

APMATH64 MANUAL

VOLUME 3 OF 4

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

860-7482-001C



FLOATING POINT SYSTEMS, INC.

APMATH64 MANUAL

VOLUME 3 OF 4

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

860-7482-001C

by FPS Technical Publications Staff

Publication No. 86Ø-7482-ØØ1C
December, 1987

NOTICE

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

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

Copyright © 1987 by Floating Point Systems, Inc.

All rights reserved. No part of this publication may
be reproduced in any form without written permission
from the publisher.

Printed in USA

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

REVISION HISTORY

This manual is the *APMATH64 Manual*, Volume 3, 860-7482-001. The letter shown under the revision number column indicates the portion of the part number that changes for each revision. The last entry is the latest revision to this manual.

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

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

NOTE TO READER

This is the 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.

CONTENTS (VOLUME 3)

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

CONTENTS

APPENDIX G	LIST OF SUPERSEDED ROUTINES	Page
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

```

*****
*           *
* MMTMUL *   --- 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 C
- 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

```

```

*****
*      *
* MTIMOV * --- VECTOR MOVE WITH INCREMENT (MD TO TM) --- * 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.

 * *
 * MTMMUL *
 * *

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

 * *
 * MTMMUL *
 * *

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

 * *
 * MTMSUB *
 * *

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

 * *
 * MTMSUB *
 * *

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

 * *
 * MTTMUL *
 * *

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

 * *
 * MTTMUL *
 * *

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

```
*****
*      *
* TMDOT *
*      *
*****
```

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

```
*****
*      *
* TMDOT *
*      *
*****
```

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:

$$C = A(1)*B(1) + A(2)*B(2) + \dots + A(N)*B(N)$$

$$C = 0.0, \text{ if } N < 1$$

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
```

```
*****
*      *
*  TMM  *
*      *
*****
```

--- MATRIX MULTIPLY (TM WORKSPACE) ---

```
*****
*      *
*  TMM  *
*      *
*****
```

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 TMM (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: TMM 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:

$$A = \begin{vmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{vmatrix} \quad B = \begin{vmatrix} 2.0 & 6.0 & 9.0 \\ 3.0 & 7.0 & 4.0 \end{vmatrix}$$

```
*****
*      *
* TMMSUB *
*      *
*****
```

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

```
*****
*      *
* TMMSUB *
*      *
*****
```

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
```

```

*****
*      *
* TMVLC2 *      --- VECTOR LINEAR COMBINATION ---
*      *
*****
*****
*      *
* TMVLC2 *
*      *
*****

```

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, S2, 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

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

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

```

```

*****
*          *
* TTMADD *      --- VECTOR ADD (TM+TM TO MD) ---
*          *
*****
*****
*          *
* TTMADD *
*          *
*****

```

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

```

```

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

```

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

```

```
*****
*      *
* TTTMUL *
*      *
*****
```

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

```
*****
*      *
* TTTMUL *
*      *
*****
```

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
```

 * *
 * TTVLC2 *
 * *

--- VECTOR LINEAR COMBINATION ---

 * *
 * TTVLC2 *
 * *

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

SPECIAL UTILITIES LIBRARY

```

*****
*      *
* PEEK *
*      *
*****

```

--- MEMORY FETCH ---

```

*****
*      *
* PEEK *
*      *
*****

```

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

DATA FORMATTING LIBRARY

```
*****
*      *
* VIFIX *
*      *
*****
```

--- VECTOR INTEGER FIX ---

```
*****
*      *
* VIFIX *
*      *
*****
```

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
```

```

*****
*           *
*  VPK16  *   --- VECTOR 16-BIT BYTE PACK ---
*           *
*****
*****
*           *
*  VPK16  *
*           *
*****

```

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 : 0008FFF800060006  00040003FFFE0001

F = 1

A : 8.3  -7.9  6.5  5.6  4.1  3.4  -2.5  1.1

C : 0008FFF900060005  00040003FFFE0001

```

```

*****
*          *
* VPKI32 *   --- VECTOR 32-BIT INTEGER PACK ---
*          *
*****
*****
*          *
* VPKI32 *
*          *
*****

```

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) bits 32 to 63
 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 (XXX indicates 'undefined')

```

A: 80C0000000000006 C: 0000000600000004
80A0000000000005 XXXXXXXXXXXXXXXXXXXX
8080000000000004 XXXXXXXXXXXXXXXXXXXX
8060000000000003 0000000200000000
0000000000000002 XXXXXXXXXXXXXXXXXXXX
0000000000000001 XXXXXXXXXXXXXXXXXXXX
8000000000000000 FFFFFFFFEEEEEEFC
7FFFFFFFFFFFFFFF
7FDFFFFFFFFFFFFFFE
00000000FFFFFFFFFD
00000000FFFFFFFFFC

```

```
*****
*           *
* VSCALE *
*           *
*****
```

--- VECTOR SCALE AND FIX ---

```
*****
*           *
* VSCALE *
*           *
*****
```

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: $C(m) = \text{FIX}(A(m) \cdot \{2^{IEXP}\})$ for $m=0$ to $N-1$
 where $IEXP = NB - E - 1$,
 and $B = \text{FRAC}(2^{IEXP})$.
 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 : 10.0  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: 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
 NS = Integer power of 2 (May be negative)

DESCRIPTION: $C(m) = \text{FIX}\{A(m) \cdot (2^{**NS})\}$; for $m=1$ to N

EXAMPLE:

```
N = 3
NS = 2

A : 1.0  2.0  3.2
C : 4    8   12
```

 * *
 * VUP8 *
 * *

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

 * *
 * VUP8 *
 * *

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 : 0807060504030201 0807060504030201

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

```
*****
*      *
* VUP32 *
*      *
*****
```

— VECTOR 32-BIT BYTE UNPACK —

```
*****
*      *
* VUP32 *
*      *
*****
```

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 : 00000000800000007
 00000000600000005
 00000000400000003
 00000000200000001

C : 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0

```

*****
*           *
* VUPS8 *   --- VECTOR 8-BIT SIGNED BYTE UNPACK ---
*           *
*****
*****
*           *
* VUPS8 *
*           *
*****

```

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 : 08F9060504FD0201 0807FA050403FE01

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

```

*****
*           *
* VUPS32 *   — VECTOR 32-BIT SIGNED BYTE UNPACK —   * VUPS32 *
*           *
*****

```

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 : 00000008FFFFFFFF9
 0000000600000005
 FFFFFFFCFFFFFFFFD
 FFFFFFFE00000001

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

```

*****
*           *
* VUUI32 *   --- VECTOR 32-BIT UNSIGNED UNPACK ---   * VUUI32 *
*           *
*****
    
```

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 (XXX indicates 'undefined')

A: 0000000800000007	C: 0000000000000008
0000000600000005	XXXXXXXXXXXXXXXXXX
0000000400000003	0000000000000007
0000000200000001	XXXXXXXXXXXXXXXXXX
FFFFFFFFFFFFFFFFFE	0000000000000002
FFFFFFFFDFFFFFFFFC	XXXXXXXXXXXXXXXXXX
FFFFFFFFBFFFFFFFFA	0000000000000001
	XXXXXXXXXXXXXXXXXX
	00000000FFFFFFFFFB
	XXXXXXXXXXXXXXXXXX
	00000000FFFFFFFFFA

```

*****
*          *
*  DADD  *   --- DOUBLE TO DOUBLE-PRECISION ADD ---
*          *
*****

```

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).

```

*****
*          *
* DADOT   *   ----- DOUBLE ACCUMULATE DOT PRODUCT -----
*          *
*          *
*****

```

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) * 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) * INC + 1 to the first element (BLAS convention).

```

*****
*           *
*  DMUL  *  — DOUBLE TO DOUBLE-PRECISION MULTIPLY —  *  DMUL  *
*           *
*****

```

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).

```

*****
*           *
*  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).

```

*****
*          *
* 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).

```
*****  
*      *  
*  ABS  *      --- REAL NUMBER ABSOLUTE VALUE ---  
*      *  
*****
```

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 = $|arg|$

* *
* AINT *
* *

--- TRUNCATE REAL NUMBER ---

* *
* AINT *
* *

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))

 * *
 * ALOG1Ø *
 * *

--- REAL NUMBER LOGARITHM ---

 * *
 * ALOG1Ø *
 * *

PURPOSE: To compute the logarithm of a real number.

CALL FORMAT: Function-value = ALOG(arg) or ALOG1Ø(arg)

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

DESCRIPTION: Function-value = Ln(arg); for ALOG
 = Log(1Ø)(arg); for ALOG1Ø

```
*****  
*          *  
* ANINT *  --- ROUND REAL NUMBER TO NEAREST WHOLE --- * ANINT *  
*          *  
*****
```

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))

```

*****
*      *
*  ATAN  *      --- ARCTANGENT OF REAL NUMBER ---
*      *
*****
*****
*      *
*  ATAN  *
*      *
*****

```

PURPOSE: To compute the arctangent of a real number
or of the ratio of two real numbers.

CALL FORMAT: Function-value = ATAN(arg1) or ATAN2(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)

```

*****
*      *
*  CABS *      --- COMPLEX NUMBER ABSOLUTE VALUE ---
*      *
*****

```

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 = $\text{SQRT} (R(\text{arg})^2 + I(\text{arg})^2)$

 * *
 * CDIV *
 * *

--- COMPLEX/COMPLEX DIVIDE ---

 * *
 * CDIV *
 * *

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 = $\{R(\text{arg2})+I(\text{arg2})\}/\{R(\text{arg1})+I(\text{arg1})\}$

 * *
 * CDIVRC *
 * *

— COMPLEX/REAL DIVIDE —

 * *
 * CDIVRC *
 * *

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 = $R(\arg(2)) + I(\arg(2)) / \arg(1)$

```

*****
*      *
*  CLOG *      --- COMPLEX NUMBER LOGARITHM ---
*      *
*****
*****
*      *
*  CLOG *
*      *
*****

```

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))

* *
* CONJG *
* *

— CONJUGATE OF COMPLEX NUMBER —

* *
* CONJG *
* *

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 = $R(\text{arg}) - I(\text{arg})$

* * *

* 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)

 * *
 * 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(arg1)+I(arg1)}**arg2

```
*****  
* *  
* CPOWRC *      --- REAL TO COMPLEX POWER ---  
* *  
*****
```

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 = $\text{arg1}^{**}(\text{R}(\text{arg2})+\text{I}(\text{arg2}))$

```

*****
*          *
* CSQRT *   — SQUARE ROOT OF COMPLEX NUMBER —
*          *
*****

```

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 $R(\text{arg}) \geq 0$ $R(\text{function value}) = F$
 $I(\text{function value}) = I(\text{arg})/(2*F)$
 if $R(\text{arg}) < 0$ $R(\text{function value}) = I(\text{arg})/(2*F)$
 $I(\text{function value}) = \text{SIGN}(I(\text{arg}))*F$

where $F = \text{SQRT}((\text{ABS}(R(\text{arg}))+\text{CABS}(\text{arg}))/2)$

```

*****
*           *
*  EXP      *      --- EXPONENTIAL OF REAL NUMBER ---
*           *
*           *
*****

```

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 > 709.089 traps with an overflow error condition.

```

*****
*      *
*  IDIM *  --- INTEGER/INTEGER POSITIVE DIFFERENCE ---
*      *
*****

```

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)

```
*****  
*      *  
* IPOW *  
*      *  
*****
```

--- INTEGER TO INTEGER POWER ---

```
*****  
*      *  
* IPOW *  
*      *  
*****
```

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

```

*****
*      *
* MOD *  --- INTEGER/INTEGER DIVIDE REMAINDER --- * MOD *
*      *
*****

```

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

```

*****
*          *
*   RAN   *   --- SCALAR RANDOM NUMBER GENERATOR ---
*          *
*****
*****
*          *
*   RAN   *
*          *
*****

```

PURPOSE: To generate one pseudo-random number.

CALL FORMAT: Function-value = RAN(SEED)

PARAMETERS: Function-value = Floating-point output scalar
Output random number
SEED = Integer input/output scalar
Input: random number seed
Output: last integer generated

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 = RAN(SEED)
and
CALL VRAND(SEED,C,1,1)
are equivalent.

EXAMPLE:

```

SEED = 1000

RAN(SEED): 0.8004849404096603
SEED      : 53719635

```

```

*****
*           *
*  RPOW   *
*           *
*****

```

--- REAL TO REAL POWER ---

```

*****
*           *
*  RPOW   *
*           *
*****

```

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.)

* *
* RRCP *
* *

--- REAL RECIPROCAL ---

* *
* RRCP *
* *

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

```

*****
*           *
*  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 *
 * *

--- REAL SINE AND COSINE ---

 * *
 * 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 = CMLX(COS(A),SIN(A)).

This is convenient for converting polar coordinates to rectangular coordinates.

EXAMPLE:

A = 0.0

CA = 1.0

SA = 0.0

```
*****  
*      *  
*  SQRT *  
*      *  
*****
```

— SQUARE ROOT OF REAL NUMBER —

```
*****  
*      *  
*  SQRT *  
*      *  
*****
```

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)

```
*****  
*      *  
*  TANH  *   --- REAL NUMBER HYPERBOLIC TANGENT ---   *  TANH  *  
*      *  
*****
```

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:

N	a scalar, is the dimension of V.
NS	a scalar, is the number of nonzero values in V.
S	a vector of length NS, contains the nonzero values of V.
IEN	a vector of length NS, contains the location in V of each corresponding element in S [i.e., $V(IEN(k)) = S(k)$ for $k=1, NS$].

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:

N:	8
NS:	3
S:	[3.2 7.8 -19.3]
IEN:	[2 4 8]

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 <u>row</u> 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 N+1, contains one element for every <u>column</u> in A. Each element indicates the location in S that holds that column's first nonzero value (exception: empty columns). IP's N+1st 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:	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

as expressed in Type I Format:

M:	5
N:	6
NS:	8

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., $IN(k) = \text{row in A of } S(k)$ 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:

0.0	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	0.0	0.0	0.0
0.0	9.9	7.1	5.8	0.0	0.0	0.0	0.0
0.0	1.3	0.0	8.3	0.0	0.0	0.0	0.0

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

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 FØ3 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 " \neq ".

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:

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

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

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,

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

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:

$$M(i,j) = L(i,j) / L(j,j) \quad \text{eq(2a)}$$

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

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 later 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:

```

      SUBROUTINE FILLIN(N, A, IA)
C
C   GIVEN AN N BY N MATRIX, A, THIS ROUTINE RETURNS AN N BY
C   N LOGICAL MATRIX, IA, WHERE IA(I,J) IS TRUE IF A(I,J) IS
C   A SPARSE ELEMENT AND FALSE OTHERWISE
C
      REAL A(N,N)
      LOGICAL IA(N,N)
C
      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
C
      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. IA(K,J)) GO TO 130
120  CONTINUE
                  IA(I,J) = .FALSE.
                  GO TO 140
130  CONTINUE
                  IA(I,J) = .TRUE.
140  CONTINUE
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(2*k-1) and S(2*k), then IDP(j) = k.

Consider the following example; let A be the real matrix.

2.0	0.0	0.0	4.0	0.0
0.0	1.0	0.0	0.0	2.0
0.0	0.0	-1.0	0.0	0.0
0.0	3.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	5.0

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:

WORD	S	IRN	ICP	IDP
1	2.0	1	1	1
2	1.0	2	2	2
3	3.0	4	4	4
4	-1.0	3	5	6
5	4.0	1	7	9
6	1.0	4	10	
7	2.0	2		
8	0.0	4		
9	5.0	5		

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.)

	2.0	0.0	0.0	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	1.0	0.0	0.0	2.0
L =	0.0	0.0	-1.0	0.0	0.0	U =	0.0	0.0	1.0	0.0	0.0
	0.0	3.0	0.0	1.0	0.0		0.0	0.0	0.0	1.0	-6.0
	0.0	0.0	0.0	0.0	5.0		0.0	0.0	0.0	0.0	1.0

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

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	3.0	0.0	1.0	-6.0
0.0	0.0	0.0	0.0	0.2

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 copys) 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 APPTN64 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

$$\begin{aligned} CR &= SC*CA+CSIN*CB \\ \emptyset &= CSIN'*CA+ SC*CB. \end{aligned}$$

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 |Re CX(I)|+|Im CX(I)| is largest of the N values |Re CX(m)|+|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).

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

If PARAM(1) = 1.0 then

$$\begin{aligned} \text{SX}(m) &:= \text{PARAM}(2)*\text{SX}(m) + \text{SY}(m) \\ \text{SY}(m) &:= -\text{SX}(m) + \text{PARAM}(5)*\text{SY}(m), \end{aligned}$$

for the N vector elements indexed by m.

If PARAM(1) = 0.0 then

$$\begin{aligned} \text{SX}(m) &:= \text{SX}(m) + \text{PARAM}(4)*\text{SY}(m) \\ \text{SY}(m) &:= \text{PARAM}(3)*\text{SY}(m) + \text{SY}(m), \end{aligned}$$

for the N vector elements indexed by m.

If PARAM(1) = -1.0 then

$$\begin{aligned} \text{SX}(m) &:= \text{PARAM}(2)*\text{SX}(m) + \text{PARAM}(4)*\text{SY}(m) \\ \text{SY}(m) &:= \text{PARAM}(3)*\text{SY}(m) + \text{PARAM}(5)*\text{SY}(m), \end{aligned}$$

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 $|D1*B1*B1| > |D2*B2*B2|$ and $D2*B2 \neq 0$ then

$$\begin{aligned} \text{PARAM}(1) &:= 0.0 \\ \text{PARAM}(3,4) &:= -B2/B1, D2*B2 / D1*B1, \text{ so that the SROTM matrix} \\ &\text{becomes } (H11,H21,H12,H22) = (1,-B2/B1,D2*B2/D1*B1,1). \end{aligned}$$

$$D1 := D1/U$$

$$D2 := D2/U$$

$$B1 := B1*U \text{ where } U = 1.0 + (D1*B1*B1)/(D2*B2*B2).$$

If $|D1*B1*B1| \leq |D2*B2*B2|$ and $D2*B2 \neq 0$ then

$$\begin{aligned} \text{PARAM}(1) &:= 1.0 \\ \text{PARAM}(2,5) &:= D1*B1/(D2*B2), B1/B2 \text{ so that the SROTM matrix} \\ &\text{becomes } (H11,H21,H12,H22) = (D1*B1/D2*B2,-1,1,B1/B2). \end{aligned}$$

$$D1,D2,B1 := D2/U,D1/U,B2*U \text{ where } U = 1 + D1*B1*B1/(D2*B2*B2).$$

If $D2*B2 = 0$, then

the rotation matrix in SROTM becomes the identity, 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 $Z(j) = 0.0$ 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)

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

$$\begin{aligned} \text{where: } ix &= (j-1) * IXO + (i-1) * IXI + 1 \\ iy &= (j-1) * IYO + (i-1) * IYI + 1 \\ jz &= (j-1) * IZO + 1 \end{aligned}$$

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

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

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

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

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

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)

$$Y(iy) = s * B(ja) * Z(ix) + Y(iy), \quad i=1,M \quad j=1,N$$

$$\begin{aligned} \text{where: } ja &= (j-1) * IAO + 1 \\ ix &= (j-1) * IXO + (i-1) * IXI + 1 \\ iy &= (j-1) * IYO + (i-1) * IYI + 1 \end{aligned}$$

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

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

$$\begin{aligned} Z &= X, \text{ if ISW}[3] = 0 \\ &= \text{Conjg}(X), \text{ if ISW}[3] = 1 \end{aligned}$$

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.

```

      DO 50 J=1,N
50   CALL SCOPY(N,0.E0,0,A(1,J),1)
      CALL SCOPY(N,1.E0,0,A,MDA+1)
      DO 60 J=1,N
60   CALL SCOPY(N,A(1,J),1,B(1,J),1)

```

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).

```

      DO 70 J=1,N
      L=IP(J)
      IF(J.NE.L) CALL SSWAP(M,C(1,J),1,C(1,L),1)
70   CONTINUE

```

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(*,*).

```

      IF(N.EQ.1) GOTO 85
      DO 80 J=1,N-1
80   CALL SSWAP(N-J,A(J,J+1),MDA,A(J+1,J),1)
85   CONTINUE

```

D.4.8 Column Vector Circular Shift

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

(x₁, ..., x_K, x_{K+1}, ..., 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 x 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
C
      IF(NI.LE.1) RETURN
      JINV=1
      NOP=NO+1
C
      DO 100 I=1,NI-1
        AINV=1./A(JINV)
        JC=JINV+NO
C
      C      COMPUTE THE NEXT COLUMN OF L
C
        CALL VSMUL(A(JC),NO,AINV,A(JC),NO,NI-I)
        MN=NI-I
C
      C      PERFORM THE ELIMINATION GETTING A NEW LOWER RIGHT MINOR
C
        CALL SAXPYN(1,MN,MN,A(JC),NO,A(JINV+1),1,0,A(JC+1),1,NO)
C
        JINV=JINV+NOP
100  CONTINUE
C
      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 x 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) $(x(1), x(2), \dots)$, 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 $x(i) \leq x \leq x(i+1)$.

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:

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

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:

XBRK(1,1) = X1	YBRK(1,1) = Y1
(2,1) = X2	(2,1) = Y2
(3,1) = X3	(3,1) = Y3
(1,2) = 1.0/(X2-X1)	(4,1) = Y4
(2,2) = 1.0/(X3-X2)	(1,2) = 1.0/(Y2-Y1)
(3,2) = 0.0	(2,2) = 1.0/(Y3-Y2)
	(3,2) = 1.0/(Y4-Y3)
X1 < X2 < X3	(4,2) = 0.0
	Y1 < Y2 < Y3 < Y4

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

Function value breakpoint table:

FBRK(1,1,1) = F1(X1,Y1)

(2,1,1) = F1(X2,Y1)

(3,1,1) = F1(X3,Y1)

(1,2,1) = F1(X1,Y2)

(2,2,1) = F1(X2,Y2)

(3,2,1) = F1(X3,Y2)

(1,3,1) = F1(X1,Y3)

(2,3,1) = F1(X2,Y3)

(3,3,1) = F1(X3,Y3)

(1,4,1) = F1(X1,Y4)

(2,4,1) = F1(X2,Y4)

(3,4,1) = F1(X3,Y4)

(1,1,2) = F2(X1,Y1)

.

.

(3,4,2) = F2(X3,Y4)

(1,1,3) = F3(X1,Y1)

.

.

(3,4,3) = F3(X3,Y4)

-6747-

Figure E-1 Example Coordinate and Function Value Breakpoint Tables

where

XY(1,1) = X coordinate value of the first input point

XY(2,1) = Y coordinate value of the first input point

. . .

E.3 CALLING APFN64 FUNCTION GENERATION ROUTINES

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

APFN64	<----->	Advanced Math
driver		Library routines

The user must supply the APFN64 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 APFN64 driver. The APFN64 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)
(0056)      WBRK(1,1)=-25.0
(0057)      WBRK(2,1)=-15.0
(0058)      WBRK(3,1)=0.0
(0059)      WBRK(1,2)=1.0/(WBRK(2,1)-WBRK(1,1))
(0060)      WBRK(2,2)=1.0/(WBRK(3,1)-WBRK(2,1))
(0061)      WBRK(3,2)=0.0
(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)+WBRK(I4,1)*YBRK(I2,1)+ZBRK(I3,1)
(0072)      FBRK(I1,I2,I3,I4,4)=XBRK(I1,1)*ZBRK(I3,1)+WBRK(I4,1)*YBRK(I2,1)
(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)      W(5)=-7.5
(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(1,I1))

```

APPENDIX F

SIMULATION LIBRARY ROUTINES

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

- RKGTKF 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)
C
C   ***  DFUN  ***  SAMPLE APFTN64 ROUTINE  ***
C
      DIMENSION Y(N), F(N)
C
      DO 1Ø I=1,N
         F(I)=Y(I)
1Ø CONTINUE
C
C   CORRESPONDS TO SOLUTIONS OF FORM:
C
C    $Y(I) = YØ * EXP(T)$ 
C
      RETURN
      END
```

APPENDIX G

LIST OF SUPERSEDED ROUTINES

F00 RELEASEOLD ROUTINES

FMMM32
 MMUL32
 ZVABS, VABS
 ZVADD, VADD
 ZVFLT, VFLOAT
 ZVIFIX, VIFIX
 ZVMSA, VMSA
 ZVMUL, VMUL
 ZVNEG, VNEG
 ZVRVRS, VRVRS
 ZVSADD, VSADD
 ZVSMA, VSMA
 ZVSMSA, VSMSA
 ZVSMSB, VSMSB
 ZVSMUL, VSMUL
 ZVSQ, VSQ
 ZVSUB, VSUB
 ZVSWAP, VSWAP

NEW ROUTINES

FMMM or FMMMV
 MMUL, FMMM, or FMMMV
 VABS
 VADD
 VFLOAT
 VIFIX
 VMSA
 VMUL
 VNEG
 VRVRS
 VSADD
 VSMA
 VSMSA
 VSMSB
 VSMUL
 VSQ
 VSUB
 VSWAP

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

F03 RELEASEOLD ROUTINES

AIMAG
 CSFR
 CSFS
 CSSV
 CUFR
 CUFS
 CUSV
 EXTRU
 FLOAT
 IFIX
 INSERT
 LOC
 RSFR
 RSFS
 RSSV

NEW ROUTINES

AIMAG (APFTN64 intrinsic)
 CSFR2
 CSFS2
 CSSV2
 CUFR2
 CUFS2
 CUSV2
 EXTRACT (APFTN64 intrinsic)
 FLOAT (APFTN64 intrinsic)
 IFIX (APFTN64 intrinsic)
 INSERT (APFTN64 intrinsic)
 LOC (APFTN64 intrinsic)
 RSFR2
 RSFS2
 RSSV2

APPENDIX H

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.

<u>INTERNAL SUBROUTINE</u>	<u>CALLING ROUTINE(S)</u>
ADV2	CFFT, CFFTB, CFFTI, XCFFT
ADV4	CFFT, CFFTB, CFFTI, XCFFT
ALTINP	CCEPS
BITREV	CFFT, CFFTI
CBEAJY	CBEJYH
CBEDH	RRGTF
CBEDJ	RRGTF
CBERHY	CBEJYH
CBERJS	CBEJYH
CBERYH	CBEJYH
CLSTAT	CFFT, CFFTB, HAMM, REALTR, STSTAT, BLKMAN, HANN, CFFTI, IIRELT, IREALT, XCFFT
CTOR	RFFT2D
ENTVAR	SIMPLE
FFT2	CFFT, CFFTB, FFT2B, CFFTI, XCFFT
FFT2B	CFFTB
FFT4	CFFT, CFFTB, FFT4B, CFFTI
FFT4B	CFFTB
IFFT4	CFFTI
IIRELT	RFTII
INTEG	CBEJYH
IREALT	RFFTI, RFTII, IIRELT
LPSPFI	SIMPLE
PHAUNW	CCEPS
PHCHECK	PHAUNW
REALTR	RFFT, RFFTB, RFFTI, RFTII
RRGTF	INTEG
RTOC	RFFT2D
SET24B	FFT2B, FFT4B
SPCVAL	PHAUNW
STSTAT	CFFT, CFFTB, HAMM, REALTR, STSTAT, BLKMAN,

APPENDIX I

APMATH64 ROUTINES IN PAGE ORDER AND BY TYPE

BASIC MATH LIBRARY (VOLUME 1)

CCMMUL	COMPLEX MATRIX MULTIPLY	A - 2
CDET	COMPLEX MATRIX DETERMINANT	A - 4
CDOTPR	COMPLEX DOT PRODUCT	A - 6
CFFT	COMPLEX-TO-COMPLEX FFT (IN PLACE)	A - 7
CFFTB	COMPLEX-TO-COMPLEX FFT (NOT IN PLACE)	A - 8
CFFTM	MIXED-RADIX COMPLEX FFT (NOT-IN-PLACE)	A - 9
CFFTSC	COMPLEX FFT SCALE	A - 11
CGMMUL	COMPLEX GENERAL MATRIX MULTIPLY	A - 12
CMATIN	COMPLEX MATRIX INVERSE	A - 14
CMDET	COMPLEX MATRIX DETERMINANT	A - 15
CMFACT	COMPLEX MATRIX L/U FACTORIZATION	A - 17
CMMTRC	COMPLEX MATRIX MULTIPLY TRACE	A - 19
CMMUL	COMPLEX MATRIX MULTIPLY	A - 20
CMSOLV	COMPLEX MATRIX EQUATION SOLVER	A - 21
CMTRAC	COMPLEX SUB-MATRIX TRACE	A - 23
CMTRAN	COMPLEX SUB-MATRIX TRANSPOSE	A - 24
CMVML3	COMPLEX 3X3 MATRIX MULT. 3D VECTORS	A - 25
CMVML4	COMPLEX 4X4 MATRIX MULT. 4D VECTORS	A - 26
CONV	CONVOLUTION (CORRELATION)	A - 27
CRMMUL	COMPLEX-REAL MATRIX MULTIPLY	A - 28
CROSSP	COMPLEX 3D CROSS PRODUCT	A - 30
CRVADD	COMPLEX AND REAL VECTOR ADD	A - 31
CRVDIV	COMPLEX AND REAL VECTOR DIVIDE	A - 32
CRVMUL	COMPLEX AND REAL VECTOR MULTIPLY	A - 33
CRVSUB	COMPLEX AND REAL VECTOR SUBTRACT	A - 34
CSOLV	COMPLEX SYSTEM SOLVER	A - 35
CSOLVQ	COMPLEX MATRIX EQUATION SOLVER	A - 36
CTRN2	2-D COORDINATE TRANSFORM	A - 39
CTRN3	3-DIMENSIONAL COORDINATE TRANSFORMATION	A - 40
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

BASIC MATH LIBRARY (cont.)

RMSQV	ROOT-MEAN-SQUARE OF VECTOR ELEMENTS	A - 125
SCJMA	SELF-CONJUGATE MULTIPLY AND ADD	A - 126
SGEFA	REAL GENERAL MATRIX FACTOR	A - 127
SGESL	REAL GENERAL MATRIX SOLVE	A - 129
SGTSL	TRIDIAGONAL MATRIX SOLVER	A - 131
SMMM	SUBMATRIX MULTIPLY	A - 133
SMMMV	SUBMATRIX MULTIPLY	A - 135
SN2	SQUARED DISTANCE BETWEEN TWO VECTORS	A - 137
SOLVEQ	LINEAR EQUATION SOLVER	A - 138
STMM	SUBMATRIX TRANSPOSE & MULTIPLY	A - 140
SVE	SUM OF VECTOR ELEMENTS	A - 142
SVEMG	SUM OF VECTOR ELEMENT MAGNITUDES	A - 143
SVESQ	SUM OF VECTOR ELEMENT SQUARES	A - 144
SVS	SUM OF VECTOR SIGNED SQUARES	A - 145
TRIDIA	TRIDIAGONAL MATRIX SOLVER	A - 146
VAAM	VECTOR ADD, ADD, AND MULTIPLY	A - 147
VABS	VECTOR ABSOLUTE VALUE	A - 148
VACOS	VECTOR ARCCOSINE	A - 149
VADD	VECTOR ADD	A - 150
VAINT	VECTOR TRUNCATE	A - 151
VALG	VECTOR LOGARITHM	A - 152
VALG10	VECTOR BASE 10 LOGARITHM	A - 153
VAM	VECTOR ADD AND MULTIPLY	A - 154
VASIN	VECTOR ARCSINE	A - 155
VASM	VECTOR ADD AND SCALAR MULTIPLY	A - 156
VATAN	VECTOR ARCTANGENT	A - 157
VATAN2	VECTOR ARCTANGENT (2 ARGUMENTS)	A - 158
VCLIP	VECTOR CLIP	A - 159
VCLR	VECTOR CLEAR	A - 160
VCOS	VECTOR COSINE	A - 161
VCOSH	VECTOR COSINE (HYPERBOLIC)	A - 162
VDIV	VECTOR DIVIDE	A - 163
VEUCL2	VECTOR EUCLIDEAN DISTANCE	A - 164
VEXP	VECTOR EXPONENTIAL	A - 165
VEXP10	VECTOR EXPONENTIAL (BASE 10)	A - 166
VFILL	VECTOR FILL	A - 167
VFRAC	VECTOR TRUNCATE TO FRACTION	A - 168
VIABS	VECTOR ABSOLUTE VALUE	A - 169
VIADD	VECTOR INTEGER ADD	A - 170
VICLIP	VECTOR INVERTED CLIP	A - 171
VIDIV	VECTOR INTEGER DIVIDE	A - 172
VIMAG	EXTRACT IMAGINARIES OF COMPLEX VECTOR	A - 173
VIMUL	VECTOR INTEGER MULTIPLY	A - 174
VINDEX	VECTOR INDEX	A - 175
VINEG	VECTOR INTEGER NEGATE	A - 176
VISUB	VECTOR INTEGER SUBTRACT	A - 177
VLAND	VECTOR LOGICAL ADD	A - 178
VLEQV	VECTOR LOGICAL EQUIVALENCE	A - 179
VLM	VECTOR LIMIT	A - 180
VLMERG	LOGICAL VECTOR MERGE	A - 181

 BASIC MATH LIBRARY (cont.)

VTSADD	VECTOR TM SCALAR ADD	A - 230
VTSMA	VECTOR TM SCALAR MULTIPLY AND ADD	A - 231
VTSMUL	VECTOR TM SCALAR MULTIPLY	A - 232

 ADVANCED MATH LIBRARY (VOLUME 2)

CH	COMPLEX HERMITIAN EIGENSYSTEM SOLVER	A - 234
EIGRS	REAL SYMMETRIC EIGENSYSTEM SOLVER	A - 237
HTRIBK	COMPLEX HERMITIAN EIGENVECTORS	A - 239
HTRIDI	COMPLEX HERMITIAN TRIDIAGONALIZATION	A - 241
IMTQL1	DIAGONALIZE TRIDIAGONAL MATRIX	A - 243
IMTQL2	DIAGONALIZE A TRIDIAGONAL MATRIX	A - 245
RS	REAL SYMMETRIC EIGENSYSTEM SOLVER	A - 247
SIMPLE	REVISED SIMPLEX	A - 249
SKYSOL	SKYLINE FORMAT EQUATION SOLVER	A - 254
TRED1	TRIDIAGONALIZE SYMMETRIC MATRIX	A - 256
TRED2	TRIDIAGONALIZE A SYMMETRIC MATRIX	A - 258
VASORT	VECTOR SORT ALGEBRAIC VALUES	A - 259
VISORT	VECTOR SORT INTEGER VALUES	A - 260
VSORT	VECTOR SORT WITH INDICES	A - 261

 SIGNAL PROCESSING LIBRARY

ACORF	AUTO-CORRELATION (FREQUENCY-DOMAIN)	A - 264
ACORT	AUTO-CORRELATION (TIME-DOMAIN)	A - 265
ASPEC	ACCUMULATING AUTO-SPECTRUM	A - 266
BLKMAN	BLACKMAN WINDOW MULTIPLY	A - 267
CCEPS	PHASE UNWRAP AND COMPLEX CEPSTRUM	A - 268
CCORF	CROSS-CORRELATION (FREQUENCY-DOMAIN)	A - 272
CCORT	CROSS-CORRELATION (TIME-DOMAIN)	A - 273
CFFTI	COMPLEX FFT WITH INTERPOLATION	A - 274
COHER	COHERENCE FUNCTION	A - 275
CSPEC	ACCUMULATING CROSS-SPECTRUM	A - 276
DECFIR	DECIMATION	A - 277
ENVEL	ENVELOPE DETECTOR	A - 279
HAMM	HAMMING WINDOW MULTIPLY	A - 280
HANN	HANNING WINDOW MULTIPLY	A - 281
HIST	HISTOGRAM	A - 282
HLBRT	HILBERT TRANSFORMER	A - 283
LPAUTO	LINEAR PREDICTION AUTOCORRELATION	A - 284
PKVAL	PEAK AND VALLEY PICKING	A - 286
RDFT	REAL DISCRETE FOURIER TRANSFORM	A - 288
RFFTI	REAL FFT WITH INTERPOLATION	A - 290

 LINPACK BLAS LIBRARY (cont.)

SROT	PLANE ROTATION	A - 355
SROTG	GIVENS PLANE ROTATION	A - 356
SROTM	MODIFIED GIVENS ROTATIONS	A - 357
SROTMG	MODIFIED GIVENS PLANE ROTATION SETUP	A - 358
SSCAL	REAL SCALAR TIMES VECTOR	A - 360
SSWAP	INTERCHANGES VECTORS	A - 361

 SIMULATION LIBRARY

ABP1	ADAMS-BASHFORTH PREDICTOR (ORDER 1)	A - 363
ABP2	ADAMS-BASHFORTH PREDICTOR (ORDER 2)	A - 364
ABP3	ADAMS-BASHFORTH PREDICTOR (ORDER 3)	A - 365
ABP4	ADAMS-BASHFORTH PREDICTOR (ORDER 4)	A - 366
ADAMS4	ADAMS VARIABLE STEP INTEG.(ORD 4)	A - 368
AMC1	ADAMS-MOULTON CORRECTOR (ORDER 1)	A - 371
AMC2	ADAMS-MOULTON CORRECTOR (ORDER 2)	A - 372
AMC3	ADAMS-MOULTON CORRECTOR (ORDER 3)	A - 373
AMC4	ADAMS-MOULTON CORRECTOR (ORDER 4)	A - 374
BIN	BINARY SEARCH	A - 376
CBEIK	COMPLEX BESSEL I AND K	A - 378
CBEJYH	COMPLEX BESSEL J, Y, AND H	A - 381
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
RKGIL	RUNGE-KUTTA-GILL INTEGRATION	A - 392
RKGTF	R-K-GILL-THOMPSON INTEG. (ORDER 4)	A - 395
ROT3	3D ROTATION MATRIX, 3-ANGLE	A - 397
SCS1	SCALAR COS/SIN, TM INTERP.(ORD 1)	A - 399
STEP	STEP SEARCH	A - 400

 GEOPHYSICAL LIBRARY

CONNMO	NMO WITH CONSTANT VELOCITY	A - 403
IIR30	RECURSIVE FILTER	A - 405
KSMLV	K-TH SMALLEST ELEMENT IN VECTOR	A - 407
NMOLI	NMO LINEAR INTERPOLATION	A - 408
NMOQI	NMO QUADRATIC INTERPOLATION	A - 410
POST64	POST BITS TO RASTER	A - 412
RESNMO	RESIDUAL NORMAL MOVEOUT	A - 413
TMCONV	CONVOLUTION (CORRELATION)	A - 415
V01	VECTOR ZERO TRENDS	A - 417
VARNMO	NMO WITH VARIABLE VELOCITY	A - 418
VRNAV	VECTOR RUNNING AVERAGE	A - 420
VSCAN0	VECTOR SCAN FOR ZEROS	A - 421

TABLE MEMORY LIBRARY (cont.)

TTTSUB	VECTOR SUBTRACT (TM-TM TO TM)	A - 500
TTVLC2	VECTOR LINEAR COMBINATION	A - 501
TVCLR	TABLE MEMORY VECTOR CLEAR	A - 502

SPECIAL UTILITIES LIBRARY

EXTRS	EXTRACT A SIGNED BIT-FIELD	A - 504
PEEK	MEMORY FETCH	A - 505
POKE	STORE INTO MEMORY	A - 506

DATA FORMATTING LIBRARY

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

DOUBLE PRECISION LIBRARY

DADD	DOUBLE TO DOUBLE-PRECISION ADD	A - 529
DADDRR	SINGLE TO DOUBLE-PRECISION ADD	A - 530
DADOT	DOUBLE ACCUMULATE DOT PRODUCT	A - 531
DDOTRR	DOUBLE DOT PRODUCT REAL REAL	A - 532
DMUL	DOUBLE TO DOUBLE-PRECISION MULTIPLY	A - 533
DMULRR	SINGLE TO DOUBLE PRECISION MULTIPLY	A - 534
DNEG	NEGATE DOUBLE-PRECISION NUMBER	A - 535
DSUB	DOUBLE TO DOUBLE-PRECISION SUBTRACT	A - 536
DSUBRR	SINGLE TO DOUBLE-PRECISION SUBTRACT	A - 537

APPENDIX J

APMATH64 ROUTINES IN ALPHABETICAL ORDER

NAME	DESCRIPTION	PAGE
ABP1	ADAMS-BASHFORTH PREDICTOR (ORDER 1)	A - 363
ABP2	ADAMS-BASHFORTH PREDICTOR (ORDER 2)	A - 364
ABP3	ADAMS-BASHFORTH PREDICTOR (ORDER 3)	A - 365
ABP4	ADAMS-BASHFORTH PREDICTOR (ORDER 4)	A - 366
ABS	REAL NUMBER ABSOLUTE VALUE	A - 539
ACORF	AUTO-CORRELATION (FREQUENCY-DOMAIN)	A - 264
ACORT	AUTO-CORRELATION (TIME-DOMAIN)	A - 265
ACOS	REAL NUMBER ARCCOSINE	A - 548
ADAMS4	ADAMS VARIABLE STEP INTEG.(ORD 4)	A - 368
AINT	TRUNCATE REAL NUMBER	A - 541
ALOG10	REAL NUMBER LOGARITHM	A - 543
ALOG	REAL NUMBER LOGARITHM	A - 542
AMC1	ADAMS-MOULTON CORRECTOR (ORDER 1)	A - 371
AMC2	ADAMS-MOULTON CORRECTOR (ORDER 2)	A - 372
AMC3	ADAMS-MOULTON CORRECTOR (ORDER 3)	A - 373
AMC4	ADAMS-MOULTON CORRECTOR (ORDER 4)	A - 374
AMOD	REAL/REAL DIVIDE REMAINDER	A - 544
ANINT	ROUND REAL NUMBER TO NEAREST WHOLE	A - 545
ASIN	REAL NUMBER ARCSINE	A - 546
ASPEC	ACCUMULATING AUTO-SPECTRUM	A - 266
ATAN2	ARCTANGENT OF RATIO OF REAL NUMBERS	A - 548
ATAN	ARCTANGENT OF REAL NUMBER	A - 547
BIN	BINARY SEARCH	A - 376
BLKMAN	BLACKMAN WINDOW MULTIPLY	A - 267
CABS	COMPLEX NUMBER ABSOLUTE VALUE	A - 549
CAXPY	COMPLEX $A * X + Y$	A - 326
CAXPYN	NESTED COMPLEX $A * X + Y$	A - 327
CBEIK	COMPLEX BESSEL I AND K	A - 378
CBEJYH	COMPLEX BESSEL J, Y, AND H	A - 381
CCEPS	PHASE UNWRAP AND COMPLEX CEPSTRUM	A - 268
CCMMUL	COMPLEX MATRIX MULTIPLY	A - 2
CCOPY	COMPLEX VECTOR COPY	A - 338
CCORF	CROSS-CORRELATION (FREQUENCY-DOMAIN)	A - 272
CCORT	CROSS-CORRELATION (TIME-DOMAIN)	A - 273
CCOS	COMPLEX NUMBER COSINE	A - 558
CDET	COMPLEX MATRIX DETERMINANT	A - 4
CDIV	COMPLEX/COMPLEX DIVIDE	A - 551
CDIVCR	REAL/COMPLEX DIVIDE	A - 552
CDIVRC	COMPLEX/REAL DIVIDE	A - 553
CDOTC	COMPLEX INNER PRODUCT	A - 331
CDOTN	NESTED COMPLEX DOT PRODUCT	A - 332
CDOTPR	COMPLEX DOT PRODUCT	A - 6
CDOTU	COMPLEX DOT PRODUCT	A - 335

APPENDIX J

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
CVSMA	COMPLEX VECTOR SCALAR MULTIPLY AND ADD	A - 57
CVSMUL	COMPLEX VECTOR SCALAR MULTIPLY	A - 58
CVSUB	COMPLEX VECTOR SUBTRACT	A - 59
DADD	DOUBLE TO DOUBLE-PRECISION ADD	A - 529
DADDRR	SINGLE TO DOUBLE-PRECISION ADD	A - 530
DADOT	DOUBLE ACCUMULATE DOT PRODUCT	A - 531
DDOTRR	DOUBLE DOT PRODUCT REAL REAL	A - 532
DEC FIR	DECIMATION	A - 277
DEQ22	DIFFERENCE EQUATION, 2 POLES, 2 ZEROS	A - 60
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
ENVEL	ENVELOPE DETECTOR	A - 279
EXP	EXPONENTIAL OF REAL NUMBER	A - 567
EXTRS	EXTRACT A SIGNED BIT-FIELD	A - 504
FMMM	FAST MATRIX MULTIPLY	A - 63
FMMMV	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
HTRIBK	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 - 506
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
RFFTB	REAL-TO-COMPLEX FFT (NOT IN PLACE)	A - 117
RFFTI	REAL FFT WITH INTERPOLATION	A - 290
RFFTM	MIXED-RADIX REAL FFT (NOT-IN-PLACE)	A - 119
RFFTSC	REAL FFT SCALE AND FORMAT	A - 121
RFTII	REAL FFT WITH QUARTER INTERPOLATION	A - 291
RGMMUL	REAL GENERAL MATRIX MULTIPLY	A - 123
RKGIL	RUNGE-KUTTA-GILL INTEGRATION	A - 392
RKGTF	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
RPOWRI	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
SCS1	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

APPENDIX J

TTTSUB	VECTOR SUBTRACT (TM-TM TO TM)	A - 500
TTVLC2	VECTOR LINEAR COMBINATION	A - 501
TVCLR	TABLE MEMORY VECTOR CLEAR	A - 502
VØ1	VECTOR ZERO TRENDS	A - 417
VAAM	VECTOR ADD, ADD, AND MULTIPLY	A - 147
VABS	VECTOR ABSOLUTE VALUE	A - 148
VACOS	VECTOR ARCCOSINE	A - 149
VADD	VECTOR ADD	A - 150
VAINT	VECTOR TRUNCATE	A - 151
VALG1Ø	VECTOR BASE 1Ø LOGARITHM	A - 153
VALG	VECTOR LOGARITHM	A - 152
VAM	VECTOR ADD AND MULTIPLY	A - 154
VARNMO	NMO WITH VARIABLE VELOCITY	A - 418
VASIN	VECTOR ARCSINE	A - 155
VASM	VECTOR ADD AND SCALAR MULTIPLY	A - 156
VASORT	VECTOR SORT ALGEBRAIC VALUES	A - 259
VATAN2	VECTOR ARCTANGENT (2 ARGUMENTS)	A - 158
VATAN	VECTOR ARCTANGENT	A - 157
VAVEXP	VECTOR EXPONENTIAL AVERAGING	A - 296
VAVLIN	VECTOR LINEAR AVERAGING	A - 297
VCLIP	VECTOR CLIP	A - 159
VCLR	VECTOR CLEAR	A - 160
VCOS	VECTOR COSINE	A - 161
VCOSH	VECTOR COSINE (HYPERBOLIC)	A - 162
VDBPWR	VECTOR CONVERSION TO DB (POWER)	A - 298
VDIV	VECTOR DIVIDE	A - 163
VEUCL2	VECTOR EUCLIDEAN DISTANCE	A - 164
VEXPLØ	VECTOR EXPONENTIAL (BASE 1Ø)	A - 166
VEXP	VECTOR EXPONENTIAL	A - 165
VFILL	VECTOR FILL	A - 167
VFLOAT	CONVERT INTEGER TO FLOATING-POINT	A - 508
VFRAC	VECTOR TRUNCATE TO FRACTION	A - 168
VIABS	VECTOR ABSOLUTE VALUE	A - 169
VIADD	VECTOR INTEGER ADD	A - 170
VICLIP	VECTOR INVERTED CLIP	A - 171
VIDIV	VECTOR INTEGER DIVIDE	A - 172
VIFIX	VECTOR INTEGER FIX	A - 509
VIMAG	EXTRACT IMAGINARIES OF COMPLEX VECTOR	A - 173
VIMUL	VECTOR INTEGER MULTIPLY	A - 174
VINDEX	VECTOR INDEX	A - 175
VINEG	VECTOR INTEGER NEGATE	A - 176
VISORT	VECTOR SORT INTEGER VALUES	A - 260
VISUB	VECTOR INTEGER SUBTRACT	A - 177
VLAND	VECTOR LOGICAL ADD	A - 178
VLEQV	VECTOR LOGICAL EQUIVALENCE	A - 179
VLIM	VECTOR LIMIT	A - 180
VLMERG	LOGICAL VECTOR MERGE	A - 181
VLNOT	VECTOR LOGICAL NOT	A - 182
VLOR	VECTOR LOGICAL OR	A - 183
VLXOR	VECTOR LOGICAL EXCLUSIVE OR	A - 184
VMA	VECTOR MULTIPLY AND ADD	A - 185
VMAX	VECTOR MAXIMUM	A - 186
VMAXMG	VECTOR MAXIMUM MAGNITUDE	A - 187
VMIN	VECTOR MINIMUM	A - 188

APPENDIX J

VT SMA	VECTOR TM SCALAR MULTIPLY AND ADD	A - 231
VT SMUL	VECTOR TM SCALAR MULTIPLY	A - 232
VUP16	VECTOR 16-BIT BYTE UNPACK	A - 520
VUP32	VECTOR 32-BIT BYTE UNPACK	A - 521
VUP8	VECTOR 8-BIT BYTE UNPACK	A - 519
VUPR32	VECTOR HALFWORD REAL UNPACK	A - 522
VUPS16	VECTOR 16-BIT SIGNED BYTE UNPACK	A - 524
VUPS32	VECTOR 32-BIT SIGNED BYTE UNPACK	A - 525
VUPS8	VECTOR 8-BIT SIGNED BYTE UNPACK	A - 523
VUSI32	VECTOR 32-BIT SIGNED INTEGER UNPACK	A - 526
VUUI32	VECTOR 32-BIT UNSIGNED UNPACK	A - 527
VXCS	VECTOR MULTIPLIED BY SIN AND COS	A - 299
WIENER	WIENER LEVINSON ALGORITHM	A - 301

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
	EXPONENTIAL BASE	1Ø...[VEXPLØ]VECTOR
[VALG1Ø]	VECTOR BASE	1Ø LOGARITHM
	[VPK16]VECTOR	16-BIT BYTE PACK
	[VUP16]VECTOR	16-BIT BYTE UNPACK
	[VUPS16]VECTOR	16-BIT SIGNED BYTE UNPACK
	[CSROT]COMPLEX	2-D ROTATION
[CFFT2D]	COMPLEX TO COMPLEX	2-DIMENSIONAL FFT
[RFFT2D]	REAL TO COMPLEX	2-DIMENSIONAL FFT
[ROT3]	3D ROTATION MATRIX,	3-ANGLE
	[VPK32]VECTOR	32-BIT BYTE PACK
	[VUP32]VECTOR	32-BIT BYTE UNPACK
	[VPKI32]VECTOR	32-BIT INTEGER PACK
	[VUPS32]VECTOR	32-BIT SIGNED BYTE UNPACK
	[VUSI32]VECTOR	32-BIT SIGNED INTEGER UNPACK
	[VUII32]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
	[IABS]INTEGER	ABSOLUTE VALUE
[ISAMAX]	INDEX OF MAXIMUM	ABSOLUTE VALUE
	[VABS]VECTOR	ABSOLUTE VALUE
	[VIABS]VECTOR	ABSOLUTE VALUE
	[DADOT]DOUBLE	ACCUMULATE DOT PRODUCT

APMATH64 KEY WORD INDEX

[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
[CAXPYN]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	COMPLEX FFT NOT-IN-PLACE
[PAS2F]RADIX-2 FORWARD	COMPLEX FFT PASS
[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
[CPOWRC]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
[CUFS2]SPARSE	COMPLEX UNSYM FACTOR & SOLVE
[CUFR2]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
[TCONV]POST-TAPERED	CONVOLUTION CORRELATION
[CTR2]2-D	COORDINATE TRANSFORM
[CTR3]3-DIMENSIONAL	COORDINATE TRANSFORMATION
[CCOPY]COMPLEX VECTOR	COPY

[DNEG]NEGATE DOUBLE-PRECISION NUMBER
 [DSUBRR]SINGLE TO DOUBLE-PRECISION SUBTRACT
 [DSUB]DOUBLE TO DOUBLE-PRECISION SUBTRACT
 [CH]COMPLEX HERMITIAN EIGENSYSTEM SOLVER
 [EIGRS]REAL SYMMETRIC EIGENSYSTEM SOLVER
 [RS]REAL SYMMETRIC EIGENSYSTEM SOLVER
 [HTRIBK]COMPLEX HERMITIAN EIGENVECTORS
 OF LARGEST COMPLEX ELEMENT...[ICAMAX]INDEX
 [KSMLV]K-TH SMALLEST ELEMENT IN VECTOR
 [MAXMGV]MAXIMUM MAGNITUDE ELEMENT IN VECTOR
 [MAXV]MAXIMUM ELEMENT IN VECTOR
 [MINMGV]MINIMUM MAGNITUDE ELEMENT IN VECTOR
 [MINV]MINIMUM ELEMENT IN VECTOR
 [MEAMGV]MEAN OF VECTOR ELEMENT MAGNITUDES
 [SVEMG]SUM OF VECTOR ELEMENT MAGNITUDES
 [MEASQV]MEAN OF VECTOR ELEMENT SQUARES
 [SVESQ]SUM OF VECTOR ELEMENT SQUARES
 [MEANV]MEAN VALUE OF VECTOR ELEMENTS
 OF VECTOR ELEMENTS...[RMSQV]ROOT-MEAN-SQUARE
 [SVE]SUM OF VECTOR ELEMENTS
 [VSUM]VECTOR SUM OF ELEMENTS INTEGRATION
 [LVEQ]LOGICAL VECTOR EQUAL
 VECTOR GREATER THAN OR EQUAL...[LVGE]LOGICAL
 [LVNE]LOGICAL VECTOR NOT EQUAL
 [CMSOLV]COMPLEX MATRIX EQUATION SOLVER
 [CSOLVQ]COMPLEX MATRIX EQUATION SOLVER
 [SKYSOL]SKYLINE FORMAT EQUATION SOLVER
 [SOLVEQ]LINEAR EQUATION SOLVER
 [DEQ22]DIFFERENCE EQUATION, 2 POLES, 2 ZEROS
 [VLEQV]VECTOR LOGICAL EQUIVALENCE
 [VEUCL2]VECTOR EUCLIDEAN DISTANCE
 [SCNRM2]COMPLEX EUCLIDEAN NORM
 [VPOLY]VECTOR POLYNOMIAL EVALUATION
 [VLXOR]VECTOR LOGICAL EXCLUSIVE OR
 [CVEXP]COMPLEX VECTOR EXPONENTIAL
 VECTOR MULTIPLY EXPONENTIAL...[CVMEXP]COMPLEX
 [VEXP]VECTOR EXPONENTIAL
 [VAVEXP]VECTOR EXPONENTIAL AVERAGING
 [VEXP10]VECTOR EXPONENTIAL BASE 10
 COMPLEX SYMMETRIC FACTOR...[CSFR2]SPARSE
 COMPLEX UNSYMMETRIC FACTOR...[CUFR2]SPARSE
 [RSFR2]SPARSE REAL SYMMETRIC FACTOR
 [RUF2]SPARSE REAL UNSYMMETRIC FACTOR
 [SGEFA]REAL GENERAL MATRIX FACTOR
 [CSFS2]SPARSE COMPLEX SYMM FACTOR & SOLVE
 [CUFS2]SPARSE COMPLEX UNSYM FACTOR & SOLVE
 [RSFS2]SPARSE REAL SYMM FACTOR & SOLVE
 [RUF2]SPARSE REAL UNSYM FACTOR & SOLVE
 [GENTAB]GENERATE TWIDDLE FACTOR TABLE
 [CMFACT]COMPLEX MATRIX L/U FACTORIZATION
 [LUF]LU MATRIX FACTORIZATION CROUT
 [PEEK]MEMORY FETCH
 TO COMPLEX 2-DIMENSIONAL FFT...[CFFT2D]COMPLEX
 TO COMPLEX 2-DIMENSIONAL FFT...[RFFT2D]REAL

APMATH64 KEY WORD INDEX

[GRAD2D]MAXIMUM	GRADIENT FILTER
[GRD2DB]MAXIMUM	GRADIENT FILTER WITH BOUND
[LVGT]LOGICAL VECTOR	GREATER THAN
[LVGE]LOGICAL VECTOR	GREATER THAN OR EQUAL
[VPKR32]VECTOR REAL	HALFWORD PACK
[VUPR32]VECTOR	HALFWORD REAL UNPACK
[CH]COMPLEX	HERMITIAN EIGENSYSTEM SOLVER
[HTRIBK]COMPLEX	HERMITIAN EIGENVECTORS
[HTRIDI]COMPLEX	HERMITIAN TRIDIAGONALIZATION
[VCOSH]VECTOR COSINE	HYPERBOLIC
[VSINH]VECTOR SINE	HYPERBOLIC
[VTANH]VECTOR TANGENT	HYPERBOLIC
[COSH]REAL NUMBER	HYPERBOLIC COSINE
[SINH]REAL NUMBER	HYPERBOLIC SINE
[TANH]REAL NUMBER	HYPERBOLIC TANGENT
[VIMAG]EXTRACT	IMAGINARIES OF COMPLEX VECTOR
[SCASUM]SUM OF REAL AND	IMAGINARY MAGNITUDES
[MTIMOV]VECTOR MOVE WITH	INCREMENT MD TO TM
[TMIMOV]VECTOR MOVE WITH	INCREMENT TM TO MD
[TTIMOV]VECTOR MOVE WITH	INCREMENT TM TO TM
[VINDE]VECTOR	INDEX
[VSORT]VECTOR SORT WITH	INDICES
[CDOTC]COMPLEX	INNER PRODUCT
[RKGTF]R-K-GILL-THOMPSON	INTEG. ORDER 4
[ADAMS4]ADAMS VARIABLE STEP	INTEG.ORD 4
REAL NUMBER TO NEAREST	INTEGER...[NINT]ROUND
[VIADD]VECTOR	INTEGER ADD
[VIDIV]VECTOR	INTEGER DIVIDE
[VIFIX]VECTOR	INTEGER FIX
[VIMUL]VECTOR	INTEGER MULTIPLY
[VINEG]VECTOR	INTEGER NEGATE
[VPKI32]VECTOR 32-BIT	INTEGER PACK
[CPOWCI]COMPLEX TO	INTEGER POWER
[IPOW]INTEGER TO	INTEGER POWER
[RPOWRI]REAL TO	INTEGER POWER
[VISUB]VECTOR	INTEGER SUBTRACT
[VFLOAT]CONVERT	INTEGER TO FLOATING-POINT
[VUSI32]VECTOR 32-BIT SIGNED	INTEGER UNPACK
[VISORT]VECTOR SORT	INTEGER VALUES
[RKGIL]RUNGE-KUTTA-GILL	INTEGRATION
SIMPSON'S 1/3 RULE	INTEGRATION...[VSIMPS]VECTOR
[VSUM]VECTOR SUM OF ELEMENTS	INTEGRATION
TRAPEZOIDAL RULE	INTEGRATION...[VTRAPZ]VECTOR
[SCS1]SCALAR COS/SIN, TM	INTERP.ORD 1
[CFFTI]COMPLEX FFT WITH	INTERPOLATION
[NMOLI]NMO LINEAR	INTERPOLATION
[NMOQI]NMO QUADRATIC	INTERPOLATION
[RFFTI]REAL FFT WITH	INTERPOLATION
[RFTII]REAL FFT WITH QUARTER	INTERPOLATION
[CMATIN]COMPLEX MATRIX	INVERSE
[MATINV]MATRIX	INVERSE
[VRECIP]VECTOR	INVERSE
[PAS2I]RADIX-2	INVERSE COMPLEX FFT PASS
[PAS3I]RADIX-3	INVERSE COMPLEX FFT PASS

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
[SMUPCK]SPARSE	MATRIX UNPACK
[SMVMUL]SPARSE	MATRIX VECTOR MULTIPLY
[ROT3]3D ROTATION	MATRIX, 3-ANGLE
SYMMETRIC	MATRIX...[TRED1]TRIDIAGONALIZE
[VMAX]VECTOR	MAXIMUM
[ISAMAX]INDEX OF	MAXIMUM ABSOLUTE VALUE
[VMAXMG]VECTOR	MAXIMUM MAGNITUDE
[MMTMUL]VECTOR MULTIPLY	MD*MD TO TM
[MTMMUL]VECTOR MULTIPLY	MD*TM TO MD
[MTTMUL]VECTOR MULTIPLY	MD*TM TO TM
[MMTADD]VECTOR ADD	MD+MD TO TM
[MTMADD]VECTOR ADD	MD+TM TO MD
[MTTADD]VECTOR ADD	MD+TM TO TM
[MMTSUB]VECTOR SUBTRACT	MD-MD TO TM
[MTMSUB]VECTOR SUBTRACT	MD-TM TO MD
[MTTSUB]VECTOR SUBTRACT	MD-TM TO TM
[POKE]STORE INTO	MEMORY
[TVCLR]TABLE	MEMORY VECTOR CLEAR
[VLMERG]LOGICAL VECTOR	MERGE
[VMIN]VECTOR	MINIMUM
[VMINMG]VECTOR	MINIMUM MAGNITUDE
[CVMOV]COMPLEX VECTOR	MOVE
[SVMOV]SPARSE VECTOR	MOVE
[VMOV]VECTOR	MOVE
[MOVREP]SUB-IMAGE	MOVE AND LEVEL REPLACE
[MTMOV]VECTOR	MOVE MD TO TM
[TMMOV]VECTOR	MOVE TM TO MD
[MTIMOV]VECTOR	MOVE WITH INCREMENT MD TO TM
[TMIMOV]VECTOR	MOVE WITH INCREMENT TM TO MD
[TTIMOV]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	MULT. 3D VECTORS
[CMVML4]COMPLEX 4X4 MATRIX	MULT. 4D VECTORS
[VXCS]VECTOR	MULTIPLIED BY SIN AND COS
[BLKMAN]BLACKMAN WINDOW	MULTIPLY
[CCMMUL]COMPLEX MATRIX	MULTIPLY
[CGMMUL]COMPLEX GENERAL MATRIX	MULTIPLY
[CMMUL]COMPLEX MATRIX	MULTIPLY
[CMUL]COMPLEX	MULTIPLY
[CRMMUL]COMPLEX-REAL MATRIX	MULTIPLY
AND REAL VECTOR	MULTIPLY...[CRVMUL]COMPLEX
[CVMUL]COMPLEX VECTOR	MULTIPLY
[CVSMUL]COMPLEX VECTOR SCALAR	MULTIPLY
TO DOUBLE PRECISION	MULTIPLY...[DMULRR]SINGLE
TO DOUBLE-PRECISION	MULTIPLY...[DMUL]DOUBLE
[FMMV]FAST MATRIX	MULTIPLY

APMATH64 KEY WORD INDEX

[VNEG]VECTOR	NEGATE
[MNAXB]SUB-MATRIX	NEGATIVE MULTIPLY
[MNATXB]SUBMATRIX	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
PREDICTOR	ORDER 2...[ABP2]ADAMS-BASHFORTH
[AMC2]ADAMS-MOULTON CORRECTOR	ORDER 2
PREDICTOR	ORDER 3...[ABP3]ADAMS-BASHFORTH
[AMC3]ADAMS-MOULTON CORRECTOR	ORDER 3
PREDICTOR	ORDER 4...[ABP4]ADAMS-BASHFORTH
[AMC4]ADAMS-MOULTON CORRECTOR	ORDER 4

[VRAMP]VECTOR	RAMP
[RAN]SCALAR	RANDOM NUMBER GENERATOR
[VRAND]VECTOR	RANDOM NUMBERS
[POST64]POST BITS TO	RASTER
[ATAN2]ARCTANGENT OF	RATIO OF REAL NUMBERS
[MRRUNR]MIXED-RADIX RFFT	RAVEL/UNRAVEL PASS
DOT PRODUCT REAL	REAL...[DDOTRR]DOUBLE
[SAXPYN]NESTED	REAL A * X + Y
[SCASUM]SUM OF	REAL AND IMAGINARY MAGNITUDES
[SDOTN]NESTED	REAL DOT PRODUCT
[RFFTM]MIXED-RADIX	REAL FFT NOT-IN-PLACE
[VPKR32]VECTOR	REAL HALFWORD PACK
[AINT]TRUNCATE	REAL NUMBER
[ATAN]ARCTANGENT OF	REAL NUMBER
[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
[CPOWCR]COMPLEX TO	REAL POWER
[RPOW]REAL TO	REAL POWER
[DDOTRR]DOUBLE DOT PRODUCT	REAL REAL
[RSFS2]SPARSE	REAL SYMM FACTOR & SOLVE
[RSFR2]SPARSE	REAL SYMMETRIC FACTOR
[RSSV2]SPARSE	REAL SYMMETRIC SOLVE
[VUPR32]VECTOR HALFWORD	REAL UNPACK
[RUF2]SPARSE	REAL UNSYM FACTOR & SOLVE
[RUF2]SPARSE	REAL UNSYMMETRIC FACTOR
[RUSV2]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
[VREAL]EXTRACT	REALS OF COMPLEX VECTOR
[CVCIP]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
MOVE AND LEVEL	REPLACE...[MOVREP]SUB-IMAGE
[VRVRS]VECTOR	REVERSE ORDERING
[MRRUNR]MIXED-RADIX	RFFT RAVEL/UNRAVEL PASS
[RSQRT]RECIPROCAL SQUARE	ROOT
RECIPROCAL SQUARE	ROOT...[VRSQRT]VECTOR
[VSQRT]VECTOR SQUARE	ROOT
[CSQRT]SQUARE	ROOT OF COMPLEX NUMBER
[SQRT]SQUARE	ROOT OF REAL NUMBER
[CROTG]COMPLEX GIVENS	ROTATION
[CSROT]COMPLEX 2-D	ROTATION
[SROTG]GIVENS PLANE	ROTATION
[SROT]PLANE	ROTATION

[VSINH]VECTOR	SINE HYPERBOLIC
[KSM LV]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
[RSSV2]SPARSE REAL SYMMETRIC	SOLVE
REAL UNSYM FACTOR AND	SOLVE...[RUFVS2]SPARSE
[RUSV2]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
[SGTSL]TRIDIAGONAL MATRIX	SOLVER
[SITSOL]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...[CRV SUB]COMPLEX
[CVS SUB]COMPLEX VECTOR	SUBTRACT
TO DOUBLE-PRECISION	SUBTRACT...[DSUBRR]SINGLE
TO DOUBLE-PRECISION	SUBTRACT...[DSUB]DOUBLE
[VISUB]VECTOR INTEGER	SUBTRACT
MULTIPLY, MULTIPLY, AND	SUBTRACT...[VMMSB]VECTOR
[VMSB]VECTOR MULTIPLY AND	SUBTRACT
SCALAR MULTIPLY AND	SUBTRACT...[VMSB]VECTOR
[VSUB]VECTOR	SUBTRACT
[VSBM]VECTOR	SUBTRACT AND MULTIPLY
[VSBSM]VECTOR	SUBTRACT AND SCALAR MULTIPLY
[MMTSUB]VECTOR	SUBTRACT MD-MD TO TM
[MTMSUB]VECTOR	SUBTRACT MD-TM TO MD
[MTTSUB]VECTOR	SUBTRACT MD-TM TO TM
[TMTMSUB]VECTOR	SUBTRACT TM-MD TO MD
[TMTSUB]VECTOR	SUBTRACT TM-MD TO TM

[TTTADD]VECTOR ADD	TM+TM TO TM
[TMMSUB]VECTOR SUBTRACT	TM-MD TO MD
[TMTSUB]VECTOR SUBTRACT	TM-MD TO TM
[TTMSUB]VECTOR SUBTRACT	TM-TM TO MD
[TTTSUB]VECTOR SUBTRACT	TM-TM TO TM
MATRIX MULTIPLY	TRACE...[CMMTRC]COMPLEX
[CMTRAC]COMPLEX SUB-MATRIX	TRACE
[ISIGN]INTEGER SIGN	TRANSFER
[SIGN]REAL NUMBER SIGN	TRANSFER
[CTRN2]2-D COORDINATE	TRANSFORM
[RDFT]REAL DISCRETE FOURIER	TRANSFORM
COORDINATE	TRANSFORMATION...[CTRN3]
[HLBRT]HILBERT	TRANSFORMER
[CMTRAN]COMPLEX SUB-MATRIX	TRANSPOSE
[MAXBT]MATRIX A TIMES B	TRANSPOSE
[MTRANS]MATRIX	TRANSPOSE
[STMM]SUBMATRIX	TRANSPOSE & MULTIPLY
[MATXBT]SUBMATRIX TRANSPOSE	TRANSPOSE MULTIPLY
[MNATXB]SUBMATRIX NEGATIVE	TRANSPOSE MULTIPLY
[MATXBT]SUBMATRIX	TRANSPOSE TRANSPOSE MULTIPLY
[VTRAPZ]VECTOR	TRAPEZOIDAL RULE INTEGRATION
[VØ1]VECTOR ZERO	TRENDS
[IMTQL1]DIAGONALIZE	TRIDIAGONAL MATRIX
[IMTQL2]DIAGONALIZE A	TRIDIAGONAL MATRIX
[HTRIDI]COMPLEX HERMITIAN	TRIDIAGONALIZATION
[VAINT]VECTOR	TRUNCATE
[VFRAC]VECTOR	TRUNCATE TO FRACTION
[GENTAB]GENERATE	TWIDDLE FACTOR TABLE
[SMUPCK]SPARSE MATRIX	UNPACK
[SVUPCK]SPARSE VECTOR	UNPACK
[VUP16]VECTOR 16-BIT BYTE	UNPACK
[VUP32]VECTOR 32-BIT BYTE	UNPACK
[VUP8]VECTOR 8-BIT BYTE	UNPACK
[VUPR32]VECTOR HALFWORD REAL	UNPACK
16-BIT SIGNED BYTE	UNPACK...[VUPS16]VECTOR
32-BIT SIGNED BYTE	UNPACK...[VUPS32]VECTOR
8-BIT SIGNED BYTE	UNPACK...[VUPS8]VECTOR
32-BIT SIGNED INTEGER	UNPACK...[VUSI32]VECTOR
[VUUI32]VECTOR 32-BIT UNSIGNED	UNPACK
[VUUI32]VECTOR 32-BIT	UNSIGNED UNPACK
[CUFS2]SPARSE COMPLEX	UNSYM FACTOR & SOLVE
[RUF2]SPARSE REAL	UNSYM FACTOR & SOLVE
[CUFR2]SPARSE COMPLEX	UNSYMMETRIC FACTOR
[RUF2]SPARSE REAL	UNSYMMETRIC FACTOR
[CUSV2]SPARSE COMPLEX	UNSYMMETRIC SOLVE
[RUSV2]SPARSE REAL	UNSYMMETRIC SOLVE
[CCEPS]PHASE	UNWRAP AND COMPLEX CEPSTRUM
[SHPHU]SCHAFFER'S PHASE	UNWRAPPING
[PKVAL]PEAK AND	VALLEY PICKING
[VASORT]VECTOR SORT ALGEBRAIC	VALUES
[VISORT]VECTOR SORT INTEGER	VALUES
[FUN1]FUNCTION OF ONE	VARIABLE
[ADAMS4]ADAMS	VARIABLE STEP INTEG.ORD 4
[VARNMO]NMO WITH	VARIABLE VELOCITY

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
[TMM]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
[VØ1]VECTOR	ZERO TRENDS
EQUATION, 2 POLES, 2	ZEROS...[DEQ2]DIFFERENCE
[VSCANØ]VECTOR SCAN FOR	ZEROS

Please detach cards along perforations.

READER'S COMMENT FORM

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

Title of document: _____
Your Name and Title: _____ Date: _____
Firm: _____ Department: _____
Address: _____
City: _____ State: _____ Zip Code: _____
Telephone Number: (____) _____ Extension: _____

I used this manual. . .

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

I found this material. . .

- | | Yes | No |
|------------------|--------------------------|--------------------------|
| accurate | <input type="checkbox"/> | <input type="checkbox"/> |
| complete | <input type="checkbox"/> | <input type="checkbox"/> |
| written clearly | <input type="checkbox"/> | <input type="checkbox"/> |
| well illustrated | <input type="checkbox"/> | <input type="checkbox"/> |
| well indexed | <input type="checkbox"/> | <input type="checkbox"/> |

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

ARRAY

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

Your Name and Title: _____ Date: _____
Firm: _____ Department: _____
Address: _____
City: _____ State: _____ Zip Code: _____
Telephone Number: (____) _____ Extension: _____

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

FLOATING POINT
SYSTEMS, INC.

