A Method for Calculating Inverse Trigonometric Functions

Introduction. The problem of calculating an angle from a known trigonometric function of that angle occurs frequently in the numerical analysis of physical systems. The purpose of this paper is to introduce a new method which is particularly suitable for the I. B. M. Card Programmed Calculator.

Methods. There are several methods of determining an angle from its trigonometric functions; the more commonly employed ones are:

1. Table look up.
2. Infinite processes
   (a) Power series
   (b) Continued fractions.
3. Approximations.

Which of these methods is the best to use depends primarily on the computing equipment that is to be used. For desk calculating, tables are usually best; for computers with limited storage capacity, the infinite processes are usually used; on large scale computers, the approximations are most efficient. Since the IBM-CPC has a very limited amount of high speed storage, the common practice has been to use a Maclaurin's series with a variable number of terms. Other methods have been devised that speed up the process by first transforming the argument into a smaller number so that the series will converge more rapidly.

The New Method. The above discussion helps to illustrate the problem facing the programmer when programming a method for determining an angle from its trigonometric function. An entirely different method for finding inverse functions will now be described. This method is more of a logical process than a mathematical one. The following notation will be used:

\[ y = \text{trig}^{-1} (A, B) \]

which reads, "y is the inverse trigonometric function of A and B."

Furthermore,

\[ A = \sin y, \]
\[ B = \cos y. \]

In the case both the sine and cosine of y may be found independently, this method will determine the angle in its proper quadrant. If the cosine must be found by an identity from the sine, then only the principal value will be computed.

Let us assume that the sine and cosine of some angle, y, is known and it is subject to the following restrictions:

\[ 0 < y < \pi, \]
\[ y \neq \frac{\pi}{2m} \quad (m = 0, 1, 2, \ldots, 25). \]

The excluded values are special cases and will be discussed later. We first calculate \( \sin 2y \) and \( \cos 2y \) by suitable trigonometric formulae. From the
first restriction, $0 < y < \pi$, we have $\sin y > 0$; if

\[
\sin 2y > 0, \quad \text{then} \quad 0 < 2y < \pi \quad \text{and} \quad 0 < y < \frac{\pi}{2},
\]

but if

\[
\sin 2y < 0, \quad \text{then} \quad \pi < 2y < 2\pi \quad \text{and} \quad \frac{\pi}{2} < y < \pi.
\]

It is obvious then that a balance test of $\sin 2y$ will determine the quadrant of $y$; this information may be retained by storing the initial value of the quadrant. Calling the stored quantity $\Sigma_1$, it is seen that if

\[
0 < y < \frac{\pi}{2}, \quad \text{then} \quad \Sigma_1 = 0,
\]

but if

\[
\frac{\pi}{2} < y < \pi, \quad \text{then} \quad \Sigma_1 = \frac{\pi}{2}.
\]

From $\sin 2y$ and $\cos 2y$ we compute $\sin 4y$ and $\cos 4y$ and analyze in the same manner.

If $\sin 2y > 0$ and $\sin 4y > 0$, then $0 < y < \frac{\pi}{4}$ and $\Sigma_2 = 0$,

if $\sin 2y > 0$ and $\sin 4y < 0$, then $\frac{\pi}{4} < y < \frac{\pi}{2}$ and $\Sigma_2 = \frac{\pi}{4}$,

if $\sin 2y < 0$ and $\sin 4y > 0$, then $\frac{\pi}{2} < y < \frac{3\pi}{4}$ and $\Sigma_2 = \frac{\pi}{2}$,

if $\sin 2y < 0$ and $\sin 4y < 0$, then $\frac{3\pi}{4} < y < \pi$ and $\Sigma_2 = \frac{3\pi}{4}$,

where $\Sigma_2$ is the initial value of the octant in which $y$ lies. A comparison of the $\Sigma_2$'s and $\Sigma_1$'s will show that when

\[
\sin 4y > 0, \quad \text{then} \quad \Sigma_2 = \Sigma_1
\]

and when

\[
\sin 4y < 0, \quad \text{then} \quad \Sigma_2 = \Sigma_1 + \frac{\pi}{4}.
\]

In fact, if the same analysis is extended the following generalization will be apparent:

When

\[
\sin 2^ny > 0, \quad \text{then} \quad \Sigma_n = \Sigma_{n-1},
\]

and when

\[
\sin 2^ny < 0, \quad \text{then} \quad \Sigma_n = \Sigma_{n-1} + \frac{\pi}{2^n}.
\]

The $\Sigma_n$ will be the initial value of the interval, of width $\pi/2^n$, in which $y$ lies. If this process is continued until $\pi/2^n$ is less than the maximum allowable error, then $\Sigma_n$ will be the value of $y$. If the error is to be less than $5 \times 10^{-8}$ then $n$ must take on all values up to 26.
The cases where \( y = \pi/2^n \) cause difficulty because
\[
\sin 2^ny = 0 \quad \text{if} \quad n \geq m.
\]
Actually, the number \( \Sigma_n \) is the answer, \( y \), when \( n = m \) since \( \Sigma_n = \Sigma_m = \pi/2^n \).
Therefore, the first zero should be treated as a “minus” and all subsequent zeroes as “plus”.
If
\[
y = 0 \quad \text{or} \quad y = \pi,
\]
then
\[
\sin y = 0
\]
and
\[
\cos y = 1 \quad \text{or} \quad \cos y = -1;
\]
therefore, if \( \sin y \) is zero, then a separate test must be made on the cosine to determine if \( y = 0 \) or \( y = \pi \).

The discussion thus far has assumed that \( y \) is in the first two quadrants; the method can be extended to third and fourth quadrant angles by three methods:

1. The absolute value of \( \sin y \) may be used and the final answer multiplied by minus one if \( \sin y \) was originally negative. This gives answers in the range \( -\pi < y \leq \pi \).
2. If \( \sin y \) is negative, set \( \Sigma_0 = \pi \); in the previous discussion it was implied that \( \Sigma_0 = 0 \). This will give answers in the range \( 0 \leq y < 2\pi \).
3. If \( \sin y \) is negative, set \( \Sigma_0 = -\pi \). This puts the same limits on \( y \) as method 1.

In most cases, either method 1 or method 3 is preferable to method 2.

Method 1 is illustrated in the flow diagram, Figure 1. In practice it is more convenient to store \( \sigma_n = \Sigma_n/\pi \) and \( p_n = 1/2^n \) instead of \( \Sigma_n \) and \( \pi/2^n \), respectively. Since \( p_n = \frac{1}{2}p_{n-1} \), any rounding error will be positive and the sum \( \sigma_n \) will include the errors of all \( p_k \)'s where \( k \leq n \). This error may be as large as six in the last digit. This error may be reduced by always rounding to an even number or by carrying an extra digit in \( p_n \) and \( \sigma_n \); obviously both methods have disadvantages. The following is still another way in which the rounding error may be reduced and also save storage space. Let \( a_n = 0 \) or \( a_n = 1 \), depending on whether the term is to be omitted or added; also let \( n = 0, 1, 2, \ldots, N \); then
\[
y = \Sigma_N = \pi\sigma_N
\]
and
\[
\sigma_N = \frac{a_1}{2} + \frac{a_2}{4} + \frac{a_3}{8} + \cdots + \frac{a_N}{2^N};
\]
factoring out \( \frac{1}{2^N} \), then
\[
\sigma_N = \frac{1}{2^N}(a_12^{N-1} + a_22^{N-2} + a_32^{N-3} + \cdots + a_N)
\]
or
\[
2^N \sigma_N = \{[(2a_1 + a_2)2 + a_3]2 + a_4\}2 + \cdots + a_N\}.
A METHOD FOR CALCULATING INVERSE TRIGONOMETRIC FUNCTIONS 135

Fig. 1.

Store A = sin y
Store B = cos y
Store $\sigma_0 = 0$
Store $p_0 = \frac{1}{2}$
Balance test $\sin y$

- Store $k = +\pi$
- Store $k = -\pi$

Store $|\sin y|$  
Zero Test $\sin 2^n y$

Zero  Non Zero

- Compute $\sin 2^{n+1} y$ and $\cos 2^{n+1} y$

Balance Test $\sin 2^{n+1} y$

$\sigma_{n+1} = \sigma_n + 0$
$\sigma_{n+1} = \sigma_n + p_n$

Store $\sigma_{n+1}$

$P_{n+1} = \frac{1}{2} P_n$

Zero test $p_{n+1}$

Zero  Non-zero

Store $p_{n+1}$

$y = k \sigma_n$
The quantity \( Z_N = 2^N a_N \) may be computed by the following formula:

\[
Z_{n+1} = 2Z_n + a_{n+1}
\]

where \( Z_0 = 0 \) and \( n \) has the same meaning as before. The quantity thus computed is a whole number and is therefore exact, the only rounding error will be in the final multiplication by \( \pi/2^N \). The number of times \( Z \) has been computed is subtracted from \( N \) so that the process may be stopped when this quantity, \( b_n \), is equal to zero. This also saves storage space since only two digits of \( N \) need to be stored rather than seven or more of \( p_n \). The flow diagram illustrating this method is shown in Figure 2.

In computing \( \sin 2^n+1y \) and \( \cos 2^n+1y \) both \( \sin 2^ny \) and \( \cos 2^ny \) must be used in each; otherwise the condition

\[
(sin 2^n+1y)^2 + (cos 2^n+1y)^2 = 1
\]

will not always be satisfied. As an example, suppose the following formula is used:

\[
\cos 2^n+1y = 2(\cos 2^ny)^2 - 1.
\]

If \( \sin y \leq .0003162 \), then \( \cos y = 1.0000000 \) (the error being less than \( 5 \times 10^{-8} \)) and \( \cos 2^ny = 1.0000000 \) for all \( n \); this is obviously incorrect. A better formula to use is

\[
\cos 2^n+1y = (\cos 2^ny)^2 - (\sin 2^ny)^2.
\]

By defining the quantities

\[
F(n, y) = \cos 2^ny + \sin 2^ny
\]

and

\[
G(n, y) = \cos 2^ny - \sin 2^ny,
\]

then

\[
\cos 2^n+1y = F(n, y) \cdot G(n, y)
\]

and

\[
\sin 2^n+1y = F(n, y)^2 - 1.
\]

Although the above formulations are clumsy, they are necessary in order that the calculation can be made on the IBM 605 calculator. The programming for an unmodified 605 is given at the end of the article. All three-digit storage units are put on an 8-6 assignment and one full sweep of program steps is impulsed. The quantities \( \sin y \) and \( \cos y \) are read into Factor Storage 1, 2, 3 and 4 and the answer is read out of the third through tenth positions of the counter. During computation, F1, 2 is used to store \( \sin 2^n+1y \) and \( F(n, y) \); F3, 4 is used to store \( \cos 2^n+1y \); \( Z_n \) is stored in G1, 2 and \( b_n \) in G3. The following modifications would save program steps but would require additional tabulator wiring. General storage 1 and 2 can be cleared by the tabulator instead of on program step 10 to set \( Z_0 = 0 \). The quantity \( b_0 = N \) may be read from a card or emitted from the tabulator into G3. If the tube located at panel 1-7T is removed, the absolute value of a number may be read in on one program step instead of taking three. If the final multiplication is carried out on a separate card cycle, steps 50
through 60 may also be eliminated. This programming allows $y$ to be any value in the region $-\pi < y < \pi$, but excludes the possibility $y = \pi$.

In the worst cases, this method requires a little over two seconds which in CPC operation is less than six card cycles. This compares favorably with the time required by other methods to attain the same accuracy, and unlike most other methods does not require extra card cycles for determining the quadrant of the answer.
A METHOD FOR CALCULATING INVERSE TRIGONOMETRIC FUNCTIONS

605 PROGRAMMING

Step | Instruction | Suppress on | Notes
--- | ----------- | ----------- | ---
1  | Emit 2 (MQRI) | Ri 2nd | 
2  | MQRO Emit 6 (Prog. Exp No. 1) | | Program expansion No. 1 is G3RI, Ri 6th
3  | F12RO ERI (Bal. Test) | 1, 1 - | 
5  | F12RO ECR + | 1, 1 - | 
6  | ECRO F12RI | | 
7  | Zero Test (Reset) | | 
8  | F12RO ECR + (Bal. Test) | 1, Non zero | 
9  | G3RI G12RI (P. U. Gp. Sup. 1) | 1 | 
10 | F34RO ECR + | * | D.O. Rpt. Del. if Repeat selector one is up.
11 | ECRO MQRI | | 
12 | ECRO G4RI | Ro 6th | 
13 | F12RO ECR - | | 
14 | F12RO ECR - | | 
15 | ECRR F12RI | | 
16 | F12RO Mult + | | 
17 | ECRR F34RI | Ro 6th | 
18 | G4RO MQRI | | 
19 | F12RO Mult + | | 
20 | F34RO ECR + | | 
21 | § adj. | Ri 2nd | 
22 | ECRR F34 RI | RO 3rd | 
23 | Emit 1 ECR + | 1, 2 | 
24 | G12RO ECR + | | 
25 | G12RO ECR + | | 
26 | ECRR G12RI | | 
27 | P. U. Gp. Sup 2 | Non zero | 
28 | § adj. | Ri 2nd | 
29 | F12RO ECR + | | 
30 | ECRO MQRI | | 
31 | ECRR G4RI | Ro 6th | 
32 | F12RO Mult + | | 
33 | ECRR F12RI | Ro 6th | 
34 | G3RO ECR + | Ri 6th | 
35 | G4RO G3RI | Ri 6th | 
36 | F12RO ECR + | | 
37 | G3RO MQRO | F12RI | 
38 | G4RO MQRI | | 
39 | F12RO Mult + | | 
40 | § adj. | Ri 2nd | 
41 | Emit 1 MQRI | Ri 5th | 
42 | ECRO G3RI | Ro 6th | 
43 | G3RO ECR - | Ri 6th | 
44 | MQRO ECR - | Ri 6th | 
45 | ECRR F12RI | Ro 3rd | 
46 | G3RO ECR + | | 
47 | Zero Test (Ri 6th) | | 
48 | ECRR G3RI | | 
49 | F12RO ECR - (Prog. Rpt.) | Zero | Program expansion 2 is ECR +, Ri 6th
50 | Emit 4 F34RI | Ri 2nd | Non zero
51 | Emit 8 Prog. Exp 2 | Ri 2nd | Non zero
52 | Emit 3 ECR + (Prog. Exp 3) | Non zero | Program expansion 3 is Ri 3rd, Ri 4th
53 | Emit 1 ECR + | Ri 5th | Non zero
54 | ECRR MQRI | Ro 2nd | Non zero
55 | G12RO * | Non zero | * Multiply plus if repeat selector 1 is normal, minus if transferred
56 | F34RO MQRI Emit 6 | Non zero | 
57 | ECRR F34RI | Ro 6th | Non zero
58 | F34RO ECR + | Non zero | 
59 | G12RO * | Non zero | 
60 | § adj. | Ri 2nd | Non zero

Notes:
- D.O. Rpt. Del. if Repeat selector one is up.
- Multiply plus if repeat selector 1 is normal, minus if transferred
A METHOD OF INVERTING SYMMETRIC DEFINITE MATRICES

Robert L. LaFara

U. S. Naval Ordnance Plant
Indianapolis 18, Indiana

Although certain standard lines of approach seem to exist in all methods of inverting matrices, the problem of inversion has not yet been solved with finality, and modifications on “well-worn” solutions often prove to be of value. This paper discusses a partitioning method for inverting matrices which is particularly suited to the International Business Machines’ Card Programmed Calculator, (C.P.C.), and subsidiary equipment.

The general problem is to invert an $n^{th}$ order matrix

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

where $a$ and $d$ are each square submatrices of order $n_1$ and $n_2$ such that $n_1 + n_2 = n$. Let the inverse of $M$ be

$$M^{-1} = \begin{pmatrix} A & C \\ B & D \end{pmatrix}$$

partitioned in the same manner as the original matrix. Post-multiplying $M$ by its inverse, we have

$$\begin{pmatrix} aA + bB & aC + bD \\ cA + dB & cC + dD \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

or in equation form

$$aA + bB = 1 \quad \text{(identity matrix of order } n_1)$$
$$cA + dB = 0 \quad \text{(zero matrix } n_2 \times n_1)$$
$$aC + bD = 0 \quad \text{(zero matrix } n_1 \times n_2)$$
$$cC + dD = 1 \quad \text{(identity matrix of order } n_2).$$