8. SIN/COS

a. General Discussion

Let $|X| = N*\text{pi}+f$, where $|f| \leq \text{pi}/2$. Then

```
sin(X) = sign(X) * sin(f) * (-1) ** N
```

and

```
cos(X) = sin(X+pi/2).
```

The computation of sin or cos thus involves three numerically distinct steps: the reduction of the given argument X to a related argument f, the evaluation of sin(f) over a small interval symmetric about the origin, and the reconstruction of the desired function value from these results. Sin(f) can be evaluated in a variety of ways. Here we use a minimax polynomial approximation derived from one in the collection of Hart et al. [1968].

The accuracy of the function values depends critically upon the accuracy of the argument reduction. Under the assumption that X is free of rounding or other errors, there is a possibility of loss of significance in f unless the difference $|X|-N*\text{pi}$ is evaluated using extra precision. Higher precision floating-point arithmetic can be used on some machines. When this is not practical, the computation

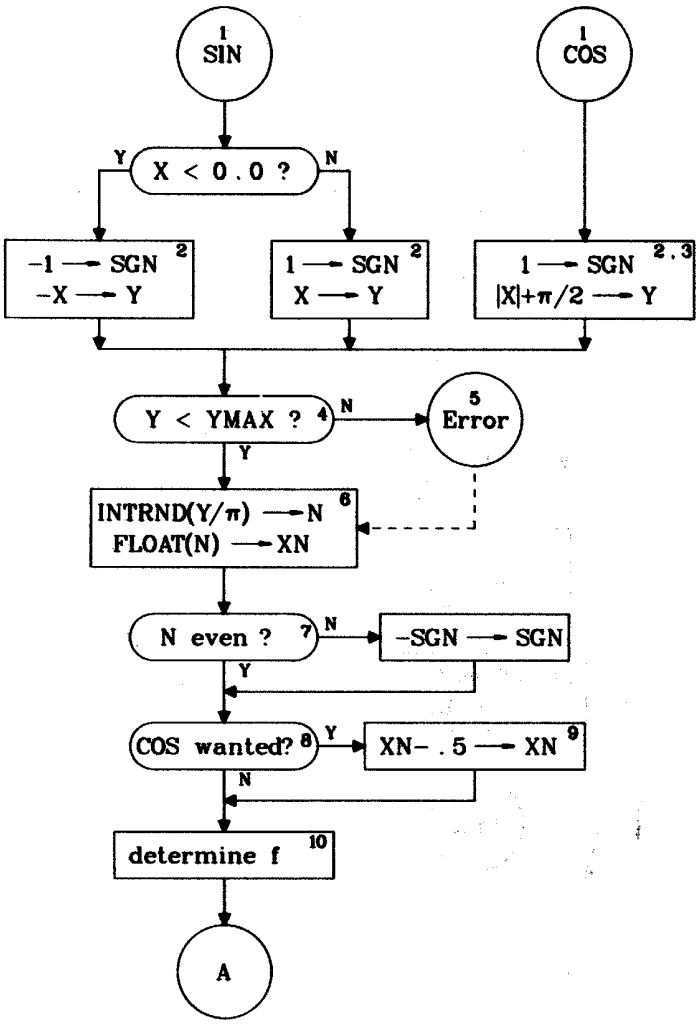
```
f = [(X1-N*C1) + X2] - N*C2,
```

where $X1 + X2 = |X|$ and $C1 + C2$ represents pi to more than working precision, preserves significance in f even though several leading significant figures may be lost in the subtraction. The protection afforded, however, is limited to the number of extra digits in this representation of pi. Should $|X|$ agree with an integer multiple of pi

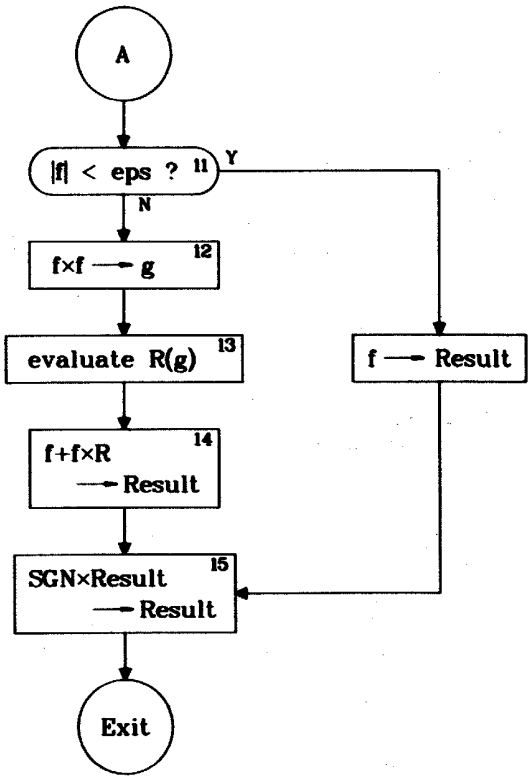
to more than this number of extra digits, the reduced argument *f* will lose a corresponding number of digits of precision. The precision also degenerates to that of the unaltered computation |*X*| - *N**pi whenever *N***C*1 cannot be represented *exactly* in the machine, e.g., when *N* becomes too large. Indeed, there will be a quantum drop in the precision of the function value as *N* passes this threshold. Note also that there is no way to compensate for inaccuracies in *X*.

For |*X*| sufficiently large, |*X*| and *N**pi agree to full machine precision. In that extreme case there is no precision in SIN(*X*) and an error condition exists. Clearly there is a general erosion of the precision of SIN(*X*) for smaller |*X*| despite the careful argument reduction just outlined. The algorithm presented here suggests an error return before the quantum loss in significance in *f* associated with the threshold on *N* mentioned above. This seems reasonable because the function is "grainy" for larger |*X*|; i.e., even the correct function values for neighboring floating-point arguments beyond this point probably disagree by more than half of the machine precision.

b. Flow Chart for SIN(X)/COS(X)



Note: Small integers indicate an implementation note.



Note: Small integers indicate an implementation note.

c. Implementation Notes, Non-Decimal Fixed-Point Machines

1) These may be alternate entries to one subroutine or entries to two different subroutines if multiple entries are not supported. See Note 4.

2) Using *SGN* as a flag may be more efficient than assigning a floating-point value to it. See Note 15.

3) Use of |*X*| here guarantees that the identity COS(-*X*) = COS(*X*) will hold. Use the following constant to machine precision:

```
pi/2 = 1.57079 63267 94896 61923
```

4) On machines that do not support multiple entry points, this is a natural point at which to begin a new subroutine. The parameters to be passed in that case are *X*, *Y* and *SGN*.

   The value of *YMAX* depends upon the number of bits *b* in the representation of the floating-point significand (see Glossary) and the number of bits in an integer. *YMAX* must satisfy the following conditions:

   a) *XN***C*1 and (*XN*-.5)**C*1 (see Notes 9 and 10) must both be exactly representable for *Y* < *YMAX*;
   b) a rounding error in the last bit of *Y* should not lead to a relative error in sin(*Y*) much greater in magnitude than 2**(*b*/2) for *Y* < *YMAX*; and
   c) *N* (see Notes 6 and 7) should be representable as an integer.

   A reasonable choice for *YMAX*, subject to condition (c), is the integer part of pi*2**(*b*/2).

5) If execution is to continue rather than terminate, then a default function value should be provided in addition to an error message. Two possibilities exist: A default value of 0.0 can be returned (especially appropriate for extremely large arguments), or processing can continue normally, provided *N* is still representable as an integer. The latter alternative may produce a few significant figures in the function value if the argument is not too large in

magnitude, but returns essentially random results for very large arguments. In either case the error message should be returned first.

6) This rounds *Y*/pi to the nearest integer (see Chapter 2 for definitions of INTRND and FLOAT). On some machines it may be more efficient to form *XN* with AINTRND and then use INT to obtain *N*. To avoid the expensive floating-point divide, form *Y*/pi by multiplication with the stored constant 1/pi. Use the following value to machine precision:

1/pi = 0.31830 98861 83790 67154.

The computation of *N* may impose a restriction on *YMAX* (see Note 4) when there are fewer than *b* bits available for the representation of an integer.

7) *N* is a positive integer. If machine instructions for testing specific bits are available, then the parity of *N* can be determined from its low-order bit; i.e., *N* is even if the bit is 0 and odd if the bit is 1.

8) The condition 'COS wanted' is equivalent to the condition $|X| \neq Y$.

9) This is equivalent to adding pi/2 to *X* for the COS entry but leads to greater accuracy in *f* when done at this point. Note that *XN* is no longer integer.

10) We assume that the multiple-step argument reduction scheme is to be used. On machines with a floating-point guard digit for addition (see Glossary), the computation becomes

```
f = (|X|-XN*C1) - XN*C2.
```

Special care must be taken on machines that lack the guard digit, or the argument reduction may lose significance for arguments slightly less than an integer power of the radix *B*. Full precision is retained on such machines with the computation

```
f = [(X1-XN*C1) + X2] - XN*C2,
```

where, in terms of our operations (see Chapter 2),

```
X1 = AINT(|X|)
```

and

```
X2 = |X| - X1.
```

Recall that b is the number of bits in the significand of a floating-point number. For $b \leq 32$ the constants

```
C1 = 201/64
   = 3.11 (octal)
   = 3.140625 (decimal),
C2 = 96765 35897 93 E-4
```

provide an extra 8 bits of precision in the argument reduction. For $b \geq 33$ the constants

```
C1 = 3217/1024
   = 3.1104 (octal)
   = 3.14160 15625 (decimal),
C2 = -8.9089 10206 76153 73566 17 E-6
```

provide an extra 12 bits of precision. *Exact* representation of $C1$ is crucial, but $C2$ need only be represented to the significance of the machine. At this point $|f| \leq$ pi/2.

11) Fixed-point underflow will not hurt here, but we want to be efficient. *Eps* should be chosen small enough that $\sin(f) = f$ to machine precision for $|f| <$ *eps*, but large enough that the shorter computational path ordinarily will be followed whenever $r1*f^3$ would underflow (see Note 13). We suggest *eps* = 2**(-b/2).

12) We convert to fixed point here, scaling as we go to avoid overflow. Let h = FIX(f/2) (see Chapter 2), and form $g = h*h$ in fixed point. Then

```
g = (f/2)² < pi²/16 < .62.
```

13) The following fixed-point polynomial approximations, $R(g)$, were derived from approximations in Hart et al. [1968]:

for b ≤ 24

```
r1 = -0.66666 62674
r2 =  0.13332 84022
r3 = -0.01267 67480
r4 =  0.00066 60872
```

for 25 ≤ b ≤ 32

```
r1 = -0.66666 66643 530
r2 =  0.13333 32915 289
r3 = -0.01269 81330 068
r4 =  0.00070 46136 593
r5 = -0.00002 44411 867
```

for 33 ≤ b ≤ 50

```
r1 = -0.66666 66666 66638 613
r2 =  0.13333 33333 32414 742
r3 = -0.01269 84126 86862 404
r4 =  0.00070 54673 00385 092
r5 = -0.00002 56531 15784 674
r6 =  0.00000 06573 19716 142
r7 = -0.00000 00120 76093 891
```

for 51 ≤ b ≤ 60

```
r1 = -0.66666 66666 66666 60209
r2 =  0.13333 33333 33330 64050
r3 = -0.01269 84126 98369 17789
r4 =  0.00070 54673 71779 91056
r5 = -0.00002 56533 57361 43317
r6 =  0.00000 06577 74038 64562
r7 = -0.00000 00125 22156 53481
r8 =  0.00000 00001 78289 31802
```

Evaluate $R(g) = g*P(g)$, where P is a polynomial, in fixed point using nested multiplication. For example, for $b \leq 24$,

```
R(g) = (((r4 * g + r3) * g + r2) * g + r1) * g.
```

14) We now convert back to floating point. Let R = REFLOAT[$R(g)$] (see Chapter 2), and form $f+f*R$ in floating point. The algebraically equivalent form $f*(1+R)$ may be less accurate and should be avoided.

15) Because floating-point multiplies are expensive, the more efficient approach may be to use *SGN* earlier as a flag and to change the sign of *Result* here if the flag is set.

d. Implementation Notes, All Floating-Point Machines

1) These may be alternate entries to one subroutine, or entries to two different subroutines if multiple entries are not supported. See Note 4.

2) Assigning a floating-point value to *SGN* will probably be more efficient than using it as a flag. See Note 15.

3) Use of |*X*| here guarantees that the identity COS(-*X*) = COS(*X*) will hold. Use the following constant to machine precision:

pi/2 = 1.57079 63267 94896 61923.

4) On machines that do not support multiple entry points, this is a natural point at which to begin a new routine. The parameters to be passed in that case are *X*, *Y* and *SGN*.

The value of *YMAX* depends upon the number *t* of base *B* digits in the representation of the floating-point significand (see Glossary) and the number of digits in an integer. *YMAX* must satisfy the following conditions:

a) *XN***C*1 and (*XN*-.5)**C*1 (see Notes 9 and 10) must both be exactly representable for *Y* < *YMAX*;
b) a rounding error in the last digit of *Y* should not lead to a relative error in sin(*Y*) much greater in magnitude than *B***(*t*/2) for *Y* < *YMAX*;
c) *N* (see Notes 6 and 7) should be representable as an integer.

A reasonable choice for *YMAX*, subject to condition (c), is the integer part of pi**B***(*t*/2).

5) If execution is to continue rather than terminate, then a default function value should be provided in addition to an error message. Two possibilities exist: A default value of 0.0 can be returned (especially appropriate for extremely large arguments), or processing can continue normally, provided *N* is still representable as an integer. The latter alternative may produce a few significant figures in the function value if the argument is not too large in

magnitude, but returns essentially random results for very large arguments. In either case the error message should be returned first.

6) This rounds *Y*/pi to the nearest integer (see Chapter 2 for definitions of INTRND and FLOAT). On some machines it may be more efficient to form *XN* with AINTRND and then use INT to obtain *N*. To avoid the floating-point divide, form *Y*/pi by multiplication with the stored constant 1/pi. Use the following value to machine precision:

1/pi = 0.31830 98861 83790 67154.

The computation of *N* may impose a restriction on *YMAX* (see Note 4) when there are fewer than *t* digits available for the representation of an integer.

7) *N* is a positive integer. If machine instructions for testing specific bits are available on non-decimal machines, then the parity of *N* can be determined from its low-order bit; i.e., *N* is even if the bit is 0 and odd if the bit is 1.

8) The condition 'COS wanted' is equivalent to the condition $|X| \neq Y$.

9) This is equivalent to adding pi/2 to *X* for the COS entry but leads to greater accuracy in *f* when done at this point. Note that *XN* is no longer integer.

10) If higher precision floating point is available and reasonably efficient, consider converting *X* and *XN* to the higher precision and using it to form

```
f = |X| - XN*pi,
```

where

pi = 3.14159 26535 89793 23846

is also given in the higher precision. Then convert *f* back to the working precision.

The multiple-step argument reduction scheme is used when the above scheme is impractical. On machines with a floating-point guard digit for addition (see Glossary), the computation becomes

```
f = (|X|-XN*C1) - XN*C2.
```

Special care must be taken on machines that lack the guard digit, or the argument reduction may lose significance for arguments slightly less than an integer power of the radix *B*. Full precision is retained on such machines with the computation

```
f = [(X1-XN*C1) + X2] - XN*C2,
```

where, in terms of our operations (see Chapter 2),

```
X1 = AINT(|X|)
```

and

```
X2 = |X| - X1.
```

Let *b* be the number of significant bits in the floating-point significand on a non-decimal machine. For $b \leq 32$ the constants

```
C1 = 201/64
   = 3.11 (octal)
   = 3.140625 (decimal),
C2 = 9.6765 35897 93 E-4.
```

provide an extra 8 bits of precision in the argument reduction. For $b \geq 33$ the constants

```
C1 = 3217/1024
   = 3.1104 (octal)
   = 3.14160 15625 (decimal),
C2 = -8.9089 10206 76153 73566 17 E-6
```

provide an extra 12 bits of precision. On decimal machines *C*1 should be the first 3 or 4 significant decimal figures of pi, and *C*2 should be determined so that *C*1 + *C*2 represents pi to 3 or 4 decimal places beyond machine precision. *Exact* representation of *C*1 is crucial, but *C*2 need only be represented to the significance of the machine. At this point $|f| \leq$ pi/2.

11) *Eps* should be chosen so that sin(*f*) = *f* to machine precision for |*f*| < *eps*, and so that r1**f*³ will not underflow for |*f*| ≥ *eps* (see Note 13). We suggest *eps* = *B***(-*t*/2).

12) There is no possibility of underflow or overflow at this point.

13) The following polynomial approximations, *R*(*g*), were derived from approximations in Hart et al. [1968]. Let *b* be the number of bits in the significand of a floating-point number on a non-decimal machine, and *d* be the number of digits in the significand on a decimal machine. Then

for b ≤ 24, or d ≤ 8

r1 = -0.16666 65668 E+0
r2 = 0.83330 25139 E-2
r3 = -0.19807 41872 E-3
r4 = 0.26019 03036 E-5

for 25 ≤ b ≤ 32, or
9 ≤ d ≤ 10

r1 = -0.16666 66660 883 E+0
r2 = 0.83333 30720 556 E-2
r3 = -0.19840 83282 313 E-3
r4 = 0.27523 97106 775 E-5
r5 = -0.23868 34640 601 E-7

for 33 ≤ b ≤ 50, or
11 ≤ d ≤ 15

r1 = -0.16666 66666 66659 653 E+0
r2 = 0.83333 33333 27592 139 E-2
r3 = -0.19841 26982 32225 068 E-3
r4 = 0.27557 31642 12926 457 E-5
r5 = -0.25051 87088 34705 760 E-7
r6 = 0.16047 84463 23816 900 E-9
r7 = -0.73706 62775 07114 174 E-12

for $51 \leq b \leq 60$, or
$16 \leq d \leq 18$

```
r1 = -0.16666 66666 66666 65052 E+0
r2 =  0.83333 33333 33316 50314 E-2
r3 = -0.19841 26984 12018 40457 E-3
r4 =  0.27557 31921 01527 56119 E-5
r5 = -0.25052 10679 82745 84544 E-7
r6 =  0.16058 93649 03715 89114 E-9
r7 = -0.76429 17806 89104 67734 E-12
r8 =  0.27204 79095 78888 46175 E-14
```

Evaluate $R(g) = g*P(g)$, where P is a polynomial, using nested multiplication. For example, for $b \leq 24$,

```
R(g) = (((r4 * g + r3) * g + r2) * g + r1) * g.
```

14) The algebraically equivalent form $f*(1+R)$ may be less accurate and should be avoided.

15) Multiplying by a floating-point value of *SGN* is probably more efficient than using *SGN* as a flag.

e. Testing

The tests are divided into four major parts. First is a random argument test to determine the accuracy of the basic computation, i.e., the evaluation of sin(x) where no argument reduction is needed. Second are similar tests of both sin(x) and cos(x) but using arguments which require that argument reduction be performed. Third is a series of short tests with special arguments. These include cursory checks of the three properties

$$\sin(-x) = -\sin(x),$$

$$\cos(-x) = \cos(x),$$

and

$$\sin(x) = x$$

to machine precision for $|x|<<1$. In addition, there is a check for underflow during the evaluation of sin(x) for very small x and a demonstration of the "granularity" of the function values for large x, that is, of the large changes induced in the function values by small changes in large arguments. Finally, there is a test for an error return when x is so large that there is no significance in the reduced argument f.

The random argument tests for sin(x) use the identity

$$\sin(x) = \sin(x/3)[3 - 4\sin^2(x/3)].$$

It is important that the subtraction not introduce a loss in significance from cancellation of leading significant digits. This can be assured by proper selection of arguments. When x is drawn from the interval $[3m*\text{pi},(3m+1/2)*\text{pi}]$, $x/3$ is in the interval $[m*\text{pi},(m+1/6)*\text{pi}]$, $4\sin^2(x/3) \leq 1$, and there is no cancellation. We henceforth assume these limitations on the arguments.

The tests measure the relative difference between the two sides of the identity. Thus we measure

$$E = \{\sin(x) - \sin(x/3)[3 - 4\sin^2(x/3)]\} / \sin(x).$$

Assume for the moment that $|x| \leq \text{pi}/2$. Because x is a random argument manufactured in the machine, we can safely assume that it is error free. However, there may be an error e in evaluating $x/3$. Now

$$\sin(x/3 + e) = \sin(x/3)\cos(e) + \cos(x/3)\sin(e).$$

Noting that

$$\sin(e) = e$$

and

$$\cos(e) = 1$$

to machine precision for e small, and that

$$|\cos(x/3)| < 1,$$

we can estimate

$$\sin(x/3+e) = \sin(x/3) + e.$$

Assume that relative errors of D and d are made in the evaluations of $\sin(x)$ and $\sin(x/3)$, respectively. Then

$$E = \frac{\sin(x)(1+D)-\sin(x/3+e)(1+d)[3-4\sin^2(x/3+e)(1+d)^2]}{\sin(x)(1+D)}.$$

Using the identity and estimate for $\sin(x/3+e)$, and keeping only terms linear in e, d and D, we can write

$$E = D - d\left[1-8\frac{\sin^3(x/3)}{\sin(x)}\right] + e\left[\frac{1}{\sin(x/3)} - 8\frac{\sin^2(x/3)}{\sin(x)}\right].$$

Because $\sin^3(x/3)/\sin(x)$ is bounded above by 1/8 for the interval under consideration, the coefficient of d is crudely bounded between 0 and 1. However, the coefficient of e is unbounded and could dominate the measured error, especially when x lies outside the primary interval. It is therefore necessary to "purify" the test arguments to ensure that $e = 0$, i.e., to perturb the original random argument x to a nearby x' such that both x' and $x'/3$ are *exactly* representable in the machine.

The following Fortran statements do the job on most computers:

```
Y = X / 3.0E0

Y = (Y + X) - X

X = 3.0E0 * Y     .
```

The exceptions are those machines where the active arithmetic registers carry more significance than the storage registers. On such machines it is necessary to force the storage and retrieval of intermediate results (see Gentleman and Marovich [1974]).

If purified arguments are used, the error measured in our tests is given by

```
E = D - c*d,
```

where $0 \leq c \leq 1$. Crude bounds indicate $0 \leq |E| \leq |D|+|d|$. In practice E turns out to be a reasonable estimate of D. The significant statistic obtained from the tests is MRE = max$|E|$ (see Chapter 3). A large value of MRE indicates a large value of D and/or d. The converse is not necessarily true; there may be a cancellation of leading digits in forming E from D and d. Our experience has been that this first test reports accuracies only slightly less than machine precision for good implementations of the basic computation of sin(x) (see Table 8.1). Typically the MRE reports a loss of about k+1 bits of precision on machines with a radix of B = 2**k, and a little over one digit on decimal machines. The tabulated loss of 1.18 digits on the IBM machine, for example, is a loss of between four and five bits, three of which are probably due to "wobbling precision" (see Glossary). The RMS typically reports a loss of a small fraction of a base-B digit.

Our second test draws x from [6*pi, 6.5*pi]. Assuming that the first test gave a small MRE, the MRE in the second test measures the accuracy of the argument reduction scheme. A good implementation should return MRE and RMS values comparable to those obtained in the first test. The large errors reported in Table 8.1 for the Varian 72, for example, probably indicate trouble in argument reduction. On the other hand, the unexpectedly large MRE reported for our program on the GP L3055 occurred for an argument which agreed with 6*pi to five decimal places. The extra precision in our multistep argument reduction was

TABLE 8.1

Typical Results for SIN/COS Tests

Test	Machine	B	Library or Program	Reported Loss of Base-B Digits in MRE	RMS
1	CDC 6400	2	Ours	2.00	0.73
	GP L3055	10	Ours	1.07	0.51
	IBM/370	16	Ours	1.08	0.71
	PDP/11	2	DOS 8.02	1.99	0.10
	Varian 72	2	Fort E3	1.87	0.00
	IBM/370	16	Argonne	1.18	0.69
2	CDC 6400	2	Ours	2.20	0.80
	GP L3055	10	Ours	2.28	0.77
	IBM/370	16	Ours	1.16	0.72
	PDP/11	2	DOS 8.02	1.74	0.09
	Varian 72	2	Fort E3	13.54	8.55
	IBM/370	16	Argonne	1.16	0.70
3	CDC 6400	2	Ours	2.39	0.68
	GP L3055	10	Ours	1.38	0.58
	IBM/370	16	Ours	1.11	0.70
	PDP/11	2	DOS 8.02	12.63	7.66
	Varian 72	2	Fort E3	12.69	7.31
	IBM/370	16	Argonne	1.16	0.69

insufficient to protect the precision of the reduced argument in this case. Note that the corresponding RMS value is reassurance that a programming error has not occurred and that the large MRE is an isolated occurrence.

The third test checks the accuracy of $\cos(x)$ with the identity

$$\cos(x) = \cos(x/3)[4\cos^2(x/3) - 3].$$

Analysis similar to that for $\sin(x)$ shows that x should be drawn from the interval $[(3m+1)*\text{pi},(3m+3/2)*\text{pi}]$ to preserve significance in the right-hand side. (Our test uses m=2.) Similarly, when the arguments are

purified, the error measured is

$$E = D - c*d,$$

where E, D and d are defined analogously to the $\sin(x)$ case and where $0 \leq c \leq 2$. A good implementation should again return MRE and RMS values comparable to those of the first test (see Table 8.1). Errors significantly larger than those reported for the previous test are usually due to carelessness in adjusting the argument by pi/2 for the COS entry. The results reported in Table 8.1 for the program on the PDP/11, for example, suggest that the argument reduction has been carefully done for the SIN entry, but not for the COS entry.

The above tests do not detect gross errors in the period of the function because the identities used are also satisfied by $\sin(kx)$ for arbitrary k. Our test program verifies that k=1 by making a finite difference approximation to the derivative near $x = 6*\text{pi}$.

```
C     PROGRAM TO TEST SIN/COS
C
C     DATA REQUIRED
C
C        NONE
C
C     SUBPROGRAMS REQUIRED FROM THIS PACKAGE
C
C        MACHAR - AN ENVIRONMENTAL INQUIRY PROGRAM PROVIDING
C                 INFORMATION ON THE FLOATING-POINT ARITHMETIC
C                 SYSTEM.  NOTE THAT THE CALL TO MACHAR CAN
C                 BE DELETED PROVIDED THE FOLLOWING FIVE
C                 PARAMETERS ARE ASSIGNED THE VALUES INDICATED
C
C                 IBETA  - THE RADIX OF THE FLOATING-POINT SYSTEM
C                 IT     - THE NUMBER OF BASE-IBETA DIGITS IN THE
C                          SIGNIFICAND OF A FLOATING-POINT NUMBER
C                 MINEXP - THE LARGEST IN MAGNITUDE NEGATIVE
C                          INTEGER SUCH THAT  FLOAT(IBETA)**MINEXP
C                          IS A POSITIVE FLOATING-POINT NUMBER
C                 EPS    - THE SMALLEST POSITIVE FLOATING-POINT
C                          NUMBER SUCH THAT 1.0+EPS .NE. 1.0
C                 EPSNEG - THE SMALLEST POSITIVE FLOATING-POINT
C                          NUMBER SUCH THAT 1.0-EPSNEG .NE. 1.0
C
C        RAN(K) - A FUNCTION SUBPROGRAM RETURNING RANDOM REAL
C                 NUMBERS UNIFORMLY DISTRIBUTED OVER (0,1)
C
C
C     STANDARD FORTRAN SUBPROGRAMS REQUIRED
C
C         ABS, ALOG, AMAX1, COS, FLOAT, SIN, SQRT
C
C
C     LATEST REVISION - DECEMBER 6, 1979
C
C     AUTHOR - W. J. CODY
C              ARGONNE NATIONAL LABORATORY
C
C
      INTEGER I,IBETA,IEXP,IOUT,IRND,IT,I1,J,K1,K2,K3,MACHEP,
     1        MAXEXP,MINEXP,N,NEGEP,NGRD
```

```
      REAL A,AIT,ALBETA,B,BETA,BETAP,C,DEL,EPS,EPSNEG,EXPON,ONE,RAN,
     1     R6,R7,THREE,W,X,XL,XMAX,XMIN,XN,X1,Y,Z,ZERO,ZZ
C
      IOUT = 6
      CALL MACHAR(IBETA,IT,IRND,NGRD,MACHEP,NEGEP,IEXP,MINEXP,
     1            MAXEXP,EPS,EPSNEG,XMIN,XMAX)
      BETA = FLOAT(IBETA)
      ALBETA = ALOG(BETA)
      AIT = FLOAT(IT)
      ONE = 1.0E0
      ZERO = 0.0E0
      THREE = 3.0E0
      A = ZERO
      B = 1.570796327E0
      C = B
      N = 2000
      XN = FLOAT(N)
      I1 = 0
C-----------------------------------------------------------------------
C     RANDOM ARGUMENT ACCURACY TESTS
C-----------------------------------------------------------------------
      DO 300 J = 1, 3
         K1 = 0
         K3 = 0
         X1 = ZERO
         R6 = ZERO
         R7 = ZERO
         DEL = (B - A) / XN
         XL = A
C
         DO 200 I = 1, N
            X = DEL * RAN(I1) + XL
            Y = X / THREE
            Y = (X + Y) - X
            X = THREE * Y
            IF (J .EQ. 3) GO TO 100
            Z = SIN(X)
            ZZ = SIN(Y)
            W = ONE
            IF (Z .NE. ZERO) W = (Z - ZZ*(THREE-4.0E0*ZZ*ZZ)) / Z
            GO TO 110
  100       Z = COS(X)
            ZZ = COS(Y)
```

```
            W = ONE
            IF (Z .NE. ZERO) W = (Z + ZZ*(THREE-4.0E0*ZZ*ZZ)) / Z
  110       IF (W .GT. ZERO) K1 = K1 + 1
            IF (W .LT. ZERO) K3 = K3 + 1
            W = ABS(W)
            IF (W .LE. R6) GO TO 120
            R6 = W
            X1 = X
  120       R7 = R7 + W * W
            XL = XL + DEL
  200     CONTINUE
C
          K2 = N - K3 - K1
          R7 = SQRT(R7/XN)
          IF (J .EQ. 3) GO TO 210
          WRITE (IOUT,1000)
          WRITE (IOUT,1010) N,A,B
          WRITE (IOUT,1011) K1,K2,K3
          GO TO 220
  210     WRITE (IOUT,1005)
          WRITE (IOUT,1010) N,A,B
          WRITE (IOUT,1012) K1,K2,K3
  220     WRITE (IOUT,1020) IT,IBETA
          W = -999.0E0
          IF (R6 .NE. ZERO) W = ALOG(ABS(R6))/ALBETA
          WRITE (IOUT,1021) R6,IBETA,W,X1
          W = AMAX1(AIT+W,ZERO)
          WRITE (IOUT,1022) IBETA,W
          W = -999.0E0
          IF (R7 .NE. ZERO) W = ALOG(ABS(R7))/ALBETA
          WRITE (IOUT,1023) R7,IBETA,W
          W = AMAX1(AIT+W,ZERO)
          WRITE (IOUT,1022) IBETA,W
          A = 18.84955592E0
          IF (J .EQ. 2) A = B + C
          B = A + C
  300 CONTINUE
C-------------------------------------------------------------------
C     SPECIAL TESTS
C-------------------------------------------------------------------
      WRITE (IOUT,1025)
      C = ONE / BETA ** (IT/2)
      Z = (SIN(A+C) - SIN(A-C)) / (C + C)
```

```
      WRITE (IOUT,1026) Z
C
      WRITE (IOUT,1030)
C
      DO 320 I = 1, 5
         X = RAN(I1) * A
         Z = SIN(X) + SIN(-X)
         WRITE (IOUT,1060) X, Z
  320 CONTINUE
C
      WRITE (IOUT,1031)
      BETAP = BETA ** IT
      X = RAN(I1) / BETAP
C
      DO 330 I = 1, 5
         Z = X - SIN(X)
         WRITE (IOUT,1060) X, Z
         X = X / BETA
  330 CONTINUE
C
      WRITE (IOUT,1032)
C
      DO 340 I = 1, 5
         X = RAN(I1) * A
         Z = COS(X) - COS(-X)
         WRITE (IOUT,1060) X, Z
  340 CONTINUE
C
      WRITE (IOUT,1035)
      EXPON = FLOAT(MINEXP) * 0.75E0
      X = BETA ** EXPON
      Y = SIN(X)
      WRITE (IOUT,1061) X, Y
      WRITE (IOUT,1040)
      Z = SQRT(BETAP)
      X = Z * (ONE - EPSNEG)
      Y = SIN(X)
      WRITE (IOUT,1061) X, Y
      Y = SIN(Z)
      WRITE (IOUT,1061) Z, Y
      X = Z * (ONE + EPS)
      Y = SIN(X)
      WRITE (IOUT,1061) X, Y
```

```
C-------------------------------------------------------------------
C     TEST OF ERROR RETURNS
C-------------------------------------------------------------------
      WRITE (IOUT,1050)
      X = BETAP
      WRITE (IOUT,1052) X
      Y = SIN(X)
      WRITE (IOUT,1055) Y
      WRITE (IOUT,1100)
      STOP
 1000 FORMAT(43H1TEST OF SIN(X) VS 3*SIN(X/3)-4*SIN(X/3)**3  //)
 1005 FORMAT(43H1TEST OF COS(X) VS 4*COS(X/3)**3-3*COS(X/3)  //)
 1010 FORMAT(I7,47H RANDOM ARGUMENTS WERE TESTED FROM THE INTERVAL /
     1 6X,1H(,E15.4,1H,,E15.4,1H)//)
 1011 FORMAT(18H SIN(X) WAS LARGER,I6,7H TIMES, /
     1      11X,7H AGREED,I6,11H TIMES, AND /
     1   7X,11HWAS SMALLER,I6,7H TIMES.//)
 1012 FORMAT(18H COS(X) WAS LARGER,I6,7H TIMES, /
     1      11X,7H AGREED,I6,11H TIMES, AND /
     1   7X,11HWAS SMALLER,I6,7H TIMES.//)
 1020 FORMAT(10H THERE ARE,I4,5H BASE,I4,
     1    46H SIGNIFICANT DIGITS IN A FLOATING-POINT NUMBER  //)
 1021 FORMAT(30H THE MAXIMUM RELATIVE ERROR OF,E15.4,3H = ,I4,3H **,
     1  F7.2/4X,16HOCCURRED FOR X =,E17.6)
 1022 FORMAT(27H THE ESTIMATED LOSS OF BASE,I4,
     1  22H SIGNIFICANT DIGITS IS,F7.2//)
 1023 FORMAT(40H THE ROOT MEAN SQUARE RELATIVE ERROR WAS,E15.4,
     1    3H = ,I4,3H **,F7.2)
 1025 FORMAT(14H1SPECIAL TESTS//)
 1026 FORMAT(4H IF ,E13.6,21H IS NOT ALMOST 1.0E0,,
     1   4X,25HSIN HAS THE WRONG PERIOD.  //)
 1030 FORMAT(51H THE IDENTITY   SIN(-X) = -SIN(X)   WILL BE TESTED.//
     1      8X,1HX,9X,12HF(X) + F(-X)/)
 1031 FORMAT(51H THE IDENTITY SIN(X) = X , X SMALL, WILL BE TESTED.//
     1       8X,1HX,9X,8HX - F(X)/)
 1032 FORMAT(50H THE IDENTITY   COS(-X) = COS(X)   WILL BE TESTED.//
     1      8X,1HX,9X,12HF(X) - F(-X)/)
 1035 FORMAT(43H TEST OF UNDERFLOW FOR VERY SMALL ARGUMENT.)
 1040 FORMAT(49H THE FOLLOWING THREE LINES ILLUSTRATE THE LOSS IN,
     1  13H SIGNIFICANCE/36H FOR LARGE ARGUMENTS.  THE ARGUMENTS,
     2  17H ARE CONSECUTIVE.)
 1050 FORMAT(22H1TEST OF ERROR RETURNS//)
 1052 FORMAT(37H SIN WILL BE CALLED WITH THE ARGUMENT,E15.4/
```

```
     1       37H THIS SHOULD TRIGGER AN ERROR MESSAGE//)
 1055 FORMAT(23H SIN RETURNED THE VALUE,E15.4///)
 1060 FORMAT(2E15.7/)
 1061 FORMAT(/6X,5H SIN(,E13.6,3H) =,E13.6)
 1100 FORMAT(25H THIS CONCLUDES THE TESTS )
C     ---------- LAST CARD OF SIN/COS TEST PROGRAM ----------
      END
```