NATIONAL RADIO ASTRONOMY OBSERVATORY
CHARLOTTESVILLE, VIRGINIA

ELECTRONICS DIVISION INTERNAL REPORT NO. 226

COMPLEX MATH PACKAGE FOR APPLE II PLUS COMPUTERS

S. KELLER AND L. D'ADDARIO

JANUARY 1982

NUMBER OF COPIES: 150

COMPLEX MATH PACKAGE FOR APPLE II PLUS COMPUTERS

Table of Contents

Tables

COMPLEX MATH PACKAGE FOR APPLE II PLUS COMPUTERS

S. Keller and L. D'Addario

I. Introduction

The complex math package extends Applesoft Basic to include the operations of adding, multiplying, dividing, and converting coordinate systems with complex numbers, as well as adding, subtracting, multiplying and inverting complex 2-by-2 matrices. The routines are accessed through a basic CALL statement, with a special format to allow passing of parameters. In addition to eliminating the need for Basic subroutines to perform these operations, the execution times for these machine language routines are from 2 to 8 times faster than Basic.

Two terms that will need defining are:

1) Complex variable: two Applesoft floating point numbers (5 bytes each) which are adjacent in memory, with the first being interpreted as the real part and the second as the imaginary part of a complex number. Within Basic, the complex number must be part of an array; it is referenced by stating the array element which corresponds to the real part.

2) Complex matrix: a set of complex variables which are adjacent in memory, stored row-by-row. That is, the first complex number is interpreted as being in the first row, first column of a matrix; the second number in the first row, second column; ...; and the last complex number is in the last row, last column. Within Basic, the matrix is referenced by stating the array element corresponding to the real part of the first matrix element. In this report, only 2-by-2 matrices will be considered.

Note that these definitions are independent of the number of dimensions specified for the array in any DIM statement. Only the order of storage is important. (Page 137 of the Applesoft manual describes the order of storage for multi-dimensional arrays.)

II. Calling Sequence

Any of the subroutines can be invoked with an Applesoft Basic statement of the following form:

CALL <address> <delim> <result> <delim> <first operand> [<delim> <second operand>]

where <address> is an expression whose value is the entry point address of the subroutine to be invoked.

<delim> is usually a comma, but may be nearly any single-byte token. Clever use of "=", "+", etc. as delimiters can make a program more readable, as shown later in examples, but the choice of delimiter characters has no effect on the operation of the subroutine.

<result>, <first operand>, and <second operand> are the names of the Applesoft real array elements where a complex variable or matrix begins. The subroutines assume, without checking, that the rest of the complex variable or matrix immediately follows the specified element in memory.

Two routines, POLAR and RECT, do not use the last <delim> and <second operand>. In one routine, INV2X2 (which inverts a 2-by-2 complex matrix), "second operand" actually refers to a complex variable which will be set equal to the determinant of the matrix.

In all cases, <result> may be the same complex variable or matrix as one or both operands.

These routines should only be used in deferred mode (i.e., within an Applesoft program), not immediate mode.

III. Descriptions of Individual Subroutines

1. Complex Add: CADD

   CALL 5141, <result>, <operand1>, <operand2>

   The complex sum of <operand1> and <operand2> is stored in <result>.

2. Complex Multiply: CMUL

   CALL 5144, <result>, <operand1>, <operand2>

   The complex product of <operand1> and <operand2> is computed, then stored in <result>.

3. Complex Divide: CDIV

   CALL 5147, <result>, <operand1>, <operand2>

   Complex <operand1> is divided by <operand2>, then the result is stored in <result>.

4. Convert (real,imag) to (magnitude,phase): POLAR

   CALL 5150, <result>, <operand>

   In this routine, <operand> is a complex variable as usual, but <result> is an array element which will be set equal to the operand's magnitude; the very next array element will be set equal to the operand's phase in radians. The phase will be in the range $[-\pi/2, 3\pi/2)$; thus, the unavoidable discontinuity in phase occurs at $3\pi/2$. The computation performed is

$$\text{magnitude} = \text{SQR}(\text{real}^2 + \text{imag}^2)$$

$$\text{phase} = \begin{cases} \text{ATN}(\text{imag}/\text{real}), & \text{real} > 0 \\ (\pi/2)*\text{SGN}(\text{imag}), & \text{real} = 0 \\ \text{ATN}(\text{imag}/\text{real}) + \pi, & \text{real} < 0. \end{cases}$$

   where SQR, ATN and SGN are Applesoft functions.

5. Convert (magnitude, phase) to (real, imag): RECT

CALL 5153, <result>, <operand>

Here <operand> is an array element containing the magnitude of a complex number, with the next array element containing the phase in radians (as in POLAR). The phase need not be in the range $[-\pi/2, 3\pi/2)$. The subroutine computes

real = magnitude*COS(phase)

imag = magnitude*SIN(phase)

using the Applesoft COS and SIN functions.

6. 2-by-2 Complex Matrix Add: ADD2X2

CALL 5159, <result>, <matrix1>, <matrix2>

<matrix1> and <matrix2> are added, element by element.

7. 2-by-2 Complex Matrix Subtract: SUB2X2

CALL 5162, <result>, <matrix1>, <matrix 2>

<matrix2> is subtracted from <matrix1>, element by element.

8. 2-by-2 Complex Matrix Multiply: MUL2X2

CALL 5165, <result>, <matrix1>, <matrix 2>

The product of the two complex matrices is computed and placed in temporary storage. The product is then copied to <result>. This subroutine uses CADD and CMUL.

9. 2-by-2 Complex Matrix Invert: INV2X2

CALL 5168, <inverse>, <matrix>, <determinant>

First, the (complex) determinant of <matrix> is computed and placed in <determinant>. If the result is non-zero, the inverse of <matrix> is computed and placed in temporary storage, then copied to <inverse>. If the determinant is zero, all elements of <inverse> (both real and imaginary parts) are set to $+2^{96}$ ($\approx$7.9228E + 28).

If the matrix is

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

then the determinant is d = AD-BC and the inverse is

$$\begin{bmatrix} D/d & -B/d \\ -C/d & A/d \end{bmatrix}$$

where A, B, C, D and d are complex.

This subroutine uses CMUL and CDIV.

IV. Memory Requirements and Addresses

All of the Basic-callable subroutines, plus a utility routine for obtaining the passed parameters, must be assembled together because they call upon each other to some extent. They also depend on numerous routines in the Applesoft ROM, and therefore will not run in other environments (RAM Applesoft or Integer Basic, for example, although Applesoft in a RAM card should still work).

The routines described here plus some others have been assembled into a single binary file which loads into addresses $1400 = 5120 through $1DBA = 7130. This file will be referred to as the NRAO Binary Library, Version 2.0. Table I gives a list of the length and entry point addresses for each routine. Locations $1400 through $14FE are reserved for JMP instructions to the actual entry points of the subroutines; Table I lists the location of the JMP to each routine under "Indirect Entry." It is recommended that Basic programs use these indirect entries in CALL statements, since the actual entry addresses may change in later versions of the library.

Further information about the use of the Binary Library is given in a separate report [Internal Report No. 225].

TABLE I. Entry Point Addresses in Binary Library Version 2.0

	Length, Bytes		Indirect Entry		Actual Entry	
Subroutine Name	Hex	Decimal	Hex	Decimal	Hex	Decimal
CADD	$41	65	$1415	5141	$1628	5672
CMUL	$85	133	$1418	5144	$1669	5737
CDIV	$DØ	208	$141B	5147	$16EE	5870
POLAR	$C9	201	$141E	5150	$17BE	6078
RECT	$4E	78	$1421	5153	$1887	6279
ADD2X2	$4F	79	$1427	5159	$191A	6426
SUB2X2	$4F	79	$142A	5162	$1969	6505
MUL2X2	$1DØ	464	$142D	5165	$19B8	6584
INV2X2	$1C1	449	$1430	5168	$1B88	7048

V. Timing

Table II gives the results of timing tests on each of the subroutines using a short Applesoft Basic program. In all cases, the arguments were non-zero, small integers and the complex variables and matrices started at the beginning of an array. The code timed was a loop of this form:

```
10  AD = <address of subroutine's indirect entry>
20  FOR I = 1 to 1500
30  CALL AD, A(0) = B(0) + C(0)
40  NEXT
```

The time to execute the CALL statement was determined by subtracting the FOR-NEXT overhead (separately determined) and dividing by the number of loops.

In some cases, a straightforward Basic subroutine was written to perform the same function, and its timing was measured for comparison. These results are also given in Table II.

TABLE II. Subroutine Execution Times

Subroutine Name	Subroutine Time, msec	Equivalent Basic time, msec
CADD	10	
CMUL	17.5	35
CDIV	26	126
POLAR	∿ 100	∿ 220
RECT	62	72
ADD2X2	16	76.5
SUB2X2	17	
MUL2X2	62	380
INV2X2	62	

VI. Examples and Suggestions

As mentioned earlier, only the order of storage is significant in the arrays passed to the subroutines. Therefore, any of the following DIM statements will reserve space for a single 2-by-2 complex matrix (8 floating point numbers):

```
DIM  A(7)
DIM  B(1, 3)
DIM  C(1, 1, 1)
```

Recall that the range of each subscript starts at 0. In the last example, the first subscript selects real or imaginary part; the second is the matrix column; and the third is the row. Furthermore, an array of 50 matrices could be reserved by these statements

```
DIM  X(399)
DIM  Y(7, 49)
```

or others with up to 4 subscripts. The Kth matrix could be passed to a subroutine

by writing X(8*K) or Y(0,K) in the CALL statement. Calls with multidimensional arrays will execute more slowly than with one-dimensional arrays, but might result in clearer Basic code.

It has been found that CALL statements execute fastest when the address is given as a simple real variable. Thus CALL ADR, where ADR has previously been set to the desired address, is preferable to CALL 5150, for example.

The delimeters <delim> which separate elements of the CALL statement can be any single character which will not be interpreted by Applesoft Basic as part of a variable name or the end of the statement. The colon (:) and quote (") must be avoided, but =, *, +, / are useful in clarifying the code. The order of parameters in the subroutines has been chosen to facilitate this, as shown in the following example.

```
10  DIM A(25), B(25), C(25), X(1)
20  CADD = 5141  :  CMUL = 5144
30  FOR I = 0 to 24 STEP 2
40  CALL CMUL, X(0) = A(I)*B(I)
50  CALL CADD, X(0) = X(0) + C(I)
60  PRINT X(0), X(1)
70  NEXT
```

Notice in the above example that X is both an operand and the result in line 50. All of the subroutines allow this by placing the result in temporary storage (when necessary) until the computation is complete.