BIT BY BIT METHODS
FOR
ELEMENTARY FUNCTIONS

A thesis submitted

by

Karabi Sarkar, M.Sc. (Jad.)

to

the Faculty of Science and Engineering

of the University of Ottawa

in partial fulfillment of the requirements

for the degree of

Master of Science

in the subject of

Mathematics

May 1972

To my parents

ABSTRACT

Methods are known for the computation of the inverses of elementary functions such as 2^x, cos x, tan x, etc. These methods are not readily applicable to the computation of direct functions such as cos x, tan x, 2^x, etc., which are at least as interesting as the inverse functions. In this thesis, we propose an algorithm for computing these functions.

ACKNOWLEDGEMENTS

It is my pleasure to express my sincere gratitude to my supervisor, Professor J.L. Howland for the advice and guidance given to me for the preparation of this thesis. It was Dr. Howland, who aroused my interest in the subject and whose valuable criticisms and guidance made the work successful.

I would like to thank Professor S. Bainbridge of the Department of Mathematics and Dr. G.M. White of the Department of Computer Science for their assistance during various phases of the work.

Thanks are also due to Professor B. Plaskacza of the Department of Slavic Studies who made a translation of a valuable manuscript in Russian and Mrs. O. Heywood, who typed this thesis with great patience.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

The word 'Bit' means a binary digit, 0 or 1, and we call an iterative method a 'Bit by Bit method' if, after each iteration, we obtain a Bit, either 0 or 1. Though some authors have called such methods 'Digit by Digit methods' in the literature, the name 'Bit by Bit method' in our opinion, is more justified.

There are various methods for calculating the elementary functions [1, p. 51 - 205]. Among these there is an important class known as iterative methods. Within this class there is a subclass in which the magnitude of the correction to be applied does not depend upon the previous steps but only on the number of iterations which have been performed. 'Bit by Bit methods' belong to this subclass.

In 1956 D.R. Morrison [7] first published a paper called "A Method for Computing Certain Inverse Functions." In his paper he showed how to calculate $\log_2 x$, and arc cos x using 'Bit by Bit methods'. In 1957, V.S. Liniskii [3] published a paper called "The Calculation of Elementary Functions on Automatic Digital Computers". He computed the functions p/q, $\sqrt{x}$, arc sin x, $\log_2 x$, and arc tan x. In 1958, L.A. Lyusternik [4] published a paper "On the Computation of Values of a Function

of a Single Variable". He calculated $\log_2 x$, arc cos x by 'Bit by Bit methods'. In 1959, J.H. Wensley [10] published a generalised paper on 'Bit by Bit methods' called "A Class of Non-Analytical Iterative Processes". According to him, it is possible to solve a functional equation f(x) = p where p is known and x is unknown if

(i) f(x) is non decreasing in the interval in which a solution is known to lie i.e. if t > s then $f(t) \geq f(s)$.

(ii) The function f(x) possesses an addition theorem of the form

$$f(s + t) = G[f(s), t].$$

He treated p/q, $\log_2 x$, $\sqrt{x}$, arc cos x, arc tan x cube root, fourth root and an inverse Jacobi elliptic function.

All of these are the inverses of functions such as 2^x, x^2, cos x which admit an addition formula, as required by condition (ii) above.

Unfortunately, the direct functions such as cos x, tan x, 2^x are of as much, or more, interest than the inverse functions, but cannot be computed by Wensley's method as arc cos x, arc tan x, $\log_2 x$ do not possess suitable addition formulae. In this thesis we propose an algorithm for evaluating such direct elementary functions by solving the equation $f^{-1}(p) = x$ where p is unknown, x is known and f(x) obeys the two sufficient conditions of Wensley.

In Chapter II we describe Bit by Bit methods for the computation of inverse functions with the help of some suitable examples adapted from the work of Wensley.

In Chapter III we describe our proposed algorithm and work out a few numerical examples.

CHAPTER II

COMPUTATION OF INVERSE FUNCTIONS

A comparative study of the procedures given in references [4], [8] and [10] for the bit by bit evaluation of various inverse functions shows that the algorithms proposed by these authors for a particular function are essentially identical; the only exceptions to this general rule being the pairs of algorithms given by Liniskii [3] for the arc sine etc., which employ different test functions. In every case however Wensley's two conditions are satisfied. These methods may be illustrated by the following examples adapted from Wensley.

2.1 Computation of Arc Cosine.

Let us consider the equation

$$\cos \pi x = p \tag{2.1.1}$$

where x $(0, \leq x < 1)$ is unknown and p $(-1 < p \leq 1)$ is known. The solution of the equation will be

$$x = \left(\frac{\text{arc cos } p}{\pi}\right) \tag{2.1.2}$$

Let

$$x = .\alpha_1\alpha_2 \ldots \alpha_t \text{ where } \alpha_j = \begin{cases} 0 \\ 1 \end{cases}$$

$$x_n = .\alpha_1\alpha_2 \ldots \alpha_n,\ n < t$$

$$y_n = 2^n(x - x_n) = .\alpha_{n+1} \ldots \alpha_t$$

$$a_n = \cos \pi y_n \qquad (2.1.3)$$

therefore $a_0 = \cos \pi y_0 = \cos \pi x = p$

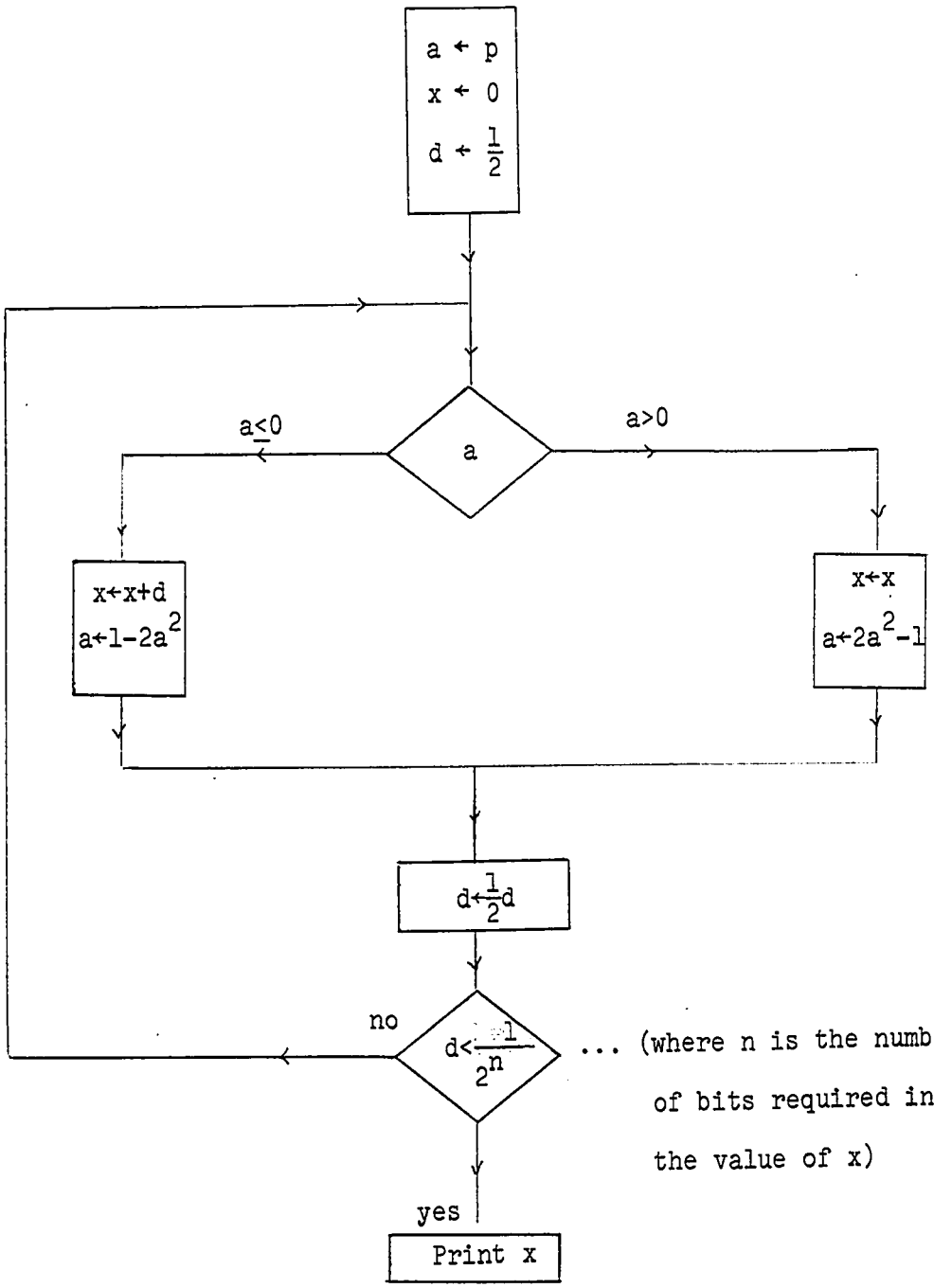
Now $y_n \geq \frac{1}{2} \Leftrightarrow \alpha_{n+1} = 1 \Leftrightarrow a_n \leq 0$

and $y_n < \frac{1}{2} \Leftrightarrow \alpha_{n+1} = 0 \Leftrightarrow a_n > 0$

If

$$\left.\begin{array}{l} \alpha_{n+1} = 0 \left\{ \begin{array}{l} y_{n+1} = 2y_n \\ a_{n+1} = 2a_n^2 - 1 \end{array} \right. \\ \alpha_{n+1} = 1 \left\{ \begin{array}{l} y_{n+1} = 2y_n - 1 \\ a_{n+1} = 1 - 2a_n^2 \end{array} \right. \end{array}\right\} \quad \ldots, (2.1.4)$$

Suppose we know a_n, then from its sign we can tell whether α_{n+1} is 0 or 1. When α_{n+1} is known, from equation (2.1.4) we can find a_{n+1}, and so on. So if a_n is known, y_n is unknown, then by knowing the sign of a_n, we can find the first digit of y_n. We call this function a_n the 'test function'. We explain the working principle of this method with the help of the following diagram:



... (where n is the number of bits required in the value of x)

$$x \rightarrow \left(\frac{\text{arc cos } p}{\pi}\right)$$

2.2 Computation of $\sqrt{x}$.

Let $$x^2 = p \qquad (2.2.1)$$

where p ($0 \leq p < 1$) is known and x is unknown. The solution of the equation will be

$$x = \sqrt{p} \qquad (2.2.2)$$

Let

$$x = .\alpha_1\alpha_2 \ldots \alpha_t \quad \text{where } \alpha_j = \begin{cases} 0 \\ 1 \end{cases}$$

$$x_n = .\alpha_1\alpha_2 \ldots \alpha_n, \ n < t$$

$$d_{n+1} = \frac{1}{2} d_n$$

$$c_n = \frac{p}{d_{n+1}} - \frac{x_n^2}{d_{n+1}}$$

$$a_n = c_n - 2x_n - d_{n+1}$$

Let $$x_0 = 0, \ d_0 = 1$$

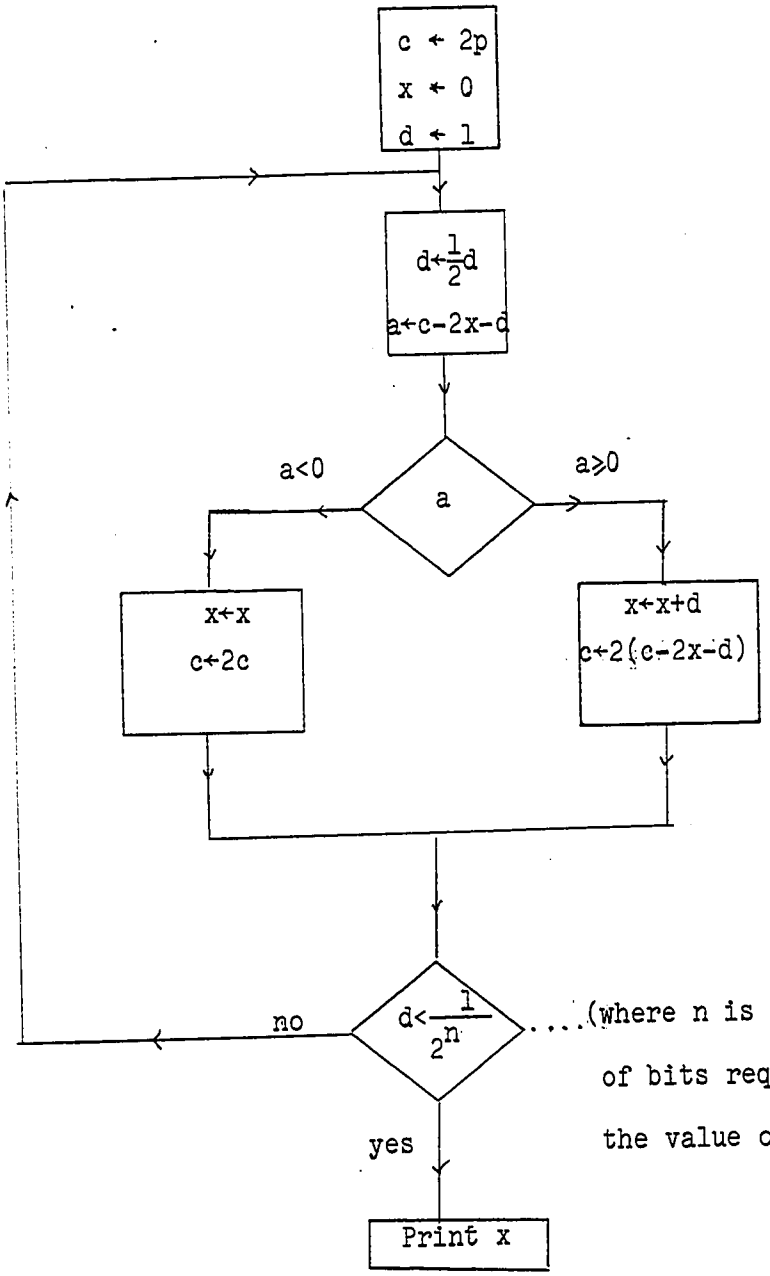
therefore $$c_0 = 2p, \ a_0 = 2p - \frac{1}{2}$$

Now

$$a_n \geq 0 \Leftrightarrow \alpha_{n+1} = 1 \Leftrightarrow c_{n+1} = 2(c_n - 2x_n - d_{n+1})$$

$$a_n < 0 \Leftrightarrow \alpha_{n+1} = 0 \Leftrightarrow c_{n+1} = 2c_n$$

So by knowing the sign of a_n we can tell whether α_{n+1} is 0 or 1. When α_{n+1} is known, we can find c_{n+1} and thus a_{n+1} and so on. We explain the working principle of this method with the help of the following diagram:



....(where n is the number of bits required in the value of x)

$x \rightarrow \sqrt{p}$

Remark. Since the procedure of computing the other inverse functions namely $\log_2 x$, arc tan x, etc. is similar to that of computing the function arc cosine, this is not described in detail here. However, the test functions used to compute these functions are derived in the next chapter, where the direct functions are computed.

CHAPTER III

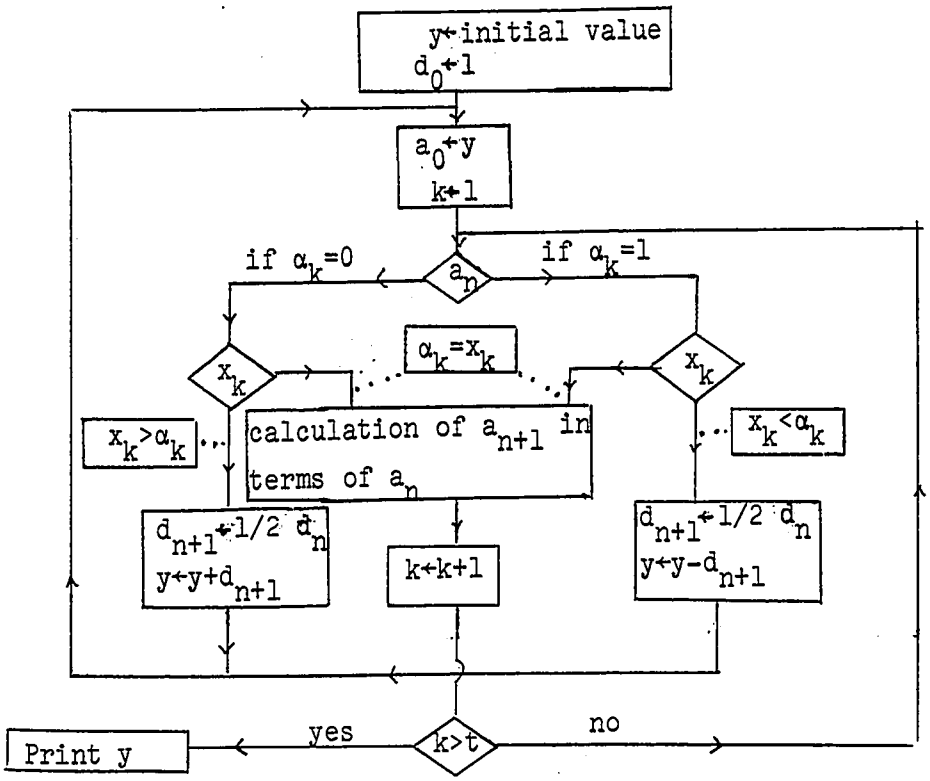
COMPUTATION OF DIRECT FUNCTIONS

3.1 The Proposed Algorithm.

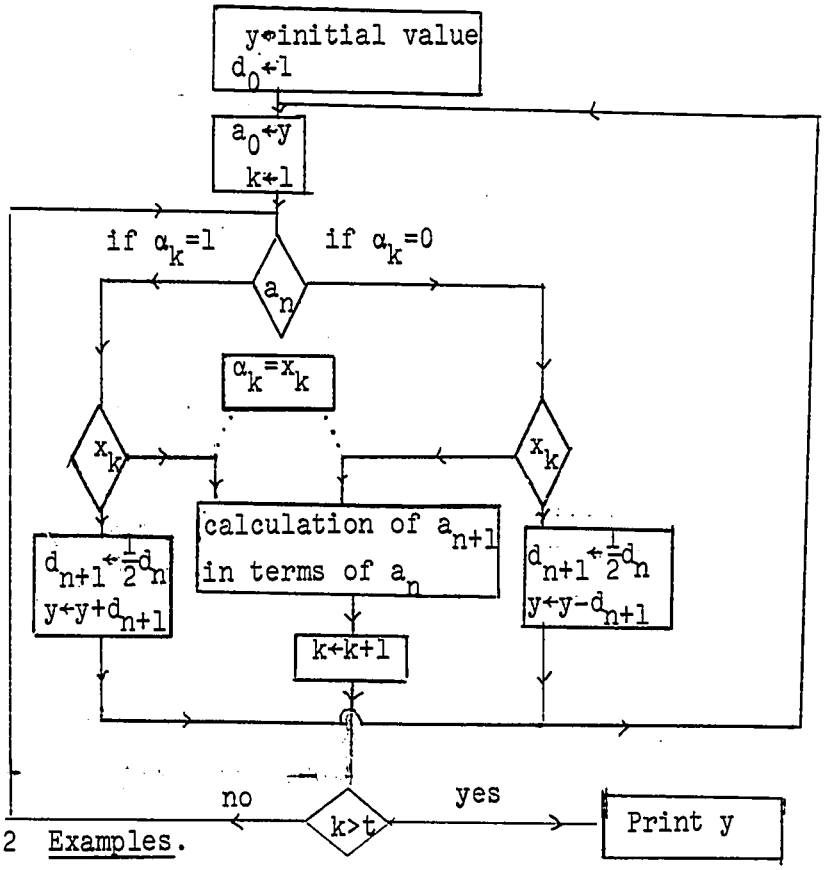
Let $$f(x) = y \qquad (3.1.1)$$

where x is known and y is unknown. Let a_n be the test function of the equation f(x) = y when x is unknown, and y is known and f(x) satisfies the two sufficient conditions of Wensley. Since x is known in (3.1.1) let $x = x_1\ x_2 \ldots x_t$ where $x_j = \begin{cases} 0 \\ 1 \end{cases}$ and let for a particular value of y, $f^{-1}(y) = \alpha_1\ \alpha_2 \ldots \alpha_t$ where $\alpha_j = \begin{cases} 0 \\ 1 \end{cases}$. Then to find the exact value of y for which each $\alpha_k = x_k$ we may use one of the following algorithms:

Algorithm 1. (The given function is monotonically increasing)



Algorithm 2. (The given function is monotonically decreasing)



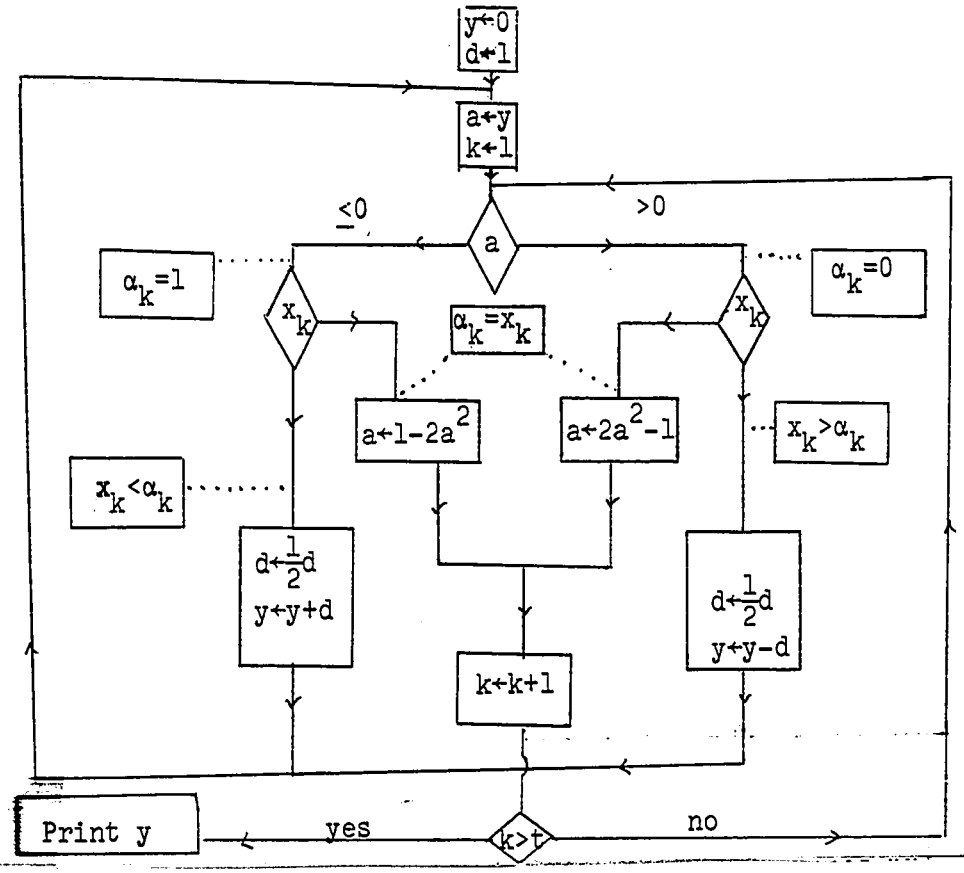
3.2 Examples.

Example 1. $f(x) = \cos \pi x = y$ (3.2.1)

where x ($0 \leq x < 1$) is known and y is unknown. The test function a_n has already been determined in (2.1.4). Let us now consider equation (3.2.1) and let $x = \cdot x_1 x_2 \ldots x_t$ where $x_j = \{^0_1$. For a particular value of y let

$$\frac{\arccos y}{\pi} = \cdot \alpha_1 \alpha_2 \ldots \alpha_k$$

Then to find the exact value of y for which each $\alpha_k = x_k$, we use algorithm 2.



Remark. If we use n correct bits of y in the arc cos programme we are able to find n corresponding correct bits of x in the angle θ. This can be easily explained from the following example:

$$\cos\,(.28506728\ldots)\pi = .625$$

$$\text{or } \cos\,(.0100100011\ldots)\pi = .101$$

3 Bit Fixed Point Arithmetic.

$a > 0$	x	a^2
.101	.0	.011|001

$2a^2 - 1 = .11 - 1 = -.01$

$a \leq 0$		
-.01	.01	.000|1

$1 - 2a^2 = 1$

$a > 0$		
1	.010	1

$2a^2 - 1 = 1$

therefore $x = .010$

4 Bit Fixed Point Arithmetic.

$a > 0$	x	a^2
.1010	.0	.0110|01

$2a^2 - 1 = -.01$

$a \leq 0$	x	a^2
-.01	.01	.0001

$1 - 2a^2 = 1 - .001 = .111$

$a > 0$	x	a^2
.111	.010	.1100|01

$2a^2 - 1 = 1.1 - 1 = .1$

$a > 0$	x	a^2
.1	.0100	.01

$x = .0100$ and so on.

But some time it may happen [11] [12] that due to losing

some significant digits we are not able to find the correct answer. For example,

Example 1.

$$\cos(.230000520340\ldots)\pi = .75$$

$$\cos(.0011101\ldots)\pi = .11$$

3 Bit Fixed Point Arithmetic.

$a > 0$	x	a^2
.11	.0	.100|1

$$2a^2 - 1 = 0$$

$a \leq 0$	x	a^2
0	.01	0.00

So instability starts from 2nd binary place of x. This instability can be avoided by using 4 Bit arithmetic.

4 Bit Arithmetic.

$a > 0$	x	a^2
.11	.0	.1001

$$2a^2 - 1 = .001$$

$a > 0$	x	a^2
.001	.00	.0000|01

$$2a^2 - 1 = -1$$

$a \leq 0$	x	a^2
-1	.001	1

$$1 - 2a^2 = -1$$

$a \leq 0$	x	a^2
-1	.0011	1

Answer x = .0011

Example 2.

$$\cos\,(.244629\ \ldots)\pi = .71875$$

$$\text{or } \cos\,(.0011111\ \ldots)\pi = .10111$$

5 Bit Arithmetic.

$\frac{a > 0}{.10111}$	$\frac{x}{.0}$	$\frac{a^2}{.10000\|10001}$

$$2a^2 - 1 = 1 - 1 = 0$$

$\frac{a \leq 0}{0}$	$\frac{x}{.01}$	$\frac{a^2}{0}$

So here again instability starts when we use 5 Bit arithmetic. But if we use 6 Bit arithmetic this difficulty can be avoided. Since we get maximum precision of 56 binary bits in Apl, it is enough to represent the correct answer up to 10 decimal places. Thus the difficulty mentioned by Van Wijngaarden [11], Wilkes and Wheeler [12], does not arise at all in our case.

Example 2.

$$f(x) = 2^x = y \qquad (3.2.2)$$

where x is known ($0 \leq x < 1$) and y is unknown.

Determination of Test Function a_n. Let us consider equation (3.2.2) where y is known ($1 \leq y < 2$) and x is unknown. Let

$$x = .\alpha_1\ \alpha_2\ \ldots\ \alpha_t;\quad x_j = \{_t^0$$

$$x_n = .\alpha_1\ \alpha_2\ \ldots\ \alpha_n,\ n < t$$

$$y_n = 2^n(x - x_n) = .\alpha_{n+1}\ \ldots\ t$$

and $$a_n = 2^{2y_n}$$

Now $$y_n \geq \frac{1}{2} \Leftrightarrow \alpha_{n+1} = 1 \Leftrightarrow a_n \geq 2$$

$$y_n < \frac{1}{2} \Leftrightarrow \alpha_{n+1} = 0 \Leftrightarrow a_n < 2$$

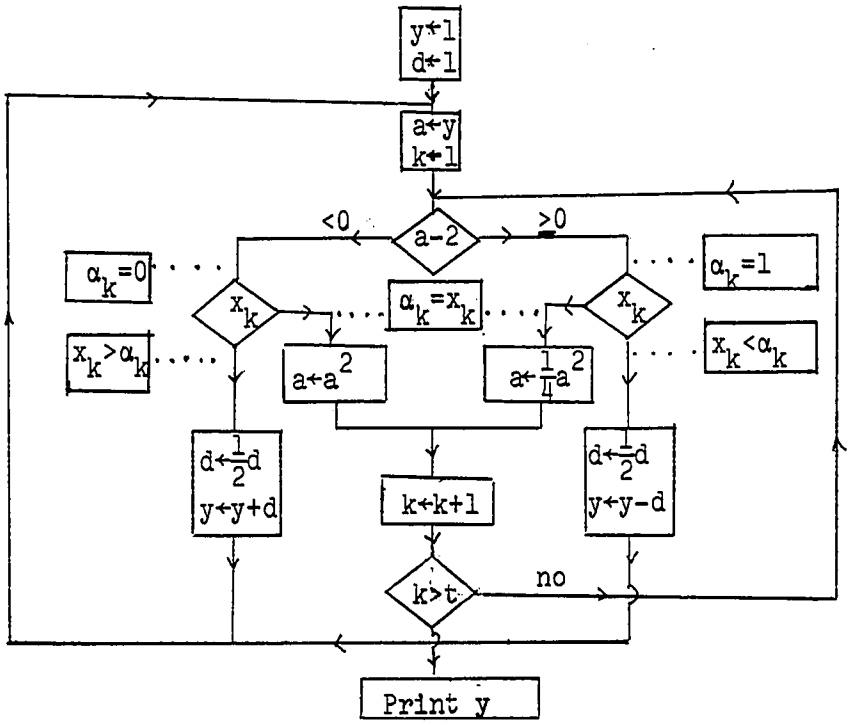
If
$$\left.\begin{array}{l} \alpha_{n+1} = 0 \begin{cases} y_{n+1} = 2y_n \\ a_{n+1} = 2^{2.2y_n} = a_n^2 \end{cases} \\ \alpha_{n+1} = 1 \begin{cases} y_{n+1} = 2y_n - 1 \\ a_{n+1} = 2^{2(2y_n-1)} = 2^{4y_n-2} = \frac{1}{4}a_n^2 \end{cases} \end{array}\right\} \quad (3.2.4)$$

So, as soon as $a_0 = 2^{2y_0} = 2^{2x} = y^2$ is known, we can find out each Bit of the corresponding unknown x from the formula (3.2.4).

Now let us consider (3.2.2) and let

$$x = .x_1 x_2 \ldots x_t \qquad \text{where } x_j = \begin{cases} 0 \\ 1 \end{cases}$$

Then to find the exact value of y for which each $\alpha_k = x_k$ we use the algorithm 1 in the following way:



Example 3. We will now show how the method of computation of 2^X by 'Bit by Bit methods' cam be adapted to compute e^X.

It is possible to reduce the value of x of e^X in the interval (0, ln 2) [2] and reduction can be done in the following way:

$$x \geq 0 \qquad u = \{\frac{x}{\log_e 2}\}, \quad v = [\frac{x}{\log_e 2}]$$

$$x < 0 \qquad u = \{\frac{x}{\log_e 2}\} + 1, \quad v = [\frac{x}{\log_e 2}] - 1$$

where the symbol [x] represents the integral part of x (for x < 0, [x] < 0) and {x} is the fractional part of x. Now,

$$e^x = e^{\frac{x}{\log_e 2}\log_e 2} = 2^{\{\frac{x}{\log_e 2}\}+[\frac{x}{\log_e 2}]}$$

$$= 2^u \cdot 2^v$$

where 2^u can be calculated by 'Bit by Bit methods' developed earlier.

Example 4.

$$f(x) = \tan \pi x = y \tag{3.2.5}$$

where x is known, and πx lies in $(-\pi/4, \pi/4)$ and y is unknown. The range of (πx) can be reduced to the interval $(-\pi/4, \pi/4)$ in the following way [2]:

$$u = \{x - \tfrac{1}{4}\}$$

$$v = |u| - \tfrac{1}{2}$$

$$z = |v| - \tfrac{1}{4} = \left|\left|\{x - \tfrac{1}{4}\}\right| - \tfrac{1}{2}\right| - \tfrac{1}{4}$$

$$\tan \pi x = \begin{cases} \tan \pi z & uv \geq 0 \\ \dfrac{1}{\tan \pi z} & uv < 0 \end{cases}$$

where $\{x\}$ represents the fractional part of x, so that $\{x\} \leq 0$ for $x < 0$.

Determination of the Value of Test Function a_n. Let us consider equation (3.2.5) where x is unknown, y is known and y lies in the interval [0, 1). Let

$$x = .\alpha_1 \alpha_2 \cdots \alpha_t \quad \text{where } \alpha_j = \begin{cases} 0 \\ 1 \end{cases}$$

$$x_n = .\alpha_1 \alpha_2 \cdots \alpha_n \quad n < t$$

$$y_n = 2^n(x - x_n) = .\alpha_{n+1} \cdots \alpha_t$$

and $a_n = \tan \pi y_n$

Now $$y_n < \frac{1}{2} \Leftrightarrow \alpha_{n+1} = 0 \Leftrightarrow a_n \geq 0$$

$$y_n > \frac{1}{2} \Leftrightarrow \alpha_{n+1} = 1 \Leftrightarrow a_n < 0$$

If
$$\left.\begin{aligned} \alpha_{n+1} = 0 &\begin{cases} y_{n+1} = 2y_n \\ a_{n+1} = \dfrac{2a_n}{1-a_n^2} \end{cases} \\ \alpha_{n+1} = 1 &\begin{cases} y_{n+1} = 2y_n - 1 \\ a_{n+1} = \dfrac{2a_n}{1-a_n^2} \end{cases} \end{aligned}\right\} \qquad (3.2.7)$$

Here also as soon as $a_0 = \tan \pi x = y$ is known, by using (3.2.7) we can find each unknown digit of x.

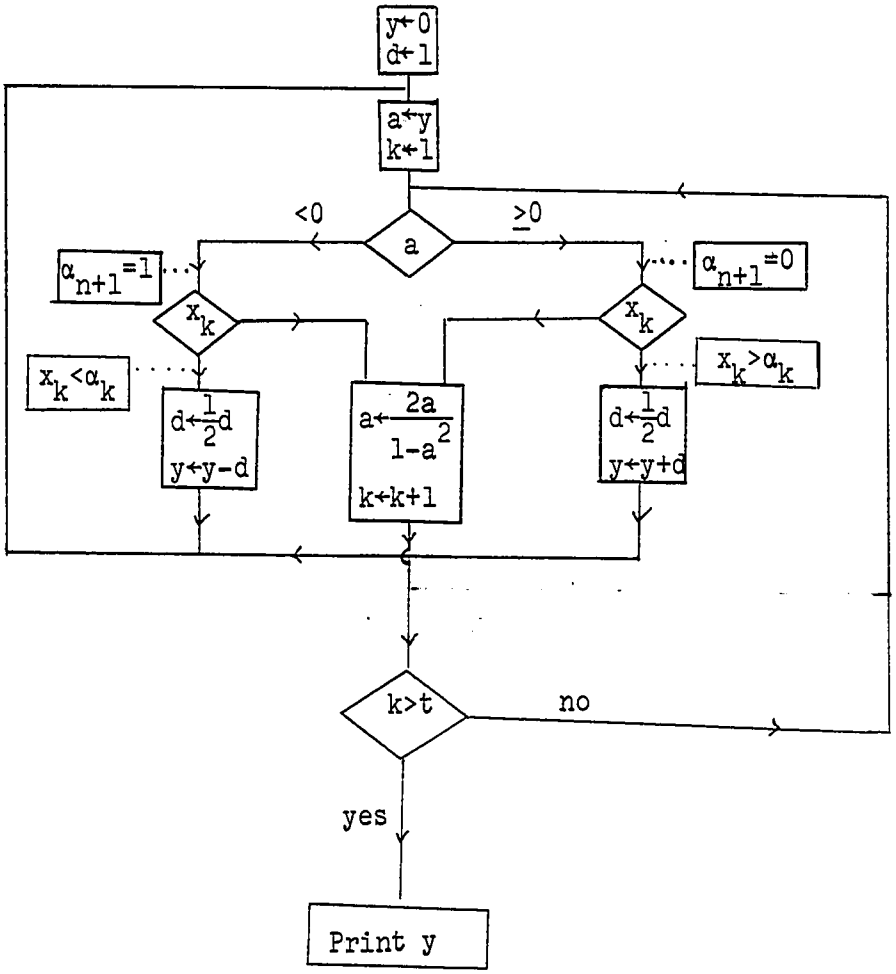
Now let us consider equation (3.2.5). Let

$$x = .x_1\ x_2 \ \ldots\ x_t \quad \text{where } x_j = \begin{cases} 0 \\ 1 \end{cases}$$

By giving a particular value of y_k of y we can find

$$\frac{\arctan y_k}{\pi} = .\alpha_1\ \alpha_2 \ \ldots\ \alpha_k \quad \text{where } \alpha_j = \begin{cases} 0 \\ 1 \end{cases}$$

To find the exact value of y where $\alpha_k = x_k$ we use the algorithm 1 in the following way:


y←0
d←1
a←y
k←1
a
<0
≥0
$\alpha_{n+1}=1$
$\alpha_{n+1}=0$
x_k
x_k
$x_k<\alpha_k$
$x_k>\alpha_k$
$d\leftarrow\frac{1}{2}d$
$y\leftarrow y-d$
$a\leftarrow\frac{2a}{1-a^2}$
$k\leftarrow k+1$
$d\leftarrow\frac{1}{2}d$
$y\leftarrow y+d$
k>t
no
yes
Print y

APPENDIX

Listings for programs:

1. Program for tan x.

```
      )FNS PAN
∇ Z←PAN P
 [1] D←1
 [2] X←0
 [3] K←1
 [4] A←X
 [5] →(A≥0)/L1
 [6] →(P[K]=1)/L2
 [7] D←D×.5
 [8] X←X-D
 [9] →3
 [10] L1:→(P[K]=0)/L2
 [11] D←D×.5
 [12] X←X+D
 [13] →3
 [14] L2:K←K+1
 [15] A←(2×A)÷(1-(A*2))
 [16] →(K>30)/L3
 [17] →5
 [18] L3:Z←X
∇
```

2. Program for 2^X.

```
      )FNS POWER
∇ Z←POWER P
 [1] Y←1
 [2] D←1
 [3] K←1
 [4] A←Y*2
 [5] →((A-2)≥0)/L2
 [6] →(P[K]=0)/L3
 [7] D←D×.5
 [8] Y←Y+D
 [9] →3
 [10] L3:K←K+1
 [11] A←A*2
 [12] →(K>30)/L5
 [13] →5
 [14] L2:→(P[K]=1)/L4
 [15] D←D×.5
 [16] Y←Y-D
 [17] →3
 [18] L4:A←.25×(A*2)
 [19] K←K+1
 [20] →(K>30)/L5
 [21] →5
 [22] L5:Z←Y
∇
```

3. Program for cos x.

```
      )FNS COSINE
∇ Z←COSINE P
 [1] X←0
 [2] D←1
 [3] K←1
 [4] A←X
 [5] →(A≤0)/L2
 [6] →(P[K]=0)/L3
 [7] D←D×.5
 [8] X←X-D
 [9] →3
 [10] L3:K←K+1
 [11] A←(2×(A*2))-1
 [12] →(K>25)/L5
 [13] →5
 [14] L2:→(P[K]=1)/L4
 [15] D←D×.5
 [16] X←X+D
 [17] →3
 [18] L4:A←1-2×(A*2)
 [19] K←K+1
 [20] →(K>25)/L5
 [21] →5
 [22] L5:Z←X
∇
```

BIBLIOGRAPHY

[1] FIKE, C.T. Computer Evaluation of Mathematical Functions. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[2] GILL, S. A Binary Form of Horner's Method. Comput. J. 1 (1958), 58-59.

[3] LINISKII, V.S. The Calculation of Elementary Functions on Automatic Digital Computer (Russian). Vychislitel'naya Mathematika 2 (1957), 90-119. Translation, abridged, by K.W. Smillie. Quart. Bull. C.D.P.S.C. 2, 4(1962) 4-7; 3, 1(1962) 3-6; 3, 2(1963) 4-9.

[4] LYUSTERNIK, L.A. On the Computation of Values of a Function of a Single Variable (Russian). Matem. Prosvesheh. Vyp 3, 63-76 (1958).

[5] LYUSTERNIK, L.A., CHERVONENKIS, O.A., and YANPOL'SKII, A.R. Handbook for Computing Elementary Functions. Pergamon Press, New York (1965). English translation by G.J. Tee.

[6] MEGGITT, J.E. Pseudo Division and Pseudo Multiplication Processes. IBM Jour. Res. and Dev., 6 (1962).

[7] MEGGITT, J.E. Digit-by-Digit Methods for Polynomials. IBM Jour. Res. and Dev., 7 (1963) 237-245.

[8] MORRISON, D.R. A Method for Computing Certain Inverse Functions. MTAC, 10 (1956) 202-208.

[9] SALE, A.H.J. The Calculation of e to Many Significant Digits. Comput. J. 2, 2(1968) 229-230.

[10] WENSLEY, J.H. A Class of Non-Analytical Iterative Processes. Comput. J. 1, 4(1959) 163-167.

[11] WIJNGAARDEN, A. VAN Erreurs D'Arrondiment dans les Calculs Systématiques, C.N.R.S. Colloques Internationaux, 37 (1953) 285-293.

[12] WILKES, M.V. and WHEELER, D.J. Note on "A Method for Computing Certain Inverse Functions". MTAC 11, 59 (1957) 204.

[13] WILKES, M.V., WHEELER, D.J. and GILL, S. The Preparation of Programs for an Electronic Digital Computer. Addison-Wesley, Cambridge, Mass. 1951 (Part III, pp. 152-3).