Steps	Description of Operation
8, 11–13	Compute $\cos x - \sin x$ and store
14–16	Compute $\cos x + \sin x$ and store
17–23	Compute $\cos 2x$ and store
24–27	Compute Z_{n+1} and store
29–45	Compute $\sin 2x$ and store
46–48	Modify b_n
50–60	Multiply Z_N by $\pi/2^N$

ROBERT L. LAFARA

U. S. Naval Ordnance Plant
Indianapolis 18, Indiana

[1] D. TEICHROEW, "Use of continued fractions in high speed computing," *MTAC*, v. 6, p. 127–133.
[2] C. HASTINGS, *Approximations in Numerical Analysis*. (Form 15's). Rand Corporation, Santa Monica. Sheets 8–12, 35–37, 39.
[3] W. P. HEISING, "An eight-digit general-purpose control panel," IBM Tech. News Letter No. 3, 1951.
[4] D. W. SWEENEY, "A Model II four address floating-decimal coding system," IBM Tech. News Letter No. 5, 1953.

A Partitioning Method of Inverting Symmetric Definite Matrices on a Card-Programmed Calculator

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 $n_2[n_1 + n_2 = n]$. 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

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

Solutions of these equations are

$$(1)\qquad \begin{aligned} D &= (d - ca^{-1}b)^{-1} \\ C &= -a^{-1}bD \\ B &= -Dca^{-1} \\ A &= a^{-1} - a^{-1}bB. \end{aligned}$$

Thus to find the inverse M^{-1}, the procedure is to determine a^{-1} from which we can derive D, C, B, and finally A. If the original matrix is symmetric, equations (1) simplify to

$$(2)\qquad \begin{aligned} D &= (d - b'a^{-1}b)^{-1} \\ B' &= C = -a^{-1}bD \\ A &= a^{-1} - a^{-1}bB. \end{aligned}$$

The restriction to definite matrices M assures us that $(d - b'a^{-1}) \neq 0$.

Let b and c be a column and row vector, respectively, and d a scalar; then the solution of (2) is obtained relatively simply once a^{-1} is known. (Note that D is merely the reciprocal of a scalar.) a^{-1} can be found by partitioning a in an analogous manner and a^{-1}, which is of the order $n - 1$, depends on the inverse of an $n - 2$ submatrix. The process carried to its completion implies that the inverse of the original matrix M is a function of the inverses of smaller and smaller submatrices. Thus a well known method for inverting matrices is suggested: start by inverting a small submatrix, say of order 2; from the 2×2 inverse, derive the 3×3 inverse, and continue until the desired $n \times n$ inverse is computed. This method turns out to be efficient and very simple to handle on an automatic computer.

The programming suggested assumes the following C.P.C. set-up: (a) Products can be accumulated in the 605 Electronic Calculating Unit counter; e.g., if $\alpha = x_1y_1$ is derived on card p and $\beta = x_2y_2$ is derived on card $p + 1$, then on card $p + 1$, $\alpha + \beta$ can be accumulated in the counter. In other words, the C.P.C. must be able to perform both a multiplication and an addition on one card.[1] (b) A second required feature is that a card may be read from either a normal or an alternate instruction field.

The procedure employs $(2n - 1)$ storage locations.

The manual operating scheme is:

1. Load the 527 Summary Punch with cards pre-punched in their instruction fields. There are n stages and at the r^{th} stage, the 527 summary punches r^2 cards which are fed back into the 418 Tabulator at stage $(r + 1)$; the r^2 cards are the r^{th} order inverse a^{-1}.

2. At the beginning of the r^{th} stage, the operator removes $(r - 1)^2$ cards summary punched on the $(r - 1)^{\text{th}}$ stage and runs them through the collator which assembles them in row-by-row sequence. These $(r - 1)^2$ cards are manually put in the program deck for stage r. The first part of this deck is run through the tabulator; when the $(r - 1)^2$ cards have gone through the tabulator once, they are removed and placed at the end of the program deck for the r^{th} stage and the deck finishes its run. A new set of r^2 cards will have been summary punched and the operator is ready to go on to stage $(r + 1)$.

The table below gives the functional computing procedure at the r^{th} stage. The following notation is used: x is a storage bank having n locations $[x + 1, \cdots, x + n]$ and y a storage bank having $n - 1$ locations, $[y + 1, \cdots, y + (n - 1)]$.

Table 1. Symmetric Matrix—Stage r

Step	Function	Put into Storage Location	Operation	Number of Cards
1	Load b	$x + 1$ ⋮ $x + (r - 1)$	(Add)	$r - 1$
	Load d	$x + r$	(Add)	1
2	Form $a^{-1}b$	$y + 1$ ⋮ $y + (r - 1)$	Row by row, the $(r - 1)^2$ cards containing a^{-1} are put through the tab. The first instruction field is punched so as to give the product of each element of a row of a^{-1}, from the card, times the corresponding element of b, from storage x, accumulate these products for each row and store the $(r - 1)$ sums in y.	$(r - 1)^2$
3	Form $(d - b'a^{-1}b)$	Keep in 605 Counter	Call out the element of b from storage x, negative multiply by the elements of $a^{-1}b$ in storage y, and accumulate the products with d which was called in on the first card from $x + r$.	$r - 1$
	Take $\frac{1}{(d - b'a^{-1}b)} = D$	$x + r$	Division	
4	Calculate $-a^{-1}bD = C$	Summary Punch	Call out each $a^{-1}b$ from storage y, negative multiply by D and punch out.	$r - 1$
5	Calculate $-a^{-1}bD = B$	Summary Punch and Store: $x + 1$ ⋮ $x + (r - 1)$	Same as 4.	$r - 1$
6	D	Summary Punch	Take D from storage $x + r$ and summary punch.	1
7	Calculate $a^{-1} - a^{-1}bB = A$	Summary Punch	Run the $(r - 1)^2$ cards through the tab again, this time using second instruction field. As each row of a^{-1} goes through, take the element of $a^{-1}b$ from storage y, and negative multiply it by the element of B in storage x, add it to the element of a^{-1} from the card-summary punch the answer.	$(r - 1)^2$
			Total number of cards on the r^{th} stage.	$2r^2 + 1$

Notice the order of punching out: we first obtain the last element in each row of a^{-1} for the first $(r - 1)$ rows (C); then we obtain the entire r^{th} row (B, D); and finally the first $(r - 1)$ rows less their last element (A). If we break this deck into a deck of C, B, and D (already in order) and a deck of A

Table 2. Approximate C.P.C. Running Time for Symmetric Matrices

Order n	10	16	20	25	30	35	40
number of cards	816	3112	5928	11,344	19,332	30,392	45,024
running time	14 min.	52 min.	1 hr. 35 min.	3 hrs. 10 min.	5 hrs. 20 min.	8 hrs. 20 min.	12 hrs. 30 min.

we can put one deck into each feed of the collator and merge them into proper row-by-row order.

The approximate total number of cards[2] passing through the C.P.C. is $\frac{2}{3}n^3 + \frac{1}{2}n^2$; the *average* speed of the C.P.C. for this method is 60 cards/minute.

If a high speed automatic calculator is used where storage is less limited, then the major time consuming factor, summary punching, can be eliminated and the relative computing time shortened.

In addition to the speed of the method, the procedure has several other recommending features: if inverses of submatrices are needed, these are available by-products of the calculations. Similarly the programming of an $n \times n$ inverse includes the programming of the inverses of smaller matrices.

An example of a 5×5 matrix inverted by this procedure is

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 8 & 11 & 14 \\ 3 & 8 & 14 & 20 & 26 \\ 4 & 11 & 20 & 30 & 40 \\ 5 & 14 & 26 & 40 & 55 \end{bmatrix} \quad M^{-1} = \begin{bmatrix} 6 & -4 & 1 & 0 & 0 \\ -4 & 6 & -4 & 1 & 0 \\ 1 & -4 & 6 & -4 & 1 \\ 0 & 1 & -4 & 5 & 2 \\ 0 & 0 & 1 & 2 & 1 \end{bmatrix}.$$

The procedure gave the exact inverse in less than .75 of a minute.

Under the assumption that in general it takes n^3 operations to invert an n^{th} order matrix, H. HOTELLING[3] asserts, "In partitioning a matrix [for inversion], there is an advantage in dividing the rows into *equally numerous* groups, since when the sum of two positive numbers is fixed, the sum of their cubes is a minimum when they are equal." If Hotelling's inference were correct, the method presented in this article of partitioning by single rows and columns would be inefficient. But in actuality the number of calculations required to invert any matrix by partitioning is invariant under the mode of partitioning:

With respect to computing time, it is most efficient to use single row and column partitioning since this method enables us to store b and d, and also at later steps $a^{-1}b$, D, and B. If b were not a vector and d not a scalar, then storage of these quantities would become impractical and the speed of

Table 3

Function	Approximate Number of Operations
a^{-1}	n_1^3
$a^{-1}b$	$n_1^2 n_2$
ca^{-1}	$n_1^2 n_2$
$ca^{-1}b$	$n_1 n_2^2$
$(d - ca^{-1}b)^{-1}$	n_2^3
$- a^{-1}bD$	$n_1 n_2^2$
$- Dca^{-1}$	$n_1 n_2^2$
$a^{-1} - a^{-1}bB$	$n_1^2 n_2$
Total	$n_1^3 + 3n_1^2 n_2 + 3n_1 n_2^2 + n_2^3 = (n_1 + n_2)^3 = n^3$

operation would be slowed down further by more intermediate summary punch operations.

It can be shown that if n_0 is the order of the largest matrix which can be inverted given the storage capacity, and $n_0 < n$, then the most efficient method of inverting by partitioning requires inverting of, say, k submatrices, where the first $(k-1)$ matrices inverted are of order n_0, and the k^{th} inverted matrix is of order $n-(k-1)n_0$.

HARVEY M. WAGNER

The RAND Corporation
Santa Monica, Calif.
and
Stanford University
Stanford, Calif.

This article is contained in H. M. WAGNER, "Matrix Inversion on an Automatic Calculator by Row and Column Partitioning," P-417, The RAND Corporation, July 14, 1953; P-417 also discusses in detail (1) the inversion by partitioning of non-symmetric matrices; (2) optimal partitioning; (3) inversion of n^{th} order matrices where n exceeds storage limitations.

[1] W. ORCHARD-HAYS, "The Duplex System for IBM's Model II CPC, A Fast Four Address, Double Operation, Floating-Decimal Set-Up," RM-1044, The RAND Corporation, February 23, 1953; H. M. WAGNER, "Coder's Manual for Duplex System" and "Stanford's Revised Duplex System—Wiring Manual," Technical Reports 1 and 4, Stanford Computation Center. These references give set-ups which meet the two operations per card requirement.

[2] The formula and table following do not take into account any saving in cards and time at the beginning of the procedure when r is small enough to invert without any summary punching. E.g., with 19 storage locations, a 5×5 matrix can be inverted without any summary punching in 90 cards at 150 cards/minute. In general, if S is the number of storage locations available, then all n^{th} order symmetric matrices for which $n \leqq N$ where $\frac{N^2+3N-2}{2} = S$ can be inverted without any summary punching.

[3] H. HOTELLING, "Some computational devices," Chapter X in *Statistical Inference in Dynamic Economic Models*, Cowles Commission Monograph 10, T. C. KOOPMANS, ed. New York, 1950, p. 323–328.

Polynomial Approximations to Elementary Functions

The increasing use of high-speed computing machines has revived interest in the approximation of functions of a real variable, particularly by polynomials. An orthodox table of function values at equidistant arguments may require considerable storage space in an electronic machine. In contrast, the coefficients of a polynomial which represents the function to a desired accuracy over a specified range may require very little storage space, and simple instructions will suffice for the evaluation of the polynomial.

If a function is bounded and continuous in a given finite range of argument, a powerful polynomial approximation is usually obtained by truncation of the infinite expansion of the function in Chebyshev polynomials. Many properties of these polynomials, and the means by which the coefficients in the infinite expansions can be derived, are given by LANCZOS,[1] whose notation for the Chebyshev polynomials in the range $0 \leq x \leq 1$ will be used here

$$T_n^*(x) = \cos n\theta, \quad \cos\theta = 2x - 1.$$

In this note, tables are given which ease the calculation of polynomial approximations to some common functions. Coefficients in the infinite