Letters to the Editor

Improvements Based on Computed Errors

EDITOR:

The paper "On the Computation of Least Squares Polynomials," by Morton Goldberg [*Comm. ACM 10*, 1 (Jan. 1967), 56–57] contains an interesting idea about the possibility of improving a computed least squares polynomial to obtain a better approximation to the true least squares polynomial, the difference between the two being due to errors accumulated during digital computations. The procedure suggested may be summarized as follows: a computed polynomial $p(x)$ fitted to the set of points $S = \{(x_n, y_n): n=1, \cdots, N\}$ can be improved by modifying $p(x)$ to $p(x) + \bar{\delta}$, i.e., by adding $\bar{\delta} = 1/N \sum_{n=1}^{N} (y_n - p(x_n))$ to the zeroth degree coefficient.

Unfortunately the better approximation to the true polynomial which was promised seems to me difficult to obtain by this method on a digital computer. The numerical example in Table I shows that the method can lead to an approximation to the true polynomial that is worse than $p(x)$. The problem solved is the following: the fitting of the polynomial of zero degree, in the least squares sense, to the data y_n. The true polynomial to be found is $P(x) = b$ and the "normal" equation to be solved is

$$\sum_{n=1}^{N} y_n - Nb = 0.$$

The computations are performed with nonrounding floating point operations (operation symbols given in circles), a mantissa of only one decimal digit, and no limitation on the length of the exponent. With the exceptions of the base and the word length, the arithmetic used is the nonrounding normalized floating point arithmetic of the IBM 7094 computer, and operations are performed in two registers working in the same way as the AC and the MQ register of this computer. In Table I the data and the results of the operations are machine numbers; furthermore, in Table I we assign to a^2 the meaning $a \otimes a$ and to $\sum_{n=1}^{N} \oplus a_n$ the meaning of the floating point summation, performed in the order $(\cdots((a_1 \oplus a_2) \oplus a_3) \cdots \oplus a_n)$.

TABLE I

n	1	2	3	4
y_n	5.	−5.	20.	0.
$\sum_{n=1}^{4} \oplus y_n = 20.$ $b_1 = 20. \oslash 4. = 5.$				
$y_n \ominus b_1$	0.	−10.	10.	−5.
$(y_n \ominus b_1)^2$	0.	100.	100.	20.
$\sum_{n=1}^{4} \oplus (y_n \ominus b_1)^2 = 200.$ $\bar{\delta} = \sum_{n=1}^{4} \oplus (y_n \ominus b_1) \oslash 4 = -1.$				
$b_2 = b_1 \oplus \bar{\delta} = 4.$				
$y_n \ominus b_2$	1.	−9.	10.	−4.
$(y_n \ominus b_2)^2$	1.	80.	100.	10.
$\sum_{n=1}^{4} \oplus (y_n \ominus b_2)^2 = 100.$				

The first result, $b_1 = 5.$, *is* the best approximation to the *true* polynomial, being better than the "improved" result $b_2 = 4$. What has apparently been improved in this example is the computed sum of the square deviations (a satisfying result if we solve such problems by minimum direct search!). However it has been achieved at the expense of the best approximation. This is not surprising since the method suggested states things about numbers and operations and not about machine numbers and machine operations.

It is a simple matter to give an algorithm to produce similar examples for computations with larger mantissas, of course. We may conclude that when the method is applied to digital computers the "improvement" need not be taken literally, but rather in a probabilistic sense. What the probability is of obtaining an improvement based on an accuracy measure such as the computed $\bar{\delta}$ (a sum of deviations with different signs, whose value should be zero) I do not know.

I am not so pessimistic as to believe that we can do nothing against the metamorphosis of theorems in probable theorems in digital computations. A more adequate formalism, like that used by Mr. Goldberg in the second part of his paper to show that the procedure cannot be repeated, could prevent us from making hasty conclusions.

ARTENIO DE MATTEIS
Centro di Calcolo del CNEN
Bologna, Italy

A Rational Approximation Optimal by Moursund's Criterion

EDITOR:

In his interesting article [1] on the subject of optimal starting approximations for square-root calculation, Professor Moursund gave several illustrative polynomial approximations but no rational approximations. In order to obtain an illustrative rational approximation that would be optimal by the criterion of the article, I wrote an *ac hoc* computer program based on the technique of equating maxima, with which I obtained the approximation

$$R^*(x) = 1.68212586 - \frac{1.28977371}{x + .84106293},$$

where the coefficients have been rounded to eight decimal places. Although there is no supporting proof, it seems reasonable to believe that for a rational approximation $R(x)$ expressible in the form $A + B/(x+C)$, the maximum in the interval [1/16, 1] of

$$\left| \frac{\frac{1}{2}\left[R(x) + \frac{x}{R(x)} \right] - \sqrt{x}}{\sqrt{x}} \right|$$

is as small as possible when $R(x)$ is $R^*(x)$. The maximum of the

above expression, when $R(x)$ is replaced by $R^*(x)$, is approximately $2^{-12.496}$ for $1/16 \leqq x \leqq 1$. For the comparable starting approximation given in [2], namely (IIa), the maximum relative error after one iteration with Newton's method is approximately $2^{-12.470}$.

REFERENCES:

1. MOURSUND, D. G. Optimal starting values for Newton-Raphson calculation of $\sqrt{x}$. *Comm. ACM 10*, 7 (July, 1967, 430–432.
2. FIKE, C. T. Starting approximations for square-root calculation on IBM System/360. *Comm. ACM 9*, 4 (April, 1966), 297–299.

C. T. FIKE
IBM Systems Research Institute
New York, N. Y. 10017

Remark on Langdon's Algorithm

EDITOR:

This letter concerns the paper entitled "An Algorithm for Generating Permutations" by G. G. Langdon Jr. [*Comm. ACM 10* (May 1967), 298].

Although the algorithm given is simple to describe, its efficiency in terms of the number of transpositions required to generate a complete set of $K!$ permutations is very poor. A number of simple and efficient algorithms have been published in the Algorithms section of the *Communications*. Algorithm 115 requires only $K!$ transpositions. Several lexicographic generators have been described. Lexicographic generation requires a number of transpositions which tends asymptotically to $1.53K!$ This algorithm, however, requires transpositions in excess of $(K-1)K!$ There does not appear to be any combinatorial advantage of circular ordering over lexicographic order.

R. J. ORD-SMITH
University of Bradford
Bradford, England

M in Z: A Game for Man-Machine Studies

EDITOR:

Readers of *CACM* will be interested in the great flexibility of using the game "M in Z" for introductory studies in artificial intelligence. Special instances of the game are Tic-Tac-Toe ($M = 3$, $Z = 3$) and the famous oriental game GOMOKU or RENJU ($M = 3$, $Z = 19$).

Optimal methods of playing can be easily determined for $M = 3$ and $M = 4$. However, when $M \geqq 5$ no general optimal methods are known. (It is not known, for example, whether the first player can ensure a win in GOMOKU.) Readers not familiar with GOMOKU may try to play a simplified version of the game on an 8×8 chess board. They will quickly discover that this is not a trivial game.

At Hopkins, the game is being used successfully both on a time sharing system (GE 265) and on a batch processing system (IBM 7094). Although the time sharing system is restricted to programs of 6000 characters, we have been successful in implementing the following variations:

1. A general program that never loses when $M = 3$ and $M = 4$. It plays a challenging game when $M = 5$. This program has won many GOMOKU games against good opponents each lasting about half an hour and requiring about 2 minutes of CPU time.
2. A program that can learn to improve its strategy. In a reasonable number of games with $M \leqq 4$, the program can ensure no losses against good opponents.

The learning is accomplished by changes in the probabilities of selection of strategies. A win (or a loss) with a specific strategy increases (or decreases) its probability of being selected. A simple device allows one parameter to determine a unique strategy. The parameters are supplied as data. Other variations can easily be programmed.

3. A program that can "look ahead." (In the previous two programs the "look ahead" feature is not incorporated.) This feature greatly improves the play of the machine, naturally, but it also takes much longer to run.

The "look ahead" feature examines all available plays by one player, it finds the "best" response by the opponent and then computes a "utility" for each next play. The play with the highest "utility" is then selected.

Heuristics can easily be added to this program.

All programs allow students to play against one another or against a variety of machine strategies. This can be done online or offline. Games may start after any number of predetermined moves.

Sample outputs, as well as BASIC tapes and FORTRAN cards, are available for distribution. They may be used for demonstrations (man against machine), for instruction (students can easily incorporate their own strategies) and for research in machine learning and the development of heuristics.

ELIEZER NADDOR
The Johns Hopkins University
Baltimore, Maryland 21218

Character Coding for Information Processing Interchange

EDITOR:

The International Organization for Standardization has issued a draft recommendation [No. 1052] for 6- and 7-bit coded character sets for information processing interchange [ISO/TC 97 (Secr-90 141E, June 1966]. In this recommendation, a number of characters needed for computer programming are not defined. The result of this is that programming groups are forced to make their own choices of codes for these characters. This is equivalent to making graphic substitutions, whether or not the printing device actually prints the programming character. The ISO standard forbids reallocation of any character already in the code table but does not control the allocations for other characters. The only general rule we can follow is that allocating such characters is a matter of international convention for programming usage, and the characters affected should not lie in "National Use" positions.

In an attempt to avoid the confusing situation of these characters being substituted at different positions by different groups, I submit the attached proposal for alternatives to the standard characters, which are introduced for programming, and would normally be used in connection with computers rather than for general information interchange. The various alternative characters are independent of one another and are to be used only by agreement between sender and recipient. It is an objective of this proposal to get into the center 64 positions of the code table the principal characters needed, on the basis that simple equipment will be able to print only these.

Proposed alternative characters for programming in ISO7: comparison with Standard ISO7

Bit Pattern	Position in table	ISO7		Alternative for programming
0100001	2/1	!	|	Vertical line (PL/I)
0100010	2/2	"	$\vee$	Logical Or (ALGOL)
0100101	2/5	%	$\supset$	Implies (ALGOL)
0100110	2/6	&	$\wedge$	Logical And (ALGOL)
1000000	4/0	@	10	Decimal exponent (ALGOL)
1011110	5/14	^	$\neg$	Logical Not (ALGOL and PL/I)

I. C. PYLE
AERE
Harwell, England