REVISION HISTORY

This manual is the APMATH64 Manual, Volume 3, 860-7482-601. 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.

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

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 third volume of the APMATH64 Manual. It is comprised of part 3 of Appendix A and Appendix B through Appendix J. 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.
APPENDIX A  APMATH64 ROUTINES

TABLE MEMORY RAM LIBRARY  A-475
SPECIAL UTILITIES LIBRARY  A-503
DATA FORMATTING LIBRARY  A-507
DOUBLE PRECISION LIBRARY  A-528
FORTRAN SUPPORT LIBRARY  A-538

APPENDIX B  DATA REPRESENTATIONS FOR STORING SPARSE VECTORS AND MATRICES

B.1 INTRODUCTION  B-1
B.2 SPARSE VECTOR STORAGE  B-1
B.3 SPARSE MATRIX STORAGE  B-2
B.3.1 Matrix Format Type I (COL_ORDER_PTRS_ONLY)  B-3
B.3.2 Matrix Format Type II (ROW_ORDER_PTRS_ONLY)  B-4
B.3.3 Matrix Format Type III (COL_ORDER_PTRS_SUMS)  B-5
B.3.4 Matrix Format Type IV (ROW_ORDER_PTRS_SUMS)  B-6

APPENDIX C  SPARSE LINEAR SYSTEM ROUTINES

C.1 INTRODUCTION  C-1
C.2 SUMMARY OF VERSION 2 FEATURES  C-2
C.3 MATHEMATICAL BACKGROUND  C-2
C.3.1 LU Theorem  C-4
C.3.2 LDU Theorem  C-5
C.4 FILL-IN  C-5
C.5 DATA FORMAT  C-8
C.6 NUMERICAL STABILITY  C-10

APPENDIX D  BASIC LINEAR ALGEBRA SUBPROGRAMS

D.1 INTRODUCTION  D-1
D.2 DATA STRUCTURES FOR VECTORS AND ARRAYS  D-2
D.3 ROUTINE CALLING SEQUENCES, ALGORITHMS, TIMINGS  D-3
D.3.1 Subroutine CAXPY(N,CA,CX,INCX,CY,INCY)  D-4
D.3.2 Subroutine CCOPY(N,CX,INCX,CY,INCY)  D-4
D.3.3 Complex Function CDOTC(N,CX,INCX,CY,INCY)  D-4
D.3.4 Complex Function CDOTU(N,CX,INCX,CY,INCY)  D-5
D.3.5 Subroutine CROTG(CA,CB,SC,CSIN)  D-5
D.3.6 Subroutine CSCAL(N,CA,CX,INCX)  D-5
D.3.7 Subroutine CSSCAL(N,SA,CX,INCX)  D-5
D.3.8 Subroutine CSROT(N,CX,INCX,CY,INCY,SC,SS)  D-5
D.3.9 Subroutine CSWAP(N,CX,INCX,CY,INCY)  D-5
D.3.10 Integer Function ICAMAX(N,CX,INCX)  D-5
D.3.11 Integer Function ISAMAX(N,SX,INCX)  D-5
D.3.12 Real Function SASUM(N,SX,INCX)  D-5
D.3.13 Subroutine SAXPY(N,SA,SX,INCX,SY,INCY)  D-6
APPENDIX G  LIST OF SUPERSEDED ROUTINES

APPENDIX H  EXCEPTIONS ENABLED ROUTINES INFORMATION AND INTERNAL SUBROUTINES
H.1 EXCEPTIONS ENABLED ROUTINES INFORMATION  H-1
H.2 INTERNAL SUBROUTINES  H-1

APPENDIX I  APMATH64 ROUTINES IN PAGE ORDER AND BY TYPE

APPENDIX J  APMATH64 ROUTINES IN ALPHABETICAL ORDER

APMATH64 KEY WORD INDEX

ILLUSTRATIONS

Figure No.  Title  Page
E-1  Example Coordinate and Function Value Breakpoint Tables  E-3

TABLES

Table No.  Title  Page
B-1  Format Types and Attributes  B-2
APMATH64 ROUTINES (VOLUME 3)
TABLE MEMORY RAM LIBRARY
**APPENDIX A**

--- VECTOR MULTIPLY (MD*MD TO TM) ---

**PURPOSE:** To multiply the elements of two vectors in Main Memory and store the resultant vector in Table Memory.

**CALL FORMAT:**

```
CALL MMTMUL(A,I,B,J,ITMC,K,N)
```

**PARAMETERS:**

- `A` = Floating-point Main Memory input vector
- `I` = Integer element step for `A`
- `B` = Floating-point Main Memory input vector
- `J` = Integer element step for `B`
- `ITMC` = Integer base address of TM output vector
- `K` = Integer element step for `C`
- `N` = Integer element count

**DESCRIPTION:** MMTMUL multiplies `N` elements of the vector `A` with `N` elements of the vector `B`, where `A` and `B` are in Main Memory, and stores the results in a vector with base address `ITMC` and increment `K` in Table Memory.

**NOTE:** Writable Table Memory begins at address 8192.

**EXAMPLE:**

```
N=3
I=J=K=1
ITMC = 8192

A :  1.0  2.0  3.0
B :  3.0  4.0  5.0

TMLOC:  8192  8193  8194
C :  3.0  8.0 15.0
```
APPENDIX A

*******
*      *
* MTIMOV *
*      *
*****

PURPOSE: To move elements of a vector from Main Memory to Table Memory, where the increments between the elements are specified.

CALL FORMAT: CALL MTIMOV(A,I,ITMC,K,N)

PARAMETERS:
A = Floating-point Main Memory input vector
I = Integer element step for A
ITMC = Integer base address of TM output vector C
K = Integer element step for C
N = Integer element count

DESCRIPTION: MTIMOV moves the elements of an input vector A with increment I in Main Memory to an output vector with base address ITMC and increment K in Table Memory.

NOTE: Writable Table Memory begins at address 8192.

EXAMPLE:

N = 3
K = 2
ITMC = 8192

A : 1.0 2.0 3.0

TMLOC: 8192 8193 8194 8195 8196 8197

C : 1.0 X 2.0 X 3.0 X

X represents unchanged values.
**** ****  VECTOR MULTIPLY (MD*TM TO MD) **** ****

PURPOSE: To multiply elements of a vector in Main Memory by elements of a vector in Table Memory and store the products in Main Memory.

CALL FORMAT: CALL MTMMUL(A,I,ITMB,J,C,K,N)

PARAMETERS: A = Floating-point Main Memory input vector
I = Integer element step for A
ITMB = Integer base address of TM input vector B
J = Integer element step for B
C = Floating-point Main Memory output vector
K = Integer element step for C
N = Integer element count

DESCRIPTION: MTMMUL multiplies N elements of the vector A in Main Memory by N elements of the vector with base address ITMB in Table Memory, and stores the products in N elements of the vector C in Main Memory.

EXAMPLE:

N=3
I=J=K=1
ITMB=8192

A : 1.0 2.0 3.0
TMLOC: 8192 8193 8194
B : 2.0 3.0 4.0
C : 2.0 6.0 12.0
APPENDIX A

**********
*        *
* MTMSUB *  ---- VECTOR SUBTRACT (MD-TM TO MD) ----
*        *
**********

PURPOSE: To subtract the elements of a vector in Table Memory from the elements of a vector in Main Memory and store the results in a vector in Main Memory.

CALL FORMAT: CALL MTMSUB(A,I,ITMB,J,C,K,N)

PARAMETERS:

A = Floating-point Main Memory input vector
I = Integer element step for A
ITMB = Integer base address of TM input vector B
J = Integer element step for B
C = Floating-point Main Memory output vector
K = Integer element step for C
N = Integer element count

DESCRIPTION: MTMSUB subtracts N elements of a vector with base address ITMB in Table Memory from N elements of the vector A in Main Memory, and stores the results in N elements of the vector C in Main Memory.

EXAMPLE:

N=3
I=J=K=1
ITMB = 8192

A : 3.0 4.0 5.0
TMLOC: 8192 8193 8194
B : 2.0 1.0 1.0
C : 1.0 3.0 4.0
APPENDIX A

********** **********
*MTRMUL *   VECTOR MULTIPLY (MD*TM TO TM)   *
**********

PURPOSE: To multiply the elements of a vector in Main Memory by the elements of a vector in Table Memory and store the products in a vector in Table Memory.

CALL FORMAT: CALL MTTMUL(A,I,ITMB,J,ITMC,K,N)

PARAMETERS:
A = Floating-point Main Memory input vector
I = Integer element step for A
ITMB = Integer base address of TM input vector B
J = Integer element step for B
ITMC = Integer base address of TM output vector C
K = Integer element step for C
N = Integer element count

DESCRIPTION: MTTMUL multiplies N elements of the vector A in Main Memory by N elements of the vector with base address ITMB in Table Memory, and stores the products in N elements of a vector with base address ITMC in Table Memory.

NOTE: Writable Table Memory begins at address 8192.

EXAMPLE:

N=3
I=J=K=1
ITMB = 8192
ITMC = 8292

A : 3.0 4.0 5.0
TMLOC: 8191 8193 8194
B : 2.0 1.0 3.0
TMLOC: 8292 8293 8294
C : 6.0 4.0 15.0
APPENDIX A

********** **********
* * *
* TMDOT * -- REAL DOT-PRODUCT (TM AND MD) ----
* * *
********** **********

PURPOSE: Computes the real dot-product of two vectors where one vector is stored in Main Memory and the other vector is stored in Table Memory. Both vectors are assumed to be stored compactly.

CALL FORMAT: CALL TMDOT (ITMA,B,C,N)

PARAMETERS: ITMA = Integer base address of TM input vector A
B = Floating-point Main Memory input vector
C = Floating-point Main Memory output scalar
N = Integer element count

DESCRIPTION: TMDOT computes the real dot-product of N elements of the vector with base address ITMA in Table Memory with N elements of the vector B in Main Memory, and stores the resultant scalar in Main Memory.

Formula:
C = A(1)*B(1) + A(2)*B(2) + ... + A(N)*B(N)
C = 0.0, if N < 1

EXAMPLE:
N = 3
ITMA = 8192

TMLOC: 8192 8193 8194
A : 1.0 2.0 3.0
B : 3.0 4.0 5.0
C = 26.0
PURPOSE: Multiplies two matrices A and B in Main Memory to form a matrix C in Main Memory. This routine uses a workspace in Table Memory to achieve high speed.

CALL FORMAT: CALL TMMM (A,B,C,MC,NC,NA,ITMW)

PARAMETERS: A = Floating-point Main Memory input matrix  
B = Floating-point Main Memory input matrix  
C = Floating-point Main Memory output matrix  
MC = Integer number of rows in output matrix C (and input matrix A)  
NC = Integer number of columns in output matrix C (and input matrix B)  
NA = Integer number of columns in input matrix A (and number of rows of input matrix B)  
ITMW = Integer base address of TM work area of length NA

DESCRIPTION: TMMM computes the product of the MC-row by NA-column matrix A and the NA-row by NC-column matrix B (both in Main Memory) and stores the result in the MC-row by NC-column matrix B in Main Memory. This routine uses a workspace of length NA in Table Memory to achieve high speed. All matrices are assumed to be stored in column order.

NOTE: Writable Table Memory begins at location 8192.

EXAMPLE:

A = | 1.0 2.0 | B = | 2.0 6.0 9.0 |
    | 3.0 4.0 |     | 3.0 7.0 4.0 |
PURPOSE: To subtract the elements of a vector in Main Memory from the elements of a vector in Table Memory and store the results in a vector in Main Memory.

CALL FORMAT: CALL TMMSUB(ITMA, I, B, J, C, K, N)

PARAMETERS:
- ITMA = Integer base address of TM input vector A
- I = Integer element step for A
- B = Floating-point Main Memory input vector
- J = Integer element step for B
- C = Floating-point Main Memory output vector
- K = Integer element step for C
- N = Integer element count

DESCRIPTION: TMMSUB subtracts N elements of the vector B in Main Memory from N elements of the vector with base address ITMA in Table Memory, and stores the differences in the vector C in Main Memory.

EXAMPLE:

N=3
I=J=K=1
ITMA=8192

TMLOC: 8192 8193 8194
A : 3.0 4.0 5.0
B : 1.0 3.0 2.0
C : 2.0 1.0 3.0
APPENDIX A

PURPOSE: To compute the linear combination of two vectors, one in Table Memory and the other in main memory, and store the resultant vector in main memory.

CALL FORMAT: CALL TMVLC2 (S1, ITMA, 52, B, J, C, K, N)

PARAMETERS:
- S1 = Floating-point scalar coefficient for the TM input vector A
- ITMA = Integer base address of the TM input vector A
- S2 = Floating-point scalar coefficient for the MD input vector B
- B = Floating-point MD input vector
- J = Integer element step for B
- C = Floating-point MD output vector
- K = Integer element step for C
- N = Integer element count

DESCRIPTION: C(m) = S1 * A(m) + S2 * B(m); for m = 1 to N

EXAMPLE:

N = 3
S1 = -1.0
S2 = 2.0
J = 1
K = 1
ITMA = 8192

TMLOC: 8192 8193 8194
A : 1.0 2.0 3.0
B : 4.0 0.5 0.0
C : 7.0 -1.0 -3.0
PURPOSE: To add the elements of two vectors in Table Memory and store the sums in Main Memory.

CALL FORMAT: CALL TTMADD(ITMA,I,ITMB,J,C,K,N)

PARAMETERS:
- ITMA = Integer base address of TM input vector A
- I = Integer element step for A
- ITMB = Integer base address of TM input vector B
- J = Integer element step for B
- C = Floating-point Main Memory output vector
- K = Integer element step for C
- N = Integer element count

DESCRIPTION: TTMADD adds N elements of the vector with base address ITMA in Table Memory to N elements of the vector with base address ITMB in Table Memory, and stores the sums in N elements of the vector C in Main Memory.

EXAMPLE:

N=3
I=J=K=1
ITMA = 8192
ITMB = 8292

TMLOC: 8192 8193 8194
A     : 1.0 2.0 3.0

TMLOC: 8292 8293 8294
B     : 4.0 5.0 6.0

C     : 5.0 7.0 9.0
PURPOSE: To subtract the elements of two vectors in Table Memory and store the differences in a vector in Main Memory.

CALL FORMAT: CALL TTMSUB(ITMA,I,ITMB,J,C,K,N)

PARAMETERS:
- ITMA = Integer base address of TM input vector A
- I = Integer element step for A
- ITMB = Integer base address of TM input vector B
- J = Integer element step for B
- C = Floating-point Main Memory output vector
- K = Integer element step for C
- N = Integer element count

DESCRIPTION: TTMSUB subtracts N elements of the vector with base address ITMB in Table Memory from N elements of the vector with base address ITMA in Table Memory, and stores the resulting differences in a vector C in Main Memory.

EXAMPLE:

N=3
I=J=K=1
ITMA = 8192
ITMB = 8292

TMLOC: 8192 8193 8194
A : 3.0 4.0 5.0

TMLOC: 8292 8293 8294
B : 2.0 1.0 1.0

C : 1.0 3.0 4.0
APPENDIX A

PURPOSE: To multiply the elements of two vectors in Table Memory and store the resulting products in a vector in Table Memory.

CALL FORMAT: CALL TTTMUL(ITMA,I,ITMB,J,ITMC,K,N)

PARAMETERS:
- ITMA = Integer base address of TM input vector A
- I = Integer element step for A
- ITMB = Integer base address of TM input vector B
- J = Integer element step for B
- ITMC = Integer base address of TM output vector C
- K = Integer element step for C
- N = Integer element count

DESCRIPTION: TTTMUL multiplies N elements of the vector with base address ITMA in Table Memory by N elements of the vector with base address ITMB in Table Memory, and stores the resultant products in the vector with base address ITMC in Table Memory.

NOTE: Writable Table Memory begins at address 8192.

EXAMPLE:

N=3
I=J=K=1
ITMA = 8192
ITMB = 8292
ITMC = 8392

TMLOC: 8192 8193 8194
A : 1.0 2.0 3.0

TMLOC: 8292 8293 8294
B : 3.0 4.0 5.0

TMLOC: 8392 8393 8394
C : 3.0 8.0 15.0
PURPOSE: To compute the linear combination of two vectors, one in Table Memory and the other in main memory, and store the resultant vector in Table Memory.

CALL FORMAT: CALL TTVLC2 (S1, ITMA, S2, B, J, ITMC, N)

PARAMETERS:

- **S1** = Floating-point scalar coefficient for the TM input vector A
- **ITMA** = Integer base address of the TM input vector A
- **S2** = Floating-point scalar coefficient for the MD input vector B
- **B** = Floating-point MD input vector
- **J** = Integer element step for B
- **ITMC** = Integer base address of the TM output vector C
- **N** = Integer element count

DESCRIPTION: C(m) = S1 * A(m) + S2 * B(m); for m = 1 to N

Where A and C are in Table Memory, and B, S1, and S2 are in main memory.

Note: Writable Table Memory begins at address 8192.

EXAMPLE:

| N  | 3 |
| S1 | -1.0 |
| S2 | 2.0 |
| J  | 1 |
| ITMA | 8192 |
| ITMC | 8195 |

TMLOC: 8192 8193 8194
A : 1.0 2.0 3.0
B : 4.0 0.5 0.0

TMLOC: 8195 8196 8197
C : 7.0 -1.0 -3.0
PURPOSE: To fetch the contents of a specified memory word.

CALL FORMAT: Function Value = PEEK(Addr)

PARAMETERS: Function Value = The unformatted contents of the specified memory location
Addr = An integer specifying the address to be accessed

DESCRIPTION: The specified memory location is accessed and its contents returned as the function-value output. The output is the unformatted word. That is, no format conversion is performed by the function.

EXAMPLE:

(Assuming location 1000 contains 01 23 34 56 78 9A BC DE (hex) )

Addr : 1000
Function Value : 01 23 34 56 78 9A BC DE
APPENDIX A

**********
* * *
* VIFIX *
* * *
**********

--- VECTOR INTEGER FIX ---

PURPOSE: To fix to 53-bit integers the elements of a floating-point vector.

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

PARAMETERS: A = Floating-point input vector
I = Integer element step for A
C = Long-integer output vector
K = Integer element step for C
N = Integer element count
F = Integer flag (0 to round, 1 to truncate)

DESCRIPTION: C(m)=FIX(A(m)); for m=1 to N

EXAMPLE:

N = 4
F = 0

A : 1.7 -1.5 -3.2 3.5
C : 2 -2 -3 4.0

N = 4
F = 1

A : 1.7 -1.5 -3.2 3.5
C : 1 -1 -3 3.0
PURPOSE: To pack each four 64-bit floating-point numbers into one destination word as 16-bit quarter words.

CALL FORMAT: CALL VPK16(A,I,C,K,N,F)

PARAMETERS:  
A = Floating-point input vector  
I = Integer element step for A  
C = Signed-quarterword-integer output vector  
K = Integer element step for C  
N = Integer element count (destination words)  
F = Integer flag (0 to round, 1 to truncate)

DESCRIPTION: VPK16 fixes and packs four floating-point numbers from vector A into 16-bit quarter words in a single word of vector C, packing an array of positive integers with values from 0 to 65535, or an array of signed two's complement integers with values from -32768 to 32767, but does not check for out-of-range values.

EXAMPLE:

N = 2  
F = 0  
A : 8.3 -7.9 6.5 5.6 4.1 3.4 -2.5 1.1  
C : 0000FF80006000 00040003FFE001  
F = 1  
A : 8.3 -7.9 6.5 5.6 4.1 3.4 -2.5 1.1  
C : 0000FF90006005 00040003FFE001
APPENDIX A

--- VECTOR 32-BIT INTEGER PACK ---

PURPOSE: To pack each two 32 bit halfword integer source words into one destination word as halfword-integers-packed.

CALL FORMAT: CALL VPKI32(A,I,C,K,N)

PARAMETERS: A = Halfword integer input vector
I = Integer element step for A
C = Halfword-integer-packed output vector
K = Integer element step for C
N = Integer element count (destination words)

DESCRIPTION: C(m) bits 0 to 31 = A(2m-1) bits 32 to 63
C(m) bits 32 to 63 = A(2m) for m=1 to N

(Bits are numbered 0-63 from left to right).

VPKI32 packs two halfword integers from vector A into 32-bit halfwords in a single word of vector C. It packs an array of positive integers with values from 0 to 4294967295, or an array of signed 2's complement integers with values from -2147483648 to 2147483647. VPKI32 does not check for values out of range.

EXAMPLE: N = 3
I = 2
K = 3

(A indicates 'undefined')

A: 80C000000000006 80A000000000005 808000000000004 806000000000003 800000000000002 800000000000001 800000000000000 7FFFFFFF00000000 7FDFFFFFFF000000 800000000000000 800000000000000 C: 0000000000000004 XXXXXXXXXX0000 0000000000000008 0000000000000004 XXXXXXXXXX0000 0000000000000003 XXXXXXXXXX0000 0000000000000002 XXXXXXXXXX0000 0000000000000001 XXXXXXXXXX0000 0000000000000000 FFFFFFEEFFFFFFFC 7FFFFFFF00000000 7FDFFFFFFF000000 800000000000000 800000000000000
PURPOSE: To scale the elements of a vector by a power of 2 such that a selected scalar will just fit into a specified integer bit width, and then fix the scaled elements to integers.

CALL FORMAT: CALL VSCALE(A,I,B,C,K,N,NB,IEXP)

PARAMETERS:
- A = Floating-point input vector
- I = Element step for A
- B = Floating-point input scalar
- C = Long-integer output vector
- K = Element step for C
- N = Element count
- NB = Long-integer input scalar
  (Desired width, 2 to 28 bits, of integers)
- IEXP = Long-integer output scalar
  (Exponent of scale factor used)

DESCRIPTION: C(m) = FIX (A(m)*{2**IEXP}) for m=0 to N-1
where IEXP=NB-E-1,
and B = FRAC*(2**E).
VSCALE scales by a power of 2 every element of the vector A so that the scalar B will just fit into an NB-bit width integer, and then fixes the scaled elements and stores them in vector C. IEXP is set to the scale factor chosen. If the scalar is larger in magnitude than any element of A, no fixing overflows will occur.

EXAMPLE:

(with N=5, NB=12)

B : 10.0
A : 12.0 5.0 0.2 -4.0 0.01
C : 1280 640 25 -512 1
IEXP : 7
*** VSHFX ***
--- VECTOR SHIFT AND FIX ---
*** VSHFX ***

PURPOSE: To shift (multiply by a power of 2) and then fix (truncate) to integers the elements of a floating-point vector.

CALL FORMAT: CALL VSHFX(A,I,C,K,N,NS)

PARAMETERS:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Floating-point input vector</td>
</tr>
<tr>
<td>I</td>
<td>Integer element step for A</td>
</tr>
<tr>
<td>C</td>
<td>Long-integer output vector</td>
</tr>
<tr>
<td>K</td>
<td>Integer element step for C</td>
</tr>
<tr>
<td>N</td>
<td>Integer element count</td>
</tr>
<tr>
<td>NS</td>
<td>Integer power of 2 (May be negative)</td>
</tr>
</tbody>
</table>

DESCRIPTION: \[ C(m) = \text{FIX} \{ A(m) \times (2^\text{NS}) \} \]; for \( m = 1 \) to \( N \)

EXAMPLE:

\[
\begin{align*}
N & = 3 \\
\text{NS} & = 2 \\
A & = 1.0 \ 2.0 \ 3.2 \\
C & = 4 \ 8 \ 12
\end{align*}
\]
APPENDIX A

********** **********
*   *           --- VECTOR 8-BIT BYTE UNPACK ---
*   *
**********

PURPOSE: To unpack eight 8-bit unsigned bytes from each source word and store them in eight destination words as 64-bit floating-point numbers.

CALL FORMAT: CALL VUP8(A,I,C,K,N)

PARAMETERS: A = Unsigned-byte-integer input vector
I = Integer element step for A
C = Floating-point output vector
K = Integer element step for C
N = Integer element count (source words)

DESCRIPTION: Unpacks eight 8-bit bytes from a single word of vector A storing them as eight floating-point numbers in vector C. The unpacked bytes have values from 0 to 255.

EXAMPLE:

N = 2
A : 0807660504030201 0807660504030201
C : 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
     8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
APPENDIX A

**********  **********
*            *
* VUP32    *  ------- VECTOR 32-BIT BYTE UNPACK -------
*            *
**********  **********

PURPOSE: To unpack two 32-bit unsigned halfwords from each source word and store them in two destination words as 64-bit floating-point positive numbers.

CALL FORMAT: CALL VUP32(A,I,C,K,N)

PARAMETERS: A = Unsigned-halfword-integer input vector
I = Integer element step for A
C = Floating-point output vector
K = Integer element step for C
N = Integer element count (source words)

DESCRIPTION: VUP32 unpacks two 32-bit halfwords from a single word of vector A, storing them as two positive 64-bit floating-point integers in vector C. The unpacked halfwords have values from 0 to 4294967295.

EXAMPLE:

N = 4

A : 0000000000000007
    0000000000000005
    0000000000000003
    0000000000000001

C : 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
**********
* * 
* VUPS8 * ------- VECTOR 8-BIT SIGNED BYTE UNPACK -------
* * 
**********

PURPOSE: To unpack eight 8-bit signed bytes from each source word and store them in eight destination words as 64-bit floating-point numbers.

CALL FORMAT: CALL VUPS8(A,I,C,K,N)

PARAMETERS: A = Signed-byte-integer input vector
I = Integer element step for A
C = Floating-point output vector
K = Integer element step for C
N = Integer element count (source words)

DESCRIPTION: VUPS8 unpacks eight 8-bit signed bytes from a single word of vector A, storing them as eight floating-point numbers in vector C. The unpacked bytes have values from -128 to 127.

EXAMPLE:

N = 2

A : 08F9060554FD0201 087FA50403FE01

C : 8.0 -7.0 6.0 5.0 4.0 -3.0 2.0 1.0
     8.0 7.0 -6.0 5.0 4.0 3.0 -2.0 1.0
PURPOSE: To unpack two 32-bit signed two's complement halfwords from each source word and store them in two destination words as signed 64-bit floating-point numbers.

CALL FORMAT: CALL VUPS32(A,I,C,K,N)

PARAMETERS: A = Signed-halfword-integer input vector
I = Integer element step for A
C = Floating-point output vector
K = Integer element step for C
N = Integer element count (source words)

DESCRIPTION: VUPS32 unpacks two 32-bit signed two's complement halfwords from a single word of vector A, storing them as two floating-point numbers in vector C. The unpacked halfwords have values from -2147483648 to 2147483647.

EXAMPLE:

N = 4

A : 00000008FFFFFFFF
     0000006000000005
     FFFFFFFCFEEEEEEDD
     FFFFFFFF20000001

C : 8.0 -7.0 6.0 5.0 -4.0 -3.0 -2.0 1.0
PURPOSE:
To unpack two 32-bit halfword integers from each source word and store them as two destination words, in unsigned integer format.

CALL FORMAT: CALL VUUI32(A,I,C,K,N)

PARAMETERS:
A = Halfword integer packed input vector
I = A address increment
C = 32 bit integer output vector
K = C address increment
N = Integer element count (source words)

DESCRIPTION:
C(2m-1) = A(m) bits 0 to 31
C(2m) = A(m) bits 32 to 63
for m=0 to N-1
(Bits are numbered 0-63 from left to right).

VUUI32 unpacks two 32-bit unsigned halfword integers from a single word of vector A and stores them as two unsigned halfword integers in vector C. The unpacked halfwords have values from 0 to 4294967295.

EXAMPLE:
N = 3
I = 3
K = 2

A: 0000000000000007 C: 0000000000000008
0000000000000005 XXXXXXXXXXXXXXXX
0000000000000003 XXXXXXXXXXXXXXXX
0000000000000001 XXXXXXXXXXXXXXXX
FFFFFFFFFFFFFFFFFE 0000000000000002
FFFFFFFFDDDDDDDDFC XXXXXXXXXXXXXXXX
FFFFFFFFBBBBBBBBFCA 0000000000000001
XXXXXXXXXXXXXXXXX 0000000000000000
0000000000000000 0000000000000000
0000000000000000

(XXX indicates 'undefined')
********** ******"****
DADD
**********

PURPOSE: To form a double-precision sum of two double-precision numbers.

CALL FORMAT: CALL DADD(XDBLE,YDBLE,ZDBLE)

PARAMETERS: XDBLE = Real vector input (double precision)
             YDBLE = Real vector input (double precision)
             ZDBLE = Real vector output (double precision)
(A double-precision value is stored in a 2-element real array. First element contains high word, second element contains low word.)

DESCRIPTION: Adds the double-precision number in XDBLE to the double-precision number in YDBLE and stores the high word of the double-precision sum in ZDBLE(1) and the low word in ZDBLE(2).
**APPENDIX A**

**Purpose:**
To perform the dot product of two real vectors, accumulating the result in double precision (128 bits), and returning the result in single precision (64 bits).

**Call Format:**
```
SW = DADOT(N,A,I,B,J)
```

**Parameters:**
- **N**: Integer element count
- **A**: Real input vector
- **I**: Integer element step for A
- **B**: Real input vector
- **J**: Integer element step for B
- **SW**: Real output result

**Description:**
- \( SW = \sum (A(m) \times B(m)) \) for \( m = 1 \) to \( N \)
- \( SW = 0.0 \) for \( N < 1 \)
- If the element increment, INC, of a vector is negative, then the vector is indexed in reverse order, i.e., element \((N-1) \times INC + 1\) to the first element (BLAS convention).
PURPOSE: To form a double-precision product of two double-precision numbers.

CALL FORMAT: CALL DMUL(XDBLE,YDBLE,ZDBLE)

PARAMETERS: XDBLE = Real vector input (double precision)
YDBLE = Real vector input (double precision)
ZDBLE = Real vector output (double precision)
   (A double-precision value is stored in a 2-element real array. First element contains high word, second element contains low word.)

DESCRIPTION: Multiplies the double-precision number in XDBLE by the double-precision number in YDBLE and stores the high word of the double-precision product in ZDBLE(1) and the low word in ZDBLE(2).
APPENDIX A

********** **********
* * DNEG * --- NEGATE DOUBLE-PRECISION NUMBER --- * DNEG *
* * **********

PURPOSE: To negate a double-precision number.

CALL FORMAT: CALL DNEG(XDBLE,ZDBLE)

PARAMETERS: XDBLE = Real vector input (double precision)
ZDBLE = Real vector output (double precision)
(A double-precision value is stored in a 2-element real array. First element contains high word, second element contains low word.)

DESCRIPTION: Negates the double-precision number in XDBLE and stores the high word of the double-precision result in ZDBLE(1) and the low word in ZDBLE(2).
APPENDIX A

********** **********
* * * DSUBRR * --- SINGLE TO DOUBLE-PRECISION SUBTRACT --- * DSUBRR * *
* * * ********** **********

PURPOSE: To form a double-precision difference of two single-precision numbers.

CALL FORMAT: CALL DSUBRR(X,Y,ZDBLE)

PARAMETERS: X = Real scalar input
            Y = Real scalar input
            ZDBLE = Real vector output (double precision)
                    (A double-precision value is stored in a 2-element real array. First element contains high word, second element contains low word.)

DESCRIPTION: Subtracts the single-precision number in Y from the single-precision number in X and stores the high word of the double-precision difference in ZDBLE(1) and the low word in ZDBLE(2).
PURPOSE: To compute the absolute value of a real number.

CALL FORMAT: Function-value = ABS(arg)

PARAMETERS: Function-value = Real Floating-point scalar output
Arg = Real Floating-point scalar input

DESCRIPTION: Function-value = |arg|
APPENDIX A

**********
*        *
*   AINT  *
*        *
**********  

--- TRUNCATE REAL NUMBER ---

PURPOSE: To truncate a real number.

CALL FORMAT: Function-value = AINT(arg)

PARAMETERS: Function-value = Real floating-point scalar output
Arg = Real floating-point scalar input

DESCRIPTION: Function-value = FLOAT(FIXT(arg))
PURPOSE: To compute the logarithm of a real number.

CALL FORMAT: Function-value = ALOG(arg) or ALOG10(arg)

PARAMETERS: Function-value = Real Floating-point scalar output
Arg = Real Floating-point scalar input

DESCRIPTION: Function-value = Ln(arg); for ALOG
= Log(10)(arg); for ALOG10
PURPOSE: To round a real number to the nearest whole number.

CALL FORMAT: Function-value = ANINT(arg)

PARAMETERS:
- Function-value = Real floating-point scalar output
- Arg = Real floating-point scalar input

DESCRIPTION: Function-value = FLOAT(FIX(arg))
PURPOSE: To compute the arctangent of a real number or of the ratio of two real numbers.

CALL FORMAT: Function-value = ATAN(arg1) or ATAN(arg1/arg2)

PARAMETERS: Function-value = Real Floating-point scalar output
Arg1 = Real Floating-point scalar input
Arg2 = Real Floating-point scalar input

DESCRIPTION: Function-value = ATAN(arg1) or ATAN(arg1/arg2)
PURPOSE: To compute the absolute value (magnitude) of a complex number.

CALL FORMAT: Function-value = CABS(arg)

PARAMETERS: Function-value = Floating-point scalar output
Arg = Complex floating scalar input

DESCRIPTION: Function-value = SQRT (R(arg)**2 + I(arg)**2)
CDIV — COMPLEX/COMPLEX DIVIDE

PURPOSE: To divide a complex number into a complex number.

CALL FORMAT: Function Value = Arg2/Arg1

PARAMETERS:
- Function Value = Complex Floating scalar output
- Arg1 = Complex Floating scalar input
- Arg2 = Complex Floating scalar input

DESCRIPTION: Function Value = \( \frac{R(\text{arg2}) + I(\text{arg2})}{R(\text{arg1}) + I(\text{arg1})} \)
PURPOSE: To divide a real number into a complex number.

CALL FORMAT: Function Value = Arg2/Arg1

PARAMETER: Function Value = Complex Floating scalar output
Arg1 = Real Floating-point scalar input
Arg2 = Complex Floating scalar input

DESCRIPTION: Function Value = R(arg(2))+I(arg(2))/arg(1)
PURPOSE: To compute the natural logarithm of a complex number.

CALL FORMAT: Function-value = CLOG(arg)

PARAMETERS: Function-value = Complex floating scalar output
Arg = Complex floating scalar input

DESCRIPTION: R(Function-value) = ALOG((CABS(arg))
I(Function-value) = ATAN(I(arg)/R(arg))
PURPOSE: To compute the conjugate of a complex number.

CALL FORMAT: Function-value = CONJG(arg)

PARAMETERS: Function-value = Complex floating scalar output
             Arg      = Complex floating scalar input

DESCRIPTION: Function-value = R(arg)-I(arg)
APPENDIX A

**********
*     *
*  COSH  *
*     *
**********

--- REAL NUMBER HYPERBOLIC COSINE ---

**********
*     *
*  COSH  *
*     *
**********

PURPOSE: To compute the hyperbolic sine or cosine of a real number.

CALL FORMAT: Function-value = SINH(arg) or COSH(arg)

PARAMETERS: Function-value = Real Floating-point scalar output
Arg = Real Floating-point scalar input

DESCRIPTION: Function-value = SINH(arg) or COSH(arg)
APPENDIX A

*****
* *
* CPOWCI *
* *
*****

--- COMPLEX TO INTEGER POWER ---

*****
* *
* CPOWCI *
* *
*****

PURPOSE: To raise a complex number to an integer power.

CALL FORMAT: Function Value = Arg1**Arg2

PARAMETERS: Function Value = Complex Floating scalar output
Arg1 = Complex Floating scalar input
Arg2 = Integer scalar input

DESCRIPTION: Function Value = \((R(\text{Arg1})+I(\text{Arg1}))^{\text{Arg2}}\)
PURPOSE: To raise a real number to a complex power.

CALL FORMAT

Function Value = Arg1**Arg2

PARAMETERS:

Function Value = Complex Floating scalar output
Arg1 = Real Floating-point scalar input
Arg2 = Complex Floating scalar input

DESCRIPTION: Function Value = arg1**(R(arg2)+I(arg2))
PURPOSE: To compute the square root of a complex number.

CALL FORMAT: Function-value = CSQRT(arg)

PARAMETERS:

Function-value = Complex floating scalar output
Arg = Complex floating scalar input

DESCRIPTION:

if R(arg) ≥ 0
R(function value) = F
I(function value) = I(arg)/(2*F)

if R(arg) < 0
R(function value) = I(arg)/(2*F)
I(function value) = SIGN(I(arg))*F

where F = SQRT((ABS(R(arg))+CABS(arg))/2)
PURPOSE: To compute the exponential of a real number.

CALL FORMAT: Function-value = EXP(arg)

PARAMETERS: Function-value = Real Floating-point scalar output
Arg = Real Floating-point scalar input

DESCRIPTION: Function-value = Exp(arg)

NOTE: arg>789.089 traps with an overflow error condition.
PURPOSE: To compute the integer positive difference of two integers.

CALL FORMAT: Function-value = IDIM(arg1, arg2)

PARAMETERS: Function-value = Integer scalar output
Arg1 = Integer scalar input
Arg2 = Integer scalar input

DESCRIPTION: Function-value = MAX((arg1 - arg2), 0)
PURPOSE:  To raise an integer number to an integer power.

CALL FORMAT: Function Value = Arg1**Arg2

PARAMETERS: Function Value = Integer scalar output  
Arg1    = Integer scalar input  
Arg2    = Integer scalar input

DESCRIPTION: Function Value = arg1**arg2
PURPOSE: To compute the remainder when one integer is divided by another.

CALL FORMAT: Function-value = MOD(arg1,arg2)

PARAMETERS: Function-value = Integer scalar output  
Arg1 = Integer scalar input  
Arg2 = Integer scalar input

DESCRIPTION: Function-value = Arg1-INT(arg1/arg2)*arg2
**SCALAR RANDOM NUMBER GENERATOR**

To generate one pseudo-random number.

**CALL FORMAT:**

\[
\text{Function-value} = \text{RAN(SEED)}
\]

**PARAMETERS:**

- **Function-value** = Floating-point output scalar
- **SEED** = Integer input/output scalar

**DESCRIPTION:**

RAN returns one pseudo-random floating-point number between \(0.0\) and \(1.0\). The routine uses a linear congruential generator initialized by SEED to generate an integer, which is then scaled to produce the function-value. SEED is replaced with the integer generated. SEED may be any integer between 0 and \(2^{26}-1\).

RAN generates the same sequence of integers as VRAND. Thus the two statements

\[
C = \text{RAN(SEED)}
\]

and

\[
\text{CALL VRAND(SEED,C,1,1)}
\]

are equivalent.

**EXAMPLE:**

\[
\begin{align*}
\text{SEED} & = 1000 \\
\text{RAN(SEED)}: & = 0.800484940496603 \\
\text{SEED} & = 53719635
\end{align*}
\]
PURPOSE: To raise a non-negative real number to a real power.

CALL FORMAT: Function Value = Arg1**Arg2

PARAMETERS: Function Value = Real Floating-point scalar output
Arg1 = Real Floating-point scalar input
Arg2 = Real Floating-point scalar input

DESCRIPTION: Function Value = arg1**arg2

(If Arg2 is a whole number, Arg1 can be negative.)
PURPOSE: To divide a real number into a real number or into 1.

CALL FORMAT: Function Value = Arg2/Arg1
or 1.0/Arg1

PARAMETERS: Function Value = Real Floating-point scalar output
Arg1 = Real Floating-point scalar input
Arg2 = Real Floating-point scalar input

DESCRIPTION: Function Value = arg2/arg1 for RDIV
or 1.0/arg1 for RRCP
**APPENDIX A**

-------------------------  -------------------------
* *  * SIGN *  *  *   REAL NUMBER SIGN TRANSFER   *
* *  * SIGN *  *  *
-------------------------  -------------------------

**PURPOSE:** To give the magnitude of a real number with the sign of a second real number.

**CALL FORMAT:** Function-value = SIGN(arg1,arg2)

**PARAMETERS:**
- Function-value = Real Floating-point scalar output
- Arg1 = Real Floating-point scalar input
- Arg2 = Real Floating-point scalar input

**DESCRIPTION:** Function-value = Sign(arg2)*ABS(arg1)
SINCOS

PURPOSE:
To compute the sine and cosine of a real number.

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

PARAMETERS:
A = Floating-point input scalar
CA = Floating-point output scalar
SA = Floating-point output scalar

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

SINCOS computes both the sine and the cosine in about the same time as the SIN function alone.

NOTE: A 32-bit integer overflow exception is generated if the input argument is too large (greater than approximately 8.0E+5). In this case, the output result has less than six decimal digits of precision.

An added feature of this routine is that it can also be called as a complex function. If FIF$PR_SINCOS is declared as complex, the call

Function-value = FIF$PR_SINCOS(A)

returns the complex value

Function-value = CMPLX(COS(A),SIN(A)).

This is convenient for converting polar coordinates to rectangular coordinates.

EXAMPLE:

A = 0.0
CA = 1.0
SA = 0.0
PURPOSE: To compute the square root of a real number.

CALL FORMAT: Function-value = SQRT(arg)

PARAMETERS: Function-value = Real Floating-point scalar output
             Arg = Real Floating-point scalar input

DESCRIPTION: Function-value = SQRT(arg)
PURPOSE: To compute the hyperbolic tangent of a real number.

CALL FORMAT: Function-value = TANH(arg)

PARAMETERS: Function-value = Real Floating-point scalar output
Arg = Real Floating-point scalar input

DESCRIPTION: Function-value = TANH(arg)
APPENDIX B

DATA REPRESENTATIONS FOR STORING SPARSE VECTORS AND MATRICES

B.1 INTRODUCTION

This appendix presents information to help the user understand and use the sparse vector and sparse matrix subroutines. It describes the data representations (or formats) both accepted as input and produced as output by these routines. This appendix also spells out parameter naming conventions common to many of these subroutines.

There are four subroutines that convert sparse vectors and matrices between their packed and full representations: Sparse Vector Pack (SVPACK), Sparse Vector Unpack (SVUPCK), Sparse Matrix Pack (SMPACK), and Sparse Matrix Unpack (SMUPCK).

B.2 SPARSE VECTOR STORAGE

An N-dimensional sparse vector \( V \) is represented in packed-vector format by \( N, NS, S, \) and \( IEN \) where:

\[
\begin{align*}
N & \quad \text{a scalar, is the dimension of } V. \\
NS & \quad \text{a scalar, is the number of nonzero values in } V. \\
S & \quad \text{a vector of length } NS, \text{ contains the nonzero values of } V. \\
IEN & \quad \text{a vector of length } NS, \text{ contains the location in } V \text{ of each corresponding element in } S \text{ [i.e., } V(IEN(k)) = S(k) \text{ for } k=1,NS].
\end{align*}
\]

For example, the following sparse vector

\[
[0.0 \ 3.2 \ 0.0 \ 7.8 \ 0.0 \ 0.0 \ 0.0 \ -19.3]
\]

can be represented in packed-vector format as follows:

\[
\begin{align*}
N: & \quad 8 \\
NS: & \quad 3 \\
S: & \quad [3.2 \ 7.8 \ -19.3] \\
IEN: & \quad [2 \ 4 \ 8 ]
\end{align*}
\]

So, \( S(1) \)'s location in \( V \) can be found in \( IEN(1) \), \( S(2) \)'s in \( IEN(2) \), ..., \( S(NS) \)'s in \( IEN(NS) \).

The nonzero values in \( S \) are generally ordered as they appear in \( V \). However, they can be ordered differently if the order is compatible with the subroutine to be used.
Except for differences in the IP vector, formats I and III are the same, as are formats II and IV.

Each attribute associated with a particular format type and the consequences of using that attribute are explained in detail in the sections that follow.

B.3.1 Matrix Format Type I (COL-ORDER PTRS-ONLY)

A sparse matrix $A$ is represented by $M$, $N$, $NS$, $S$, $IN$, and $IP$, in format I where:

- $M$ a scalar, is the number of rows in $A$.
- $N$ a scalar, is the number of columns in $A$.
- $NS$ a scalar, is the number of nonzero values in $A$.
- $S$ a real vector of length $NS$, contains the nonzero values of $A$ in column order.
- $IN$ an integer vector of length $NS$, contains the row in $A$ of each corresponding value in $S$ [i.e., $IN(k) = \text{row in } A \text{ of } S(k)$ for $k=1,NS$].
- $IP$ an integer vector of length $N+1$, contains one element for every column in $A$.
  Each element indicates the location in $S$ that holds that column's first nonzero value (exception: empty columns).
  $IP$'s $N+1$st element is a sentinel.

The sentinel element $IP(N+1)$ holds the number $NS+1$.
In general, $IP(i)$ contains the location in $S$ that refers to $A$'s $i$-th column, for $i=1,N$.

If a column in $A$ is empty, then the entry in $IP$ for that column is the same as the entry for the next nonempty column, or if there is no such column, sentinel value in $IP(N+1)$ is used.

The matrix:

\[
\begin{array}{ccccccc}
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 4.5 & 0.0 & 0.2 & 3.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 9.9 & 7.1 & 5.8 & 0.0 & 0.0 \\
0.0 & 1.3 & 0.0 & 8.3 & 0.0 & 0.0 \\
\end{array}
\]

as expressed in Type I Format:

\[
\begin{array}{c}
M: 5 \\
N: 6 \\
NS: 8 \\
\end{array}
\]
B.3.3 Matrix Format Type III (COL-ORDER PTRS-SUMS)

A sparse matrix A is represented by M, N, NS, S, IN, and IP, in format III where:

- M a scalar, is the number of rows in A.
- N a scalar, is the number of columns in A.
- NS a scalar, is the number of nonzero values in A.
- S a real vector of length NS, contains the nonzero values of A in column order.
- IN an integer vector of length NS, contains the row in A of each corresponding value in S [i.e., IN(k) = row in A of S(k) for \( k=1,NS \).]
- IP an integer vector of length 2*N, contains two elements for every column in A:
  - (a) the location in S that holds that column's first nonzero value (exception: an empty column).
  - (b) that column's total number of nonzero elements.

IP(i) and IP(i+N) always refer to the i-th column in A, for i=1,N. IP(1) to IP(N) holds locations as in (a) above and IP(N+1) to IP(2*N) holds sums as in (b) above.

If a column in A is empty, then the (a)-entry in IP for that column is the same as the (a)-entry for the next nonempty column, or if there is no such column, the number NS+1. (Note that the (b)-entry is zero.)

The matrix:

```
H.H  H.H  0.0  H.H  0.H  0.H  0.H
H.H  4.5  0.H  .0'.2  3.H  0.H
0.H  0.0  .0'.H  0.H  0.H
r.r  r.r  0   0   .r   r.r  r.r
```

as expressed in Type III Format:

```
M:  5
N:  6
NS:  8

S:  [4.5  9.9  1.3  7.1  0.2  5.8  8.3  3.0]
IN: [2  4  5  4  2  4  5  2 ]
IP: [1  1  4  5  8  9  0  3  1  3  1  0]
```

Note that lengths of S and IN equal NS (=8); S is in column order; the length of IP equals 2*N (=12); IP contains both locations and sums; IN contains row numbers.
APPENDIX C

APPENDIX C

SPARSE LINEAR SYSTEM ROUTINES

C.1 INTRODUCTION

This appendix contains information to help the user understand and use the sparse linear system routines in the Advanced Math Library. The sparse linear system routines are APAL64 routines that provide an efficient method for solving the linear system $Ax = b$ where the coefficient matrix is sparse and is stored in packed form.

There are twelve generic sparse linear system routines in all. The name of each routine consists of a four-letter generic name followed by the single digit "2". The first two letters of the name indicate the coefficient matrix type (i.e., the problem domain), and the last two letters indicate its function. The single digit is a version number and is not included on the names of the original routines, which were superseded as of the F63 release (see Appendix G).

The types of coefficient matrices are:

- RU A is real.
- RS A is real and symmetric.
- CU A is complex.
- CS A is complex and symmetric.

The functions performed are:

- FR Factor the coefficient matrix.
- SV Solve the system given the factorization of the coefficient matrix.
- FS Factor and solve (combines FR and SV).

In general, the time required to factor the coefficient matrix is much greater than the time required to solve the factored system. Therefore, by having separate routines for each of these functions, the factorization need only be performed once when solving a number of systems that all have the same coefficient matrix.
Denote the determinant of a square matrix $A$ by $\text{Det}(A)$. The "not equal" relation will be denoted by the symbol "#".

Assume an $n \times n$ lower-triangular matrix $L$, and $n \times n$ upper-triangular matrix $U$, such that $A = LU$. Then the system $Ax = b$ is equivalent to $LUx = b$. Letting $Ux = y$, where $y$ is an $n$-dimensional vector, then the system becomes $Ly = b$. Thus, it is possible to decompose the original system into two triangular systems which, in general, are easier to solve. It is then possible to find the solution to the original system $x$, by the following two steps:

1) Solve $Ly = b$ for $y$ by forward elimination

2) Solve $Ux = y$ for $x$ by backward substitution

If there does exist an $L$ and $U$ such that $LU = A$, then $L$ and $U$ are not uniquely determined unless additional conditions are imposed. One such set of conditions is to require the following:

$$U(i,i) = 1 \text{ for } i = 1 \text{ to } n.$$ 

By imposing this restriction on $U$, the remaining elements of $L$ and $U$ can now be solved obtaining the following:

$$L(i,j) = A(i,j) - \text{Sum}[L(i,k) * U(k,j), k=1,j-1]$$
for $i = 1$ to $n$, $j = 1$ to $n$, and $i \geq j$ \hspace{1cm} eq(1a)

$$U(i,j) = (A(i,j) - \text{Sum}[L(i,k) * U(k,j), k=1,i-1]) / L(i,i)$$
for $i = 1$ to $n-1$, $j = 2$ to $n$, and $j > i$ \hspace{1cm} eq(1b)

It is clear from an examination of the expressions above that a unique $L$ and $U$ exist if and only if $L(i,i) \neq 0$ for $i = 1$ to $n-1$. Letting $A(k)$ denote the $k$-th order principle submatrix of $A$ (i.e., the submatrix formed by the intersection of the first $k$ rows and the first $k$ columns of $A$), then it follows from equation (1) that $A(k) = L(k)U(k)$. Recall from elementary linear algebra that:

(a) if $A = BC$, then $\text{Det}(A) = \text{Det}(B)\text{Det}(C)$; and

(b) if $T$ is an $n \times n$ triangular matrix,

then $\text{Det}(T) = \text{Prod}[T(i,i), i=1,n]$. 

A common variation of the method of LU factorization involves the further factorization of L into MD where M is a lower-triangular matrix with \(M(i,i) = 1\) for \(i=1\) to \(n\) and D is a diagonal matrix. The elements of M and D are found to be:

\[
M(i,j) = \frac{L(i,j)}{L(j,j)} \quad \text{eq}(2a)
\]

\[
D(i,i) = L(i,i) \quad \text{eq}(2b)
\]

Equations (1) and (2) can be used to show that M is the transpose of U if A is symmetric. The LDU theorem can now be stated.

C.3.2 LDU Theorem

If A is an \(n \times n\) matrix, then there exist unique matrices L, D, and U, where L is lower-triangular with \(L(i,i) = 1\), D is diagonal with \(D(i,i) \neq 0\), and U is upper-triangular with \(U(i,i) = 1\) such that \(A = LDU\) if and only if \(\text{Det}(A[k]) \neq 0\) for \(k = 1\) to \(n\). Furthermore, if \(A = LDU\) and A is symmetric, then L is the transpose of U.

If A is factored into LDU, then the original system, \(Ax = b\), is equivalent to \(LDUx = b\). Letting \(Ux = y\) and \(Dy = z\) where \(y\) and \(z\) are \(n\)-dimensional vectors, then the original system decomposes into two triangular systems and a diagonal system that are solved by the following three steps:

1) Solve \(Lz = b\) for \(z\) by forward elimination.
2) Solve \(Dy = z\) for \(y\).
3) Solve \(Ux = y\) for \(x\) by backward substitution.

Since LDU-factorization requires more work than LU-factorization, the latter is preferable unless A is symmetric. In that case, the direct computation and storage of U is unnecessary since U is the transpose of L and the factors are written LDL'.

C.4 FILL-IN

If the coefficient matrix A is sparse, (this is assumed when using the sparse system routines) store only the nonzero elements of A with information about the location of the nonzero elements. (The manner in which this is done is described in Section C.5.) It is very desirable to do this since both storage requirements and execution time can be greatly reduced.
The following algorithm is given in the form of a FORTRAN subroutine for determining fill-in:

```fortran
SUBROUTINE FILLIN(N, A, IA)
    REAL A(N,N)
    LOGICAL IA(N,N)
    DO 110 I = 1, N
        IA(I,1) = .FALSE.
        IA(1,I) = .FALSE.
        IF(A(I,1) .NE. 0.0) IA(I,1) = .TRUE.
        IF(A(1,I) .NE. 0.0) IA(1,I) = .TRUE.
    110 CONTINUE
    DO 150 J = 2, N
        DO 140 I = 2, N
            IF(A(I,J) .NE. 0.0) GO TO 130
            K2 = MIN0(I,J) - 1
            DO 120 K = 1, K2
                IF(IA(I,K) .AND. A(K,J)) GO TO 130
            120 CONTINUE
            IA(I,J) = .FALSE.
        130 CONTINUE
        GO TO 140
    140 CONTINUE
    IA(I,J) = .TRUE.
    150 CONTINUE
    RETURN
END
```

The amount of fill-in varies as the rows and columns of A are permuted and algorithms exist to minimize the fill-in. However, any permuting of the rows and columns of A to decrease fill-in may be detrimental to the numerical stability.

Before leaving the subject of fill-in, note that if A is a band matrix, then the superposition of L and U will also be a band matrix and will have the same bandwidth as A. Therefore, if A is a band matrix where the nonzero elements are dense within the band consider every element within the band to be sparse without introducing a great number of unnecessary sparse elements.
Finally, if A is unsymmetric, then an additional integer vector IDP of length \( N \) is required for pointers into \( S \) to the diagonal elements of \( A \). For example,

- If \( A \) is real and \( A(j,j) \) is stored in \( S(k) \), then \( IDP(j) = k \).
- If \( A \) is complex and \( A(j,j) \) is stored in \( S(2k-1) \) and \( S(2k) \), then \( IDP(j) = k \).

Consider the following example; let \( A \) be the real matrix:

\[
\begin{bmatrix}
2 & 0 & 0 & 0 & 4 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Note that \( A(4,5) \) is a sparse element since it is a fill-in element.

The vectors \( S \), IRN, ICP, and IDP that are required to represent \( A \) are:

<table>
<thead>
<tr>
<th>WORD</th>
<th>S</th>
<th>IRN</th>
<th>ICP</th>
<th>IDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3.0</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>-1.0</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4.0</td>
<td>1</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>1.0</td>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2.0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5.0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The output from the factorization routines and the input to the solution routines require these same vectors except that \( S \) then contains the sparse elements of the superposition of \( L \) and \( U \) on \( A \) (\( L' \), \( D \), and \( U \) if \( A \) is symmetric) with the diagonal elements replaced by their reciprocals. (See the example above.)

\[
L = \begin{bmatrix}
2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\
0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Therefore, the superposition of \( L \) and \( U \) with the diagonal elements replaced with their reciprocals is

\[
\begin{bmatrix}
0.5 & 0.0 & 0.0 & 2.0 & 0.0 \\
0.0 & 1.0 & 0.0 & 0.0 & 2.0 \\
0.0 & 0.0 & -1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 1.0 & -6.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.2 \\
\end{bmatrix}
\]
APPENDIX D

BASIC LINEAR ALGEBRA SUBPROGRAMS

D.1 INTRODUCTION

This appendix contains information to help the user understand and use the routines, which constitute the basic linear algebra subprograms (BLAS) as implemented within the LINPACK Users' Guide Manual, Appendix A. These routines are a subset of the basic linear algebra subprograms developed by Lawson, Hanson, Kincaid, and Krogh (refer to ACM Trans. Math. Software 5, 3 (Sept. 1979) pp. 324-325) for many of the basic vector operations of numerical linear algebra. The package was intended to be called from FORTRAN programs, and was developed to focus on performance improvements of the well known set of LINPACK routines (refer to the LINPACK Users' Guide, Appendix A).

In addition, four routines have been added which are extensions to four of the BLAS routines (real and complex versions of the dot product and scalar times vector plus vector) which provide for repeated invocations with only one subroutine call. These are useful in many applications including matrix multiply and matrix factoring (refer to examples D.4.9, D.4.10, and D.4.11).

Double precision entry points allow the routines to handle standard calls to BLAS double-precision routines. There are no specific double-precision routines implemented, since the single precision routines use the standard 64-bit wide floating-point numbers.

When called from FORTRAN, the BLAS routines perform according to the algorithmic description in Appendix A, LINPACK User's Guide. In particular, negative subscript increment specification results in adjustment of the vector base address, as described in Section D.2. (No such base address adjustment needs to take place when the MLSP entries are used. However, when calling the routines from APAL64 base address adjustment is used.)

Much of the information in Sections D.2 and D.4 is taken from Appendix 3 of the NTIS-distributed Sandia National Labs. report, SAND77-0898, Basic Linear Algebra Subprograms for Fortran Usage, by Lawson, Hanson, Kincaid, and Krogh, and is reprinted with their kind permission. Floating Point Systems, Inc., gratefully acknowledges the suggestions given by R. J. Hanson.
D.3 ROUTINE CALLING SEQUENCES, ALGORITHMS, TIMINGS

The names of entities used in BLAS calls conform in general to standard FORTRAN conventions. In particular, names that begin with I or N pertain to integer data types; names that begin with C pertain to complex data types, and names that begin with S (for scalar) pertain to real (floating-point) data types.

The roots of the names pertain to function. The routines with -DOT- as root calculate different versions of the dot product, SDOT calculating the inner product of real vectors, CDOTC and CDOTU calculating complex inner products conjugated and unconjugated respectively.

**COPY** Replaces (moves or copies) elements of a vector with elements of another.

**AXPY** Stands for "aX+Y". It is intended to perform the elementary matrix operation of adding to the elements of a vector the scalar multiple of another vector.

**SCAL** Multiplies a vector by a scalar.

**SWAP** Interchanges (or swaps) elements of two vectors.

**ASUM** Calculates the absolute sum of a vector; that is, the sum of the absolute values of each element.

**I-AMAX** Calculates the index, or subscript, of the component of a vector of the largest absolute value.

**S-NRM2** Calculates the 2-norm, or Euclidean length of a vector. It carefully concerns itself with scaling problems to maintain accuracy and exponent range, by testing each component before adding its square to the accumulating partial sum. Usually it would be appropriate to use SQRT(DOT) for the same operation with greater speed but less robustness.

**ROT** Rotates a vector of pairs of points.

The parameter names are also standardized. These routines all deal with one or two vectors, usually coming from matrix rows or columns. The first vector is X; the second, -Y. Increments between consecutive elements of a vector are named INCX and INCY. Scalars are named A and -B.

Speed values reflect average values, without regard for vector placement, for typical APFTN64 compilations. Often much improvement is possible by judicious placement of elements among memory modules. Also, initial setup times are not included, only the loop values; which results in a value which is a constant multiple of N, the number of elements in the destination vector.
D.3.4 Complex Function CDOTU(N,CX,INCX,CY,INCY)

Function value = sum( CX(m)*CY(m), for the N vector elements indexed by m).

D.3.5 Subroutine CROTG(CA,CB,SC,CSIN)

SC := |CA|/r, CSIN := conjugate(CB)*CA/|CA|/r, CA := CR
where: r=\sqrt{|CA|^2 + |CB|^2} and SC,CSIN chosen to satisfy

\[ CR = SC*CA + CSIN*CB \]
\[ \theta = CSIN'*CA + SC*CB. \]

D.3.6 Subroutine CSCAL(N,CA,CX,INCX)

CX(m) := CA*CX(m), for the N vector elements indexed by m.

D.3.7 Subroutine CSSCAL(N,SA,CX,INCX)

CX(m) := SA*CX(m), for the N vector elements indexed by m.

D.3.8 Subroutine CSROT(N,CX,INCX,CY,INCY,SC,SS)

CX(m) := SC*CX(m)+SS*CY(m)
CY(m) := -SS*CX(m)+SC*CY(m), for the N vector elements indexed by m.

D.3.9 Subroutine CSWAP(N,CX,INCX,CY,INCY)

CX(m) := CY(m), for the N vector elements indexed by m.

D.3.10 Integer Function ICAMAX(N,CX,INCX)

Function value = I such that \(|\text{Re } CX(I)|+|\text{Im } CX(I)|\) is largest of the N values \(|\text{Re } CX(m)|+|\text{Im } CX(m)|\).

D.3.11 Integer Function ISAMAX(N,SX,INCX)

Function value = smallest I such that \(|SX(I)|\) is largest of all N values \(|SX(m)|\).

D.3.12 Real Function SASUM(N,SX,INCX)

Function value = sum( |SX(m)|, for the N values indexed by m).
APPENDIX D

D.3.21 Subroutine SROTM(N,SX,INCX,SY,INCY,PARAM)

If PARAM(1) = 1.0 then

\[ SX(m) := \text{PARAM}(2) \times SX(m) + \text{PARAM}(4) \times SY(m) \]
\[ SY(m) := \text{PARAM}(3) \times SY(m) + \text{PARAM}(5) \times SY(m), \]

for the N vector elements indexed by m.

If PARAM(1) = 0.0 then

\[ SX(m) := SX(m) + \text{PARAM}(4) \times SY(m) \]
\[ SY(m) := \text{PARAM}(3) \times SY(m) + \text{PARAM}(5) \times SY(m), \]

for the N vector elements indexed by m.

If PARAM(1) = -1.0 then

\[ SX(m) := \text{PARAM}(2) \times SX(m) + \text{PARAM}(4) \times SY(m) \]
\[ SY(m) := \text{PARAM}(3) \times SY(m) + \text{PARAM}(5) \times SY(m), \]

for the N vector elements indexed by m.

If PARAM(1) is not 1, 0, or -1, the routine returns without modifying the vector elements. It thus becomes equivalent to an identity transformation.

D.3.22 Subroutine SROTMG(D1,D2,B1,B2,PARAM)

If \(|D1 \times B1 \times B1| > |D2 \times B2 \times B2|\) and \(D2 \times B2 \neq 0\) then

\[ \text{PARAM}(1) := 0.0 \]
\[ \text{PARAM}(2,4) := -B2/B1, D2 \times B2 \times D1 \times B1, \] so that the SROTM matrix becomes \((H11,H21,H12,H22) = (1,-B2/B1,D2 \times B2/D1 \times B1,1)).\)

\[ D1 := D1/U \]
\[ D2 := D2/U \]
\[ B1 := B1 \times U \text{ where } U = 1.0 + (D1 \times B1 \times B1)/(D2 \times B2 \times B2). \]

If \(|D1 \times B1 \times B1| \leq |D2 \times B2 \times B2|\) and \(D2 \times B2 \neq 0\) then

\[ \text{PARAM}(1) := 1.0 \]
\[ \text{PARAM}(2,5) := D1 \times B1/(D2 \times B2), B1 \times B2 \text{ so that the SROTM matrix becomes } (H11,H21,H12,H22) = (D1 \times B1/D2 \times B2,-1,1,B1/B2). \]

\[ D1,D2,B1 := D2/U,D1/U,B2 \times U \text{ where } U = 1 + D1 \times B1 \times B1/(D2 \times B2 \times B2). \]

If \(D2 \times B2 = 0\), then

the rotation matrix in SROTM becomes the identity, \(\text{PARAM}(1) := -2.0\).
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 $N < 1$, SDOTN returns with no action taken.

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

If $M < 1$ and $ISW[1] = 0$, SDOTN returns with $Z(j) = 0$ for $j = 1$ to $N$.

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

D.3.26 **Complex Subroutine CDOTN(ISW,N,M,X,IXI,IXO,Y,IYI,IYO,Z,IZO)**

$$Z(jz) = r * C(jz) + s * \sum_{i=1}^{M} A(\text{ix}) * B(\text{iy}) \text{, } j=1,N$$

where:

$$\text{ix} = (j-1) * \text{IXO} + (i-1) * \text{IXI} + 1$$

$$\text{iy} = (j-1) * \text{IYO} + (i-1) * \text{IYI} + 1$$

$$\text{jz} = (j-1) * \text{IZO} + 1$$

$$s = \begin{cases} 1.0, & \text{if ISW[6]} = 0 \\ -1.0, & \text{if ISW[6]} = 1 \end{cases}$$

$$r = \begin{cases} 0.0, & \text{if ISW[1]} = 0 \\ 1.0, & \text{if ISW[1]} = 1 \end{cases}$$

$$A = \begin{cases} X, & \text{if ISW[2]} = 0 \\ \text{Conjg}(X), & \text{if ISW[2]} = 1 \end{cases}$$

$$B = \begin{cases} Y, & \text{if ISW[3]} = 0 \\ \text{Conjg}(Y), & \text{if ISW[3]} = 1 \end{cases}$$

$$C = \begin{cases} Z, & \text{if ISW[4]} = 0 \\ \text{Conjg}(Z), & \text{if ISW[4]} = 1 \end{cases}$$

and $ISW[k] = \text{bit } k \text{ of ISW}$.

$ISW$ is a one word function selector switch and is treated as a bit string with the bits numbered from the least significant bit (bit 0). If a given bit is set (equal to one), then the function option that corresponds to that bit is selected.

If $IZO = 0$, then CDOTN sets $Z(1)$ equal to the accumulated sum of all $N$ dot products. If $ISW[1] = 1$ also, then input $Z(1)$ is 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 $Y$. 
D.3.28 **Subroutine CAXPYN(ISW,N,M,A,IAO,X,IXI,IXO,Y,IYI,IYO)**

\[ Y(iy) = s \times B(ja) \times Z(ix) + Y(iy), \quad i=1,M \quad j=1,N \]

where:
\[
\begin{align*}
ja &= (j-1) \times IAO + 1 \\
ix &= (j-1) \times IXO + (i-1) \times IXI + 1 \\
iy &= (j-1) \times IYO + (i-1) \times IYI + 1 \\
s &= \begin{cases} 
1 & \text{if ISW[0]} = 0 \\
-1 & \text{if ISW[0]} = 1 
\end{cases} \\
B &= A, \quad \text{if ISW[2]} = 0 \\
&= \text{Conjg}(A), \quad \text{if ISW[2]} = 1 \\
Z &= X, \quad \text{if ISW[3]} = 0 \\
&= \text{Conjg}(X), \quad \text{if ISW[3]} = 1
\end{align*}
\]

and ISW[k] = bit k of ISW.

ISW is a one word function selector switch and is treated as a bit string with the bits numbered from the least significant bit (bit 0). If a given bit is set (equal to one), then the function option that corresponds to that bit is selected.

Memory words occupied by A may intersect those occupied by X. However, memory occupied by Y should not, in general, intersect that occupied by A or X.

Furthermore, the user will not get meaningful results when distinct "columns" of Y intersect. For instance, if M = 100, IYI = 1 and IYO = 96, then Y(97,1) = Y(1,2), Y(98,1) = Y(2,2) etc.

However, cases involving IYO = 0 produce meaningful results in that the products are accumulated to Y. That is, successive results bound for the same storage location in Y are added together rather than stored over each other. In this case, the calculation is reduced to a single call to CDOTN which executes much faster than the general case speeds given in the routine documentation.

IYI = 0 is of no real value and is omitted for speed and simplicity.

If N < 1, CAXPYN returns with no action.

If M < 1, CAXPYN returns with no action.

If IYI = 0, CAXPYN returns with no action.
D.4.5 Set to Identity

Given an N by N matrix A, to set A = the identity matrix and then B = A.

\[
\begin{align*}
\text{DO } & 50 \text{ J}=1,N \\
50 & \text{ CALL SCOPY}(N,0.0,E0,0,A(1,J),1) \\
& \text{ CALL SCOPY}(N,1.0,E0,0,A,MDA+1) \\
\text{DO } & 60 \text{ J}=1,N \\
60 & \text{ CALL SCOPY}(N,A(1,J),1,B(1,J),1)
\end{align*}
\]

D.4.6 Matrix Columns Interchange

To interchange the columns of an M by N matrix C, where the column to be interchanged with column J is in a type INTEGER array IP(*), and has the value IP(J).

\[
\begin{align*}
\text{DO } & 70 \text{ J}=1,N \\
& \text{ L=IP(J)} \\
& \text{ IF(J.NE.L) CALL SSWAP(M,C(1,J),1,C(1,L),1) \\
70 & \text{ CONTINUE}
\end{align*}
\]

D.4.7 Matrix Transposition

To transpose an N by N matrix A in-place, where MDA is the first dimensioning parameter of the array A(*, *).

\[
\begin{align*}
\text{IF(N.EQ.1) GOTO 85} \\
\text{DO } & 80 \text{ J}=1,N-1 \\
80 & \text{ CALL SSWAP(N-J,A(J,J+1),MDA,A(J+1,J),1) \\
85 & \text{ CONTINUE}
\end{align*}
\]

D.4.8 Column Vector Circular Shift

Finally, an inefficient but illustrative code segment which swaps in-place the components of the column vector

\[(x_1, \ldots, x_K, x_{K+1}, \ldots, x_N)\]
D.4.10 Matrix Factorization Using SAXPYN

This subroutine performs matrix factorization $A=LU$ without pivoting using SAXPYN. $L$ replaces the lower part of $A$ excluding the diagonal. $L$ is assumed implicitly to have 1's on its diagonal. $U$ replaces the upper part of $A$ including the diagonal. $A$ itself is treated as a doubly dimensioned array with first dimension $NO$. $A$ is assumed to contain an $NI \times NI$ matrix stored by rows rather than the usual storage by columns. This storage scheme allows SAXPYN to more efficiently process the current row being used for elimination.

```
SUBROUTINE MFBGE(A,NI,NO)
  REAL A(1)
  INTEGER NI,NO

  IF(NI.LE.1) RETURN
  JINV=1
  NOP=NO+1

  DO 100 I=1,NI-1
    AINV=1.0/A(JINV)
    JC=JINV+NO

  C    COMPUTE THE NEXT COLUMN OF L
    CALL VSMUL(A(JC),NO,AINV,A(JC),NO,NI-1)
    MN=NI-1

  C    PERFORM THE ELIMINATION GETTING A NEW LOWER RIGHT MINOR
    CALL SAXPYN(1,MN,MN,A(JC),NO,A(JINV+1),1,0,A(JC+1),1,NO)

  JINV=JINV+N0
  100 CONTINUE

RETURN
END
```

D.4.11 Matrix Factorization Using SDOTN

This subroutine performs matrix factorization $A=LU$ without pivoting using SDOTN. $L$ replaces the lower part of $A$ excluding the diagonal. $L$ is assumed implicitly to have 1's on its diagonal. $U$ replaces the upper part of $A$ including the diagonal. $A$ itself is treated as a doubly dimensioned array with first dimension $NO$. $A$ is assumed to contain an $NI \times NI$ matrix stored by columns. Doolittle's method is used.
APPENDIX E

APMATH64 FUNCTION GENERATION ROUTINES

E.1 INTRODUCTION

This appendix presents information to help the programmer understand and use the function generation routines of the Advanced Math Library. The function generation routines are APAL64 routines that provide a flexible and efficient way of evaluating functions of one, two, three, or four variables. They do this using table lookup with linear interpolation. Lookup is performed by searching for the breakpoints, using either a binary search (successive interval halving) or a step search (nearest neighbor), depending on whether the user expects the values of the input variables to be rapidly or slowly changing from call to call.

Function generation is described in the following manner:

Given the function F of one input variable x, for which the value of F is known at specific values of x (breakpoints) (x(1), x(2), ...), calculate the value of the function for an arbitrary value of x by linearly interpolating between the values of F at the pair of breakpoints x(i) <= x <= x(i+1).

After determining the pair of breakpoints (x(i), x(i+1)), between which the value of x lies, calculate the function by the following formula:

\[ F(x) = F(x(i)) + (F(x(i+1)) - F(x(i))) \times \frac{x - x(i)}{x(i+1) - x(i)} \]

This process is extended to two-variable functions by three applications of the above formula, to three-variable functions by seven applications, and four-variable functions by 15 applications.

The function generation routines are listed below (refer to Appendix A for detailed descriptions):

breakpoint search routines:  BIN
                          STEP

function evaluation routines:  FUN1
                                 FUN2
                                 FUN3
                                 FUN4
2 variables: X, Y

3 functions: F1(X,Y), F2(X,Y), F3(X,Y)

3 X breakpoints: X1, X2, X3

4 Y breakpoints: Y1, Y2, Y3, Y4

Coordinate value breakpoint tables:

\begin{align*}
X(BRK)(1,1) &= X_1 \\
(2,1) &= X_2 \\
(3,1) &= X_3 \\
(1,2) &= \frac{1.0}{(X_2 - X_1)} \\
(2,2) &= \frac{1.0}{(X_3 - X_2)} \\
(3,2) &= 0.0 \\
X_1 &< X_2 < X_3 \\
Y(BRK)(1,1) &= Y_1 \\
(2,1) &= Y_2 \\
(3,1) &= Y_3 \\
(4,1) &= Y_4 \\
(1,2) &= \frac{1.0}{(Y_2 - Y_1)} \\
(2,2) &= \frac{1.0}{(Y_3 - Y_2)} \\
(3,2) &= \frac{1.0}{(Y_4 - Y_3)} \\
Y_1 &< Y_2 < Y_3 < Y_4
\end{align*}

Taken together, these two breakpoint tables specify a 3 X 4 rectangular grid of points in the X-Y plane.

Function value breakpoint table:

\begin{align*}
F(BRK)(1,1,1) &= F_1(X_1, Y_1) \\
(2,1,1) &= F_1(X_2, Y_1) \\
(3,1,1) &= F_1(X_3, Y_1) \\
(1,2,1) &= F_1(X_1, Y_2) \\
(2,2,1) &= F_1(X_2, Y_2) \\
(3,2,1) &= F_1(X_3, Y_2) \\
(1,3,1) &= F_1(X_1, Y_3) \\
(2,3,1) &= F_1(X_2, Y_3) \\
(3,3,1) &= F_1(X_3, Y_3) \\
(1,4,1) &= F_1(X_1, Y_4) \\
(2,4,1) &= F_1(X_2, Y_4) \\
(3,4,1) &= F_1(X_3, Y_4) \\
(1,1,2) &= F_2(X_1, Y_1) \\
(3,4,2) &= F_2(X_3, Y_4) \\
(1,1,3) &= F_3(X_1, Y_1) \\
(3,4,3) &= F_3(X_3, Y_4)
\end{align*}

Figure E-1 Example Coordinate and Function Value Breakpoint Tables
where

\[ \text{XY}(1,1) = X \text{ coordinate value of the first input point} \]
\[ \text{XY}(2,1) = Y \text{ coordinate value of the first input point} \]

E.3 CALLING APMATH64 FUNCTION GENERATION ROUTINES

The function generation package is used with System Job Executive (SJE) as follows:

\[
\begin{align*}
\text{APFTN64} & \quad \leftrightarrow \quad \text{Advanced Math Library routines} \\
\text{driver} & \quad \leftrightarrow \quad \text{Library routines}
\end{align*}
\]

The user must supply the APFTN64 driver, which contains calls to the appropriate Advanced Math Library routines. The coordinate value tables, function value table, and the input points are generated in the APFTN64 driver. The APFTN64 driver routine does the following:

- Generates the coordinate value breakpoint tables.
- Generates the function value breakpoint table.
- Specifies the input points.
- Sets up a loop to process the input points.
- For each input point, determines the appropriate breakpoint pair for each of the coordinates of the input point by calling the BIN or STEP routine for each coordinate. (This feature makes input point data structure arbitrary.)
- Calls the appropriate function evaluation routine (i.e., FUN1, FUN2, FUN3, or FUN4 from the Advanced Math Library).

Refer to the Advanced Math Library documentation and the individual program headers for descriptions of these programs.
The structure of the output function value array FVAL is arbitrary to the extent that each call to the Advanced Math Library function generation routine returns the interpolated values for all of the functions at the given input point in one array. For this reason, FVAL is perhaps most conveniently dimensioned \( FVAL(NF,NIP) \).

Lines 35 through 61 load the coordinate value breakpoint tables. In the FUN4 example below, the program assumes the function values to be known (i.e., generated by the user) on the four-dimensional grid of points as specified by the coordinate value breakpoint tables.

Lines 65 through 73 load the function value breakpoint table. In this example, it is done by simply cycling through all possible coordinate value combinations, evaluating the four functions at each point.

Lines 77 through 100 specify the input points calling for interpolated values for each of the four functions.

Lines 102 through 120 call the APMATH64 BIN and FUN4 subroutines, pass the tables and other arrays as arguments, and write out the results.
(0055) WBRK(1,1) = -25.0
(0056) WBRK(2,1) = -15.0
(0057) WBRK(3,1) = 0.0
(0058) WBRK(1,2) = 1.0/(WBRK(2,1)-WBRK(1,1))
(0059) WBRK(2,2) = 1.0/(WBRK(3,1)-WBRK(2,1))
(0060) WBRK(3,2) = 0.0
(0061)
(0062)
(0063) C LOAD FBRK ARRAY
(0064)
(0065) DO 100 I4=1, NW
(0066) DO 100 I3=1, NZ
(0067) DO 100 I2=1, NY
(0068) DO 100 I1=1, NX
(0069) FBRK(I1,I2,I3,I4,1) = XBRK(I1,1) + YBRK(I2,1) + ZBRK(I3,1) * WBRK(I4,1)
(0070) FBRK(I1,I2,I3,I4,2) = XBRK(I1,1) * WBRK(I4,1) + YBRK(I2,1) + ZBRK(I3,1)
(0071) FBRK(I1,I2,I3,I4,3) = XBRK(I1,1) + YBRK(I2,1) * WBRK(I4,1) + ZBRK(I3,1)
(0072) FBRK(I1,I2,I3,I4,4) = XBRK(I1,1) * ZBRK(I3,1) + YBRK(I2,1) * WBRK(I4,1)
(0073) 100 CONTINUE
(0074)
(0075) C LOAD X,Y,Z,W ARRAYS
(0076)
(0077) X(1) = 0.3
(0078) Y(1) = -5.0
(0079) Z(1) = 5.1
(0080) W(1) = -1.5
(0081)
(0082) X(2) = 1.1
(0083) Y(2) = -3.0
(0084) Z(2) = 4.0
(0085) W(2) = -2.2
(0086)
(0087) X(3) = 0.9
(0088) Y(3) = -9.0
(0089) Z(3) = 7.5
(0090) W(3) = -13.0
(0091)
(0092) X(4) = 2.9
(0093) Y(4) = -6.0
(0094) Z(4) = 6.0
(0095) W(4) = -15.0
(0096)
(0097) X(5) = 0.4
(0098) Y(5) = -5.0
(0099) Z(5) = 4.5
(0100)
(0101)
(0102) DO 150 I1=1, NIP
(0103) CALL BIN(XBRK,X(I1),IX,DRX,NX)
(0104) CALL BIN(YBRK,Y(I1),IY,DRY,NY)
(0105) CALL BIN(ZBRK,Z(I1),IZ,DRZ,NZ)
(0106) CALL BIN(WBRK,W(I1),IW,DRW,NW)
(0107) CALL FUN4(FBRK,NX,NY,NZ,NW,NF,IX, IY, IZ, IW,
(0108) DRX,DRY,DRZ,DRW,FVAL(I1,1))
F.1 INTRODUCTION

The Simulation Library contains a set of routines which are useful in modeling various continuous systems. These continuous systems are characterized by ordinary differential equations (ODE) and three-dimensional coordinate transformations of rigid bodies, which simulate physical models.

The methods provided for solving ODE's include Runge-Kutta and Euler explicit methods, which require no previous evaluation of functions or derivatives, as well as multistep Adams implicit and explicit methods, which require previous evaluation of the function and one or more previous derivatives. These multistep methods can be started with lower order methods or with the Runge-Kutta routine. Once started, the multistep routines require only a single evaluation of the derivative functions per call. The fourth order Runge-Kutta method requires four evaluations per time step.

The three-dimensional rotation matrix routine forms a rotation matrix from a sequence of rotational specifications and can be used in conjunction with routine CTRN3 to perform three-dimensional coordinate transformations consisting of rotation plus translation.

An additional utility routine is provided to rapidly calculate the cosine and sine of an angle, both of which are often required in geometric transformations and graphic output.

F.2 SINGLE STEP METHODS

RKGTF Runge-Kutta-Gill-Thompson: a fourth order single step method to solve a system of ordinary differential equations (ODE's) using Thompson's numerical enhancement of the Runge-Kutta-Gill method. The routine requires an APFTN64 user subroutine to evaluate the derivatives.

ABP1 Adams-Bashforth predictor order one: a single step predictor method, also known as Euler's method, for solving ODE's.

AMC1 Adams-Moulton corrector order one: a single step predictor method, also known as the backward Euler method, used for corrections to "stiff" ODE's.
APFTN64 ROUTINE FOR USE WITH RKGTF

SUBROUTINE DFUN(T, N, Y, F)

*** DFUN *** SAMPLE APFTN64 ROUTINE ***

DIMENSION Y(N), F(N)

DO 10 I = 1, N
   F(I) = Y(I)
10 CONTINUE

CORRESPONDS TO SOLUTIONS OF FORM:

\[ Y(I) = \gamma * \exp(T) \]

RETURN
END
# Appendix G

## List of Superseded Routines

### F90 Release

<table>
<thead>
<tr>
<th>Old Routines</th>
<th>New Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMMM32</td>
<td>FMMM or FMMMV</td>
</tr>
<tr>
<td>MMUL32</td>
<td>MMUL, FMMM, or FMMMV</td>
</tr>
<tr>
<td>ZVABS, VABS</td>
<td>VABS</td>
</tr>
<tr>
<td>ZVADD, VADD</td>
<td>VADD</td>
</tr>
<tr>
<td>ZVFILT, VFLOAT</td>
<td>VFLOAT</td>
</tr>
<tr>
<td>ZVIFIX, VIFIX</td>
<td>VIFIX</td>
</tr>
<tr>
<td>ZVMWA, VMSA</td>
<td>VMSA</td>
</tr>
<tr>
<td>ZVMUL, VMUL</td>
<td>VMUL</td>
</tr>
<tr>
<td>ZVNEG, VNEG</td>
<td>VNEG</td>
</tr>
<tr>
<td>ZVRVRS, VRVRS</td>
<td>VRVRS</td>
</tr>
<tr>
<td>ZVSADD, VSADD</td>
<td>VSADD</td>
</tr>
<tr>
<td>ZVSMA, VSMAS</td>
<td>VSMAS</td>
</tr>
<tr>
<td>ZVSMSE, VMSB</td>
<td>VMSBS</td>
</tr>
<tr>
<td>ZVSMUL, VSMUL</td>
<td>VSMUL</td>
</tr>
<tr>
<td>ZVSMUR, VSQ</td>
<td>VSQ</td>
</tr>
<tr>
<td>ZVSUB, VSUB</td>
<td>VSUB</td>
</tr>
<tr>
<td>ZVS WA, VS WA</td>
<td>VS WA</td>
</tr>
</tbody>
</table>

The replacement routines for FMMM32 and MMUL32 include the same functionality as FMMM32 and MMUL32 and are also more general.

### F93 Release

<table>
<thead>
<tr>
<th>Old Routines</th>
<th>New Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIMAG</td>
<td>AIMAG (APFTN64 intrinsic)</td>
</tr>
<tr>
<td>CSFR</td>
<td>CSFR2</td>
</tr>
<tr>
<td>CSFS</td>
<td>CSFS2</td>
</tr>
<tr>
<td>CSSV</td>
<td>CSSV2</td>
</tr>
<tr>
<td>CSUR</td>
<td>CSUR2</td>
</tr>
<tr>
<td>CUFS</td>
<td>CUFS2</td>
</tr>
<tr>
<td>CUFSV</td>
<td>CUFSV2</td>
</tr>
<tr>
<td>EXTRU</td>
<td>EXTRACT (APFTN64 intrinsic)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT (APFTN64 intrinsic)</td>
</tr>
<tr>
<td>IFIX</td>
<td>IFIX (APFTN64 intrinsic)</td>
</tr>
<tr>
<td>INSERT</td>
<td>INSERT (APFTN64 intrinsic)</td>
</tr>
<tr>
<td>LOC</td>
<td>LOC (APFTN64 intrinsic)</td>
</tr>
<tr>
<td>RSFR</td>
<td>RSFR2</td>
</tr>
<tr>
<td>RSFS</td>
<td>RSFS2</td>
</tr>
<tr>
<td>RSSV</td>
<td>RSSV2</td>
</tr>
</tbody>
</table>
EXCEPTIONS ENABLED ROUTINES INFORMATION AND INTERNAL SUBROUTINES

H.1 EXCEPTIONS ENABLED ROUTINES INFORMATION

Beginning with the G00 Release, all APMATH64 routines report valid exceptions.

H.2 INTERNAL SUBROUTINES

The following routines are used only as internal subroutines by other APMATH64 routines. These routines are listed here to facilitate interpretation of program tracebacks.

<table>
<thead>
<tr>
<th>INTERNAL SUBROUTINE</th>
<th>CALLING ROUTINE(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV2</td>
<td>CFFT, CFFTB, CFFTI, XCFFT</td>
</tr>
<tr>
<td>ADV4</td>
<td>CFFT, CFFTB, CFFTI, XCFFT</td>
</tr>
<tr>
<td>ALTNP</td>
<td>CCEPS</td>
</tr>
<tr>
<td>BITREV</td>
<td>CFFT, CFFTI</td>
</tr>
<tr>
<td>CBEAJY</td>
<td>CBEJYH</td>
</tr>
<tr>
<td>CBEDH</td>
<td>RKGTF</td>
</tr>
<tr>
<td>CBEDJ</td>
<td>RKGTF</td>
</tr>
<tr>
<td>CBERHY</td>
<td>CBEJYH</td>
</tr>
<tr>
<td>CBERJS</td>
<td>CBEJYH</td>
</tr>
<tr>
<td>CBERYH</td>
<td>CBEJYH</td>
</tr>
<tr>
<td>CLSTAT</td>
<td>CFFT, CFFTB, HAMM, REALTR, STSTAT, BLKMAN, HANN, CFFTI, IIRELT, IREALT, XCFFT</td>
</tr>
<tr>
<td>CTOR</td>
<td>RFFT2D</td>
</tr>
<tr>
<td>ENTVAR</td>
<td>SIMPLE</td>
</tr>
<tr>
<td>FFT2</td>
<td>CFFT, CFFTB, FFT2B, CFFTI, XCFFT</td>
</tr>
<tr>
<td>FFT2B</td>
<td>CFFTB</td>
</tr>
<tr>
<td>FFT4</td>
<td>CFFT, CFFTB, FFT4B, CFFTI</td>
</tr>
<tr>
<td>FFT4B</td>
<td>CFFTB</td>
</tr>
<tr>
<td>IFFT4</td>
<td>CFFTI</td>
</tr>
<tr>
<td>IIRELT</td>
<td>RFTII</td>
</tr>
<tr>
<td>INTEG</td>
<td>CBEJYH</td>
</tr>
<tr>
<td>IREALT</td>
<td>RFFT, RFTII, IIRELT</td>
</tr>
<tr>
<td>LPSPFI</td>
<td>SIMPLE</td>
</tr>
<tr>
<td>PHAUNW</td>
<td>CCEPS</td>
</tr>
<tr>
<td>PHCHCK</td>
<td>PHAUNW</td>
</tr>
<tr>
<td>REALTR</td>
<td>RFFT, RFFTB, RFFT, RFTII</td>
</tr>
<tr>
<td>RKGTF</td>
<td>INTEG</td>
</tr>
<tr>
<td>RTOC</td>
<td>RFFT2D</td>
</tr>
<tr>
<td>SET24B</td>
<td>FFT2B, FFT4B</td>
</tr>
<tr>
<td>SPCVAL</td>
<td>PHAUNW</td>
</tr>
<tr>
<td>STSTAT</td>
<td>CFFT, CFFTB, HAMM, REALTR, STSTAT, BLKMAN,</td>
</tr>
</tbody>
</table>
## APPENDIX I

**APMATH64 Routines in Page Order and by Type**

### BASIC MATH LIBRARY (VOLUME 1)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCMMUL</td>
<td>Complex matrix multiply</td>
<td>2</td>
</tr>
<tr>
<td>CDET</td>
<td>Complex matrix determinant</td>
<td>4</td>
</tr>
<tr>
<td>CIDOTPR</td>
<td>Complex dot product</td>
<td>6</td>
</tr>
<tr>
<td>CFFT</td>
<td>Complex-to-complex FFT (in place)</td>
<td>7</td>
</tr>
<tr>
<td>CFFTB</td>
<td>Complex-to-complex FFT (not in place)</td>
<td>8</td>
</tr>
<tr>
<td>CFFTM</td>
<td>Mixed-radix complex FFT (not-in-place)</td>
<td>9</td>
</tr>
<tr>
<td>CFFTSW</td>
<td>Complex FFT scale</td>
<td>11</td>
</tr>
<tr>
<td>CGMMUL</td>
<td>Complex general matrix multiply</td>
<td>12</td>
</tr>
<tr>
<td>CMATIN</td>
<td>Complex matrix inverse</td>
<td>14</td>
</tr>
<tr>
<td>CMDET</td>
<td>Complex matrix determinant</td>
<td>15</td>
</tr>
<tr>
<td>CMFACT</td>
<td>Complex matrix L/U factorization</td>
<td>17</td>
</tr>
<tr>
<td>CMTRAC</td>
<td>Complex sub-matrix trace</td>
<td>19</td>
</tr>
<tr>
<td>CMSOLV</td>
<td>Complex matrix equation solver</td>
<td>21</td>
</tr>
<tr>
<td>CMTRAN</td>
<td>Complex sub-matrix transpose</td>
<td>23</td>
</tr>
<tr>
<td>CMVD3</td>
<td>Complex 3x3 matrix mult., 3D Vectors</td>
<td>24</td>
</tr>
<tr>
<td>CMVD4</td>
<td>Complex 4x4 matrix mult., 4D Vectors</td>
<td>25</td>
</tr>
<tr>
<td>CONV</td>
<td>Convolution (correlation)</td>
<td>26</td>
</tr>
<tr>
<td>CRMUL</td>
<td>Complex-real matrix multiply</td>
<td>27</td>
</tr>
<tr>
<td>CROSSP</td>
<td>Complex 3D cross product</td>
<td>30</td>
</tr>
<tr>
<td>CRVADD</td>
<td>Complex and real vector add</td>
<td>31</td>
</tr>
<tr>
<td>CRVDIV</td>
<td>Complex and real vector divide</td>
<td>32</td>
</tr>
<tr>
<td>CRVMUL</td>
<td>Complex and real vector multiply</td>
<td>33</td>
</tr>
<tr>
<td>CRVSUB</td>
<td>Complex and real vector subtract</td>
<td>34</td>
</tr>
<tr>
<td>CSOLV</td>
<td>Complex system solver</td>
<td>35</td>
</tr>
<tr>
<td>CSOLQV</td>
<td>Complex matrix equation solver</td>
<td>36</td>
</tr>
<tr>
<td>CTRN2</td>
<td>2-D coordinate transform</td>
<td>39</td>
</tr>
<tr>
<td>CTRN3</td>
<td>3-Dimensional coordinate transformation</td>
<td>40</td>
</tr>
<tr>
<td>CVABS</td>
<td>Complex vector absolute value</td>
<td>42</td>
</tr>
<tr>
<td>CVADD</td>
<td>Complex vector add</td>
<td>43</td>
</tr>
<tr>
<td>CVCOMB</td>
<td>Complex vector combine</td>
<td>44</td>
</tr>
<tr>
<td>CVCONJ</td>
<td>Complex vector conjugate</td>
<td>45</td>
</tr>
<tr>
<td>CVEXP</td>
<td>Complex vector exponential</td>
<td>46</td>
</tr>
<tr>
<td>CVFILL</td>
<td>Complex vector fill</td>
<td>47</td>
</tr>
<tr>
<td>CVMA</td>
<td>Complex vector multiply and add</td>
<td>48</td>
</tr>
<tr>
<td>CVMAGS</td>
<td>Complex vector magnitude squared</td>
<td>50</td>
</tr>
<tr>
<td>CVMEXP</td>
<td>Complex vector multiply exponential</td>
<td>51</td>
</tr>
<tr>
<td>CVMOV</td>
<td>Complex vector move</td>
<td>52</td>
</tr>
<tr>
<td>CVMUL</td>
<td>Complex vector multiply</td>
<td>53</td>
</tr>
<tr>
<td>CVNEG</td>
<td>Complex vector negate</td>
<td>54</td>
</tr>
<tr>
<td>CVRCIP</td>
<td>Complex vector reciprocal</td>
<td>55</td>
</tr>
<tr>
<td>CVREAL</td>
<td>Form complex vector of reals</td>
<td>56</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>DESCRIPTION</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>RMSQV</td>
<td>Root-Mean-Square of Vector Elements</td>
<td>A-125</td>
</tr>
<tr>
<td>SCJMA</td>
<td>Self-Conjugate Multiply and Add</td>
<td>A-126</td>
</tr>
<tr>
<td>SGEFA</td>
<td>Real-General Matrix Factor</td>
<td>A-127</td>
</tr>
<tr>
<td>GESL</td>
<td>Real-General Matrix Solve</td>
<td>A-129</td>
</tr>
<tr>
<td>STGSL</td>
<td>Tridiagonal Matrix Solver</td>
<td>A-131</td>
</tr>
<tr>
<td>SM3MM</td>
<td>Submatrix Multiply</td>
<td>A-133</td>
</tr>
<tr>
<td>SM3MV</td>
<td>Submatrix Multiply</td>
<td>A-135</td>
</tr>
<tr>
<td>SN2</td>
<td>Squared Distance Between Two Vectors</td>
<td>A-137</td>
</tr>
<tr>
<td>SOLVEQ</td>
<td>Linear Equation Solver</td>
<td>A-138</td>
</tr>
<tr>
<td>STMM</td>
<td>Submatrix Transpose &amp; Multiply</td>
<td>A-140</td>
</tr>
<tr>
<td>SVE</td>
<td>Sum of Vector Elements</td>
<td>A-142</td>
</tr>
<tr>
<td>SVEMG</td>
<td>Sum of Vector Element Magnitudes</td>
<td>A-143</td>
</tr>
<tr>
<td>SVESQ</td>
<td>Sum of Vector Element Squares</td>
<td>A-144</td>
</tr>
<tr>
<td>SVS</td>
<td>Sum of Vector Signed Squares</td>
<td>A-145</td>
</tr>
<tr>
<td>TRIDIA</td>
<td>Tridiagonal Matrix Solver</td>
<td>A-146</td>
</tr>
<tr>
<td>VAAM</td>
<td>Vector Add, Add, and Multiply</td>
<td>A-147</td>
</tr>
<tr>
<td>VABS</td>
<td>Vector Absolute Value</td>
<td>A-148</td>
</tr>
<tr>
<td>VACOS</td>
<td>Vector Arccosine</td>
<td>A-149</td>
</tr>
<tr>
<td>VADD</td>
<td>Vector Add</td>
<td>A-150</td>
</tr>
<tr>
<td>VAINT</td>
<td>Vector Truncate</td>
<td>A-151</td>
</tr>
<tr>
<td>VALG</td>
<td>Vector Logarithm</td>
<td>A-152</td>
</tr>
<tr>
<td>VALG10</td>
<td>Vector Base 10 Logarithm</td>
<td>A-153</td>
</tr>
<tr>
<td>VAM</td>
<td>Vector Add and Multiply</td>
<td>A-154</td>
</tr>
<tr>
<td>VASIN</td>
<td>Vector Arcsine</td>
<td>A-155</td>
</tr>
<tr>
<td>VASM</td>
<td>Vector Add and Scalar Multiply</td>
<td>A-156</td>
</tr>
<tr>
<td>VATAN</td>
<td>Vector Arctangent</td>
<td>A-157</td>
</tr>
<tr>
<td>VATAN2</td>
<td>Vector Arctangent (2 Arguments)</td>
<td>A-158</td>
</tr>
<tr>
<td>VCLIP</td>
<td>Vector Clip</td>
<td>A-159</td>
</tr>
<tr>
<td>VCLR</td>
<td>Vector Clear</td>
<td>A-160</td>
</tr>
<tr>
<td>VCONS</td>
<td>Vector Cosine</td>
<td>A-161</td>
</tr>
<tr>
<td>VCCOSH</td>
<td>Vector Cosine (Hyperbolic)</td>
<td>A-162</td>
</tr>
<tr>
<td>VDIV</td>
<td>Vector Divide</td>
<td>A-163</td>
</tr>
<tr>
<td>VEUCM2</td>
<td>Vector Euclidean Distance</td>
<td>A-164</td>
</tr>
<tr>
<td>VEXP</td>
<td>Vector Exponential</td>
<td>A-165</td>
</tr>
<tr>
<td>VEXP10</td>
<td>Vector Exponential (Base 10)</td>
<td>A-166</td>
</tr>
<tr>
<td>VFILL</td>
<td>Vector Fill</td>
<td>A-167</td>
</tr>
<tr>
<td>VFRAc</td>
<td>Vector Truncate to Fraction</td>
<td>A-168</td>
</tr>
<tr>
<td>VIABS</td>
<td>Vector Absolute Value</td>
<td>A-169</td>
</tr>
<tr>
<td>VIADD</td>
<td>Vector Integer Add</td>
<td>A-170</td>
</tr>
<tr>
<td>VICCLIP</td>
<td>Vector Inverted Clip</td>
<td>A-171</td>
</tr>
<tr>
<td>VIDIV</td>
<td>Vector Integer Divide</td>
<td>A-172</td>
</tr>
<tr>
<td>VIMAG</td>
<td>Extract Imaginaries of Complex Vector</td>
<td>A-173</td>
</tr>
<tr>
<td>VINUL</td>
<td>Vector Integer Multiply</td>
<td>A-174</td>
</tr>
<tr>
<td>VINDEX</td>
<td>Vector Index</td>
<td>A-175</td>
</tr>
<tr>
<td>VINEG</td>
<td>Vector Integer Negate</td>
<td>A-176</td>
</tr>
<tr>
<td>VISUB</td>
<td>Vector Integer Subtract</td>
<td>A-177</td>
</tr>
<tr>
<td>VLAND</td>
<td>Vector Logical Add</td>
<td>A-178</td>
</tr>
<tr>
<td>VLEEQV</td>
<td>Vector Logical Equivalence</td>
<td>A-179</td>
</tr>
<tr>
<td>VLM</td>
<td>Vector Limit</td>
<td>A-180</td>
</tr>
<tr>
<td>VLMERG</td>
<td>Logical Vector Merge</td>
<td>A-181</td>
</tr>
</tbody>
</table>
### BASIC MATH LIBRARY (cont.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTSADD</td>
<td>Vector TM Scalar Add</td>
<td>A - 230</td>
</tr>
<tr>
<td>VTSMA</td>
<td>Vector TM Scalar Multiply and Add</td>
<td>A - 231</td>
</tr>
<tr>
<td>VTSMUL</td>
<td>Vector TM Scalar Multiply</td>
<td>A - 232</td>
</tr>
</tbody>
</table>

### ADVANCED MATH LIBRARY (VOLUME 2)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH</td>
<td>Complex Hermitian Eigensystem Solver</td>
<td>A - 234</td>
</tr>
<tr>
<td>EIGRS</td>
<td>Real Symmetric Eigensystem Solver</td>
<td>A - 237</td>
</tr>
<tr>
<td>HTRIBK</td>
<td>Complex Hermitian Eigenvectors</td>
<td>A - 239</td>
</tr>
<tr>
<td>HTRIDI</td>
<td>Complex Hermitian Tridiagonalization</td>
<td>A - 241</td>
</tr>
<tr>
<td>IMTQL1</td>
<td>Diagonalize Tridiagonal Matrix</td>
<td>A - 243</td>
</tr>
<tr>
<td>IMTQL2</td>
<td>Diagonalize a Tridiagonal Matrix</td>
<td>A - 245</td>
</tr>
<tr>
<td>RS</td>
<td>Real Symmetric Eigensystem Solver</td>
<td>A - 247</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>Revised Simplex</td>
<td>A - 249</td>
</tr>
<tr>
<td>SKYSOL</td>
<td>Skyline Format Equation Solver</td>
<td>A - 254</td>
</tr>
<tr>
<td>TRED1</td>
<td>Tridiagonalize Symmetric Matrix</td>
<td>A - 256</td>
</tr>
<tr>
<td>TRED2</td>
<td>Tridiagonalize a Symmetric Matrix</td>
<td>A - 258</td>
</tr>
<tr>
<td>VASORT</td>
<td>Vector Sort Algebraic Values</td>
<td>A - 259</td>
</tr>
<tr>
<td>VISORT</td>
<td>Vector Sort Integer Values</td>
<td>A - 260</td>
</tr>
<tr>
<td>VSORT</td>
<td>Vector Sort with Indices</td>
<td>A - 261</td>
</tr>
</tbody>
</table>

### SIGNAL PROCESSING LIBRARY

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACORF</td>
<td>Auto-Correlation (Frequency-Domain)</td>
<td>A - 264</td>
</tr>
<tr>
<td>ACORT</td>
<td>Auto-Correlation (Time-Domain)</td>
<td>A - 265</td>
</tr>
<tr>
<td>ASPESC</td>
<td>Accumulating Auto-Spectrum</td>
<td>A - 266</td>
</tr>
<tr>
<td>BLKMAN</td>
<td>Blackman Window Multiply</td>
<td>A - 267</td>
</tr>
<tr>
<td>CCEPS</td>
<td>Phase Unwrap and Complex Cepstrum</td>
<td>A - 268</td>
</tr>
<tr>
<td>CCOEF</td>
<td>Cross-Correlation (Frequency-Domain)</td>
<td>A - 272</td>
</tr>
<tr>
<td>CCORT</td>
<td>Cross-Correlation (Time-Domain)</td>
<td>A - 273</td>
</tr>
<tr>
<td>CFFTI</td>
<td>Complex FFT with Interpolation</td>
<td>A - 274</td>
</tr>
<tr>
<td>COHER</td>
<td>Coherence Function</td>
<td>A - 275</td>
</tr>
<tr>
<td>CSPESC</td>
<td>Accumulating Cross-Spectrum</td>
<td>A - 276</td>
</tr>
<tr>
<td>DECFLR</td>
<td>Decimation</td>
<td>A - 277</td>
</tr>
<tr>
<td>ENVEL</td>
<td>Envelope Detector</td>
<td>A - 279</td>
</tr>
<tr>
<td>HAMM</td>
<td>Hamming Window Multiply</td>
<td>A - 280</td>
</tr>
<tr>
<td>HANN</td>
<td>Hanning Window Multiply</td>
<td>A - 281</td>
</tr>
<tr>
<td>HIST</td>
<td>Histogram</td>
<td>A - 282</td>
</tr>
<tr>
<td>HLBRT</td>
<td>Hilbert Transformer</td>
<td>A - 283</td>
</tr>
<tr>
<td>LPAUTO</td>
<td>Linear Prediction Autocorrelation</td>
<td>A - 284</td>
</tr>
<tr>
<td>PVAL</td>
<td>Peak and Valley Picking</td>
<td>A - 286</td>
</tr>
<tr>
<td>RDFT</td>
<td>Real Discrete Fourier Transform</td>
<td>A - 288</td>
</tr>
<tr>
<td>RFFTI</td>
<td>Real FFT with Interpolation</td>
<td>A - 290</td>
</tr>
</tbody>
</table>
**LINPACK BLAS LIBRARY (cont.)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SROT</td>
<td>PLANE ROTATION</td>
<td>A - 355</td>
</tr>
<tr>
<td>SROTG</td>
<td>GIVENS PLANE ROTATION</td>
<td>A - 356</td>
</tr>
<tr>
<td>SROTM</td>
<td>MODIFIED GIVENS ROTATIONS</td>
<td>A - 357</td>
</tr>
<tr>
<td>SROTMG</td>
<td>MODIFIED GIVENS PLANE ROTATION SETUP</td>
<td>A - 358</td>
</tr>
<tr>
<td>SSCAL</td>
<td>REAL SCALAR TIMES VECTOR</td>
<td>A - 360</td>
</tr>
<tr>
<td>SSWAP</td>
<td>INTERCHANGES VECTORS</td>
<td>A - 361</td>
</tr>
</tbody>
</table>

**SIMULATION LIBRARY**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABP1</td>
<td>ADAMS-BASHFORTH PREDICTOR (ORDER 1)</td>
<td>A - 363</td>
</tr>
<tr>
<td>ABP2</td>
<td>ADAMS-BASHFORTH PREDICTOR (ORDER 2)</td>
<td>A - 364</td>
</tr>
<tr>
<td>ABP3</td>
<td>ADAMS-BASHFORTH PREDICTOR (ORDER 3)</td>
<td>A - 365</td>
</tr>
<tr>
<td>ABP4</td>
<td>ADAMS-BASHFORTH PREDICTOR (ORDER 4)</td>
<td>A - 366</td>
</tr>
<tr>
<td>ADAMS4</td>
<td>ADAMS VARIABLE STEP INTEG. (ORD 4)</td>
<td>A - 367</td>
</tr>
<tr>
<td>AMC1</td>
<td>ADAMS-MOULTON CORRECTOR (ORDER 1)</td>
<td>A - 371</td>
</tr>
<tr>
<td>AMC2</td>
<td>ADAMS-MOULTON CORRECTOR (ORDER 2)</td>
<td>A - 372</td>
</tr>
<tr>
<td>AMC3</td>
<td>ADAMS-MOULTON CORRECTOR (ORDER 3)</td>
<td>A - 373</td>
</tr>
<tr>
<td>AMC4</td>
<td>ADAMS-MOULTON CORRECTOR (ORDER 4)</td>
<td>A - 374</td>
</tr>
<tr>
<td>BIN</td>
<td>BINARY SEARCH</td>
<td>A - 375</td>
</tr>
<tr>
<td>CBEIK</td>
<td>COMPLEX BESSEL I AND K</td>
<td>A - 376</td>
</tr>
<tr>
<td>CBEJYH</td>
<td>COMPLEX BESSEL J, Y, AND H</td>
<td>A - 377</td>
</tr>
<tr>
<td>FUN1</td>
<td>FUNCTION OF ONE VARIABLE</td>
<td>A - 378</td>
</tr>
<tr>
<td>FUN2</td>
<td>FUNCTION OF TWO VARIABLES</td>
<td>A - 379</td>
</tr>
<tr>
<td>FUN3</td>
<td>FUNCTION OF THREE VARIABLES</td>
<td>A - 380</td>
</tr>
<tr>
<td>FUN4</td>
<td>FUNCTION OF FOUR VARIABLES</td>
<td>A - 381</td>
</tr>
<tr>
<td>RKGIL</td>
<td>RUNGE-KUTTA-GILL INTEGRATION</td>
<td>A - 382</td>
</tr>
<tr>
<td>RKGTF</td>
<td>R-K-GILL-THOMPSON INTEG. (ORDER 4)</td>
<td>A - 383</td>
</tr>
<tr>
<td>ROT3</td>
<td>3D ROTATION MATRIX, 3-ANGLE</td>
<td>A - 384</td>
</tr>
<tr>
<td>SCS1</td>
<td>SCALAR COS/SIN, TM INTERP. (ORD 1)</td>
<td>A - 385</td>
</tr>
<tr>
<td>STEP</td>
<td>STEP SEARCH</td>
<td>A - 386</td>
</tr>
</tbody>
</table>

**GEOPHYSICAL LIBRARY**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNMO</td>
<td>NMO WITH CONSTANT VELOCITY</td>
<td>A - 403</td>
</tr>
<tr>
<td>IIR3J</td>
<td>RECURSIVE FILTER</td>
<td>A - 404</td>
</tr>
<tr>
<td>KSMLV</td>
<td>K-TH SMALLEST ELEMENT IN VECTOR</td>
<td>A - 405</td>
</tr>
<tr>
<td>NMOLI</td>
<td>NMO LINEAR INTERPOLATION</td>
<td>A - 406</td>
</tr>
<tr>
<td>NMQQI</td>
<td>NMO QUADRATIC INTERPOLATION</td>
<td>A - 407</td>
</tr>
<tr>
<td>POST54</td>
<td>POST BITS TO RASTER</td>
<td>A - 408</td>
</tr>
<tr>
<td>RESNMO</td>
<td>RESIDUAL NORMAL MOVEOUT</td>
<td>A - 409</td>
</tr>
<tr>
<td>TMCONV</td>
<td>CONVOLUTION (CORRELATION)</td>
<td>A - 410</td>
</tr>
<tr>
<td>V01</td>
<td>VECTOR ZERO TRENDS</td>
<td>A - 411</td>
</tr>
<tr>
<td>VARNMO</td>
<td>NMO WITH VARIABLE VELOCITY</td>
<td>A - 412</td>
</tr>
<tr>
<td>VRNAVG</td>
<td>VECTOR RUNNING AVERAGE</td>
<td>A - 413</td>
</tr>
<tr>
<td>VSCAN0</td>
<td>VECTOR SCAN FOR ZEROS</td>
<td>A - 414</td>
</tr>
</tbody>
</table>
### TABLE MEMORY LIBRARY (cont.)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTTSUB</td>
<td>VECTOR SUBTRACT (TM-TM TO TM)</td>
<td>A - 500</td>
</tr>
<tr>
<td>TTVLC2</td>
<td>VECTOR LINEAR COMBINATION</td>
<td>A - 501</td>
</tr>
<tr>
<td>TVCLR</td>
<td>TABLE MEMORY VECTOR CLEAR</td>
<td>A - 502</td>
</tr>
</tbody>
</table>

### SPECIAL UTILITIES LIBRARY

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTRS</td>
<td>EXTRACT A SIGNED BIT-FIELD</td>
<td>A - 504</td>
</tr>
<tr>
<td>PEEK</td>
<td>MEMORY FETCH</td>
<td>A - 505</td>
</tr>
<tr>
<td>POKE</td>
<td>STORE INTO MEMORY</td>
<td>A - 506</td>
</tr>
</tbody>
</table>

### DATA FORMATTING LIBRARY

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFLOAT</td>
<td>CONVERT INTEGER TO FLOATING-POINT</td>
<td>A - 508</td>
</tr>
<tr>
<td>VFIX</td>
<td>VECTOR INTEGER FIX</td>
<td>A - 509</td>
</tr>
<tr>
<td>VPK8</td>
<td>VECTOR 8-BIT BYTE PACK</td>
<td>A - 510</td>
</tr>
<tr>
<td>VPK16</td>
<td>VECTOR 16-BIT BYTE PACK</td>
<td>A - 511</td>
</tr>
<tr>
<td>VPK32</td>
<td>VECTOR 32-BIT BYTE PACK</td>
<td>A - 512</td>
</tr>
<tr>
<td>VPKI32</td>
<td>VECTOR 32-BIT INTEGER PACK</td>
<td>A - 513</td>
</tr>
<tr>
<td>VPKR32</td>
<td>VECTOR REAL HALFWORD PACK</td>
<td>A - 514</td>
</tr>
<tr>
<td>VSCALE</td>
<td>VECTOR SCALE AND FIX</td>
<td>A - 515</td>
</tr>
<tr>
<td>VSCSCL</td>
<td>VECTOR SCAN SCALE AND FIX</td>
<td>A - 516</td>
</tr>
<tr>
<td>VSHFX</td>
<td>VECTOR SHIFT AND FIX</td>
<td>A - 517</td>
</tr>
<tr>
<td>VSMAFX</td>
<td>VECTOR SCALAR MULTIPLY, ADD AND FIX</td>
<td>A - 518</td>
</tr>
<tr>
<td>VUP8</td>
<td>VECTOR 8-BIT BYTE UNPACK</td>
<td>A - 519</td>
</tr>
<tr>
<td>VUP16</td>
<td>VECTOR 16-BIT BYTE UNPACK</td>
<td>A - 520</td>
</tr>
<tr>
<td>VUP32</td>
<td>VECTOR 32-BIT BYTE UNPACK</td>
<td>A - 521</td>
</tr>
<tr>
<td>VUPR32</td>
<td>VECTOR HALFWORD REAL UNPACK</td>
<td>A - 522</td>
</tr>
<tr>
<td>VUPS8</td>
<td>VECTOR 8-BIT SIGNED BYTE UNPACK</td>
<td>A - 523</td>
</tr>
<tr>
<td>VUPS16</td>
<td>VECTOR 16-BIT SIGNED BYTE UNPACK</td>
<td>A - 524</td>
</tr>
<tr>
<td>VUPS32</td>
<td>VECTOR 32-BIT SIGNED BYTE UNPACK</td>
<td>A - 525</td>
</tr>
<tr>
<td>VUSI32</td>
<td>VECTOR 32-BIT SIGNED INTEGER UNPACK</td>
<td>A - 526</td>
</tr>
<tr>
<td>VUUI32</td>
<td>VECTOR 32-BIT UNSIGNED UNPACK</td>
<td>A - 527</td>
</tr>
</tbody>
</table>

### DOUBLE PRECISION LIBRARY

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DADD</td>
<td>DOUBLE TO DOUBLE-PRECISION ADD</td>
<td>A - 529</td>
</tr>
<tr>
<td>DADDRR</td>
<td>SINGLE TO DOUBLE-PRECISION ADD</td>
<td>A - 530</td>
</tr>
<tr>
<td>DADOT</td>
<td>DOUBLE ACCUMULATE DOT PRODUCT</td>
<td>A - 531</td>
</tr>
<tr>
<td>DDOTRR</td>
<td>DOUBLE DOT PRODUCT REAL REAL</td>
<td>A - 532</td>
</tr>
<tr>
<td>DMUL</td>
<td>DOUBLE TO DOUBLE-PRECISION MULTIPLY</td>
<td>A - 533</td>
</tr>
<tr>
<td>DMULRR</td>
<td>SINGLE TO DOUBLE PRECISION MULTIPLY</td>
<td>A - 534</td>
</tr>
<tr>
<td>DNEG</td>
<td>NEGATE DOUBLE-PRECISION NUMBER</td>
<td>A - 535</td>
</tr>
<tr>
<td>DSUB</td>
<td>DOUBLE TO DOUBLE-PRECISION SUBTRACT</td>
<td>A - 536</td>
</tr>
<tr>
<td>DSUBRR</td>
<td>SINGLE TO DOUBLE-PRECISION SUBTRACT</td>
<td>A - 537</td>
</tr>
</tbody>
</table>
APMATH64 ROUTINES IN ALPHABETICAL ORDER

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABP1</td>
<td>ADAMS-BASHFORTH PREDICTOR (ORDER 1)</td>
<td>A-363</td>
</tr>
<tr>
<td>ABP2</td>
<td>ADAMS-BASHFORTH PREDICTOR (ORDER 2)</td>
<td>A-364</td>
</tr>
<tr>
<td>ABP3</td>
<td>ADAMS-BASHFORTH PREDICTOR (ORDER 3)</td>
<td>A-365</td>
</tr>
<tr>
<td>ABP4</td>
<td>ADAMS-BASHFORTH PREDICTOR (ORDER 4)</td>
<td>A-366</td>
</tr>
<tr>
<td>ABS</td>
<td>REAL NUMBER ABSOLUTE VALUE</td>
<td>A-539</td>
</tr>
<tr>
<td>ACORF</td>
<td>AUTO-CORRELATION (FREQUENCY-DOMAIN)</td>
<td>A-264</td>
</tr>
<tr>
<td>ACORT</td>
<td>AUTO-CORRELATION (TIME-DOMAIN)</td>
<td>A-265</td>
</tr>
<tr>
<td>ACOS</td>
<td>REAL NUMBER ARCCOSINE</td>
<td>A-540</td>
</tr>
<tr>
<td>ADAMS4</td>
<td>ADAMS VARIABLE STEP INTEG.(ORD 4)</td>
<td>A-368</td>
</tr>
<tr>
<td>AINT</td>
<td>TRUNCATE REAL NUMBER</td>
<td>A-541</td>
</tr>
<tr>
<td>ALOG10</td>
<td>REAL NUMBER LOGARITH</td>
<td>A-543</td>
</tr>
<tr>
<td>ALOG</td>
<td>REAL NUMBER LOGARITH</td>
<td>A-542</td>
</tr>
<tr>
<td>AMC1</td>
<td>ADAMS-MOULTON CORRECTOR (ORDER 1)</td>
<td>A-371</td>
</tr>
<tr>
<td>AMC2</td>
<td>ADAMS-MOULTON CORRECTOR (ORDER 2)</td>
<td>A-372</td>
</tr>
<tr>
<td>AMC3</td>
<td>ADAMS-MOULTON CORRECTOR (ORDER 3)</td>
<td>A-373</td>
</tr>
<tr>
<td>AMC4</td>
<td>ADAMS-MOULTON CORRECTOR (ORDER 4)</td>
<td>A-374</td>
</tr>
<tr>
<td>AMOD</td>
<td>REAL/REAL DIVIDE REMAINDER</td>
<td>A-544</td>
</tr>
<tr>
<td>ANINT</td>
<td>ROUND REAL NUMBER TO NEAREST WHOLE</td>
<td>A-545</td>
</tr>
<tr>
<td>ASIN</td>
<td>REAL NUMBER ARCSINE</td>
<td>A-546</td>
</tr>
<tr>
<td>ASPEC</td>
<td>ACCUMULATING AUTO-SPECTRUM</td>
<td>A-266</td>
</tr>
<tr>
<td>ATAN2</td>
<td>ARCTANGENT OF RATIO OF REAL NUMBERS</td>
<td>A-548</td>
</tr>
<tr>
<td>ATAN</td>
<td>ARCTANGENT OF REAL NUMBER</td>
<td>A-547</td>
</tr>
<tr>
<td>BIN</td>
<td>BINARY SEARCH</td>
<td>A-376</td>
</tr>
<tr>
<td>BLKMAN</td>
<td>BLACKMAN WINDOW MULTIPLY</td>
<td>A-267</td>
</tr>
<tr>
<td>CABS</td>
<td>COMPLEX NUMBER ABSOLUTE VALUE</td>
<td>A-549</td>
</tr>
<tr>
<td>CAXPY</td>
<td>COMPLEX A * X + Y</td>
<td>A-326</td>
</tr>
<tr>
<td>CAXPYN</td>
<td>NESTED COMPLEX A * X + Y</td>
<td>A-327</td>
</tr>
<tr>
<td>CBEIK</td>
<td>COMPLEX BESSEL I AND K</td>
<td>A-378</td>
</tr>
<tr>
<td>CBEJYH</td>
<td>COMPLEX BESSEL J, Y, AND H</td>
<td>A-381</td>
</tr>
<tr>
<td>CCEPS</td>
<td>PHASE UNWRAP AND COMPLEX CEPSTRUM</td>
<td>A-268</td>
</tr>
<tr>
<td>CCMMUL</td>
<td>COMPLEX MATRIX MULTIPLY</td>
<td>A-2</td>
</tr>
<tr>
<td>CCOPY</td>
<td>COMPLEX VECTOR COPY</td>
<td>A-330</td>
</tr>
<tr>
<td>CORCF</td>
<td>CROSS-CORRELATION (FREQUENCY-DOMAIN)</td>
<td>A-272</td>
</tr>
<tr>
<td>CORCT</td>
<td>CROSS-CORRELATION (TIME-DOMAIN)</td>
<td>A-273</td>
</tr>
<tr>
<td>CCOS</td>
<td>COMPLEX NUMBER COSINE</td>
<td>A-550</td>
</tr>
<tr>
<td>CDET</td>
<td>COMPLEX MATRIX DETERMINANT</td>
<td>A-4</td>
</tr>
<tr>
<td>CDIV</td>
<td>COMPLEX/COMPLEX DIVIDE</td>
<td>A-551</td>
</tr>
<tr>
<td>CDIVCR</td>
<td>REAL/COMPLEX DIVIDE</td>
<td>A-552</td>
</tr>
<tr>
<td>CDIVRC</td>
<td>COMPLEX/REAL DIVIDE</td>
<td>A-553</td>
</tr>
<tr>
<td>CDTGC</td>
<td>COMPLEX INNER PRODUCT</td>
<td>A-331</td>
</tr>
<tr>
<td>CDOTN</td>
<td>NESTED COMPLEX DOT PRODUCT</td>
<td>A-332</td>
</tr>
<tr>
<td>CDOTPR</td>
<td>COMPLEX DOT PRODUCT</td>
<td>A-6</td>
</tr>
<tr>
<td>CDOTU</td>
<td>COMPLEX DOT PRODUCT</td>
<td>A-335</td>
</tr>
</tbody>
</table>
CUFS2  SPARSE COMPLEX UNSYM FACTOR & SOLVE  A - 434
CUSV2  SPARSE COMPLEX UNSYMMETRIC SOLVE  A - 437
CVABS  COMPLEX VECTOR ABSOLUTE VALUE  A - 42
CVADD  COMPLEX VECTOR ADD  A - 43
CVCOMB  COMPLEX VECTOR COMBINE  A - 44
CVCONJ  COMPLEX VECTOR CONJUGATE  A - 45
CVEXP  COMPLEX VECTOR EXPONENTIAL  A - 46
CVFILL  COMPLEX VECTOR FILL  A - 47
CVMA  COMPLEX VECTOR MULTIPLY AND ADD  A - 48
CVMAGS  COMPLEX VECTOR MAGNITUDE SQUARED  A - 50
CVMEXP  COMPLEX VECTOR MULTIPLY EXPONENTIAL  A - 51
CVMOV  COMPLEX VECTOR MOVE  A - 52
CVMUL  COMPLEX VECTOR MULTIPLY  A - 53
CVNEG  COMPLEX VECTOR NEGATE  A - 54
CVRCIP  COMPLEX VECTOR RECIPROCAL  A - 55
CVREAL  FORM COMPLEX VECTOR OF REALS  A - 56
CVMXSA  COMPLEX VECTOR SCALAR MULTIPLY AND ADD  A - 57
CVMXSL  COMPLEX VECTOR SCALAR MULTIPLY  A - 58
CVSUB  COMPLEX VECTOR SUBTRACT  A - 59
DADD  DOUBLE TO DOUBLE-PRECISION ADD  A - 529
DADDR  SINGLE TO DOUBLE-PRECISION ADD  A - 530
DADOT  DOUBLE ACCUMULATE DOT PRODUCT  A - 531
DDOTRR  DOUBLE DOT PRODUCT REAL REAL  A - 532
DECIR  DECIMATION  A - 277
DEQ22  DIFFERENCE EQUATION, 2 POLES, 2 ZEROS  A - 68
DIM  REAL/REAL POSITIVE DIFFERENCE  A - 566
DMUL  DOUBLE TO DOUBLE-PRECISION MULTIPLY  A - 533
DMULRR  SINGLE TO DOUBLE PRECISION MULTIPLY  A - 534
DNEG  NEGATE DOUBLE-PRECISION NUMBER  A - 535
DOTPR  DOT PRODUCT  A - 62
DSUB  DOUBLE TO DOUBLE-PRECISION SUBTRACT  A - 536
DSUBRR  SINGLE TO DOUBLE-PRECISION SUBTRACT  A - 537
EIGRS  REAL SYMMETRIC EIGENSYSTEM SOLVER  A - 237
ENTEL  ENVELOPE DETECTOR  A - 279
EXP  EXPONENTIAL OF REAL NUMBER  A - 567
EXEES  EXTRACT A SIGNED BIT-FIELD  A - 594
FMNM  FAST MATRIX MULTIPLY  A - 63
FMNNMV  FAST MATRIX MULTIPLY  A - 64
FUN1  FUNCTION OF ONE VARIABLE  A - 384
FUN2  FUNCTION OF TWO VARIABLES  A - 386
FUN3  FUNCTION OF THREE VARIABLES  A - 388
FUN4  FUNCTION OF FOUR VARIABLES  A - 390
GENTAB  GENERATE TWIDDLE FACTOR TABLE  A - 65
GRAD2D  MAXIMUM GRADIENT FILTER  A - 310
GRD2DB  MAXIMUM GRADIENT FILTER WITH BOUND  A - 312
HAMM  HAMMING WINDOW MULTIPLY  A - 280
HANN  HANNING WINDOW MULTIPLY  A - 281
HIST  HISTOGRAM  A - 282
HLBRT  HILBERT TRANSFORMER  A - 283
HTBIBK  COMPLEX HERMITIAN EIGENVECTORS  A - 239
HTRIDI  COMPLEX HERMITIAN TRIDIAGONALIZATION  A - 241
IABS  INTEGER ABSOLUTE VALUE  A - 568
ICAMAX  INDEX OF LARGEST COMPLEX ELEMENT  A - 341
IDIM  INTEGER/INTEGER POSITIVE DIFFERENCE  A - 569
PAS3F   RADIX-3 FORWARD COMPLEX FFT PASS  A - 99
PAS3I   RADIX-3 INVERSE COMPLEX FFT PASS  A - 101
PAS4F   RADIX-4 FORWARD COMPLEX FFT PASS  A - 103
PAS4I   RADIX-4 INVERSE COMPLEX FFT PASS  A - 105
PAS5F   RADIX-5 FORWARD COMPLEX FFT PASS  A - 107
PAS5I   RADIX-5 INVERSE COMPLEX FFT PASS  A - 109
PEEK    MEMORY FETCH     A - 505
PFINV   MATRIX INVERSE (PRODUCT FORM)   A - 111
PKVAL   PEAK AND VALLEY PICKING         A - 286
POKE    STORE INTO MEMORY               A - 586
POLAR   RECTANGULAR TO POLAR CONVERSION A - 112
POST64  POST BITS TO RASTER              A - 412
RAN     SCALAR RANDOM NUMBER GENERATOR   A - 575
RCMMUL  REAL-COMPLEX MATRIX MULTIPLY    A - 113
RDFT    REAL DISCRETE FOURIER TRANSFORM  A - 288
RDIV    REAL/REAL DIVIDE                 A - 576
RECT    POLAR TO RECTANGULAR CONVERSION  A - 115
RESNMO  RESIDUAL NORMAL MOVEOUT          A - 413
RFFT2D  REAL TO COMPLEX 2-DIMENSIONAL FFT A - 323
RFFT    REAL-TO-COMPLEX FFT (IN PLACE)   A - 116
RFFT2B  REAL-TO-COMPLEX FFT (NOT IN PLACE)A - 117
RFFT1   REAL FFT WITH INTERPOLATION      A - 290
RFFT2M  MIXED-RADIX REAL FFT (NOT-IN-PLACE)A - 119
RFFTS2  REAL FFT SCALE AND FORMAT       A - 121
RFTI1   REAL FFT WITH QUARTER INTERPOLATION A - 291
RGMML   REAL GENERAL MATRIX MULTIPLY     A - 123
RGKIL   RUNGE-KUTTA-GILL INTEGRATION     A - 392
RGKTF   R-K-GILL-THOMPSON INTEG. (ORDER 4) A - 395
RMSQV   ROOT-MEAN-SQUARE OF VECTOR ELEMENTS A - 125
ROT3    3D ROTATION MATRIX, 3-ANGLE       A - 397
RPOW    REAL TO REAL POWER               A - 577
RPOWR1  REAL TO INTEGER POWER            A - 578
RRCP    REAL RECIPROCAL                  A - 579
RS      REAL SYMMETRIC EIGENSYSTEM SOLVER A - 247
RSFR2   SPARSE REAL SYMMETRIC FACTOR     A - 439
RSFS2   SPARSE REAL SYMM FACTOR & SOLVE  A - 442
RSQRT   RECIPROCAL SQUARE ROOT           A - 580
RSSV2   SPARSE REAL SYMMETRIC SOLVE      A - 446
RUFR2   SPARSE REAL UNSYMMETRIC FACTOR   A - 449
RUFS2   SPARSE REAL UNSYM FACTOR & SOLVE A - 452
RUSV2   SPARSE REAL UNSYMMETRIC SOLVE    A - 456
SASUM   SUM OF MAGNITUDES                A - 343
SAXPY   REAL A * X + Y                   A - 344
SAXPYN  NESTED REAL A * X + Y            A - 345
SCASUM  SUM OF REAL AND IMAGINARY MAGNITUDES A - 348
SCJMA   SELF-CONJUGATE MULTIPLY AND ADD  A - 126
SCNRM2  COMPLEX EUCLIDEAN NORM           A - 349
SCOPY   VECTOR COPY                      A - 350
SCSI    SCALAR COS/SIN, TM INTERP.(ORD 1)A - 399
SDOT    DOT PRODUCT OF REAL VECTORS      A - 351
SDOTN   NESTED REAL DOT PRODUCT          A - 352
SDOTPR  SPARSE DOT PRODUCT               A - 459
SGEFA   REAL GENERAL MATRIX FACTOR       A - 127
SGESL   REAL GENERAL MATRIX SOLVE        A - 129
VECTOR SUBTRACT (TM-TM TO TM)
VECTOR LINEAR COMBINATION
TABLE MEMORY VECTOR CLEAR
VECTOR ZERO TRENDS
VECTOR ADD, ADD, AND MULTIPLY
VECTOR ABSOLUTE VALUE
VECTOR ARCCOSINE
VECTOR ADD
VECTOR TRUNCATE
VECTOR BASE 10 LOGARITHM
VECTOR LOGARITHM
VECTOR ADD AND MULTIPLY
NMO WITH VARIABLE VELOCITY
VECTOR ARCSINE
VECTOR ADD AND SCALAR MULTIPLY
VECTOR SORT ALGEBRAIC VALUES
VECTOR ARCTANGENT (2 ARGUMENTS)
VECTOR ARCTANGENT
VECTOR EXPONENTIAL AVERAGING
VECTOR LINEAR AVERAGING
VECTOR CLIP
VECTOR CLEAR
VECTOR COSINE
VECTOR COSINE (HYPERBOLIC)
VECTOR CONVERSION TO DB (POWER)
VECTOR DIVIDE
VECTOR EUCLIDEAN DISTANCE
VECTOR EXPONENTIAL (BASE 10)
VECTOR EXPONENTIAL
VECTOR FILL
VECTOR TRUNCATE TO FRACTION
VECTOR ABSOLUTE VALUE
VECTOR INTEGER ADD
VECTOR INVERTED CLIP
VECTOR INTEGER DIVIDE
VECTOR INTEGER FIX
EXTRACT IMAGINARIES OF COMPLEX VECTOR
VECTOR INTEGER MULTIPLY
VECTOR INDEX
VECTOR INTEGER NEGATE
VECTOR SORT INTEGER VALUES
VECTOR INTEGER SUBTRACT
VECTOR LOGICAL ADD
VECTOR LOGICAL EQUIVALENCE
VECTOR LIMIT
LOGICAL VECTOR MERGE
VECTOR LOGICAL NOT
VECTOR LOGICAL OR
VECTOR LOGICAL EXCLUSIVE OR
VECTOR MULTIPLY AND ADD
VECTOR MAXIMUM
VECTOR MAXIMUM MAGNITUDE
VECTOR MINIMUM
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTSMA</td>
<td>VECTOR TM SCALAR MULTIPLY AND ADD</td>
<td>231</td>
</tr>
<tr>
<td>VTSMUL</td>
<td>VECTOR TM SCALAR MULTIPLY</td>
<td>232</td>
</tr>
<tr>
<td>VUP16</td>
<td>VECTOR 16-BIT BYTE UNPACK</td>
<td>520</td>
</tr>
<tr>
<td>VUP32</td>
<td>VECTOR 32-BIT BYTE UNPACK</td>
<td>521</td>
</tr>
<tr>
<td>VUP8</td>
<td>VECTOR 8-BIT BYTE UNPACK</td>
<td>519</td>
</tr>
<tr>
<td>VUPR32</td>
<td>VECTOR HALFWORD REAL UNPACK</td>
<td>522</td>
</tr>
<tr>
<td>VUPSi6</td>
<td>VECTOR 16-BIT SIGNED BYTE UNPACK</td>
<td>524</td>
</tr>
<tr>
<td>VUPSi2</td>
<td>VECTOR 32-BIT SIGNED BYTE UNPACK</td>
<td>525</td>
</tr>
<tr>
<td>VUPSi8</td>
<td>VECTOR 8-BIT SIGNED BYTE UNPACK</td>
<td>523</td>
</tr>
<tr>
<td>VUSI32</td>
<td>VECTOR 32-BIT SIGNED INTEGER UNPACK</td>
<td>526</td>
</tr>
<tr>
<td>VUUi32</td>
<td>VECTOR 32-BIT UNSIGNED UNPACK</td>
<td>527</td>
</tr>
<tr>
<td>VXCS</td>
<td>VECTOR MULTIPLIED BY SIN AND COS</td>
<td>299</td>
</tr>
<tr>
<td>WIENER</td>
<td>WIENER LEVINSON ALGORITHM</td>
<td>301</td>
</tr>
</tbody>
</table>
APMATH64 KEY WORD INDEX

This index of APMATH64 routines is sorted by key words that appear in each routine title. Each title can contain more than one key word. The key words are listed alphabetically to the right of the gap running down the center of each page.

To use the key word index, locate a key word that is representative of the desired APMATH64 function. Applicable APMATH64 routine names and titles can be found on the same line with each occurrence of the key word. The routine name appears in brackets ([ ]). The routine title immediately follows the routine name and continues on the other side of the gap when necessary. The ellipsis (•••) is placed directly after the last word in the title if the line wraps around. The page where a particular routine is documented can be found in Appendix J.

[VSIMPS]VECTOR SIMPSON'S 1/3 RULE INTEGRATION
EXPOENENTIAL BASE 16...[VEXPL0]VECTOR
[VALG16]VECTOR BASE 16 LOGARITHM
[VPK16]VECTOR 16-BIT BYTE PACK
[VUP16]VECTOR 16-BIT BYTE UNPACK
[VUP816]VECTOR 16-BIT SIGNED BYTE UNPACK
[CSROT]COMPLEX 2-D ROTATION
[CFPT2D]COMPLEX TO COMPLEX 2-DIMENSIONAL FFT
[RFP2D]REAL TO COMPLEX 2-DIMENSIONAL FFT
[ROTI3]3D ROTATION MATRIX, 3-ANGLE
[VPK32]VECTOR 32-BIT BYTE PACK
[VUP32]VECTOR 32-BIT BYTE UNPACK
[VPK32]VECTOR 32-BIT INTEGER PACK
[VUP32]VECTOR 32-BIT SIGNED BYTE UNPACK
[VUSI32]VECTOR 32-BIT SIGNED INTEGER UNPACK
[VUUI32]VECTOR 32-BIT UNSIGNED UNPACK
[CROSSP]COMPLEX 3D CROSS PRODUCT
3X3 MATRIX MULT. 3D VECTORS...[CMVML3]COMPLEX
[MVML3]MATRIX VECTOR MULTIPLY 3X3
[CMVML3]COMPLEX 3X3 MATRIX MULT. 3D VECTORS
4X4 MATRIX MULT. 4D VECTORS...[CMVML4]COMPLEX
[MVML4]MATRIX VECTOR MULTIPLY 4X4
[CMVML4]COMPLEX 4X4 MATRIX MULT. 4D VECTORS
[VPK8]VECTOR 8-BIT BYTE PACK
[VUP8]VECTOR 8-BIT BYTE UNPACK
[VUPS8]VECTOR 8-BIT SIGNED BYTE UNPACK
[ABS]REAL NUMBER ABSOLUTE VALUE
[CABS]COMPLEX NUMBER ABSOLUTE VALUE
[CVABS]COMPLEX VECTOR ABSOLUTE VALUE
[IAS]INTEGER ABSOLUTE VALUE
[ISAMAX]INDEX OF MAXIMUM ABSOLUTE VALUE
[VABS]VECTOR ABSOLUTE VALUE
[VIABS]VECTOR ABSOLUTE VALUE
[DADOT]DOUBLE ACCUMULATE DOT PRODUCT
[VUP16] VECTOR 16-BIT BYTE UNPACK
[VUP32] VECTOR 32-BIT BYTE UNPACK
[VUP8] VECTOR 8-BIT BYTE UNPACK
[VUPS16] VECTOR 16-BIT SIGNED BYTE UNPACK
[VUPS32] VECTOR 32-BIT SIGNED BYTE UNPACK
[VUPS8] VECTOR 8-BIT SIGNED BYTE UNPACK
UNWRAP AND COMPLEX CEPSTRUM...[CCEPS] PHASE
[TVCLR] TABLE MEMORY VECTOR CLEAR
[VCLR] VECTOR CLEAR
[VCLIP] VECTOR CLIP
[VICLIP] VECTOR INVERTED CLIP
[TMVLC2] VECTOR LINEAR COMBINATION
[TTVLC2] VECTOR LINEAR COMBINATION
[CVCOMB] COMPLEX VECTOR COMBINE
[CFFT2D] COMPLEX TO COMPLEX 2-DIMENSIONAL FFT
[RFFT2D] REAL TO COMPLEX 2-DIMENSIONAL FFT
[CAXPY] NESTED COMPLEX A * X + Y
[CCEPS] PHASE UNWRAP AND COMPLEX CEPSTRUM
[CDOTN] NESTED COMPLEX DOT PRODUCT
[ICAMAX] INDEX OF LARGEST COMPLEX ELEMENT
[CFFTM] MIXED-RADIX
[PAS2F] RADIX-2 FORWARD COMPLEX FFT NOT-IN-PLACE
[PAS2I] RADIX-2 INVERSE COMPLEX FFT PASS
[PAS3F] RADIX-3 FORWARD COMPLEX FFT PASS
[PAS3I] RADIX-3 INVERSE COMPLEX FFT PASS
[PAS4F] RADIX-4 FORWARD COMPLEX FFT PASS
[PAS4I] RADIX-4 INVERSE COMPLEX FFT PASS
[PAS5F] RADIX-5 FORWARD COMPLEX FFT PASS
[PAS5I] RADIX-5 INVERSE COMPLEX FFT PASS
[CEXP] EXPONENTIAL OF COMPLEX NUMBER
[CONJG] CONJUGATE OF COMPLEX NUMBER
[CSQRT] SQUARE ROOT OF COMPLEX NUMBER
[CPWR] REAL TO COMPLEX POWER
[CPOW] COMPLEX TO COMPLEX POWER
[CSFS2] SPARSE COMPLEX SYMM FACTOR & SOLVE
[CSFR2] SPARSE COMPLEX SYMMETRIC FACTOR
[CSSV2] SPARSE COMPLEX SYMMETRIC SOLVE
[CUSFS2] SPARSE COMPLEX UNSYM FACTOR & SOLVE
[CUSFR2] SPARSE COMPLEX UNSYMMETRIC FACTOR
[CUSV2] SPARSE COMPLEX UNSYMMETRIC SOLVE
[VIMAG] EXTRACT IMAGINARIES OF COMPLEX VECTOR
[VREAL] EXTRACT REALS OF COMPLEX VECTOR
[CVREAL] FORM COMPLEX VECTOR OF REALS
[CSSCAL] REAL TIMES COMPLEXES
[CVCONJ] COMPLEX VECTOR CONJUGATE
[CONNMO] NMO WITH CONSTANT VELOCITY
[POLAR] RECTANGULAR TO POLAR CONVERSION
[RECT] POLAR TO RECTANGULAR CONVERSION
[VDBPWR] VECTOR CONVERSION TO DB POWER
[CONV2D] 2-D CONVOLUTION AND CORRELATION
[CONV] POST-TAPERED CONVOLUTION CORRELATION
[CTRNS3] 3-DIMENSIONAL COORDINATE TRANSFORMATION
[CCOPY] COMPLEX VECTOR COPY
<table>
<thead>
<tr>
<th>Key Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[DNEG]</td>
<td>N Egate of DOUBLE-PRECISION NUMBER</td>
</tr>
<tr>
<td>[DSUBRR]</td>
<td>S To DOUBLE-PRECISION SUBTRACT</td>
</tr>
<tr>
<td>[DSUB]</td>
<td>D To DOUBLE-PRECISION SUBTRACT</td>
</tr>
<tr>
<td>[CH]</td>
<td>CHERMITIAN EIGENSYSTEM SOLVER</td>
</tr>
<tr>
<td>[EIGRS]</td>
<td>R SYMMETRIC EIGENSYSTEM SOLVER</td>
</tr>
<tr>
<td>[RS]</td>
<td>S SYMMETRIC EIGENSYSTEM SOLVER</td>
</tr>
<tr>
<td>[HTRISB]</td>
<td>SHERMITIAN EIGENVARIABLES</td>
</tr>
<tr>
<td>[ICMAX]</td>
<td>X INDEX OF LARGEST ELEMENT [KSMLV]</td>
</tr>
<tr>
<td>[KSMLV]</td>
<td>SMALLEST ELEMENT IN VECTOR</td>
</tr>
<tr>
<td>[MAXMGV]</td>
<td>M MAXIMUM MAGNITUDE IN VECTOR</td>
</tr>
<tr>
<td>[MAXV]</td>
<td>V MAXIMUM ELEMENT IN VECTOR</td>
</tr>
<tr>
<td>[MINMGV]</td>
<td>M MINIMUM MAGNITUDE IN VECTOR</td>
</tr>
<tr>
<td>[MINV]</td>
<td>V MINIMUM ELEMENT IN VECTOR</td>
</tr>
<tr>
<td>[MEAMGV]</td>
<td>M MEAN OF VECTOR ELEMENTS</td>
</tr>
<tr>
<td>[MEANV]</td>
<td>M OF VECTOR ELEMENTS</td>
</tr>
<tr>
<td>[MEASQV]</td>
<td>S VECTOR SUM OF SQUARES</td>
</tr>
<tr>
<td>[SVE]</td>
<td>V SUM OF VECTOR ELEMENTS</td>
</tr>
<tr>
<td>[VSUM]</td>
<td>SUM OF VECTOR ELEMENTS</td>
</tr>
<tr>
<td>[LVEQ]</td>
<td>E QUAL LOGICAL VECTOR</td>
</tr>
<tr>
<td>[LVGE]</td>
<td>G EQUAL LOGICAL VECTOR</td>
</tr>
<tr>
<td>[LVNE]</td>
<td>N NOT LOGICAL VECTOR</td>
</tr>
<tr>
<td>[CMSOLV]</td>
<td>S COMPLEX MATRIX EQUATION SOLVER</td>
</tr>
<tr>
<td>[CSOLVQ]</td>
<td>Q COMPLEX MATRIX EQUATION SOLVER</td>
</tr>
<tr>
<td>[SKYSOL]</td>
<td>S SKYLINE FORMAT EQUATION SOLVER</td>
</tr>
<tr>
<td>[SOLVEQ]</td>
<td>Q LINEAR EQUATION SOLVER</td>
</tr>
<tr>
<td>[DEQ22]</td>
<td>DIFFERENCE EQUATION, 2 POLES, 2 ZEROS</td>
</tr>
<tr>
<td>[VLEQV]</td>
<td>V LOGICAL EQUATION</td>
</tr>
<tr>
<td>[VEUCL2]</td>
<td>EUCLIDEAN DISTANCE</td>
</tr>
<tr>
<td>[SCNRM2]</td>
<td>N COMPLEX EUCLIDEAN NORM</td>
</tr>
<tr>
<td>[VPOLY]</td>
<td>POLYNOMIAL EVALUATION</td>
</tr>
<tr>
<td>[VLXOR]</td>
<td>X EXCLUSIVE OR LOGICAL</td>
</tr>
<tr>
<td>[CVEXP]</td>
<td>EXPONENTIAL VECTOR</td>
</tr>
<tr>
<td>[VEXP]</td>
<td>P EXPONENTIAL VECTOR</td>
</tr>
<tr>
<td>[VAVEXP]</td>
<td>EXPONENTIAL AVERAGING</td>
</tr>
<tr>
<td>[VEXP10]</td>
<td>EXPONENTIAL BASE 10</td>
</tr>
<tr>
<td>[COMPLEX]</td>
<td>SYMMETRIC FACTOR</td>
</tr>
<tr>
<td>[COMPLEX]</td>
<td>UNSYMMETRIC FACTOR</td>
</tr>
<tr>
<td>[RSFR2]</td>
<td>SPARSE REAL SYMMETRIC</td>
</tr>
<tr>
<td>[RUFR2]</td>
<td>SPARSE REAL UNSYMMETRIC</td>
</tr>
<tr>
<td>[SGEFA]</td>
<td>GENERAL MATRIX FACTOR</td>
</tr>
<tr>
<td>[CSFS2]</td>
<td>SPARSE COMPLEX SYMMETRIC FACTOR &amp; SOLVE</td>
</tr>
<tr>
<td>[CUFS2]</td>
<td>SPARSE COMPLEX FACTOR</td>
</tr>
<tr>
<td>[RSFS2]</td>
<td>SPARSE REAL SYMMETRIC FACTOR &amp; SOLVE</td>
</tr>
<tr>
<td>[RUF2]</td>
<td>SPARSE REAL UNSYMMETRIC FACTOR &amp; SOLVE</td>
</tr>
<tr>
<td>[GENTAB]</td>
<td>GENERATE TWIDDLE FACTOR</td>
</tr>
<tr>
<td>[CMFACT]</td>
<td>COMPLEX MATRIX L/U FACTORIZATION CROUT</td>
</tr>
<tr>
<td>[LUF]</td>
<td>F FACTORIZATION</td>
</tr>
<tr>
<td>[PEEK]</td>
<td>E MEMORY FETCH</td>
</tr>
<tr>
<td>TO COMPLEX</td>
<td>FFT...[CFFT2D] COMPLEX</td>
</tr>
<tr>
<td>TO COMPLEX</td>
<td>FFT...[RFFT2D] REAL</td>
</tr>
<tr>
<td>Key Word</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>[GRAD2D]MAXIMUM</td>
<td>GRADIENT FILTER</td>
</tr>
<tr>
<td>[GRD2DB]MAXIMUM</td>
<td>GRADIENT FILTER WITH BOUND</td>
</tr>
<tr>
<td>[LVGT]LOGICAL VECTOR</td>
<td>GREATER THAN</td>
</tr>
<tr>
<td>[LVGE]LOGICAL VECTOR</td>
<td>GREATER THAN OR EQUAL</td>
</tr>
<tr>
<td>[VPR32]VECTOR</td>
<td>HALFWORD PACK</td>
</tr>
<tr>
<td>[VUPR32]VECTOR</td>
<td>HALFWORD REAL UNPACK</td>
</tr>
<tr>
<td>[CH]COMPLEX</td>
<td>HERMITIAN EIGENSYSTEM SOLVER</td>
</tr>
<tr>
<td>[HTRIBK]COMPLEX</td>
<td>HERMITIAN EIGENVECTORS</td>
</tr>
<tr>
<td>[HTRIDI]COMPLEX</td>
<td>HERMITIAN TRIDIAGONALIZATION</td>
</tr>
<tr>
<td>[VCOSH]VECTOR</td>
<td>HYPERBOLIC COSINE</td>
</tr>
<tr>
<td>[VSINH]VECTOR</td>
<td>HYPERBOLIC SINE</td>
</tr>
<tr>
<td>[VTANH]VECTOR</td>
<td>HYPERBOLIC TANGENT</td>
</tr>
<tr>
<td>[COSH]REAL NUMBER</td>
<td>HYPERBOLIC COSINE</td>
</tr>
<tr>
<td>[SINH]REAL NUMBER</td>
<td>HYPERBOLIC SINE</td>
</tr>
<tr>
<td>[TANH]REAL NUMBER</td>
<td>HYPERBOLIC TANGENT</td>
</tr>
<tr>
<td>[VIMAG]VECTOR</td>
<td>IMAGINARIES OF COMPLEX VECTOR</td>
</tr>
<tr>
<td>[SCASUM]</td>
<td>SUM OF REAL AND IMAGINARY MAGNITUDES</td>
</tr>
<tr>
<td>[MTIMOV]VECTOR</td>
<td>MOVE WITH INCREMENT MD TO TM</td>
</tr>
<tr>
<td>[TMIMOV]VECTOR</td>
<td>MOVE WITH INCREMENT TM TO MD</td>
</tr>
<tr>
<td>[TTIMOV]VECTOR</td>
<td>MOVE WITH INCREMENT TM TO TM</td>
</tr>
<tr>
<td>[VINDEX]VECTOR</td>
<td>INDEX</td>
</tr>
<tr>
<td>[VSORT]VECTOR</td>
<td>SORT WITH INDICES</td>
</tr>
<tr>
<td>[CDOTC]COMPLEX</td>
<td>INNER PRODUCT</td>
</tr>
<tr>
<td>[RKGTF]</td>
<td>R-K-GILL-THOMPSON</td>
</tr>
<tr>
<td>[ADAMS4]</td>
<td>ADAMS VARIABLE STEP</td>
</tr>
<tr>
<td>REAL NUMBER TO NEAREST</td>
<td>INTEGER...[NINT]ROUND</td>
</tr>
<tr>
<td>[VIADD]VECTOR</td>
<td>INTEGER ADD</td>
</tr>
<tr>
<td>[VIDIV]VECTOR</td>
<td>INTEGER DIVIDE</td>
</tr>
<tr>
<td>[VIFIX]VECTOR</td>
<td>INTEGER FIX</td>
</tr>
<tr>
<td>[VIMUL]VECTOR</td>
<td>INTEGER MULTIPLY</td>
</tr>
<tr>
<td>[VINEG]VECTOR</td>
<td>INTEGER NEGATE</td>
</tr>
<tr>
<td>[VPKI32]VECTOR</td>
<td>32-BIT INTEGER PACK</td>
</tr>
<tr>
<td>[CPWCI]COMPLEX</td>
<td>INTEGER POWER</td>
</tr>
<tr>
<td>[IPOW]INTEGER</td>
<td>INTEGER POWER</td>
</tr>
<tr>
<td>[RPOWRI]REAL</td>
<td>INTEGER POWER</td>
</tr>
<tr>
<td>[VISUB]VECTOR</td>
<td>INTEGER SUBTRACT</td>
</tr>
<tr>
<td>[VFLOAT]CONVERT</td>
<td>INTEGER TO FLOATING-POINT</td>
</tr>
<tr>
<td>[VUSI32]VECTOR</td>
<td>32-BIT SIGNED INTEGER UNPACK</td>
</tr>
<tr>
<td>[VISORT]VECTOR</td>
<td>INTEGER VALUES</td>
</tr>
<tr>
<td>[RKGIL]</td>
<td>RUNGE-KUTTA-GILL SIMPSON'S 1/3 RULE</td>
</tr>
<tr>
<td>[VSUM]VECTOR</td>
<td>SUM OF ELEMENTS</td>
</tr>
<tr>
<td>[SCS1]SCALAR</td>
<td>COS/SIN, TM</td>
</tr>
<tr>
<td>[CFFTI]COMPLEX</td>
<td>FFT WITH INTERPOLATION</td>
</tr>
<tr>
<td>[NMOLI]NMO</td>
<td>LINEAR INTEGRATION</td>
</tr>
<tr>
<td>[NMOQI]NMO QUADRATIC</td>
<td>INTERPOLATION</td>
</tr>
<tr>
<td>[RFFTI]REAL</td>
<td>FFT WITH INTERPOLATION</td>
</tr>
<tr>
<td>[RFTII]REAL</td>
<td>FFT WITH QUARTER</td>
</tr>
<tr>
<td>[CMATIN]COMPLEX</td>
<td>MATRIX INVERSE</td>
</tr>
<tr>
<td>[MATINV]MATRIX</td>
<td>INVERSE</td>
</tr>
<tr>
<td>[VRECP]VECTOR</td>
<td>INVERSE</td>
</tr>
<tr>
<td>[PAS2I]RADIX-2</td>
<td>INVERSE COMPLEX FFT PASS</td>
</tr>
<tr>
<td>[PAS3I]RADIX-3</td>
<td>INVERSE COMPLEX FFT PASS</td>
</tr>
</tbody>
</table>
APMATH64 KEY WORD INDEX

[CMMTRC]COMPLEX MATRIX MULTIPLY TRACE
[SMPACK]SPARSE MATRIX PACK
[SGESL]REAL GENERAL MATRIX SOLVE
[LUSN]LU MATRIX SOLVE CROUT
[SGTSL]TRIDIAGONAL MATRIX SOLVER
[TRIDIA]TRIDIAGONAL MATRIX SOLVER
[SMPCK]SPARSE MATRIX UNPACK
[SMVMUL]SPARSE MATRIX VECTOR MULTIPLY
[ROT3]3D ROTATION SYMMETRIC MATRIX...[TRED]TRIDIAGONALIZE
[VMAX]VECTOR MAXIMUM
[ISAMAX]INDEX OF MAXIMUM ABSOLUTE VALUE
[VMAXM]VECTOR MAXIMUM MAGNITUDE
[MMTMUL]VECTOR MULTIPLY MD*MD TO TM
[MTTMUL]VECTOR MULTIPLY MD*TM TO MD
[MCTMUL]VECTOR MULTIPLY MD*M TO TM
[MNTADD]VECTOR ADD MD+TM TO MD
[MMTADD]VECTOR ADD MD+TM TO TM
[MWTADD]VECTOR ADD MD+TM TO TM
[MWTSUB]VECTOR SUBTRACT MD-MD TO TM
[MWTMUL]VECTOR SUBTRACT MD-TM TO TM
[MTTMUL]VECTOR SUBTRACT MD-TM TO TM
[MPOKE]STORE INTO MEMORY
[TVCLR]TABLE MEMORY VECTOR CLEAR
[VLMERG]LOGICAL VECTOR MERGE
[VMIN]VECTOR MINIMUM
[VMINM]VECTOR MINIMUM MAGNITUDE
[CVMOV]COMPLEX VECTOR MOVE
[SVMOV]SPARSE VECTOR MOVE
[VMOV]VECTOR MOVE
[MVREP]SUB-IMAGE MOVE AND LEVEL REPLACE
[MTMOV]VECTOR MOVE MD TO TM
[TMMOV]VECTOR MOVE TM TO MD
[MTMOV]VECTOR MOVE WITH INCREMENT MD TO TM
[TTMOV]VECTOR MOVE WITH INCREMENT TM TO TM
[MTTMOV]VECTOR MOVE WITH INCREMENT TM TO TM
[RESNMO]RESIDUAL NORMAL MOVEOUT
[VSMA3]THREE VECTOR SCALAR MULT AND ADD
[VSMA4]FOUR VECTOR SCALAR MULT AND ADD
[CMVML3]COMPLEX 3X3 MATRIX MULTIPLY...3D VECTORS
[CMVML4]COMPLEX 4X4 MATRIX MULTIPLY...4D VECTORS
[VCXS]VECTOR MULTIPLIED BY SIN AND COS
[BLKMAN]BLACKMAN WINDOW MULTIPLY
[CCMUL]COMPLEX MATRIX MULTIPLY
[CMGML]COMPLEX GENERAL MATRIX MULTIPLY
[CMUL]COMPLEX MATRIX MULTIPLY
[CMUL]COMPLEX MATRIX MULTIPLY
[CRRMUL]COMPLEX-REAL MATRIX MULTIPLY
[CRRMUL]REAL VECTOR MULTIPLY...[CVMUL]COMPLEX
[CVMUL]COMPLEX VECTOR MULTIPLY
[CMVMUL]COMPLEX VECTOR SCALAR MULTIPLY
TO DOUBLE PRECISION MULTIPLY...[DMUL]SINGLE
TO DOUBLE-PRECISION MULTIPLY...[DMUL]DOUBLE
[CMVMUL]FAST MATRIX MULTIPLY
[VNEG] VECTOR NEGATE
[MNAXB] SUB-MATRIX NEGATIVE MULTIPLY
[MNATXB] SUB-MATRIX NEGATIVE TRANSPOSE MULTIPLY
[SCNRM2] COMPLEX EUCLIDEAN NORM
[SNRM2] EUCLIDEAN NORM
[RESNMO] RESIDUAL NORMAL MOVEOUT
[LVNOT] LOGICAL VECTOR NOT
[VLNOT] VECTOR LOGICAL NOT
[LVNE] LOGICAL VECTOR NOT EQUAL
[CFFTB] COMPLEX-TO-COMPLEX FFT NOT IN PLACE
[RFFTB] REAL-TO-COMPLEX FFT NOT IN PLACE
[CFFTM] MIXED-RADIX COMPLEX FFT NOT IN PLACE
[RFFTM] MIXED-RADIX REAL FFT NOT IN PLACE
[AINT] TRUNCATE REAL NUMBER
[ATAN] ARCTANGENT OF REAL NUMBER
[CEXP] EXPONENTIAL OF COMPLEX NUMBER
[CONJG] CONJUGATE OF COMPLEX NUMBER
[CSQRT] SQUARE ROOT OF COMPLEX NUMBER
[DNEG] NEGATE DOUBLE-PRECISION NUMBER
[EXP] EXPONENTIAL OF REAL NUMBER
[SQRT] SQUARE ROOT OF REAL NUMBER
[ABS] REAL NUMBER ABSOLUTE VALUE
[CABS] COMPLEX NUMBER ABSOLUTE VALUE
[ACOS] REAL NUMBER ARCCOSINE
[ASIN] REAL NUMBER ARCSINE
[CCOS] COMPLEX NUMBER COSINE
[COS] REAL NUMBER COSINE
[RAN] SCALAR RANDOM NUMBER GENERATOR
[COSH] REAL NUMBER HYPERBOLIC COSINE
[SINH] REAL NUMBER HYPERBOLIC SINE
[TANH] REAL NUMBER HYPERBOLIC TANGENT
[ALOG10] REAL NUMBER LOGARITHM
[ALOG] REAL NUMBER LOGARITHM
[CLOG] COMPLEX NUMBER LOGARITHM
[SIGN] REAL NUMBER SIGN TRANSFER
[CSIN] COMPLEX NUMBER SINE
[SIN] REAL NUMBER SINE
[TAN] REAL NUMBER TANGENT
[NINT] ROUND REAL NUMBER TO NEAREST INTEGER
[ANINT] ROUND REAL NUMBER TO NEAREST WHOLE
OF RATIO OF REAL NUMBERS...[ATAN2] ARCTANGENT
[VRAND] VECTOR RANDOM NUMBERS
[FUN1] FUNCTION OF ONE VARIABLE
[VLOR] VECTOR LOGICAL OR
LOGICAL EXCLUSIVE OR...[VLXOR] VECTOR
VECTOR GREATER THAN OR EQUAL...[LVGE] LOGICAL
PREDICTOR ORDER 1...[ABP1] ADAMS-BASHFORTH
[AMC1] ADAMS-MOULTON CORRECTOR ORDER 1
[AMC2] ADAMS-MOULTON CORRECTOR ORDER 2
[AMC3] ADAMS-MOULTON CORRECTOR ORDER 3
[AMC4] ADAMS-MOULTON CORRECTOR ORDER 4
[VRAMP]VECTOR RAMP
[VRAND]VECTOR RANDOM NUMBERS
[POST64] POST BITS TO RASTER
[ATAN2] ARCTANGENT OF RATIO OF REAL NUMBERS
[MRRUNR] MIXED-RADIX RFFT RAEL/UNRAEEL PASS
[ATAN2] ARCTANGENT OF RATIO OF REAL NUMBERS
[EXP] EXPONENTIAL OF REAL NUMBER
[SQRT] SQUARE ROOT OF REAL NUMBER
[NINT] ROUND REAL NUMBER TO NEAREST INTEGER
[ANINT] ROUND REAL NUMBER TO NEAREST WHOLE
[ATAN2] ARCTANGENT OF RATIO OF REAL NUMBERS
[CPWCR] COMPLEX TO REAL POWER
[RPOW] REAL TO REAL POWER
[DDOTRR] DOUBLE DOT PRODUCT REAL REAL
[RSFSZ] SPARSE REAL SYMM FACTOR & SOLVE
[RSFRZ] SPARSE REAL SYMMETRIC FACTOR
[RSSVZ] SPARSE REAL SYMMETRIC SOLVE
[VUPR32] VECTOR HALFWORD REAL UNPACK
[RUFSZ] SPARSE REAL UNSYM FACTOR & SOLVE
[RUFRZ] SPARSE REAL UNSYMMETRIC FACTOR
[RUSVZ] SPARSE REAL UNSYMMETRIC SOLVE
[CRVADD] COMPLEX AND REAL VECTOR ADD
[CRVDIV] COMPLEX AND REAL VECTOR DIVIDE
[CRVMUL] COMPLEX AND REAL VECTOR MULTIPLY
[CRVSUB] COMPLEX AND REAL VECTOR SUBTRACT
[SDOT] DOT PRODUCT OF REAL VECTORS
[CVREAL] FORM COMPLEX VECTOR OF REALS
[REAL] EXTRACT REALS OF COMPLEX VECTOR
[CVRCIP] COMPLEX VECTOR RECIPROCAL
[RRCP] REAL RECIPROCAL
[VRSQRT] VECTOR RECIPROCAL SQUARE ROOT
[RECT] POLAR TO RECTANGULAR CONVERSION
[AMOD] REAL/REAL DIVIDE REMAINDER
[MOD] INTEGER/INTEGER DIVIDE REMAINDER
[MOVREP] MOVE AND LEVEL REPLACE...[MOVREP]SUB-IMAGE
[MRRUNR] MIXED-RADIX RFFT RAEL/UNRAEEL PASS
[RSQRT] RECIPROCAL SQUARE ROOT
[VRSQRT] VECTOR SQUARE ROOT
[CSQRT] SQUARE ROOT OF COMPLEX NUMBER
[SQRT] SQUARE ROOT OF REAL NUMBER
[CROTG] COMPLEX GIVENS ROTATION
[CSROT] COMPLEX 2-D ROTATION
[DSROT] GIVENS PLANE ROTATION
[PSOR] PLANE ROTATION
APMATH64 KEY WORD INDEX

[VSINH]VECTOR SINE HYPERBOLIC
[KSMLV]K-TH SMALLEST ELEMENT IN VECTOR
COMPLEX SYMM FACTOR AND SOLVE ...[CSFS2]SPARSE
COMPLEX SYMMETRIC SOLVE ...[CSSV2]SPARSE
COMPLEX UNSYM FACTOR AND SOLVE ...[CUFS2]SPARSE
COMPLEX UNSYMMETRIC SOLVE ...[CUSV2]SPARSE
REAL SYMM FACTOR AND SOLVE ...[RSFS2]SPARSE
REAL UNSYM FACTOR AND SOLVE ...[RUSFS2]SPARSE
[RSSV2]SPARSE REAL SYMMETRIC SOLVE
[RUFS2]SPARSE REAL UNSYMMETRIC SOLVE
[SGESL]REAL GENERAL MATRIX SOLVE
[LUSN]LU MATRIX SOLVE CROUT
HERMITIAN EIGENSYSTEM SOLVER...[CH]COMPLEX
MATRIX EQUATION SOLVER...[CMSOLV]COMPLEX
MATRIX EQUATION SOLVER...[CSOLVQ]COMPLEX
[CSOLV]COMPLEX SYSTEM SOLVER
SYMMETRIC EIGENSYSTEM SOLVER...[EIGRS]REAL
[RS]REAL SYMMETRIC EIGENSYSTEM SOLVER
[SQTS]TRIDIAGONAL MATRIX SOLVER
[SITS]SPARSE ITERATIVE SOLVER
FORMAT EQUATION SOLVER...[SKYSOL]SKYLINE
[SOLVEQ]LINEAR EQUATION SOLVER
[TRIDIA]TRIDIAGONAL MATRIX SOLVER
[VASORT]VECTOR SORT ALGEBRAIC VALUES
VISORT]VECTOR SORT INTEGER VALUES
[VSORT]VECTOR SORT WITH INDICES
[VSQ]VECTOR SQUARE
[VSSQ]VECTOR SIGNED SQUARE
[RSQRT]RECIPROCAL SQUARE ROOT
[VRSQRT]VECTOR RECIPROCAL SQUARE ROOT
[VSQRT]VECTOR SQUARE ROOT
VECTOR MAGNITUDE SQUARED...[CVMAGS]COMPLEX
[MEASQV]MEAN OF VECTOR ELEMENT SQUARES
[SVESQ]SUM OF VECTOR ELEMENT SQUARES
[SVS]SUM OF VECTOR SIGNED SQUARES
[ADAMS4]ADAMS VARIABLE STEP INTEG.ORD 4
[CMTRAC]COMPLEX SUB-MATRIX TRACE
[CMTRAN]COMPLEX SUB-MATRIX TRANSPOSE
AND REAL VECTOR SUBTRACT...[CRVSUB]COMPLEX
[CVSUB]COMPLEX VECTOR SUBTRACT
TO DOUBLE-PRECISION SUBTRACT...[DSUBRR]SINGLE
TO DOUBLE-PRECISION SUBTRACT...[DSUB]DOUBLE
[VSUB]VECTOR INTEGER SUBTRACT
MULTIPLY, MULTIPLY, AND SUBTRACT...[VMMSB]VECTOR
[VMSB]VECTOR MULTIPLY AND SUBTRACT
SCALAR MULTIPLY AND SUBTRACT...[VSMSB]VECTOR
[VSUB]VECTOR SUBTRACT
[VSREM]VECTOR SUBTRACT AND MULTIPLY
[VBSM]VECTOR SUBTRACT AND SCALAR MULTIPLY
[MMTS]VECTOR SUBTRACT MD-MD TO TM
[MNM]VECTOR SUBTRACT MD-TM TO MD
[MNT]VECTOR SUBTRACT MD-TM TO TM
[TMMS]VECTOR SUBTRACT TM-MD TO MD
[TMNT]VECTOR SUBTRACT TM-MD TO TM
APMATH64 KEY WORD INDEX

[SVS] SUM OF VECTOR SIGNED SQUARES
[CRVSUB] COMPLEX AND REAL VECTOR SUBTRACT
[CVSUB] COMPLEX VECTOR SUBTRACT
[CSWAP] COMPLEX VECTOR SWAP
[SVUPCK] SPARSE VECTOR UNPACK
3X3 MATRIX MULT. 3D VECTORS... [CMVML3] COMPLEX
4X4 MATRIX MULT. 4D VECTORS... [CMVML4] COMPLEX
[SDOT] DOT PRODUCT OF REAL VECTORS
DISTANCE BETWEEN TWO VECTORS...[SN2] SQUARED
[SSWAP] INTERCHANGES VECTORS
[CONNMO] NMO WITH CONSTANT VELOCITY
[VARNMO] NMO WITH VARIABLE VELOCITY
REAL NUMBER TO NEAREST WHOLE...[ANINT] ROUND
[BLKMAN] BLACKMAN WINDOW MULTIPLY
[HAMM] HAMMING WINDOW MULTIPLY
[HANN] HANNING WINDOW MULTIPLY
[TMMM] MATRIX MULTIPLY TM WORKSPACE
[CAXPYN] NESTED COMPLEX A * X + Y
[CAXPY] COMPLEX A * X + Y
[SAXPYN] NESTED REAL A * X + Y
[SAXPY] REAL A * X + Y
[CAXPYN] NESTED COMPLEX A * X + Y
[CAXPY] COMPLEX A * X + Y
[SAXPYN] NESTED REAL A * X + Y
[SAXPY] REAL A * X + Y
[VNZ] VECTOR ZERO TRENDS
EQUATION, 2 POLES, 2 ZEROS...[DEQ22] DIFFERENCE
[VSCANZ] VECTOR SCAN FOR ZEROs
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: ____________________________________________________________ Date: __________________

Your Name and Title: ___________________________________________ Department: ____________________

Firm: ___________________________ 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...
☐ accurate
☐ complete
☐ written clearly
☐ well illustrated
☐ well indexed

Yes
☐
☐

No
☐
☐

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 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: ___________________________ Address: ___________________________________________

City: ___________________________ State: ___________________________ Zip Code: __________________

Telephone Number: (______) _____________________ Extension: _______________________

ARRAY