(c) At [24]: $I_n(x)$, based on manuscript of BAAS now in print.

(d) [24]: Spherical Bessel functions. Keypunched but not checked.

(e) At NOTS, KENNETH C. RICH and CHARLES RICKER, China Lake Pilot Plant, China Lake, California. Bessel functions $y_n(x)$ and $Y_n(x)$, $n \leq 20$, BAAS, v. 10.

Readers are again requested to review their keypunching loads, for possible keypunching of other basic mathematical tables during spare hours. Any one who can undertake a part of this work is requested to communicate with the undersigned.

GERTRUDE BLANCH
EVERETT C. YOWELL

National Bureau of Standards
Los Angeles 24, Calif.

The Sieve Problem for All-Purpose Computers

Introduction. The term all-purpose digital computer is often used to indicate a computing machine capable of performing the rational operations using addition and multiplication as basic functions of the arithmetic unit. Even when these operations are supplemented by some discrimination and "extract" commands, for dealing directly with the digits of numbers, there are a number of processes to which such a machine is not well suited. This includes even finite processes that are combinatorial in nature. Perhaps the most well-known of these processes is the sieve process. Although special equipment has been constructed for carrying out this algorithm[1] we shall not describe it here. Our purpose is to indicate how the all-purpose computer may be programmed to compete with sieve machines.

Let us first state the general problem to be solved by the sieve process. Let $m_1, m_2, \cdots, m_k$ be a set of k positive integers which, for the purposes we have in mind, may be assumed to be relatively prime in pairs. For each m_i we consider n_i distinct arithmetical progressions or linear forms in the variable x which we denote by

$$P_{ij}(x) = m_i x + a_{ij} \begin{cases} i = 1, 2, \cdots, k \\ j = 1, 2, \cdots, n_i \end{cases}.$$

We may assume that, for i fixed, the a_{ij} are distinct non-negative integers less than m_i. The problem is to find all integers N between given limits, say

$$A \leq N < B,$$

such that each N belongs to k arithmetical progressions.

The number k is called the *width* of the problem, the k numbers m_i will be called the *moduli* of the problem and the numbers a_{ij} $(j = 1 \cdots n_i)$ the *admissible remainders* for the modulus m_i. The solution N, or even the number of solutions, is an exceedingly complicated function of the given parameters m_i, a_{ij}, A, B. There are two extreme cases which may be mentioned.

Eratosthenes Sieve Problem. In this case $m_1 = 2$, $m_2 = 3$, and in general m_i is the i-th prime number, k is the number of primes not exceeding $A = B^{\frac{1}{2}}$, $n_i = m_i - 1$ and $a_{ij} = j$. This gives the famous sieve of Eratosthenes and has for solutions (besides $N = 1$) all the prime numbers between $B^{\frac{1}{2}}$ and B. In this sieve there is the maximum number of admissible remainders for each modulus.

Chinese Sieve Problem. In this extreme case there is the minimum number of admissible remainders for each modulus, that is, $n_i = 1$ for all i. This problem is very old[2] and often "solved" by what is known as the Chinese Remainder Theorem. If $A = 1$ and B is the product of the moduli, then there is a single solution N which may be found by any one of a number of other nearly equivalent practical methods rather than by a bona fide sieve process. On the other hand, this type of sieve problem is often useful as a checking routine for a more general sieve setup.

Quadratic Sieve. Midway between the two extreme examples we have examples in which each n_i is approximately $\frac{1}{2}m_i$ so that, to put it roughly, any number N has, a priori, an even chance of belonging or not belonging to one of the arithmetical progressions of a fixed modulus m. The expected number of solutions N is therefore approximately $(B - A)/2^k$. This kind of problem is the one most frequently met with and occurs in problems involving quadratic residues and congruences, binary quadratic forms, Diophantine equations of the second degree, etc.; hence the reason for calling this the quadratic sieve problem.

Theoretical Aspects. It is theoretically possible to combine any two arithmetical progressions with relatively prime moduli

$$P_1 = m_1x + a_1 \qquad P_2 = m_2x + a_2$$

into a single one of the form

$$P_3 = m_1m_2x + a_3$$

where P_3 consists of all numbers common to P_1 and P_2. Thus all numbers belonging to both the forms

$$5x + 4 \qquad 12x + 5$$

comprise the arithmetical progression

$$60x + 29.$$

By proceeding in this way one may combine the k sets of arithmetical progressions into a single set whose modulus is $m = m_1m_2 \cdots m_k$. The number of arithmetical progressions in this one set is clearly $n = n_1n_2 \cdots n_k$. The ratio n/m is the density of solutions of the problem. The above process becomes intolerably unwieldy even for moderate values of k except when most of the n_i are equal to unity. In the quadratic case, for example, a moderate value of k, like 20, which eliminates all numbers but one in a million, would lead to one set of at least $4 \cdot 10^{21}$ arithmetical progressions with a modulus of $4 \cdot 10^{27}$. To produce and sort for size the $4 \cdot 10^{21}$ admissible remainders is clearly out of the question. In these cases it may be, indirectly, more practical to examine all integers N between A and B excluding each integer in turn for non-membership in one of the original sets of arithmetical

progressions. This is the procedure we speak of when we refer to a sieve process.

Normalized Sieve Problem. In order to make a comparison of the effectiveness of different sifting processes we restrict ourselves to the case in which $n_i > 1$ for all i. That this is no real restriction is seen from the following consideration. If, for example, $n_1 = 1$ so that we are looking for an N of the form $m_1x + a_{11}$, we may change variable from N to N_1 where $N = m_1N_1 + a_{11}$ and thus eliminate $P_{11}(x)$. This reduces the width of the problem by unity, introduces new constants a_{ij} in the remaining $k - 1$ sets of progressions, but leaves the other n_i unaltered. The limits A and B are replaced by A/m_1 and B/m_1 so that the result is a fictitious speed-up of the sieve process by a factor of m_1. Proceeding in this way with any other cases of $n_i = 1$ we finally come to grips with the real problem in which $n_i > 1$. To be sure, a case in which $n_i = 2$ breaks up the problem into two parts each of which may be considered separately as a problem with an $n = 1$. The result is two short problems instead of one long one. This dissection of the problem is a favorite method with hand computers and accounts for the fact that fairly slow automatic sieves have considerable competition from hand methods. Of course the same dissection method is applicable to automatic sieves too, but often it is not practical because of the comparatively long set-up time involved. For these reasons, therefore, we consider as a normalized sieve problem, one in which $n_i \geq 2$. To put the matter in another way, we consider that all integers N between A and B have, a priori, an equal chance of satisfying the conditions imposed by the problem. The effectiveness of a particular process will then depend only upon the speed with which it can dispose of the average candidate N and pass on to $N + 1$.

Rates of Special Sieves. Without going into details it is desirable for purposes of comparison to mention existing sieves and to give an idea of their effectiveness. First of all we have hand methods. These involve the so-called movable strip method and its generalization, the stencil method. Here the range of natural numbers is represented by one or more sheets of paper ruled in squares, each square representing, by virtue of its position on the sheet, a definite integer. Strips of paper of length m_i are punched with $m_i - n_i$ holes, representing the inadmissible remainders modulo m_i, and are moved carefully down the columns of the sheets and those square cells which are revealed by the punched holes are crossed out by the computer. After k of these strips have been applied, those cells which still survive are the answers to the sieve problem. Rates as high as 3 numbers per second are difficult to maintain over a long period of time.

Electromechanical sieves have been built to canvass 50 numbers per second. These sieves can accommodate almost any reasonable sized moduli m_i and handle problems of width $k \leq 20$.

A photo-electric sieve has been built in 1932 which runs at 6000 numbers per second and an electronic sieve is under development which is designed for approximately 300000 numbers per second. These high-speed sieves are not equipped with high-speed output and are intended to be used on problems whose progressions give restrictions comparable to or higher than those of quadratic type. It will be seen that such high speeds are not obtainable

with all-purpose computers but that the more modest speeds of the electromechanical type sieve can be achieved and even surpassed.

Direct Attack by an All-Purpose Computer. Most all-purpose computers are equipped with a division command or at least a division program. A direct method of handling the sieve process would consist in dividing a trial value of N by each modulus m_i in turn and then inspecting the remainder for membership in the set α_{ij}. In the case of the Eratosthenes sieve, the inspection of the remainder is practically instantaneous as it requires only one or two addition times. If the number of moduli is large, for example large enough to make a list of primes above 10^7, then the number of divisions in case N is a prime will be greater than 440. Speeds are such that 50 milliseconds per division is not often realized. Allowing only 10 milliseconds per division, times of the order of several seconds are required to treat these cases. This does not compare favorably with even hand methods. However, in case the number of moduli is small a much more favorable speed prevails. A small Eratosthenes sieve is often incorporated into a problem in which a parameter p is supposed to have a prime value. In this way a large percentage of composite p's are eliminated, the remaining composites being taken care of later either from the output or by some other internal program. Beyond a certain point we get diminishing returns from each new modulus we introduce.

In the case of a typical quadratic sieve problem the examination of the remainder would be a much longer routine but still, on the average, rather shorter than a division program. Sieves of width 25 would canvass numbers at the rate of three or four per second. This is still too slow to compete with much simpler equipment. Besides, the memory capacity of many machines would be exceeded in trying to accommodate the given remainders a_{ij} in many typical problems.

Binary Method of Representing a Sieve Problem. The fact that, as far as a given modulus m is concerned, a proposed number N is either rejected or accepted, makes it possible to treat the sieve with binary methods. The sieve problem may in fact be represented by a matrix of k rows and infinitely many columns. The infinite rows are periodic, the i-th row having a period of m_i. The element situated in any column whose number is congruent to r modulo m_i is 0 or 1 according as r is or is not one of the acceptable remainders a_{ij}. The problem is then to find those columns which consist wholly of zeros. (Clearly the rôles of 0 and 1 may be interchanged here.) The matrix will be referred to as the sieve matrix.

For example, suppose that the problem is to find the least positive integer of the forms

$$\begin{cases} 5x + 1 \text{ or } 4 \\ 7x + 2 \text{ or } 3 \text{ or } 4 \\ 9x + 1 \text{ or } 6 \text{ or } 7 \end{cases}$$

The matrix corresponding to this problem is

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & \cdots \end{pmatrix}$$

Here we are looking for the first column which consists wholly of zeros.

The answer to the problem will be seen to be 16. In fact if we continue each periodic row two more steps beyond what is shown, we obtain a column of zeros.

Machine Methods of Realizing the Sieve Process. Most of the high-speed digital computers use binary numbers and almost all of the decimal type machines use a binary code for decimal digits. The purely binary machines are the most natural ones with which to apply the following methods. The use of decimal type machines entails a considerable reduction in the width of the problem or else a certain amount of indirect use of the binary code for the decimal digits. In what follows we shall assume that we are dealing with a binary type machine, leaving the modifications necessary for the decimal machine to the ingenuity of the reader. The operations needed to carry out the sieve process clearly involve shifting operations and the extraction or discrimination of digits. This does not mean that addition, subtraction, and multiplication are not needed. In fact these latter operations are frequently involved in the execution of the former ones.

Of the various possible methods to be used that one is to be preferred which gives the greatest speed in canvassing the columns of our matrix. The decision may be determined by the logical structure of the machine. In any case, there are two general steps to carry out: (a) the passage from one column of the matrix to the next and (b) the inspection of a column to see if it contains a non-zero element.

First Method. The steps (a) and (b) can be carried out simultaneously by the simple process of doubling the binary number represented by each row. Since in many cases the period m_i will exceed the number s of digits which the adder will accommodate, it will be necessary to program this doubling as a "multiple precision" process, taking account of overflow between the separate batches of s digits of the entire number. In particular, if overflow occurs in the first column there are two important consequences. First, this indicates that a 1 was standing in the first column and so the corresponding number N is to be rejected. Secondly, this overflow must be sent to the other end of the period to perpetuate the periodic pattern of the digits. This operation has to be performed k times, once for each row. If at any time no overflow is obtained in the entire set of k operations, then we have a solution of the problem and the machine is instructed to print out the answer, continuing on or halting as desired. The entire process can be carried out using only addition with detection of overflow, and involves a certain amount of "tallying" and modification of commands. The method has the disadvantage of requiring k multiple precision operations in passing from N to $N + 1$. When k is large (20 or 30 in some typical cases) and the m's large, the major portion of the time is spent in keeping the periods up-to-date. However, for small k and m, the method is both simple and fast. In fact, its use is recommended in problems in which no actual sifting is involved. Suppose, for example, that we have a problem involving one or more parameters and these are given certain irregularly spaced values such as for instance

$$x = 1, 3, 6\text{–}9, 12, 15, 18\text{–}22, 25.$$

Instead of storing all these numbers in as many memory positions, this

information may be stored in only the one word

$$1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1,$$

in which the r-th digit from the left is 1 or 0 according as r belongs to our set or not. By successive doubling and inspection for overflow the machine can inform itself as to which values of x it should consider.

Second Method. In order to eliminate the lengthy multiple precision processes of the first method one may proceed as follows. The periodic digits of each row of our matrix are stored as before in "words" of s digits. The minor of k rows and the first s columns of the original matrix are now duplicated in a special part of the memory where the operation of doubling and testing for overflow is performed as in the previous method. However, when overflow occurs the fact is recognized by the machine but the digit is not kept track of as before. After s doublings the information in the original k by s matrix is all used up and the matrix now consists wholly of zeros.

The next process consists of circulating the original matrix by moving the elements of each row s places to the left. Since s binary digits is a word, this process is simple and fast. Only one complication arises, that of disposing of the first word in each row. If for a particular modulus m_i we have

$$m_i = sq_i + r_i \qquad (0 \leq r_i < s),$$

then the first $s - r_i$ digits of this first word w_i belong in the penultimate word of the shifted matrix while the last r_i digits form the last word. This disposal of w_i can be accomplished by use of a product command. The machine computes the exact two word product of w_i by 2^{s-r}. The "most significant" word of this product is thus added to the penultimate word of the shifted matrix and the least significant word is placed at the end.

One disadvantage of the second method is the fact that all k rows of the matrix have to be treated, even though a rejection of the value of N may result from one of the first few rows. This early rejection of N is a highly probable event in the case of a quadratic sieve problem. In fact the probability of getting a rejection after h trials is $1 - 2^{-h}$, so that only one N in a thousand is apt to pass the test imposed by the first ten rows. Moreover, in many problems some moduli are much more restrictive than others and if the rows of our matrix which correspond to these moduli are put first, the probability of early rejection is very considerably increased. In order to exploit this possibility we propose another method.

Third Method. This method differs from the previous one in its more rapid treatment of the minor of k rows and s columns. Instead of using the detection of overflow we employ the more elaborate "extract" command. This command, which varies slightly from one binary computer to the next, enables the machine to extract from a given binary number those digits which occupy specified positions. These positions are specified by a number called the extractor. As extract is performed in the SWAC, for example, those digits of a given number, called the "extractee," are selected which correspond to the zero digits of the extractor; all other digits are made zero. Thus if the extractee is

$$1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\cdots$$

and the extractor is

$$0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\cdots$$

the extracted number is

$$1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\cdots.$$

In other words, the extracted number has for its i-th digit the product of the i-th digit of the extractee by the complement of the i-th digit of the extractor.

Let us now consider the minor of k rows and the first s columns of our sieve matrix. Starting with the extractee consisting of s 1's and using the first row of our minor as extractor, the extracted number records by its 1's those columns whose first row elements are zero. Using this extracted number as a new extractee and the second row as extractor, the extracted number now records those columns whose first and second elements are both zero. Continuing this process and comparing the successive extracted numbers with zero (that is the number consisting of s digits 0) after each extraction, we very soon (at least in the typical case) arrive at an extracted number which is zero. At this point the entire minor is thrown away and a new minor is created by circulating the rows of our matrix as described in the second method. If, however, our minor contains one or more columns of zeros, a non-zero number will result from all k extraction operations with digits 1 in the positions corresponding to the solution or solutions of the problem. To find these positions we may proceed successively to double the corresponding number and test for overflow as in Method 1. An obvious method of tallying will produce the solution or solutions which the extraction process has detected.

Comparison of Methods. On the basis of speed, a comparison of the three methods for the case of the quadratic type sieve problem may be made as follows.

Let

T = the average time required to dispose of a single number N and pass on to $N + 1$,
s = the number of binary digits in a machine word,
h = the average number of words for each modulus of the problem,
k = the number of moduli, or width, of the problem,
a = the addition time of the computer, that is, the time required to receive and add two numbers and to dispose of their sum.

For the three methods rough approximations to T may be given as follows:

First Method: $T = 5hka$
Second Method: $T = 5k(1 + 2hs^{-1})a$
Third Method: $T = [10kh + 6(1 + \log_2 s)]s^{-1}a$

As an example of what may be done on the SWAC for which $s = 36$ we take a typical case of $h = 2$ and $k = 17$. The addition time for the SWAC being 64 microseconds, the three values of T turn out to be 10880, 6044, and 672 microseconds respectively.

The above formulas for T do not take into account certain occasional routine operations such as resetting tallies to zero, the restoration of modified commands, the decimal conversion and printing of the solutions, etc. To take these activities into account may lengthen the time by as much as 10 percent. In actual tests on the SWAC, for instance, the third method gave $T = 695$ microseconds. It should be pointed out that h must be at least 2 in the second and third methods since a complete minor of s columns is needed.

Operating Suggestions. We conclude with a few remarks for the benefit of the programmer and operator in dealing with sieve problems. In the normal case the expected output is only a few numbers N occurring in unpredictable places between A and B. It is evident that the machine must be in absolutely perfect operation during the run in order to be sure that no solution has been overlooked. Absolute precision in the circulation of the rows of the sieve matrix is, of course, of paramount importance. The accuracy of this operation may be checked at the end of the run by printing out the entire matrix and inspecting the result. In order to check the machine during operation the programmer may deliberately insert one or more "traps" or precomputed extraneous solutions. It is not difficult to find by hand methods a number N_0 which satisfies most of the requirements imposed by our matrix. To make N_0 satisfy all the conditions we may deliberately weaken the remaining ones by changing a 1 to a 0 in each of the outstanding rows. This weakening, especially when applied to large moduli, will not change perceptibly the overall exclusion ratio but may introduce one or more unexpected extraneous solutions which, however, are easily recognized when they occur. This is a very cheap price to pay for the confidence that one gets from seeing the machine deliver the one or more predicted solutions N_0 on schedule.

In case the running of the problem is interrupted either by machine failure or by another problem of higher priority, there are three procedures available.

First, one may start the problem over from the beginning. This is especially advisable if the interruption occurs early in the run. Secondly, one may make an obvious translation in the variable N and compute by hand a new sieve matrix. Of course, this has to be done with considerable care but in case the machine is being used on another problem, there may be time to do this. In running very long problems it may be wise to prepare in advance several matrices from which a new start may be made in case of interruption. This procedure, however, is almost as much work as breaking up the problem into parts as described above, and this latter procedure always pays off in the consequent speed-up of the problem.

A third procedure is available in case there is still some memory space unused. This method is simply to make the machine compute its new starting matrix itself. This program may be based on the First Method, using detection of overflow to circulate each row of the original matrix to its new position. Of course the amount of this circulation depends on the new proposed starting point and is different for each modulus. If, instead of starting with $N = 1$, we wish to recommence with $N = N_1$, the row corresponding to the modulus m must be circulated $r\frac{1}{2}-\frac{1}{2}1$ steps where r is the remainder on division of N_1 by m. The program for carrying this out is set

up with N_1 as a free parameter which can be typed into the machine as occasion demands, no further information being needed. This elaboration of the program makes it possible to operate a lengthy sieve problem as a backlog workload without the need of a specially trained operator.

D. H. L.

[1] D. H. LEHMER, "The mechanical combination of linear forms," *Amer. Math. Monthly*, v. 35, 1928, p. 114–121, "A photo-electric number sieve," *Amer. Math. Monthly*, v. 40, 1933, p. 401–406, "A machine for combining sets of linear congruences," *Math. Annalen*, v. 109, 1934, p. 661–667.

[2] L. E. DICKSON, *History of the Theory of Numbers*, v. 2, Washington, 1923; New York, 1934, p. 57.

The Use of Large Intervals in Finite-Difference Equations

In a recent article SIR RICHARD SOUTHWELL[1] has challenged the general theory of finite differences and in particular the use of it in connection with the solution of differential equations by relaxation methods.[2] Opinions differ on the method of treatment of a numerical differentiation formula such as

$$h^2y_0'' = \left(\delta^2 - \frac{1}{12}\,\delta^4 + \frac{1}{90}\,\delta^6 - \cdots\right) y_0,$$

where h is the interval between pivotal points and where $\delta^{2n}y_0$ is the $2n^{\text{th}}$ central difference of y_0. This equation is replaced for the purpose of numerical solution by its equivalent

$$h^2y_0'' = (y_1 + y_{-1} - 2y_0) + \Delta\,(y_0),$$

where y_1, y_0, and y_{-1} refer to the values of y at $x_0 + h$, x_0, and $x_0 - h$ respectively and

$$\Delta(y) = \left(-\frac{1}{12}\,\delta^4 + \frac{1}{90}\,\delta^6 - \cdots\right) y.$$

I advocate the use of as large an interval as possible (consistent with convergence of the finite-difference equations and convenience of computation) calculating $\Delta(y)$ and incorporating it into the computation. Southwell prefers to use such a small interval that Δ is negligible. He states: "*Accuracy is not predictable from quantities computed from a finite-difference approximation unless the interval is small enough to justify belief in the convergence of the Taylor series:* in general the radius of convergence, though it exceeds one or two of the smallest intervals that are practicable, will not exceed many such intervals, and consequently *what have been claimed as closer approximations may in fact be less accurate.*"

The purpose of this note is to point out that, in the writer's opinion, this statement is unduly cautious and that its conclusion cannot be true if finite-difference equations are properly used.

In the first place it is not clear what role the Taylor series plays in finite-difference theory. If the Taylor series is not convergent, care must certainly be exercised; but finite-difference methods do not always break down in this case.[3] The point to be emphasized, however, is that an examination of the differences will tell us, in all practical cases, whether the finite-difference