CONVERGENCE GUARANTEE AND IMPROVEMENTS FOR A FAST HARDWARE EXPONENTIAL AND LOGARITHM EVALUATION SCHEME

Celia Wrathall* and Tien Chi Chen

IBM San Jose Research Laboratory
San Jose, California 95193

ABSTRACT

In one iteration, Chen's algorithm for evaluating exponentials and logarithms advances by 2 bits on the average, yet may not advance at all. Analysis reveals that the no-advance situation actually paves the way for sizable advance in the next iteration, and the guaranteed advance, after a one iteration overhead, is one bit per iteration.

Two new schemes raise the guaranteed advance to 1.5 bits per iteration, after a two-iteration overhead, while maintaining the original requirement of one stored constant per operand bit.

Adopting as a figure of merit the following quantity

$$Q = \frac{\text{advance per iteration}}{\text{memory words per operand bit}}$$

for the steady-state iterations, the new schemes appears to be better than other methods heretofore proposed.

1. Introduction

Chen's hardware-oriented scheme for evaluating exponentials and logarithms <1> has the merit of conceptual simplicity, small table size, and good average performance. It has been installed on at least one computer <2>.

For many realistic computing environments, a guaranteed performance is called for. The performance of Chen's scheme, however, is strongly dependent on the bit pattern of the input operand. Though an iteration brings the intermediate result two bit positions closer to the final answer on the average, it also may produce no visible advance at all.

An analysis shows, surprisingly, that this temporary standstill actually paves the way for a subsequent sizable advance. Further investigation shows the possibility of improving the guaranteed convergence rate by 50% without inflating the table size, at a small added cost in processing complexity.

The convergence analysis and the search for improvements are the subject of this communication.

2. Chen's scheme

Chen's algorithm involves the co-transformation of a number pair (x(k),y(k)), using a parameter m(k) found by examining x(k). It drives x(k) towards a known goal, and concurrently y(k) to the desired functional value. An "end-play" replaces half of the iterations by a truncated Taylor expansion, costing either an add (for logarithms) or a half-multiply (for exponentials).

Let

$$T(m,s) = -\ln(1+s.2^{**}-m)$$

be the possible pre-stored tabular values, also the signed fraction

$$\begin{aligned} u(k) &= x(k) && \text{for computing exp } x \\ &= 1 - x(k) && \text{for computing log } x \end{aligned}$$

We have

$$u(k) = g(k)\ .\ (2^{**}-m(k) + p(k))$$

with s, g(k) = +1 or -1
m, m(k) = positive integers
$0 \leqslant p(k) < 2^{**}-m(k)$

In Chen's method, s and g(k) are always positive, and m is chosen to match m(k).

The aim of the iterations is to drive u(k) towards the value zero. A convenient measure of the magnitude is m(k), the position of the leading 1-bit of the magnitude of u(k), counting rightward from the binary point. If

* Current address: Department of Mathematics, University of California at Santa Barbara, Santa Barbara, California.

$u(k)=1/2$, its leading 1-bit position is $m(k)=1$. An important measure of the efficacy of an iteration is the gain in leading 1-bit position, called the <u>advance</u> of the iteration.

$$A(k+1) = m(k+1) - m(k)$$

When the leading $\lceil N/2 \rceil$ bits are zero,

$$m(k) \geqslant 1 + \lceil N/2 \rceil$$

$u(k)$ is said to equal h, where

$$|h| < 2^{-N/2}$$

At this point an "end-play" based on a truncated Taylor expansion can be used to obtain the final result.

The iterations for evaluating $\exp(u)$ and $\ln(1-u)$ for the fraction u are given below.

****

THE EXPONENTIAL CASE

$$\begin{aligned}\exp u(0) &= w.\exp(u(0)) \\ &= y(0).\exp(u(0)) \\ &= (y(0).(1+2^{-m(0)})).\exp(u(0)- \\ &\qquad -\ln(1+2^{-m(0)})) \\ &= y(1).\exp(u(1)) \\ &= \ldots \\ &= (y(k).(1+2^{-m(k)})).\exp(u(k)- \\ &\qquad -\ln(1+2^{-m(k)}) \\ &= y(k+1).\exp(u(k+1)) \\ &= \ldots \\ &= y(n)\,.\,\exp h \\ &\sim y(n) + y(n).h\end{aligned}$$

with a relative truncation error measured by $(h^2)/2 < (2^{-N})/2$.

THE LOGARITHM CASE

$$\begin{aligned}\ln(1-u(0)) &= w + \ln(1-u(0)) \\ &= y(0) + \ln(1-u(0)) \\ &= (y(0)-\ln(1+2^{-m(0)}))+ \\ &\qquad +\ln[(1-u(0)).(1+2^{-m(0)})] \\ &= y(1) + \ln(1-u(1)) \\ &= \ldots \\ &= (y(k)-\ln(1+2^{-m(k)}))+ \\ &\qquad +\ln[(1-u(k)).(1+2^{-m(k)})] \\ &= y(k+1) + \ln(1-u(k+1)) \\ &= \ldots \\ &= y(n) + \ln(1-h) \\ &\sim y(n) - h\end{aligned}$$

with an absolute truncation error measured by $(h^2)/2 < (2^{-N})/2$.

****

3. <u>Convergence analysis of Chen's scheme</u>

Chen has found the average advance of his method to be 2. For some iterations, however, there may be no advance at all; curiously, the method seems to work even better after such a standstill.

3.1 Deliberate undercorrection

the quantity $u(k+1)$ depends on $u(k)$, but not on $y(k)$. Hence the convergence of Chen's scheme is seen by examining the transformation of $u(k)$ alone, as shown below.

a. Exponential case:

$$\begin{aligned}u(k+1) &= u(k) - \ln(1 + 2^{-m(k)}) \\ &= p(k) + RE(k)\end{aligned}$$

with

$$RE(k) = 2^{-m(k)} - \ln(1 + 2^{-m(k)})$$
$$0 < RE(k) < 2^{-(2m(k)+1)}$$

b. Logarithmic case:

$$\begin{aligned}u(k+1) &= u(k) - (2^{-m(k)})\,(1- u(k)) \\ &= p(k) + RL(k)\end{aligned}$$

with

$$RL(k) = u(k).2^{-m(k)}$$
$$0 < RL(k) < 2^{(-2m(k)+1)}$$

In both situations the largest bit in $u(k)$, with a magnitude of $2^{-m(k)}$, is singled out for destruction; the iteration destroys that bit, leaving the leftover bit pattern $p(k)$ plus a positive definite correction term (RE or RL, to be called R below) bounded by $2^{(-2m(k)+1)}$.

The scheme undercorrects for positive $u(k)$, making $u(k+1)$ positive also. As long as the $u(k)$'s remain positive, Chen's method never needs the tabular entries $T(m,s)$ with negative s.

The one-sided convergence means that the arithmetic unit need never perform any subtraction during iteration; this advantage is minor in view of the rapid progress of LSI electronics.

3.2. No advance brings good advance

The undercorrection also leads to the anomaly of zero advance. This is seen in Fig. 1a, where both $m(k)$ and $m(k+1)$ equal 6. Such an anomaly occurs when $p(k)$, the left-over bit pattern, has a value very close to $2^{-m(k)}$. The second order term (RE or RL), being positive, adds to $p(k)$ to create carry ripples, resurrecting the leading bit in $u(k)$ after the initial cancellation.

This standstill is actually a blessing in disguise. The rippling of carries necessarily leaves a long trail of zeros behind (Figure 1b). The next iteration will destroy the m(k)th bit, producing a very small result with about 2m(k) leading zeros. This loose discussion can be made rigorous by the following

THEOREM 1. If m(k+1) = m(k), then m(k+2) $\geq$ 2m(k)-1.

PROOF. We have

$$u(k+1) = 2^{**}-m(k+1) + p(k+1)$$
$$= 2^{**}-m(k) + p(k+1)$$

But $u(k+1) = p(k) + R(k)$
$< 2^{**}-m(k) + R(k)$
hence $p(k+1) < R(k) < 2^{**}(-2m(k)+1)$

Now $u(k+2) = 2^{**}-m(k+2) + p(k+2)$
but $u(k+2) = p(k+1) + R(k+1)$
$< R(k) + R(k+1)$
$< 2^{**}(-2m(k)+2)$
Thus $2^{**}-m(k+2) < 2^{**}(-2m(k)+2)$
hence $m(k+2) > 2m(k) - 2$

by taking logarithms of both sides. As m(k), m(k+2) are both small integers, the strict inequality means

$$m(k+2) \geq 2m(k)-1. \quad Q. E. D.$$

COROLLARY 1.1. If m(k) $\geq$ j $\geq$ 2, then either

$m(k+r) \geq j + r$
or $m(k+r) = j + r - 1$
but $m(k+r+1) \geq 2j + 2r - 3$

COROLLARY 1.2. If after k iterations, m(k) $\geq$ j $\geq$ 2, then for N > J $\geq$ j we have m(k-j+J+1) $\geq$ J. In other words, n, the minimum total number of iterations to yield m(n) $\geq$ J, is no greater than (k-j+J+1).

In Chen's method, the end-play is invoked when m(n) $\geq\lceil N/2+1\rceil$, whether N is even or odd. The number of iterations needed to reach this point is bounded by $\lceil k-j+N/2+2\rceil$.

Standard function-evaluation techniques map floating point numbers into the range [0, ln 2) for exponentials, and the range [0.5, 1) for logarithms. m(n) is bounded by $\lceil N/2\rceil$ for both cases, as shown below.

THEOREM 2. For N $\geq$ 8, using Chen's method for the exponential function with x(0) in [0, ln2), the number of iterations needed to reach m(n) $\geq\lceil N/2 + 1\rceil$ never exceeds $\lceil N/2\rceil$.

PROOF. It suffices to show that m(k)$\geq$k+2 for some k$\leq$2. For then j-k=2, and n = $\lceil k-j+N/2+2\rceil$ = $\lceil N/2\rceil$.

Detailed examination shows that there are only three processing patterns

a. m(0) $\geq$ 2
b. m(0) = 1, m(1) $\geq$ 3
c. m(0) = 1, m(1) = 2, m(2) $\geq$ 4

hence m(k)=k+2 for k=0,1,2 respectively. Q. E. D.

THEOREM 3. For N $\geq$ 6, in the evaluatiion of ln x starting with x(0) in [0.5, 1), the number of iterations before the end-play is no more than $\lceil N/2\rceil$.

PROOF. Here 0<u(0)=1-x(0)<0.5, and already and m(0)$\geq$2 for all cases but one. The lone exception is u(0) = 0.5, for which m(0) = 1, but m(1) $\geq$ 3. Q. E. D.

We observe incidentally that roundoff error will not invalidate the results, as the above theorems are protected by ample safety margins. Theorem 2 is valid for x as large as 0.7536, and Theorem 3 is valid for x as small as 0.3456.

4. The search for a better algorithm

The performance of Chen's method shows wide fluctuation; the above bound on total advance is only half of the expected average advance.

The next challenge is to find alternative methods with the same table economy as Chen's scheme, yet with stronger convergence guarantee. Despite the seeming lack of extra freedom, the answer turns out to be in the affirmative.

To obtain faster guaranteed convergence, the undercorrection in Chen's method should be replaced by a measure of overcorrection. But this latter tends to change the sign of u(k), which calls for both types of table entries, T(m,+1) and T(m,-1), for exponentials in the reduction of u(k), and for logarithms in the co-transformation of y(k). Since Chen's method uses T(m,+1) exclusively, the addition of T(m,-1) could double the cost for table storage.

It turns out that there need be no table-size increase, if one merely changes a portion of Chen's table of T(m,+1), to the alternative form T(m,-1), such that over a small range of m, both types of table entries are present, but not for the same m. This way a need for a given type of table entry will be satisfied, though the entry of the desired type may not match the desired m(k) exactly.

We now assume both positive and negative values for u(k):

$$u(k) = g(k).(2^{**}-m(k) + p(k))$$

The table entries will be taken as

$$\{T(m)\} = \{-\ln (1 + s(m).\ 2^{**}-m)\}$$

The magnitude of s(m) is 1 in all cases, but the signs are to be decided later.

4.1 A first-order theory

An iteration using the signed u(k) can be written as

$$u(k+1) = u(k) - s(m).2^{**}-m + O(2^{**}-2m)$$

A first-order theory, based on ignoring the second order O(2**-2m) terms, makes the problem vastly more tractable: an iteration has the simple effect of adding a signed bit to u(k) at a known offset.

The design of the best scheme (to first order) consists of choosing the T(m)'s which cancel bits in u(k) most effectively.

We have limited our choices to the cases with s(k)=s(k+q) for small integer q. All cases for q up to and including 5 have been studied to first order; the most fruitful one being with q=3, which is examined below.

4.2. The cancellation of octal digits.

For the case of q=3, it is most convenient to consider u(k) as a sequence of octal digits, thus

$$\begin{aligned} u(k) &= U(k).2^{**}(d-3j) \\ &= g(k).(D(k)+P(k)).2^{**}(d-3j) \end{aligned}$$

with $0 < D(k) \leqslant 7$
$0 \leqslant P(k) < 1$

where d = 0, 1 or 2 is an integer indicating the displacement of the octal digit boundary. The simple choice d = 0 turns out to be adequate.

For s(k)=s(k+3), there are only the following nontrivial cases, plus cyclic permutations already allowed for by d.

s(1), s(2), s(3)	
= +1, +1, +1	(the "ppp" scheme)
= +1, +1, -1	(the "ppn" scheme)
= -1, -1, +1	(the "nnp" scheme)
= -1, -1, -1	(the "nnn" scheme)

The ppp scheme is just the unmodified Chen algorithm. The nnn scheme handles positive u(k)'s poorly, and needs extra work to preserve the negativity of the latter; it will not be discussed further. The ppn and nnp schemes can be treated together; the former operating on U(k) has the same first-order behavior as the latter operating on -U(k). We assume zero displacement unless stated otherwise.

One can now take the bit pattern of U(k), and examine the effort required to cross out the nonzero bits, by adding one signed bit at a time. The result for nonzero P(k)'s are shown in Table I, and a graphical view is given in Fig. 2. It is clear that two iterations at most, will clear an octal digit to first-order. The case of P(k) = 0 occurs rarely, and need not be considered in the first-order theory.

Assuming all 16 cases to be equally distributed, a total of 48 bits are cleared in 22 iterations, for an average advance of 2.18 bit positions per iteration. This average advance is only marginally better than Chen's original method. The guaranteed sustainable advance, however, is fully 50% better.

5. Guaranteed convergence of the new algorithm.

We have made an extensive study of the ppn, nnp schemes. The analysis resembles that for the Chen algorithm, but requires more details based on an expanded notation. The results are stated as theorems without proof. Zero displacement is assumed throughout.

THEOREM 4. The ppn scheme clears D(k) in no more than three iterations.

THEOREM 5. For j>2, if the ppn scheme requires three iterations to clear D(k), then it takes no more than one iteration to clear the next octal, namely D(k+1).

THEOREM 6. Using the ppn scheme for exponentials with x(0) in [0, ln 2), or for logarithms with x(0) in [.5, 1), then it takes no more than $2+2.\lceil N/6 \rceil$ iterations to clear the first $\lceil N/2 \rceil$ bits.

THEOREM 7. Using the nnp scheme for exponentials and logarithms, it takes no more than three iterations to clear an octal digit.

THEOREM 8. For j>2, if it takes three iterations to clear D(k) using the nnp scheme, then the next octal digit is automatically cleared by the third iteration.

THEOREM 9. Using the nnp scheme for

logarithms with $x(0)$ in $[.5,1)$, or for exponentials with $x0$ in $[0,-\ln 2)$, it takes no more than $2+2.\lceil N/6 \rceil$ iterations to clear the first $\lceil N/2 \rceil$ bits.

6. The Fast ppn Algorithm

The ppn scheme for zero displacement is given below. The hardware (or equivalents) consists of

```
a table of ⌈N/2⌉ words,
  T(m)=-ln (1+s(m).2**-m)
  with s(m) = -1, when 3 divides m
            = +1  otherwise;
an adder (which can add and subtract);
a multiplier (perhaps shared);
a shifter which can
  select the leading nonzero octal
  and also note its position;
an octal position register J;
data registers X, Y;
short temporary registers:
  M (N/2 bits, for table address m)
  D (3 bits, for octal digit D(k))
  S (one bit, for sign of fraction).
```

****

THE ppn EXPONENTIAL ALGORITHM

```
E0.  Load the input x into X,
     set C(Y) to 1.

E1.  Fetch C(X),
       copy its sign into S,
     locate the leading octal digit,
       store its position (3j) in M,
       and put digit in D.

E2.  If C(M) ≥ N/2 + 1,
       go to Step EN for end play;
       else
         if C(D)≥+3,       subtract 2 from M
         if +3>C(D)≥+1,    subtract 1 from M
         if -1≥C(D)>-3,    leave M unchanged
         if -3≥C(D),       subtract 3 from M.

E3.  Use C(M) as address m to fetch T(m).

E4.  C(X)+T(m) becomes the new C(X).

E5.  Fetch a copy of C(Y),
     right-shift by m positions,
     change its sign if S is negative,
     result + C(Y) becomes new C(Y).

E6.  Go to E1.

EN.  (End play)
     C(Y) + C(Y).C(X) gives the answer.
```

THE ppn LOGARITHM ALGORITHM

```
L0.  Load the input x into X,
     set C(Y) to 0.

L1.  Fetch C(X), find u=1-C(X),
       put sign of u in S,
     locate the leading octal digit of u,
       store its position (3j) in M,
       and put digit in D.

L2.  If C(M) ≥ N/2 + 1,
       go to Step LN for end play;
       else
         if C(D)≥+3,       subtract 2 from M
         if +3>C(D)≥+1,    subtract 1 from M
         if -1≥C(D)>-3,    leave M unchanged
         if -3≥C(D),       subtract 3 from M.

L3.  Fetch a copy of C(X),
     right shift by C(M),
     change sign iff S is negative,
     add to C(X) to become new C(X).

L4.  Use C(M) as address m to fetch T(m).

L5.  C(Y) + T(m) becomes new C(Y)

L6.  Go to Step L1.

LN.  (End play)
     C(Y) + (1 - C(X)) gives the answer.
```

****

The algorithm is more complex than Chen's scheme, but then the performance is under much better guarantee. Also, there are many alternatives to the one above. The latter is chosen mainly for clarity of exposition.

7. Summary and Conclusions

We have examined the convergence characteristics of Chen's method and found that it can guarantee the steady-state advance of one bit per iteration. Essentially the same analysis was used to study improvements, keeping the size of the table the same as in Chen's method. The new ppn and nnp algorithms guarantee an advance of 1.5 bits per iteration, with an overhead of two extra iterations. The hardware scheme in the new methods are more complicated, but the reward, in terms of guaranteed convergence, is much higher. The theorems developed here have not allowed for roundoff error, and may not hold strictly for some machines if $m(k)$ is too close to N; this problem is always solvable by using guard digits. The use of end-plays also confines $m(k)$ essentailly to the left half of the fraction.

We now consider briefly the relative merit of the methods. In the new LSI world, the cost of arithmetic is rapidly dropping, so is the cost of memory. We note, however, that hardware can often be shared with other equipment, but the table of logarithms needed for exponentials and logarithms is seldom needed elsewhere. For fraction lengths of about 50 bits, the size of the table is about 25 words, a little more than a thousand bits. This is still not a trivial amount of memory. For high performance, this table should be permanently installed in high-speed read-only memories, carrying a fixed cost. In any case, the memory cost appears to be the only tangible cost accountable.

The performance of a function-evaluation algorithm is intimately connected to the amount of memory used. As an outrageous example, 2**50 table entries can evaluate any 50-bit function in one memory operation. To allow for the variation in word-length, the memory cost can be expressed in the amount of memory used for each bit of the operand.

An adequate measure of performance is the advance per iteration, as defined in Section 3. In the steady-state, the guaranteed advance per iteration is unity for Chen's original method, which uses a table entry for each bit in the frontal half of x. The new schemes has a 50% higher guarantee.

De Lugish <3> uses two memory words per bit of operand, for an average advance of three bits per iteration. He guarantees a one bit per iteration advance; the actual advance in the steady-state is <u>twice</u> as good as claimed. His scheme involves iterations for the entire fraction length, and does not use the end-play, but the latter clearly can be installed without any difficulty.

Using as a figure of merit

$$Q = \frac{\text{advance per iteration}}{\text{memory words per operand bit}}$$

for the steady-state, a brief comparison of the three methods are given in Table II. It is significant that for the ppn and nnp methods, Q is guaranteed to be 1.5, and is 2.2 on the average. We are aware of no other approaches with comparable advantage.

REFERENCES

1. T. C. Chen, "Automatic Computation of Exponentials, Logarithms, Ratios, and Square Roots," IBM J. Research Develop. Vol. 16, 380-388 (1972).

2. T. H. Kehl and K. Burkhardt, "A minicomputer Microprogrammable, Arithmetic Processor," Proc. Third Symposium on Computer Arithmetic, Dallas Texas, Nov., 1975, pp.174-178.

3. B. G. De Lugish, "A Class of Algorithms for Automatic Evaluation of Certain Elementary Functions in a Binary Computer," Report No. 399, Depart of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, June 1970.

TABLE I.

THE CANCELLATION OF AN OCTAL DIGIT

(-1 is represented by T; $0 < P = P(k) < 1$, $Q = 1 - P$)

T(m,s) = (T,1 1 T)	ITERATIONS	REMAINDERS	COUNT
U(k) = - 1 1 1 . P =	(T,000)	+ 0 0 0 . Q	1
- 1 1 0 . P =	(T,000) + (010)	- 0 0 0 . P	2
- 1 0 1 . P =	(T,000) + (010)	+ 0 0 0 . Q	2
- 1 0 0 . P =	(T,000) + (100)	- 0 0 0 . P	2
- 0 1 1 . P =	(T,000) + (100)	+ 0 0 0 . Q	2
- 0 1 0 . P =	(00T) + (00T)	- 0 0 0 . P	2
- 0 0 1 . P =	(00T)	+ 0 0 0 . P	1
- 0 0 0 . P =		- 0 0 0 . P	0
0 0 0 . P =		+ 0 0 0 . P	0
0 0 1 . P =	(010)	- 0 0 0 . Q	1
0 1 0 . P =	(010)	+ 0 0 0 . P	1
0 1 1 . P =	(100)	- 0 0 0 . Q	1
1 0 0 . P =	(100)	+ 0 0 0 . P	1
1 0 1 . P =	(100) + (010)	- 0 0 0 . Q	2
1 1 0 . P =	(100) + (010)	+ 0 0 0 . P	2
1 1 1 . P =	(100) + (100)	- 0 0 0 . Q	2

TABLE II.

COMPARISON OF THREE SCHEMES

(Steady-state performance)

	Chen	De Lugish	ppn or nnp
Guaranteed Advance	1	2	1.5
Average Advance	2	3	2.2
Memory Words per Bit	1	2	1
Guaranteed Merit	1	1	1.5
Average Merit	2	1.5	2.2

```
u(k)          =   .0 0 0 0 0 1̸ 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 ...
change(k)     =              -1̸               + 1 1 1 1 1 1 0 0 ...

                         a.  Recreation of destroyed 1-bit.
                             m(k) = m(k+1) = 6.


u(k+1)        =   .0 0 0 0 0 1̸ 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 ...
change(k+1)   =              -1̸               + 1 1 1 1 1 1 0 0 ...

                         b.  Limited carry on re-iteration.


u(k+2)        =   .0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 ...

                         c.  No advance leads to good advance.
                             m(k+2) = 12 > 2m(k) - 1.
```

FIGURE 1. The standstill case in Chen's scheme.

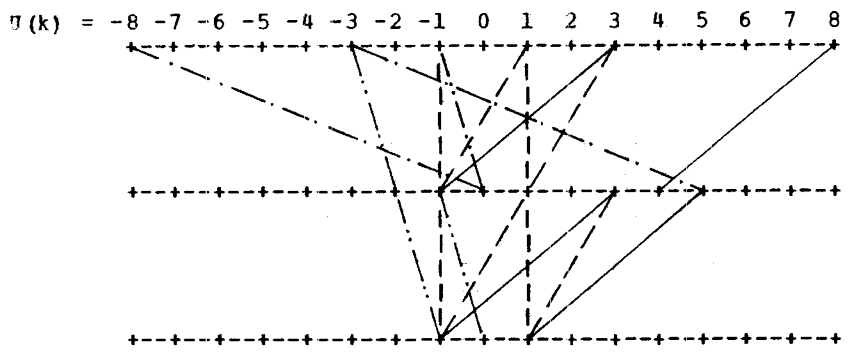

FIGURE 2. The cancellation of an octal digit by ppn iterations.