An adjustment must be made for an odd exponent in the argument, where K is the original multiplier.

$$\begin{aligned}\sqrt{X} &= \sqrt{Y} \times \sqrt{1/K} \quad \text{(even exponent)} \\ &= \sqrt{Y} \times \sqrt{2/K} \quad \text{(odd exponent)}\end{aligned}$$

Since

$$\sqrt{Y} = \frac{A + \frac{Y}{A}}{2},$$

$$\sqrt{X} = \left(A + \frac{Y}{A}\right)\sqrt{1/4K} \text{ (even exponent)},$$

$$= \left(A + \frac{Y}{A}\right)\sqrt{1/2K} \text{ (odd exponent)}.$$

Thus the final multiplication is done selectively (and indexed). Note that this correction for odd exponent would have to be done in any event, but here it is automatically incorporated into multiplication required in any case. A possible convenience in actual coding is to use (−A) throughout and change the sign of the final multipliers. Properly coded, this method can give near minimum execution times. When applied to the UNIVAC 1107, a time of 144μsec was achieved.

General

If this method is applied to many functions in a library package, the reduced range yields a saving in the number of coefficients stored for the polynomials, and the table of multipliers is amortized over the several routines.

REFERENCES

1. BEMER, R. W. A machine method for square root computation. *Comm. ACM 1* (Jan. 58), 6–7.
2. ——. A subroutine method for calculating logarithms. *Comm. ACM 1* (May 58), 5–7.
3. ——. Editor's note on series approximation truncation. *Comm. ACM 1* (Sept. 58), 3–6.

A Penny-Matching Program*

ELIZABETH WALL AND RICHARD M. BROWN
University of Illinois,† Urbana, Illinois

The logic of a penny-matching program written for the CSX-1 is described.

The penny-matching game[1] is one of the simplest learning programs that can be written for a computer. It is also an effective demonstration device for student groups in that the play is simple and the adaptive behavior readily perceived. Notwithstanding this and the fact that many computer installations have such a program hidden in a corner of their libraries, exceedingly little on such programs can be found in the literature; Hagelbarger's paper on SEER [1], which was a special device, is the only detailed description known to the authors. The present paper describes the logic of a penny-matching program written for the CSX-1 computer at the University of Illinois Coordinated Science Laboratory.

Computer Program Logic

The penny-matching program looks for correlations between the opponent's choices and the pattern of moves of both players for the previous games. To this end it generates estimates, ρ, of the probability that the opponent will play heads given the pattern b from the last n games. These estimates, ρ, are corrected and stored following each play. Using these ρ's, the program selects an appropriate move so as to match or mismatch according to its role in the game.

Since an estimate must be stored for every possible pattern of the last n games, obviously memory capacity limits the number of previous games which can be considered. In this program only the last six games are considered; furthermore, for greater flexibility, values of ρ for all n's from 0 up to 6 are generated, so that a total of 5461 values have to be stored (the value $n = 0$ considers only the opponent's previous play, irrespective of the program's play).

Initially, it is assumed by the program that its opponent's play is random; that is, all of the program's ρ's are set equal to one-half. After each game, however, the program extracts from the memory the seven ρ's corresponding to the patterns of the six games preceding the one just played. It then modifies the ρ's according to the actual last play using the following rules:

$$\rho_{i+1}\begin{cases} = \rho_i + k(1 - \rho_i) & \text{if opponent played heads,} \\ = \rho_i - k\rho_i & \text{if opponent played tails.} \end{cases} \quad (1)$$

It can be shown that if the opponent plays heads with a constant probability, P, the estimate ρ will eventually stabilize to an average value P. The smoothing constant, k, determines the rate of learning of the opponent's bias towards a given pattern. If the bias changes from a value P_0 to P_1, then the estimate of that bias will move towards the new value P; the average number of recurrences, I, of the pattern required for the estimate to move $\frac{1}{2}$ the distance to the new value is given by:

$$I = -\ln 2/\ln(1 - k) \quad (2)$$

* This work was supported by the U. S. Army Signal Corps, the Office of Naval Research, and the Air Force Office of Scientific Research.

† Coordinated Science Laboratory.

[1] This is a two person game where on each move the players each choose one of two alternatives (heads or tails); the win or loss is determined by the matching of the choices, one player having previously been selected to try to match.

Typical values for I are:

K	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
I	1	2	5	11	22	44

It should also be noted that for nearly random play, any given pattern of six games will occur $\frac{1}{64}$ as frequently as a pattern for one game only. Thus the rate of pattern learning also depends on n, the depth of games considered in the pattern; for equivalent rates of learning at all depths, the smoothing constants should vary according to factors of 2^n.

At the beginning of each play, the program must determine its own choice of heads or tails. To do this it first extracts from memory the seven values of ρ appropriate to the play of the preceding games. The program selects that ρ which deviates most from the value $\frac{1}{2}$—i.e. that ρ which most strongly suggests a bias on the part of the opponent. The other ρ's are discarded.[2]

To determine its own play the program generates a random number, R $(0 \leqq R < 1)$, and compares it to the selected most deviant ρ. It then plays heads for $R < \rho$ (assuming matching is desired), so that its own play follows a random strategy biased in accordance to the estimated bias of its opponent.

In summary of the entire cycle, the program first examines its estimates of bias on the part of its opponent towards heads or tails. The most strongly indicative estimate then determines the program's own bias in its next play. The opponent's actual play is ingested and used to adjust the bias estimates. After storing these new estimates, the program constructs the description of the pattern for the new set of previous six games, and the process begins over again.

[2] This process resembles that for the "demons" in Selfridge's *Pandemonium*, wherein the loudest demon is listened to.

Results of Play

The program has been run against human opponents and against other versions of itself. As expected, in play against humans, patterns of play were ultimately detected to the advantage of the program. In machine-machine play, two results are of interest. First, unlike human behavior, any bias in the randomness of play of one machine (produced with a biased random number generator) will be compensated for to a very high degree by the adaptive mechanisms of the program. For example, a machine playing with a bias of 5:1 towards tails will lose to an unbiased machine only by a ratio of 53:47.

A second result of interest is the fact the values of the learning time constants, k, did not affect the scores. This is in contradiction to results reported by Shannon [2] for play between an undescribed machine and Hagelbarger's SEER, wherein the faster responding machine beat its slower opponent by a ratio of 55:45. In our program the ability to learn was the only apparent factor, not the rate of learning.

REFERENCES

1. HAGELBARGER, D. W. SEER, a SEquence Extrapolating Robot. *Trans. IRE EC5* (Mar. 1956), 1–6.
2. SHANNON C. E. Game playing machines. *J. Franklin Inst. 260*, (Dec. 1955), 447–453.

A Computer Program for Analysis of Variance for a 2-Level Factorial Design

CATHERINE BRITTON AND I. F. WAGNER

Johns Hopkins University Applied Physics Laboratory Silver Spring, Maryland

Although many computer programs exist for handling analysis of variance, most of them employ the conventional methods described in statistics books for finding the sums of squares.

Yates [1] presented a simplified method for calculating the main effects and interactions for a two-level factorial experimental design. This method is readily adapted to computer use and involves only simple arithmetic operations. It has been programmed in FORTRAN and can be used to calculate the mean squares for all of the main effects and interactions of a two-level factorial design with as many as eight factors (variables) and five replications. A minor modification to the program could extend the number of variables and replications, limited only by machine storage capacity.

To simplify the discussion, the following terms are defined:

$N \equiv$ number of factors (variables) being tested
$R \equiv$ number of replications of each treatment combination
$2^N \equiv$ total number of treatment combinations
$R \times 2^N \equiv$ total number of responses
$M_{ij} \equiv$ a matrix whose maximum number of rows (i) equals 2^N and maximum number of columns (j) is C_{R+N+4}.

The program consists of constructing a $2^N \times C_{R+N+4}$ matrix from the $R \times 2^N$ responses using only simple arithmetic operations. The first column of the matrix (M) represents the responses from the first replication of the 2^N treatment combinations and must be arranged in standard order. The generally accepted statistical definition of standard order is used, see column 1 of Table I. Column two of the matrix (M) represents the responses from the second replication. The successive columns similarly represent responses for the additional replications and provide a total of C_R columns, $R \leqq 5$.

The remainder of the columns in the matrix (M) is