  4,  6,  9, 13, 17, 2Ø, 26, 3Ø, 35
IRN  =  1,  2, 1Ø,  3,  4,  3,  4,  5,  4,  5,  6,
        8,  5,  6,  8, 1Ø,  7,  8,  9,  5,  6,  7,
        8,  9, 1Ø,  7,  8,  9, 1Ø,  2,  6,  8,  9, 1Ø
IDP  =  1,  2,  4,  7, 1Ø, 14, 17, 23, 28, 34
ZTOL = 1.ØE-6
```

The output parameters are:

```
S    = Ø.125, Ø.125, 16.Ø, Ø.Ø625, 32.Ø, 2.Ø, Ø.Ø625,
       16.Ø, 1.Ø, Ø.125, 16.Ø, 8.Ø, 2.Ø, -Ø.125,
       -8.Ø, 4.Ø, Ø.Ø625, 32.Ø, 16.Ø, 1.Ø, 1.Ø, 2.Ø,
       Ø.Ø625, 8.Ø, 4.Ø, 1.Ø, Ø.5, -Ø.Ø625, -2.Ø,
       2.Ø, -Ø.5, Ø.25, Ø.125, -Ø.Ø3125
IERR = Ø
```

Thus the superposition of L and U with the diagonal elements of L replaced by their reciprocals is

```
Ø.125  Ø.Ø    Ø.Ø     Ø.Ø     Ø.Ø    Ø.Ø     Ø.Ø     Ø.Ø     Ø.Ø     Ø.Ø
Ø.Ø    Ø.125  Ø.Ø     Ø.Ø     Ø.Ø    Ø.Ø     Ø.Ø     Ø.Ø     Ø.Ø     2.Ø
Ø.Ø    Ø.Ø    Ø.Ø625  2.Ø     Ø.Ø    Ø.Ø     Ø.Ø     Ø.Ø     Ø.Ø     Ø.Ø
Ø.Ø    Ø.Ø   32.Ø     Ø.Ø625  1.Ø    Ø.Ø     Ø.Ø     Ø.Ø     Ø.Ø     Ø.Ø
Ø.Ø    Ø.Ø    Ø.Ø    16.Ø     Ø.125  2.Ø     Ø.Ø     1.Ø     Ø.Ø     Ø.Ø
Ø.Ø    Ø.Ø    Ø.Ø     Ø.Ø    16.Ø   -Ø.125  Ø.Ø     1.Ø     Ø.Ø    -Ø.5
Ø.Ø    Ø.Ø    Ø.Ø     Ø.Ø     Ø.Ø    Ø.Ø     Ø.Ø625  2.Ø     1.Ø     Ø.Ø
Ø.Ø    Ø.Ø    Ø.Ø     Ø.Ø     8.Ø   -8.Ø    32.Ø     Ø.Ø625  Ø.5     Ø.25
Ø.Ø    Ø.Ø    Ø.Ø     Ø.Ø     Ø.Ø    Ø.Ø     16.Ø     8.Ø    -Ø.Ø625  Ø.125
Ø.Ø   16.Ø    Ø.Ø     Ø.Ø     Ø.Ø    4.Ø     Ø.Ø     4.Ø    -2.Ø    -Ø.Ø3125
```

DESCRIPTION: First RUFR2 is called to factor A into LU where L is a lower triangular matrix and U is an upper triangular matrix with ones on its diagonal. The factorization is performed without any row or column interchanges. L and U are superpositioned by suppressing the ones on the diagonal of U; i.e., if the superposition of L and U is denoted by C, then C = L + U - I. The sparse elements of the superposition of L and U are stored in the corresponding locations of S with the diagonal elements of L replaced by their reciprocals. L and U may contain nonzero elements where A contains zero elements. Collectively called "fill-in", these zeros must be included in S as input sparse elements of A. Failure to properly provide for fill-in results in undetermined action by this routine.

Next, RUSV2 is called to solve the system in two steps:

(1) Solve Ly=b for y (forward elimination)
(2) Solve Ux=y for x (backward substitution)

This routine supercedes RUFS and differs from it in two important respects. First, RUFS2 is much faster than RUFS. Second, RUFS2 does not check to ensure that fill-in has been provided for properly; whereas, RUFS does.

For a more detailed discussion, refer to Appendix C.

The execution time for this routine is data dependent.

EXAMPLE: Let A be the matrix

8.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	8.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø
Ø.Ø	Ø.Ø	16.Ø	32.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	32.Ø	8Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	24.Ø	16.Ø	Ø.Ø	8.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	24.Ø	Ø.Ø	8.Ø	Ø.Ø	4.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	32.Ø	16.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	8.Ø	8.Ø	32.Ø	8Ø.Ø	4Ø.Ø	8.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	4Ø.Ø	4.Ø	Ø.Ø
Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	4.Ø	Ø.Ø	8.Ø	Ø.Ø	-1.25

NOTE: It is known apriori that fill-in occurs in elements (1Ø,9) and (9,1Ø).

Thus the superposition of L and U with the diagonal elements of L replaced by their reciprocals is

Ø.125	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.125	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	2.Ø
Ø.Ø	Ø.Ø	Ø.Ø625	2.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	32.Ø	Ø.Ø625	1.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	Ø.125	2.Ø	Ø.Ø	1.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	-Ø.125	Ø.Ø	1.Ø	Ø.Ø	-Ø.5
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø625	2.Ø	1.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	8.Ø	-8.Ø	32.Ø	Ø.Ø625	Ø.5	Ø.25
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	8.Ø	-Ø.Ø625	Ø.125
Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	4.Ø	Ø.Ø	4.Ø	-2.Ø	-Ø.Ø3125

and the solution vector, x, is

3.Ø
1.Ø
4.Ø
1.Ø
5.Ø
9.Ø
Ø.Ø
Ø.Ø
7.Ø
Ø.Ø

The execution time for this routine is data dependent.

EXAMPLE: Let A be the matrix

8.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	8.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø
Ø.Ø	Ø.Ø	16.Ø	32.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	32.Ø	8Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	24.Ø	16.Ø	Ø.Ø	8.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	24.Ø	Ø.Ø	8.Ø	Ø.Ø	4.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	32.Ø	16.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	8.Ø	8.Ø	32.Ø	8Ø.Ø	4Ø.Ø	8.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	4Ø.Ø	4.Ø	Ø.Ø
Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	4.Ø	Ø.Ø	8.Ø	Ø.Ø	-1.25

Then the superposition of L and U with the diagonal elements of L replaced by their reciprocals is

Ø.125	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.125	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	2.Ø
Ø.Ø	Ø.Ø	Ø.Ø625	2.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	32.Ø	Ø.Ø625	1.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	Ø.125	2.Ø	Ø.Ø	1.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	-Ø.125	Ø.Ø	1.Ø	Ø.Ø	-Ø.5
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø625	2.Ø	1.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	8.Ø	-8.Ø	32.Ø	Ø.Ø625	Ø.5	Ø.25
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	16.Ø	8.Ø	-Ø.Ø625	Ø.125
Ø.Ø	16.Ø	Ø.Ø	Ø.Ø	Ø.Ø	4.Ø	Ø.Ø	4.Ø	-2.Ø	-Ø.Ø3125

Let b be the vector

24.Ø
8.Ø
96.Ø
288.Ø
28Ø.Ø
296.Ø
112.Ø
392.Ø
28.Ø
52.Ø

Then the input parameters are:

N = 1Ø
NS = 34
S = Ø.125, Ø.125, 16.Ø, Ø.Ø625, 32.Ø, 2.Ø, Ø.Ø625, 16.Ø, 1.Ø, Ø.125, 16.Ø, 8.Ø, 2.Ø, -Ø.125, -8.Ø, 4.Ø, Ø.Ø625, 32.Ø, 16.Ø, 1.Ø, 1.Ø, 2.Ø, Ø.Ø625, 8.Ø, 4.Ø, 1.Ø, Ø.5, -Ø.Ø625, -2.Ø, 2.Ø, -Ø.5, Ø.25, Ø.125, -Ø.Ø3125

```
**********                                              **********
*        *                                              *        *
* SDOTPR *          --- SPARSE DOT PRODUCT ---          * SDOTPR *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To calculate the dot product of a column of A with another vector, B, given a real, sparse matrix, A, that is in packed format.

CALL FORMAT: CALL SDOTPR(M,NP1,NS,S,IRN,ICP,IC,B,J,C)

PARAMETERS:

- M = Integer input scalar
  Number of rows in A.
- NP1 = Integer input scalar
  Number of columns in A plus one.
- NS = Integer input scalar
  Number of nonzero elements in A.
- S = Floating-point input array of length NS
  Contains the nonzero elements of A stored by columns.
- IRN = Integer input array of length NS
  Contains the row numbers (in A) that correspond to the nonzero elements in S.
- ICP = Integer input array of length NP1
  Contains pointers to the elements in S that are the first nonzero elements in each column of A. ICP(NP1) = NS + 1.
- IC = Integer input scalar
  Number of the column in A that is to be used.
- B = Floating-point input vector of length M
- J = Integer input scalar
  Element step for B.
- C = Floating-point output scalar

DESCRIPTION: C = Sum[B(IRN(k)) * S(k); k=ICP(IC) to ICP(IC+1)-1]

EXAMPLE: Let A :

1.Ø	Ø.Ø	Ø.Ø	4.Ø	Ø.Ø	1.Ø	Ø.Ø
Ø.Ø	Ø.Ø	-1.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	-4.Ø	Ø.Ø	Ø.Ø	5.Ø	Ø.Ø	2.Ø
2.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø
Ø.Ø	Ø.Ø	Ø.Ø	Ø.Ø	-2.Ø	Ø.Ø	-3.Ø
Ø.Ø	Ø.Ø	Ø.Ø	-3.Ø	3.Ø	Ø.Ø	Ø.Ø

```
**********                                              **********
*        *                                              *        *
* SITSOL *      --- SPARSE ITERATIVE SOLVER ---         * SITSOL *
*        *                                              *        *
**********                                              **********
```

PURPOSE: To solve a real, sparse, linear system A * X = B, where A is in packed, row-order format.

CALL FORMAT:
```
CALL SITSOL(N,NS,S,ICN,IRP,B,W,ZTOL,NCUT,IFLG,
        X,ITER,IERR)
```

PARAMETERS:

N = Integer input scalar
Order of A.

NS = Integer input scalar
Number of nonzero elements in A.

S = Real input array of length NS
Contains the nonzero elements of A stored in row order.

ICN = Integer input array of length NS
Contains the column numbers (in A) of the corresponding elements in S.

IRP = Integer input array of length N+1
Contains pointers to the first element of each row of A in S with IRP(N+1) = NS+1.

B = Real input vector of length N
Contains the right-hand side.

W = Real input scalar
Over relaxation coefficient. If W = 1.Ø, then the Gauss-Seidel method is used to solve the system. Otherwise, the successive over relaxation (SOR) method is used with a coefficient of W.

ZTOL = Real input scalar
Zero tolerance value. The solution is considered to have converged when every element of X is within ZTOL of its value on the previous iteration.

NCUT = Integer input scalar
Iteration limit. The routine will return after NCUT iterations if the solution has not converged.

IFLG = Integer input scalar
Input flag:
- Ø - Normal input
- 1 - X contains an initial solution
- 2 - The routine is being reentered to perform additional iterations and the vectors S, ICN, IRP, B, and X contain the values that they had on return from a previous call to SITSOL.

EXAMPLE: Given the linear system A * X = B, where

```
A   :  4.Ø   Ø.Ø   2.Ø   Ø.Ø   Ø.Ø   Ø.Ø   Ø.Ø   Ø.Ø
       Ø.Ø   8.Ø   Ø.Ø   3.Ø   Ø.Ø   Ø.Ø   Ø.Ø   Ø.Ø
       3.Ø   Ø.Ø   8.Ø   Ø.Ø   1.Ø   Ø.Ø   Ø.Ø   Ø.Ø
       Ø.Ø  -3.Ø   Ø.Ø   8.Ø   1.Ø   2.Ø   Ø.Ø   Ø.Ø
       Ø.Ø   Ø.Ø   5.Ø  -2.Ø  16.Ø   3.Ø   4.Ø   Ø.Ø
       Ø.Ø   Ø.Ø   Ø.Ø  -2.Ø   4.Ø  -8.Ø   Ø.Ø   1.Ø
       Ø.Ø   Ø.Ø   Ø.Ø   Ø.Ø   Ø.Ø   Ø.Ø   2.Ø   Ø.Ø
       Ø.Ø   Ø.Ø   Ø.Ø   Ø.Ø   Ø.Ø   2.Ø   Ø.Ø   4.Ø
```

and

```
B   :  8.Ø  -5.Ø   7.Ø  18.Ø  31.Ø -22.Ø   4.Ø   6.Ø
```

then the inputs are

```
N   = 8
NS  = 23

S   :  4.Ø,  2.Ø,  8.Ø,  3.Ø,  3.Ø,  8.Ø,  1.Ø, -3.Ø,
       8.Ø,  1.Ø,  2.Ø,  5.Ø, -2.Ø, 16.Ø,  3.Ø,  4.Ø,
      -2.Ø,  4.Ø, -8.Ø,  1.Ø,  2.Ø,  2.Ø,  4.Ø

ICN :    1,    3,    2,    4,    1,    3,    5,    2,
         4,    5,    6,    3,    4,    5,    6,    7,
         4,    5,    6,    8,    7,    6,    8

IRP :   1,   3,   5,   8,  12,  17,  21,  22,  24

B   :  8.Ø, -5.Ø,  7.Ø, 18.Ø, 31.Ø,-22.Ø,  4.Ø,  6.Ø

W    =  1.Ø
ZTOL =  Ø.ØØØ1
NCUT = 2Ø
IFLG =  Ø
```

and the outputs are

```
X   :  2.ØØØØ,  -1.ØØØØ,   Ø.ØØØØ,   1.ØØØØ,
       1.ØØØØ,   3.ØØØØ,   2.ØØØØ,   Ø.ØØØØ

ITER = 8
IERR = Ø
```

EXAMPLE: Input:

```
ITYPE = 3
M     = 4
N     = 5
NS    = 7

A:  5.Ø  6.Ø  Ø.Ø  4.Ø  Ø.Ø
    Ø.Ø  Ø.Ø  Ø.Ø  3.Ø  Ø.Ø
    9.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
    Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
```

Output:

```
NS    = 5

S:  5.Ø  9.Ø  6.Ø  4.Ø  3.Ø

IN: 1    3    1    1    2

IP: 1    3    4    4    6    2    1    Ø    2    Ø

IERR  = Ø
```

EXAMPLE: Input:

```
ITYPE = 1
M     = 4
N     = 3
NS    = 5

S:  5.Ø 6.Ø 3.Ø 2.Ø 4.Ø

IN:   1   2   4   1   4

IP:   1   4   4   6
```

Output:

```
A:  5.Ø 6.Ø Ø.Ø 3.Ø Ø.Ø Ø.Ø Ø.Ø Ø.Ø 2.Ø Ø.Ø Ø.Ø 4.Ø

IERR  = Ø
```

Then the input is

```
M   = 6
NP1 = 8
NS  = 12

S   : 1.  2. -4. -1.  4. -3.  5. -2.  3.  1.  2. -3.

IRN : 1   4   3   2   1   6   3   5   6   1   3   5

ICP : 1   3   4   5   7  10  11  13

IC  = 4

B   :  5.0  -2.0   1.0   6.0   4.0   2.0
      -1.0  -7.0   8.0   3.0   2.0   2.0
       4.0   2.0   3.0  -5.0   6.0   3.0

NC  = 3
```

Output:

```
C   = 14.0 -10.0   7.0
```

Then the input is

```
M   = 6
NP1 = 8
NS  = 12

S   : 1.  2. -4. -1.  4. -3.  5. -2.  3.  1.  2. -3.

IRN : 1   4   3   2   1   6   3   5   6   1   3   5

ICP : 1   3   4   5   7   10  11  13

IC  = 5

B   : b1  b2  b3  b4  b5  b6
      where b1 to b6 are the existing values in B
```

Output:

```
B   : b1  b2  5.  b4 -2.  3.
```

Output:

```
NS   =   3
IERR =   Ø

S    : 1.5  1.25 -4.375

IEN  :   2     7     1Ø
```

Please detach cards along perforations.

READER'S COMMENT FORM

Your comments will help us improve the quality and usefulness of our publications. Please fill out and return this form. (The mailing address is on the back.)

Title of document: _____

Your Name and Title: _____ Date: _____

Firm: _____ Department: _____

Address: _____

City: _____ State: _____ Zip Code: _____

Telephone Number: (_____) _____ Extension: _____

I used this manual. . .

- ❑ as an introduction to the subject
- ❑ as an aid for advanced training
- ❑ to instruct a class
- ❑ to learn operating procedures
- ❑ as a reference manual
- ❑ other _____

I found this material. . .

	Yes	No
accurate	❑	❑
complete	❑	❑
written clearly	❑	❑
well illustrated	❑	❑
well indexed	❑	❑

Please indicate below, listing the pages, any errors you found in the manual. Also indicate if you would have liked more information about a certain subject.

ARRAY

ARRAY is an independent society of people who use FPS products. Membership is free and includes a quarterly newsletter. There is an annual conference, as well as other activities. If you are interested in becoming an ARRAY member, please fill out and return this form. (The mailing address is on the back.)

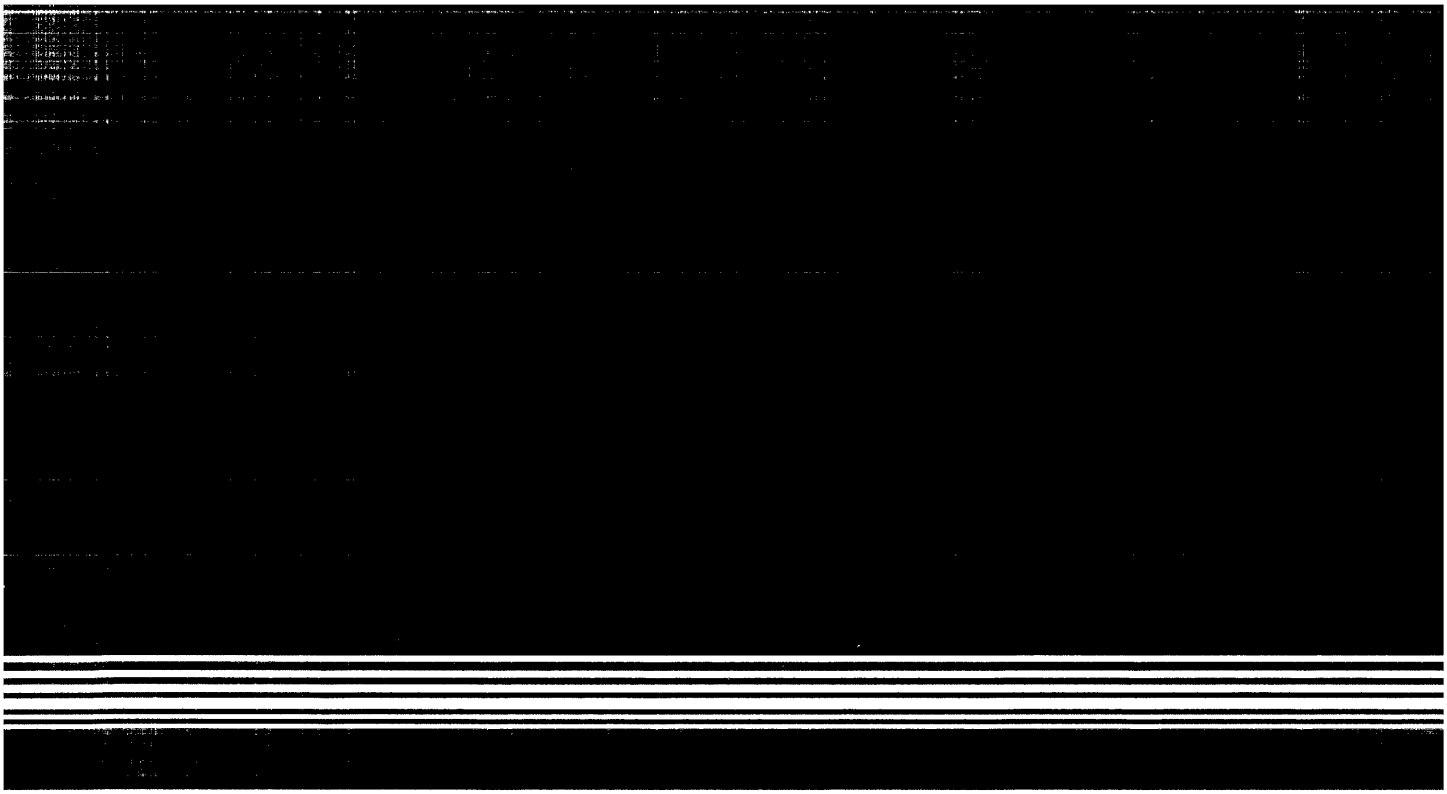
Your Name and Title: _____ Date: _____

Firm: _____ Department: _____

Address: _____

City: _____ State: _____ Zip Code: _____

Telephone Number: (_____) _____ Extension: _____



FLOATING POINT
SYSTEMS, INC.

P.O. Box 23489, Portland, Oregon 97223
Tel: 503/641-3151
Telex: 360470 FLOATPOIN BEAV
Telex: 4742018 FLPT UI