COMPUTER PROGRAMS FOR FINDING EXACT EIGENVALUES OF MATRICES AND TEST MATRICES

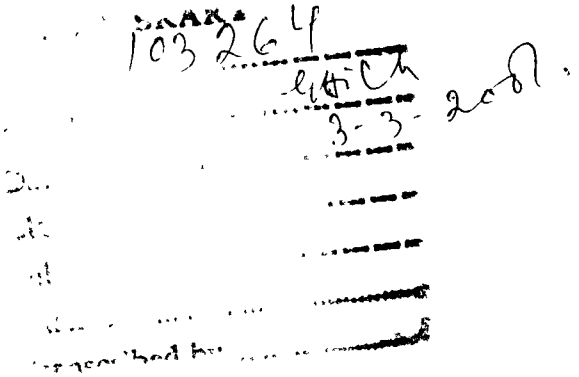
By

BIPUL SYAM PURKAYASTHA
DEPARTMENT OF MATHEMATICS

SUBMITTED
IN
FULFILMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
DOCTOR OF PHILOSOPHY

To

NORTH - EASTERN HILL UNIVERSITY
SHILLONG - 793022, INDIA
FEBRUARY, 1996



CERTIFICATE

I certify that the thesis entitled **"COMPUTER PROGRAMS FOR FINDING EXACT EIGENVALUES OF MATRICES AND TEST MATRICES"** submitted by Bipul Syam Purkayastha in fulfilment of the requirements for the degree of Doctor of Philosophy is the outcome of studies undertaken by the candidate. I certify that the sources from which the ideas have been borrowed are duly referred to.

The material in this thesis has not been presented for the award of a degree in any University before.

This thesis may be placed before the examiners for evaluation and necessary formalities.

विजय कुमार

(Prof. Vijai Kumar)
Supervisor
Shillong
The 5th February 1996

Mangesh B. Rege

(Prof. M. B. Rege)
Supervisor
Head of the Department of Mathematics
North Eastern Hill University
Shillong – 793022
INDIA

God gave us the integers.
All the rest is man's work.
Leopold Kronecker

To my uncle
B. B. Syam Purkayastha

ACKNOWLEDGEMENTS

The studies for this thesis were undertaken under the guidance of Dr. Vijai Kumar. I wish to express my deepest regards and profound gratitude for his guidance and fatherly affection. Over the years he continued to guide me with a rare sense of patience and understanding which made it much easier for me to write this thesis. I also express my indebtedness to Smt. Sulakshana Rani, his wife, who is encouragement personified. I think that I am immensely fortunate to have had the pleasure of coming across them.

I take this opportunity to express my gratitude to Dr. M. B. Rege, the Head of the Department of Mathematics for his constant help and encouragement. I express my indebtedness to Dr. P. K. Saikia whom I have consulted frequently and who was always ready with valuable suggestions. I owe special thanks to all the other faculty members of the department of Mathematics, NEHU, namely Dr. A. K. Das, Dr. B. K. Dev Sharma, Dr. C. R. Mondal, Dr. H. K. Mukherjee, Dr. S. K. Srivastava, Dr. S. L. Marbaning and Dr. S. S. Khare for their constant encouragement and for their best wishes.

My warmest thanks are also reserved for Dr. S. N. Rai, Director of Computer Centre, NEHU, for allowing me to use the computers in his Centre. My friend Mr. P. P. Dey, Senior System Analyst of the Centre, always saw to this that my work went smoothly and without a hitch.

My colleagues at Shillong College were always ready to extend their helping hands for the smooth completion of my work. Special mention must be made of Mr. T. Maitra, the Principal of the College, Mr. B. C. Goswami,

Mr. K. Choudhury, Mrs. S. Dhar and Mr. H. Dhar, my colleagues in the department of Mathematics of the College, Mrs. M. P. R. Lyngdoh, Mr. R. Dutta and Mr. B. Roy my senior collegues, Mr. M. N. Bhattacherjee, Mr. M. Dey, Miss. V. R. Solomon and Mr. T. S. Rajee, my friends and colleagues. With their constant encouragement my problems and frustrations became more bearable.

I shall like to thank all my friends, specially Mr. A. Das, Mr. H. S. Bhattacherjee, Mr. M. Chakravorty, Mr. P. Chakravorty, Mr. S. K. Singh, Mr. D. Dey and Mrs. R. Bhattacherjee. I consider myself fortunate to have friends like them. Associations with them made my hard days easy.

Special gratitude and heartfelt thanks are also due to Mr. M. Bhattacharjee, with whose help I could avail the library facilities of I.I.T. Kharagpur. I also thank Mr. P. Bhattacherjee for many stimulating discussions I had with him.

I am grateful to my aunt Smt. K. Syam Purkayastha and my late uncle Shri B. B. Syam Purkayastha and for bringing me up as their own child. With much affection and love I remember my mother Smt. A. L. Syam Purkayastha and my late father Shri B. B. Syam Purkayastha. I express my thanks to my elder brothers Mr. B. K. Syam Purkayastha and Mr. B. J. Syam Purkayastha who are extremely caring and who shielded me throughout from the non-academic problems of life. Finally I shall like to thank all my family members, past and present, who shared their lives with me and sustained me in beautiful and sad times. I shall cherish their memories forever.

Bipul Syam Purkayastha

CONTENTS

CONTENTS

Introduction

In order to employ a computer, effective algorithms are needed which can be implemented in the form of computer programs. Endeavours to devise such algorithms often yield new theoretical findings, as a by-product, which are also per se of interest. A basic task, which every mathematician faces, is the task of providing suitable numerical examples and data in order to illustrate the known theorems and in many cases to make up or to support conjectures that may lead to the discovery of new algorithms. However, it is a particular phenomenon of some topics such as Algebraic Number Theory, that even relatively simple computational problems require a great deal of numerical calculations which are not feasible to carry out without a computer [68].

The eigenvalue problem is also 1 such problem. This problem has been the subject of intensive research for many decades with proposals of new algorithms appearing frequently. These algorithms are required to be tested, among others, by providing numerical data. These numerical examples provide a systematic and rational means of evaluating the performance of programs implementing those algorithms executed by high-speed electronic computers. So equal emphasis is given on (**a**) the numerical solution of the problem, such as the computation of the zeros of the polynomial and the exact eigenvalues of the given matrix and (**b**) its converse problem, i.e., generating the problem from its known solution, which means the reconstruction of the polynomial from its known (prescribed) zeros and the generation of

matrices with preassigned eigenvalues. For the determination of eigenvalues the data to be provided is in the form of matrices with known eigenvalues. There exist many algorithms for generating such data. But, as it is known in computer parlance, that there exist algorithms which may be mathematically interesting, but which are certainly not suitable for developing a good software. For testing the accuracy of algorithms it is imperative that the matrices generated as data should be exact. So to provide exact numerical data and examples we have given emphasis on generating these matrices over the unique factorisation domain **Z** and over the exact field **Q** instead of an imprecise field such as **R**. Since floating-point computation by nature is inexact, it is not difficult to misuse it in such a way that the computed answers will consist almost entirely of 'noise'. Although the floating-point mode is widely used in scientific calculations and all the major programming languages support floating-point calculations, still it should not be used without reservation because of possible errors in its implementaion. For example, when we try to invert a Hilbert (a very ill-conditioned) matrix, even for a relatively small order, using floating-point computations, we get a result very far from the correct inverse, but it is impossible to locate how and in what way the round-off errors have piled up to such an extent. Because of the complexity of piling up of such errors, it has been reported[64] that certain critical safety systems have prohibited the use of floating-point arithmetic.

There are well-known methods for the computation of eigenvalues. These methods depend on whether we have to compute all the eigenvalues, or some particular eigenvalues, e.g., the power method for the computation of the dominant (largest in magnitude) eigenvalue, and also on the special forms of

the matrices, such as the Jacobi method for the computation of the eigenvalues of a Hermitian matrix. But most of the algorithms require the floating-point mode for their implementation and so the problem of the generation of round-off errors arises with them.

It is known that there does not exist any 'stable' method for obtaining the eigenvalues of a non-symmetric matrix. So for calculating the exact eigenvalues of any matrix, irrespective of being of any special form, such as symmetric or Hermitian, and even as ill-conditioned as it can possibly be, we have adopted the technique of calculating its characteristic polynomial and then computing the zeros of this polynomial, which are known to be the eigenvalues of the given matrix. The major effort in this endeavour is concentrated on error-free computation.

Therefore we have developed algorithms which can be implemented using only fixed-point arithmetic. Consequently there is no possibility of the generation of any round-off error whatsoever in the calculations. However, the fixed-point implementation of such algorithms poses certain problems, such as in many algorithms, the integers become greater than the fixed-point limit for the particular computer on which we are trying to implement the algorithms. So a pre-requisite for the effective implementation of these algorithms is a multi-precision package.

Regarding the algorithms for the multiplication and division of multi-precision integers, possibly the best are due to Schönhage and Strassen which run in $O(n\ ln\ n\ ln\ ln\ n)$ bit operations, which is much faster than $O(n^2)$ as n gets large. These sophisticated algorithms become practical only for numbers having more than several hundred decimal digits and they are not worth-

while for numbers having upto roughly 100 decimal digits [10]. Moreover the arithmetic techniques for use in computer programming for multi-precision numbers differ considerably from the techniques used in the hardware implementation of arithmetic operations [31].

The first chapter of this thesis begins with algorithms for the 4 fundamental arithmetic operations of addition, subtraction, multiplication and division of multi-precision integers. The division of a multi-precision non-negative integer by a multi-precision natural number is the most difficult of the 4 arithmetic operations. The main difficulty in the mechanisation of the pencil-and-paper method of division is the determination of the digits of the quotient. This we do by first finding the trial digit of the quotient and then we find the exact digit, the implementation of which becomes very time consuming and cumbersome. To overcome this difficulty we have given an algorithm and proved that the trial digit $q' \leq 10$ and the exact digit $q=q'$ or $q'-1$ in the 10^d radix division, where $d \in \mathbf{N}$ and $d \geq 2$.

The next problem in this chapter is the evaluation of the square root of a natural number, which is considered to be next in importance to the 4 basic operations, since it finds wide application in many areas of Science and Engineering. Newton-Raphson method is 1 of the most widely used methods for calculating the square root of a positive number. We have used the Newton-Raphson method for the computation of the integer part of the square root of a positive integer x in preference to the technique described in [45] as it has been found that the speed of computation of the latter[45] is proportional to the binary length of the root. It is well-known that with the close initial first estimate of the square root, the quadratic nature of

this algorithm makes it converge fast. But this method does not provide any scheme for guessing the initial first estimate of the square root. We have presented an algorithm (together with a mathematically elegant proof) for obtaining the close initial first estimate of the square root of a positive integer.

The next problem which we have dealt with in the 1st chapter is the purely number-theoretic problem of calculating all the prime numbers from 2 to a given natural number n. We have used these prime numbers for the computation of all the zeros of a given polynomial. The algorithms which purport to solve this problem either require large storage or have comparatively slow speed of computation which is a hindrance to their practical implementation. We have developed an algorithm overcoming these 2 difficulties.

Permutations, combinations and partitions are 3 important problems in combinatorics. They have received much attention and various applications have been found. For example, the calculation of the characteristic polynomial of a square matrix by the direct expansion method requires the computation of all the combinations of the 1st n natural numbers taken r at a time ($1 \leq r \leq n$). As another example, let us consider the set of all the square matrices of a particular order n such that all their eigenvalues are equal to a given number. We must break up this set into mutually disjoint equivalence classes such that all the matrices in a particular equivalence class are similar to each other and no matrix in a particular equivalence class can be similar to any matrix in any other equivalence class. To find the number of such equivalence classes and the Jordan canonical forms representing each equivalence class we require the knowledge of all partitions of n. We have solved

these problems using the control structure proposed by Skordalakis and Papakonstantinou [51], which otherwise would have been solved by altogether different algorithms, e.g., [6,15,60].

The second chapter deals with the numerical solution of polynomial equations and also with the converse problem of reconstructing polynomials from their known zeros. We have started this chapter with the latter problem. The first implementation is of generating the monic polynomial $P(x) = \prod_{i=1}^{n} (x - a_i)$, $a_i \in \mathbf{Z}$, and then we have extended this algorithm to generate the primitive polynomial $Q(x) = \prod_{i=1}^{n} (xq_i - p_i)$, $p_i, q_i \in \mathbf{Z}$, $q_i \neq 0$. Thus we have provided numerical examples for testing the algorithms for the factorisation of polynomials over $\mathbf{Z}$ (or $\mathbf{Q}$). For constructing polynomials with integer coefficients and having zeros of the form $\frac{1}{2}(a \pm \sqrt{b})$, a,b $\in \mathbf{Z}$ and b, if positive, is not a perfect square, we have formed quadratic polynomials with such zeros and developed algorithms for multiplying such polynomials together or such polynomials with linear factors or both. The reconstruction of polynomials from their zeros is useful for testing algorithmic stability (i.e., whether the computed zeros of a given polynomial generate a polynomial which is only slightly different from the original polynomial in the case of performing all calculations in real arithmetic instead of integer arithmetic) [37].

The numerical solution of a polynomial equation is a difficult computational problem. We can view it as a non-linear problem in terms of linear algebra. Many algorithms which work easily on equations with well-separated simple zeros fail to work on more difficult problems [25]. Moreover, most of the algorithms have to work in the field $\mathbf{R}$ of real numbers, which is an im-

precise field due to the reason that we cannot express its irrational elements exactly; we can only express them upto a desired degree of accuracy. Since we are interested only in exact arithmetic, we may not be able to compute the exact zeros of a polynomial (even if they exist) using the existing algorithms. So we have developed algorithms for computing the exact zeros of a polynomial.

In many algorithms it is necessary that the polynomial in question should be square-free; this condition can be fulfilled by dividing the polynomial by the G.C.D. of itself and its derivative. The algorithm for calculating the G.C.D. of polynomials over $\mathbf{Z}$ is also included in this chapter. At the end of this chapter we have given an algorithm for reducing non-monic polynomials over $\mathbf{Z}$ to monic polynomials over $\mathbf{Z}$ (for computing the rational zeros of the original polynomials), and then we have extended this algorithm to generate another algorithm which reduces a polynomial with non-integer rational coefficients to a monic polynomial with integer coefficients. We have also proved that this reduction is possible without inordinately increasing the magnitudes of the coefficients.

In the third chapter the first algorithm, which we have given, is for computing the characteristic polynomial of a prescribed square matrix with integer entries. Efficient methods for this purpose are (**c**) Leverrier-Faddeev method, the computational complexity being n^4, and (**d**) the method of calculating the Hessenberg form of the matrix by a similarity transformation and then computing the characteristic polynomial using a recursive relation, the computational complexity being n^3. But the phenomenon of coefficient explosion neutralises the advantages of an n^3 method (**d**) over an n^4 method

(**c**), when the matrix under consideration is over the base ring **Z or Q**. Moreover the algorithm (**c**) is much simpler than the algorithm (**d**). So we have coded the subroutine for calculating the characteristic polynomial using the method (**c**).

Although, the method of direct expansion, i.e., the process of computing the coefficients $p_1,p_2,p_3,...,p_{n-1}$ of the characteristic polynomial, as the sum of the principal minors of the matrix A, having orders 1,2,3,...,(n−1), and the coefficient p_n, as the determinant of A, is not considered to be very efficient, since it requires the evaluation of too many determinants. The number of such determinants of different orders is $2^n - 1$. We have shown that the number of such determinants can be drastically reduced by modifying the Gaussian elimination method of evaluation of determinants.

The eigenvalues of a square matrix, are the zeros of the characteristic polynomial obtained by the above process. The zeros are calculated using the algorithms/subroutines which we have developed in Chapters 1 and 2.

It is well-known that every square matrix, although it need not be diagonalisable, is similar to a matrix whose diagonal entries are its eigenvalues and whose super-diagonal entries consist of only 0's and 1's. This matrix is called the Jordan canonical form of the given matrix and is denoted by J. So for every square matrix M, there exists a non-singular matrix P, and a unique matrix J, such that $PMP^{-1} = J$. We have computed the matrix P using the Gaussian method of solution of a system of linear simultaneous equations. The computation of the Jordan canonical form helps in deciding whether 2 given matrices are similar or not; the calculation of the invariants corresponding to a particular eigenvalue; and so on.

The concluding chapter (i.e., Chapter 4) of this thesis deals with the generation of matrices with known (pre-assigned) spectra, which provides numerical data to the subroutines developed in Chapter 3. For constructing such matrices, we have applied what is known as the 'grand strategy' of similarity transformations in the computation of eigenvalues. The simplest way of generating these matrices accurately is to have the matrices with all integer entries. In the human domain it is a general practice[16,33,34,56] to consider matrices over the base ring **Z** for providing numerical examples for the purposes to which we have referred at the beginning of this introduction. These matrices are very useful in integer programming which, in recent years, has become increasingly popular. It is not possible to obtain such non-trivial matrices with many of the existing algorithms for generating matrices, even if it is known that their spectra are composed of integers, such as the algorithm suggested by Hall and Porsching [22] for the generation of positive test matrices. In certain cases it is not possible to have such a matrix at all, e.g., it is not possible to have a positive matrix (all elements are positive) with integer entries, even with a small order such as 2, with eigenvalues 1,1. In such cases we have deduced necessary conditions which the eigenvalues must satisfy, so as to ensure that the entries of the generated matrix will belong to the set **N** of natural numbers, the principal ideal domain **Z**, the field **Q** and so on, and the matrix will have a special form, such as symmetric, positive, positive symmetric and so on.

The straightforward implementation of the similarity transformation requires $3n^3$ multiplication/division operations. Therefore we have chosen a modal matrix with known inverse, so that the similarity transformation can

be implemented only with addition/subtraction operations. This modal matrix will also facilitate the reduction of propagation of errors if the matrices are to be generated with spectra belonging to $\mathbf{R}$, the field of real numbers (or $\mathbf{C}$, the field of complex numbers).

It is quite evident that if in the similarity transformation PMP^{-1}, $P \in GL_n(\mathbf{Z})$, the group of unimodular matrices with integer entries, then $PMP^{-1} \in M_n(\mathbf{Z})$, the set of matrices with integer entries, provided that $M \in M_n(\mathbf{Z})$. So we have started Chapter 4 with algorithms for generating such unimodular matrices P with integer entries, and then we have extended the algorithms to produce other algorithms which generate a matrix with prescribed determinant, all of whose entries are integers except perhaps the last entry. The unimodular matrix P generated in this process is used in constructing non-trivial matrices with integer entries and prescribed integer spectra. The generated matrices may be of the following different properties:- derogatory, non-derogatory, diagonalisable etc. Since determinants and inverses of square matrices play a major role in these transformations, so the algorithms for error-free computations of determinants and inverses are also included in this chapter.

In the generation of matrices we have paid particular attention so that there is no unnecessary explosion of the entries of the generated matrices which generally accompany such algorithms. As such, these matrices will better meet the basic tasks, which every mathematician faces, such as the tasks of providing numerical examples to illustrate known theorems, and in many cases, to make up or to support conjectures, which may lead to the discovery of new algorithms. The usefulness of such matrices in testing the eigenvalue routines has been demonstrated in [33,34,66,67]. We have writ-

ten programs implementing all the algorithms (The listing of the programs together with sample input/output is given in the appendix.); the systematic testing of the programs for (**a**) the numerical solution of the problems, such as the zeros of a given polynomial and the exact eigenvalues of a given matrix has been accomplished in part with the aid of programs for (**b**) the reconstruction of polynomials from their known (prescribed) zeros and the generation of matrices with pre-assigned eigenvalues; and vice versa. The methods which we have discussed are error-free and work independently of any convergence criteria. This feature amply justifies the small number of extra arithmetic operations. Error-free computation permitted us to test for exactness in all the problems (for example, whether the computed roots generated a polynomial which was exactly the same as the original polynomial, etc.). This fulfilled the objective of this thesis and may also be used for analysing the performance of algorithms which purport to solve these and similar other problems, thereby permitting comparison to be made between theoretical results and computation.

CHAPTER 1

NUMBER-THEORETIC AND COMBINATORIAL ALGORITHMS

It is well known that the computer works with limited precision in both real and integer modes. For real numbers there are both lower and upper limits beyond which we get overflow and underflow errors (respectively); and moreover even if a real number is storable in a single memory location, still it cannot be represented exactly in the computer, because of the conversion to the binary system of numeration at the time of storing and again conversion back to the decimal system at the time of retrieval. Although the upper limit for real numbers is much higher than the upper limit for integers but the presence of round-off errors in real arithmetic due to internal conversions from decimal to binary number system and vice versa results in the fact that the answers obtained by using real arithmetic are never exact. If we use double precision arithmetic, no doubt, the accuracy increases; but still we do not obtain exact answers. As we are interested only in exact arithmetic, so we have proposed algorithms which can be implemented with integer arithmetic. In most of our algorithms the numbers involved will almost always become quite large. So it becomes imperative to have a multi-precision package for the better implementation of such algorithms. This multi-precision package has interesting applications in both pure and applied mathematics. Some important applications in pure mathematics are the disproof of Mertens con-

jecture by Odlyzko and te Riele, the disproof of the Bernstein conjecture in approximation theory by Varga and Carpenter and the resolution of the one-ninth conjecture [3]. There are algorithms for this purpose e.g.[13,38], some of which are very fast (as n gets large, multiplication and division can be performed faster than $O(n^2)$). But these sophisticated algorithms for hardware implementation appear to be inapplicable to computer programs for high precision numbers [31].

As the process of the determination of the sign of any number is very simple and the process of the extension of integers to real numbers (with embedded radix point) is also very simple, therefore, for simplicity of exposition we have developed our package for non-negative integers only. Since we are mostly working with integers upto 80 decimal digits and the emphasis on the development of the programs is that they should work independently of any specific hardware (for the sake of portability of the programs), so we have developed a package which just mechanises the implementation of the paper-and-pencil methods of calculations with integers.

This multi-precision package includes the following

(a) addition of 2 m,n-piece numbers (where 1 piece stands for d decimal digits, d > 1),

(b) subtraction of an n-piece number from an m-piece number,

(c) multiplication of 2 m,n-piece numbers,

(d) division of an m-piece number by an n-piece number,

(e) left and right shifts of an n-piece number by a given number (say t) of places,

(f) input and output of n-piece numbers.

Given 2 integers $x_p \cdots x_3x_2x_1$, $y_q \cdots y_3y_2y_1$ (where each x and y stands for a single decimal digit), we divide each number into pieces, containing a fixed number of digits (this particular number depending on the particular computer and the type of integer variables on which we are working), starting from the right. For example, in IBM 1620 series computer the upper limit is 99999. So we can take 5 digits in each piece. But the upper limit for an INTEGER*2 variable in many other computers is 32767 and for simplicity of exposition we have taken 4 digits in each piece, i.e., in effect we have changed the radix from 10 (the usual decimal representation of numbers) to 10^4 and consequently $(u_n \cdots u_3u_2u_1)_{10}$ becomes $(v_m \cdots v_3v_2v_1)_{10^d}$ which is said to be an m-piece number.

1.1 Addition of 2 multi-precision non-negative integers

When we have to add 2 integers $x_i \cdots x_3x_2x_1$ and $y_j \cdots y_3y_2y_1$, we divide each number into pieces as described above. We store the i,j pieces separately in 1-dimensional arrays called L and M of maximum size z each. It means that we can take care of maximum $z \times d$ digits in each integer. Then we add M(1) to L(1), M(2) to L(2), M(3) to L(3), etc., using the following algorithm:-

We consider L and M as vectors and we store the required sum in another vector called N. We define the components of N as follows:-

A1. Initially we set $c = 0$ (c means the carrying digit).

A2. Then for $1 \leq k \leq \min(i,j)$

Let s= g − m_k − c where g = 10^d−1.

Now if l_k > s, we define n_k as l_k − (s+1) and set the carrying digit c as 1;

otherwise we define n_k as g −(s−l_k) and set c as 0.

10.3264

A3. If i≠j, then for min(i,j)+1 ≤ k ≤ max(i,j)

we set n_k to l_k or m_k depending on whether i > j or vice-versa.

Now if the carrying digit c = 0 in the end, we leave all pieces unchanged,

and if the carrying digit c = 1 in the end,

then if n_k = g, we set n_k to 0 and c remains 1;

otherwise we set n_k to n_k + 1 and c to 0 and terminate the process.

Thus we insure that no number stored in a single memory location becomes negative or ≥ 10^d (even though the actual upper limit may be much higher, namely 32767 for INTEGER*2, still we do not allow any number to become ≥ 10^d). In the process of adding the numbers if we take care of all the pieces and even then the carrying digit c remains 1, then we increase the number of pieces in the sum N by unity, i.e., the number of pieces in N becomes equal to max(i,j)+1 and we set the leftmost piece of N to 1. In this process of addition if the total number of pieces becomes greater than z in the end, we set an indicator IND to 1 and terminate the process. We print the radix-10^d sum N(1),N(2),N(3) etc. backwards in a single line. While printing, if some piece is less than 10^{d-1}, in the normal printing the computer will leave a certain number of blanks. To avoid these blanks we use a special SUBROUTINE subprogram PRINT which fills up these blanks by zeros

using dynamic formatting. We do not do so in the leftmost piece, so that all the numbers are printed in the conventional way. The program together with all the SUBROUTINE subprograms used is given in the Appendix.

1.2 Subtraction of 1 multi-precision non-negative integer from another multi-precision non-negative integer

For subtraction of $m_i \cdots m_3 m_2 m_1$, an i-piece number, from $l_j \cdots l_3 l_2 l_1$, a j-piece number, before we start the subtraction, we must know which of the 2 integers (if any) is bigger. For this purpose, first of all we compare i and j. If i is greater than j, then we decide to subtract the minuend from the subtrahend. If j is greater than i, we decide to subtract the subtrahend from the minuend. If i = j, we compare m_j with l_j, m_{j-1} with l_{j-1} and so on using a DO loop. If at any stage we find that the kth piece of the minuend is bigger, we decide to subtract the subtrahend from the minuend, and if we find otherwise, we decide the reverse. If we find that $m_k = l_k \; \forall$ k, we conclude that both the numbers are equal. In this case we set an index equal to 0 and define the remainder to 0. For this purpose we say that the remainder is a 1-piece number 0. If our decision is to subtract the subtrahend from the minuend, we again set the index to 0 and start the calculations. If our decision is the reverse, we set the index as 1 and start the subtraction in the reverse order. This technique helps in determining the sign of the remainder. If the index happens to be 1, we print a negative sign before the remainder. This happens if the subtrahend is greater than the minuend. The program sets the index to 0, if the subtrahend is less than or equal to the minuend

and consequently we do not print any negative sign in the beginning of the remainder. So in all cases the answer printed is according to the conventional method. We use the following algorithm, so that during the subtraction of v_k from u_k, no piece of the remainder becomes < 0, or $> 10^d - 1$ (i.e., $> g$), where g is the greatest number of d digits, and we take care of any possible borrowing figure, which we can easily prove to be either 0 or 1. (Here, if we are subtracting the subtrahend from the minuend, then u_k means l_k and v_k means m_k; otherwise u_k means m_k and v_k means l_k.)

A1. Initially we set the borrowing figure $b = 0$.

A2. Then for $1 \leq k \leq \min(i,j)$

Let $r = u_k - v_k$.

Now if $r > 0$, n_k is defined as $r - b$ and we set b to 0;

and if $r = b = 0$, n_k is set to 0 and b remains unchanged;

otherwise n_k is defined as $r + 1 - b + g$, where $g = 10^d - 1$ and we set b to 1.

A3. Now if $i = j$, we are through (at this stage b cannot be 1);

A4. and if $i \neq j$, then for $\min(i,j)+1 \leq k \leq \max(i,j)$

we set n_k to l_k or m_k, depending on whether $i > j$ or vice-versa.

Now if the borrowing figure $b = 0$, we leave all the pieces unchanged;

and if the borrowing figure $b = 1$,

then if $n_k = 0$, we set n_k to g and b remains unchanged;

otherwise we set n_k to $n_k - 1$ and b to 0 and terminate the process. At the time of termination of the process b cannot be 1 as in the case i = j.

Thus no number stored in a single memory location in the process of subtraction can become negative or greater than 10^d-1. The answer is printed as in the case of addition, but if in the difference N(1), N(2), N(3) ,$\cdots$, N(k), say, N(k) = N(k−1) = N(k−2) = 0 but N(k−3) > 0, then we do not print N(k), N(k−1), N(k−2) at all. In this case we print only from N(k−3) to N(1) in 1 line using dynamic formatting with the help of the SUBROUTINE subprogram called PRINT. The program of subtraction together with all the subroutines used is given in the Appendix.

1.3 Multiplication of 2 multi-precision integers

We perform the multiplication of 2 i,j piece-integers $a_i\cdots a_3a_2a_1$, $b_j\cdots b_3b_2b_1$ by combining the paper-and-pencil method with a table-lookup scheme iteratively. For this we create a table of 8 products of the multiplicand by 2,3,4,$\cdots$,9 with the exception that if the multiplier does not contain a particular digit, then we do not store the product of the multiplicand by that digit. In case of the digit 0, we do nothing and for the digit 1 we take the multiplicand as such. We perform the multiplication of the multiplicand $a_i\cdots a_3a_2a_1$ by a single digit using a SUBROUTINE subprogram called MULT developed for this purpose which uses the following algorithm for multiplying each piece a_k of the multiplicand by a single digit x :-

A1. Initially we set p_1 equal to 0.

A2. For $1 \leq k \leq i$

Let $q = [a_k/t]$ where $t = 10^{d-1}$,

$m = (a_k - tq)x + p_k$,

$n = qx$,

$q = [n / 10]$,

$y = m - (g - t(n - 10q))$ where $g = 10^d - 1$.

Now if $y > 0$ then p_k is defined as $y - 1$ and q is set to $q + 1$

otherwise p_k is defined as $y + g$ and q remains unchanged.

For both the cases we set p_{k+1} to q except for the leftmost piece, and for the leftmost piece, i.e., for the ith piece,

A3. if $q \neq 0$ we set p_{i+1} to q and correspondingly the number of pieces in the partial product of the multiplicand by a single digit x becomes i+1, i.e., 1 more than the number of pieces in the multiplicand.

([x] denotes the integer part of x, i.e., [55 / 10] = 5.)

We develop a SUBROUTINE called SHIFT for shifting a k-piece non-negative integer by r places to the left (and for padding these r places by zeros), and use it to shift the partial products by the required numbers of places, as we normally do in the human domain. Then we add the partial product obtained by the multiplication of the multiplicand by a single digit to the earlier partial product instead of the conventional method of first finding all the partial products and then adding all of them with appropriate shifting. We implement the addition using the SUBROUTINE called BIGADD

written for the addition of multi-precision integers. We take special precaution about the number of pieces obtained in the processes of multiplication by a single digit, shifting and adding. If in any of these processes this number becomes greater than z, then we immediately set an indicator IND as unity and return to the main program. The main program then prints out an appropriate message and aborts the job. The limit of z pieces is however not fixed, but it can be increased or decreased as desired. We print the answer with embedded zeros, if any, instead of blanks, through the use of the same SUBROUTINE called PRINT, which we have been using in the other programs. The main program together with all the SUBROUTINE subprograms used is given in the Appendix.

1.4 Division of a multi-precision non-negative integer by a multi-precision positive integer

The division of a multi-precision non-negative integer by a multi-precision natural number is the most difficult of the 4 arithmetic operations. The main difficulty in the mechanisation of the pencil-and-paper method of division is the determination of the digits of the quotient. We shall denote a particular digit by q_k. In order to find the digit q_k, first we find a trial digit q'_k by dividing the d leftmost digits of the partial dividend by the d or d − 1 leftmost digits of the divisor. If we form the partial dividend with n digits of the complete dividend, (n being the number of digits in the divisor), then we take the d leftmost digits of the divisor for this division. On the other hand, if we form the partial dividend with n digits of the complete dividend,

(n−1 being the number of digits in the divisor), then we take the d − 1 leftmost digits of the divisor for this division. **We shall prove that the maximum value of q'_k is 10 and the exact digit q_k in the quotient must be either q'_k or $q'_k - 1$ in the 10^d-radix division, where d > 2.** For this purpose we shall take that case in which the exact digit will be the least possible, i.e., we shall assume that all the remaining digits in the partial dividend after the leftmost d digits are 0's and all the corresponding digits in the divisor are 9's. So in this way this case will be the most favourable for the exact digit q_k in the quotient to be as small as possible. We are tacitly assuming that n > d.

Proof:-

First of all we shall consider the case in which we are dividing the first d digits of the partial dividend by the first d digits of the divisor.

We designate $x_1x_2x_3...x_d$ as x and $y_1y_2y_3...y_d$ as y. Also we assume that the trial digit of the quotient is q'_k. We shall prove that, if d > 2, the exact corresponding digit of the quotient will be either q'_k or $q'_k - 1$.

It is clear that $1 \leq q'_k \leq 9$ and $x \geq q'_k y$. (1)

The number formed by the leftmost n digits of the complete dividend is now $x \times 10^{n-d}$ and the complete divisor (containing n digits) is $10^{n-d}(y+1)-1$.

Now

$$x \times 10^{n-d} - (q'_k - 1)\{10^{n-d}(y+1) - 1\}$$

$$= 10^{n-d}(x - q'_k y) + 10^{n-d}y - 10^{n-d}q'_k + 10^{n-d} - 1 + q'_k$$

$$= 10^{n-d}(x - q'_k y) + 10^{n-d}(y - q'_k) + (10^{n-d} - 1) + q'_k. \quad (2)$$

The quantity inside the first bracket ≥ 0 because of (1), the quantity inside the second bracket > 0 because $y > q'_k$ (since $y \geq 100$ and $q'_k \leq 9$), the quantity inside the third bracket > 0 (because $10^{n-d} > 1$), and we have something more at the end.

So from (2), we have

$x \times 10^{n-d} \geq (q'_k - 1)\{10^{n-d}(y+1)-1\}$.

This proves that the exact digit of the quotient cannot be $< q'_k - 1$.

Now we shall consider the case in which we are dividing the first d digits of the partial dividend by the first $d - 1$ digits of the divisor.

As before, we designate $x_1x_2x_3...x_d$ as x and $y_1y_2y_3...y_{d-1}$ as y. Also we assume that the trial digit of the quotient is q'_k. In this case also, we shall prove that, if $d > 2$, the exact corresponding digit of the quotient will be either q'_k or $q'_k - 1$. We further denote the complete divisor (containing $n-1$ digits, say) as b and the number formed by the leftmost n digits of the complete dividend as a.

So a is actually $x_1x_2x_3...x_d000...0$ (where there are $n-d$ 0's) and b is $y_1y_2y_3...y_{d-1}999...9$ (where there are $n-d$ 9's). For the purpose of explaining (5) below, we shall now split up a as $a_1a_2a_3...a_n$ and b as $b_1b_2b_3...b_{n-1}$.

As before, it is clear that $x \geq q'_k y$. (3)

Also we have $d > 2$. (4)

Further we must have $[\frac{x}{10}] \leq y$, because otherwise there would have been no necessity of dividing a d-digit number by a $(d-1)$-digit number.

Also in case $[\frac{x}{10}] = y$, $\exists\ i \mid d \leq i \leq n - 1$ and $b_i > a_i$,

whereas $b_j = a_j\ \forall\ j \mid 1 \leq j < i$.

Since $d > 2$, $10y \leq x < 11y \Rightarrow q'_k = 10$. (5)

In the other case, i.e., $[\frac{x}{10}] < y$, we have $q'_k \leq 9$. (6)

Now, as before, we shall have

$a = x \times 10^{n-d}$ and $b = 10^{n-d}(y+1) - 1$.

As such, we have

$$\begin{aligned} & a - (q'_k - 1)b \\ &= x \times 10^{n-d} - (q'_k - 1)\{10^{n-d}(y+1) - 1\} \\ &= 10^{n-d}(x - q'_k y) + 10^{n-d}y - 10^{n-d}q'_k + 10^{n-d} - 1 + q'_k \\ &= 10^{n-d}(x - q'_k y) + 10^{n-d}(y - q'_k) + (10^{n-d} - 1) + q'_k. \end{aligned} \quad (7)$$

The quantity inside the first bracket ≥ 0 because of (3), the quantity inside the second bracket ≥ 0 because $y \geq q'_k$ (since $y \geq 10$ and $q'_k \leq 10$), the quantity inside the third bracket > 0 (because $10^{n-d} > 1$), and we have something more at the end.

So from (7), we have

$x \times 10^{n-d} \geq (q'_k - 1)\{10^{n-d}(y+1)-1\}$.

So we have proved that in all possible cases the trial digit q'_k of the quotient can be at most 10 and the exact digit q_k cannot be $< q'_k - 1$.

1.4.1 Algorithm

Given a dividend $a_i \cdots a_3 a_2 a_1$ and a divisor $b_j \cdots b_3 b_2 b_1$, to obtain the quotient and the remainder, we initialise q_1 with 0 and set the number of pieces in the quotient as 1. Then we start the process by comparing i and j; if $j > i$ then the quotient is 0 and the remainder is the dividend itself. Otherwise for $k = j(-1)1$ we store a_{k+i-j} into a_k' and if $i - j > 0$, we digitise the part

$a_{i-j}a_{i-j-1}\cdots a_1$ of the dividend using a SUBROUTINE called INTEG, coded accordingly. Now if $a_j'\cdots a_3'a_2'a_1' < b_j\cdots b_3b_2b_1$, then we try to form the first partial dividend by appending extra digit(s) from the extreme left of the remaining digit(s) in the dividend to $a_j'\cdots a_3'a_2'a_1'$. If no remaining digits are left in the dividend, then again the quotient is 0 and the remainder is the dividend itself; otherwise we determine the number of digits i', j' in the leftmost pieces of the dividend and the divisor. We now initialise IDEND and ISOR with the leftmost d digits of the dividend and the divisor (provided that the divisor contains at least 2 pieces) and then check whether any right shifts of the leftmost j − 1 pieces of $a_j'\cdots a_3'a_2'a_1'$ are required. The number of right shifts r is $i' - (j' + 1)$ if IDEND < ISOR, otherwise it is $i' - j'$. If necessary we further digitise the rightmost r digits of a_1' and correspondingly right shift the j − 1 leftmost pieces of $a_j'\cdots a_3'a_2'a_1'$. This we perform in the SUBROUTINE subprogram PARDEF. Then we divide IDEND by ISOR or IDEND by ISOR/10 depending on whether IDEND ≥ ISOR or not, to find the trial digit q_k' of the quotient. After calculation of the trial digit we find the exact digit q_k such that $0 \leq (a_j'\cdots a_3'a_2'a_1') - q_k\ (b_j\cdots b_3b_2b_1) < b_j\cdots b_3b_2b_1$. We have proved that the maximum possible decrement of q_k' is 1. This is carried out in the SUBROUTINE called EXACDQ. In the process of calculating the exact digit we also calculate the partial remainder $(a_j'\cdots a_3'a_2'a_1') - q_k\ (b_j\cdots b_3b_2b_1)$, which is implemented with the help of the SUBROUTINE subprograms called BIGMUL and BIGSUB. After calculating the exact digit of the quotient q_k we append it to the existing quotient to form a partial quotient. Then we try to make the new partial dividend greater than or equal to the divisor by appending further extra digit(s) from

the extreme left of the remaining digit(s) in the dividend to the partial remainder obtained in the process of the last division, as we do in the conventional method. We have thus developed the SUBROUTINE APPEND for appending digit(s) to the right of the existing digit(s). Now we again divide the new d leftmost digits of the dividend by the d or d−1 leftmost digits of the divisor as explained before, which gives us 1 more digit in the quotient. The second possibility arises when either the number of pieces in the partial dividend becomes > j (the number of pieces in the divisor) or the number of digits in the leftmost piece of the divisor < d and the leftmost piece of the partial dividend $a_j{}'$ is $\geq 10^{j'}$. After appending a single digit to the partial remainder from the left-over portion of the given dividend, if we find that the number to be divided still remains less than the divisor, we append 0 to the quotient, just as we do in the human domain. We continue this process till all the digits of the given dividend are exhausted. We print the desired quotient and the desired final remainder (which is the last partial dividend) using the SUBROUTINE subprogram called PRINT.

1.5 The square root

The next algorithm of this chapter is the evaluation of the square root of a positive number, which is considered to be next in importance to the 4 fundamental arithmetic operations, regarding its usefulness in diverse areas of Science and Engineering, e.g., trigonometry, solution of quadratic equations, 2-dimensional modelling, graphics, image processing, etc. [21].

The Newton-Raphson quadratically convergent iterative process is one

of the most widely used methods for the evaluation of the square root of a positive number. This method is equally suitable for integer and real arithmetic.

In many applications only the integer part of the square root of a natural number is required for further calculations. For example, in determining whether a given natural number is a perfect square, in the computation of prime numbers and similar other calculations we require only the integer part of the square root of a natural number.

In the Newton-Raphson method for calculating the square root of x, we choose the initial approximation to be a natural number $x_o \geq \sqrt{x}$. Then we apply the formula

$$x_i = \frac{1}{2}(x_{i-1} + \frac{x}{x_{i-1}}),\ i \geq 1.$$

If we do this and we find that at some stage $x_i \geq x_{i-1}$, then we can show that $x_{i-1} = [\sqrt{x}\,]$, where we have used integer arithmetic in all the calculations. With close first approximation the running time of evaluation of the integer part of the square root of a positive number x can be reduced from $O(ln^3\ x)$ to $O(ln^2\ x)$ [10].

But the Newton-Raphson method does not provide any scheme for a close estimation of the initial approximation to the square root. **We have given a mathematically elegant proof to show that the initial approximation**

y $= 2^{d-1} + [\frac{x}{2^{d+1}}]$ where x is a 2d bits' number,

$= [.71(2^{d-1} + [\frac{x}{2^d}] + 1)]$, where x is a 2d$-$1 bits' number,

is greater than or equal to $[\sqrt{x}\,]$. For an even bits' number the initial estimate y is the same as the closest known approximation suggested by Hashemian [21]. But for an odd bits' number y is much closer than the value given in his paper.

Proof:-

Case I. Let x be a 2d bits' number, i.e., an even bits' number.

Then $x = k2^{d+1} + l$, where $2^{d-2} \leq k \leq 2^{d-1} - 1$,

and $0 \leq l \leq 2^{d+1} - 1$.

Now $y = 2^{d-1} + k$; (1)

then $y^2 = 2^{2d-2} + k^2 + 2^d k$

$\geq 2^{2d-2} + 2^{2d-4} + 2^{2d-2}$

$= 2^{2d-1} + 2^{2d-4}$

$\geq 2^{2d-1}$.

Similarly by taking k $= 2^{d-1} - 1$, we can show that $y^2 \leq 2^{2d} - 1$.

Thus y^2 is a 2d bits' number.

Putting $k = 2^{d-2} + k'$ in (1) where $0 \leq k' < 2^{d-2}$,

we have y $= 2^{d-1} + 2^{d-2} + k'$.

Now

$y^2 - k2^{d+1} = (2^{d-1} + 2^{d-2} + k')^2 - 2^{d+1}(2^{d-2} + k')$

$= 2^{2d-2} + 2^{2d-4} + k'^2 + 2^{2d-2} + k'2^{d-1} + k'2^d - 2^{2d-1} - k'2^{d+1}$

$= k'^2 + k'(2^{d-1} + 2^d - 2^{d+1}) + 2^{2d-4}$

$= k'^2 - 2^{d-1}k' + 2^{2d-4}$

$= (2^{d-2} - k')^2 > 0$

$\Rightarrow \frac{y^2}{2^{d+1}} > k$

$\Rightarrow [\frac{y^2}{2^{d+1}}] \geq k$

$\Rightarrow$ the number formed by the most significant (d−1) bits of y^2

$\geq$ the number formed by the most significant (d−1) bits of x

$\Rightarrow y^2 > x$ (when the least significant (d+1) bits of x are all zeros)

$$\Rightarrow y - 1 \geq [\sqrt{x}] \tag{2}$$

Now $(y+1)^2 - x = (2^{d-1} + k + 1)^2 - (k2^{d+1} + l)$

$$= 2^{2d-2} + 2^d k + k^2 + 2^d + 2k + 1 - k2^{d+1} - l$$

$$= 2^{2d-2} - 2^d k + 2k + k^2 + 2^d + 1 - l$$

$$\geq 2^{2d-2} - 2^d k + 2k + k^2 + 2^d + 1 - 2^{d+1} + 1$$

$$= 2^{2d-2} - 2^d k + 2k + k^2 - 2^d + 2$$

$$= (2^{d-1} - 1 - k)^2 + 1 \geq 1 \tag{2'}$$

$$\Rightarrow (y+1)^2 > x \tag{3}$$

$$\Rightarrow y \geq [\sqrt{x}]. \tag{4}$$

Thus it is clear from (2) and (4) that the integer part of the square root differs at most by 1 from the number formed by the least significant (d+1) bits of x. Since $2^{d-2} \leq k \leq 2^{d-1} - 1$, it follows from (2′) and (4) that the difference between $[\sqrt{x}]$ and y is least (i.e., $[\sqrt{x}]$ = y or y − 1) when k = $2^{d-1}-1$ and is greatest when k = 2^{d-2}. And the difference between $[\sqrt{x}]$ and y becomes smaller and smaller as the value of k increases (for many values of x, $[\sqrt{x}]$ = y or y − 1).

Case II. Let x be a 2d−1 bits' number, i.e., an odd bits' number.

Then $2x = k2^{d+1} + l$, where $2^{d-2} \leq k \leq 2^{d-1} - 1$,

$$\text{and } 0 \leq l \leq 2^{d+1} - 2.$$

Now $y = 2^{d-1} + k$.

So (3) $\Rightarrow (y+1)^2 > 2x$

$\Rightarrow (y+1) > \sqrt{2x}$

$\Rightarrow [.71(y+1)] \geq [\sqrt{x}\,]$.

In this process the explicit computation of the binary representation of the number x is not required. To determine d it is only required to find m such that $2^m \leq x < 2^{m+1}$, (which can be found out simply by $O(ln\ ln\ x)$ searches[10]). For an even bits' number, the initial approximation is computed by a division and an addition and for an odd bits' number, by a division and two additions followed by a multiplication. This method of computation of an initial close estimate of the integer part of the square root of a positive integer x is equally applicable for computing an initial close estimate of the square root of a real positive number which is required in the process of calculating its exact square root.

1.6 Prime numbers

The computation of all the prime numbers from 2 to a given natural number N has captured the attention of pure and applied mathematicians alike for centuries because there are many important results associated with prime numbers, e.g., every natural number > 1 can be expressed as a product of primes (and this expression is unique upto the order of the primes), the number of primes not exceeding x is asymptotic in its behaviour to $\frac{x}{logx}$, i.e., $\pi(x) \sim \frac{x}{logx}$, etc. Moreover it has wide applications in many computations, e.g., in modular arithmetic, in the factorisation of polynomials over **Z** and in computational algebraic number theory. There exist many algorithms for

the computation of prime numbers, e.g., [23,28,39,46]. But many algorithms have more theoretical significance than practical [28,39]. Luo [39] has made a significant practical improvement in the sieve of Eratosthenes (3rd century B.C.) in which he has obtained gain in speed by reducing the auxiliary storage requirement, the storage requirement of the improved algorithm being $N/3$, whereas the arithmetic complexity of computation and the storage requirement of the sieve of Eratosthenes are $O(NloglogN)$ and $O(N)$ respectively. However Kumar's algorithm [28] does not require any extra storage but it is very slow as compared to the improved sieve algorithm due to the large number of divisions and multiplications required for the testing of primality. It requires k−1 divisions and an equal number of multiplications for testing the primality of a prime x such that $p_k p_{k+1} < x < p_{k+1} p_{k+2}$, where p_k denotes the kth prime. But the practical significance of these 3 algorithms is limited, the first 2 due to their relatively large storage requirements and the 3rd due to its comparatively slow speed of computation. We have developed an optimal algorithm with regard to running time and storage space.

Pritchard [46] has shown that the programs attributed to Wirth and Misra for the computation of all prime numbers from 2 to N are wrong because these programs accept even composite numbers as primes. He has also constructed counter-examples to disprove a conjecture of Misra. The program which originated with Dijkstra has also been discussed in the paper by Pritchard for constructing counter-examples. It works as follows:-

First the only even prime 2 is accepted. Then each odd integer x = 3,5,7,... is taken in turn and tested for primality. The program maintains 2 arrays. The first array consists of all accepted primes, that is, those less than x.

$p[j] = p_j$, $1 \leq j \leq \pi(x-1)$

(Here $\pi(x)$ denotes the number of primes less than or equal to x.)

The second array V is used to avoid dividing x by p[j]. The intention is that the assertion

V[j] = the least multiple of p_j not less than x

should hold whenever it would have been necessary to divide x by p[j] if the array V would not have been maintained. In Dijkstra's program V[j] is initialised with $p[j]^2$ as soon as $x = p[j]^2$. In this way the primality test is performed avoiding the division operation. The program tests the equality of x with V[j] for all j such that $2 \leq j \leq \pi(\sqrt{x})$. If for some j, V[j] = x, then it means that $p_j | x$, in which case x is rejected as composite; otherwise x is accepted as prime and is added to the already existing array p containing all the primes. If V[j] < x, then V[j] is set to V[j]+p[j] and the test is repeated and if V[j] = x, then x is declared to be composite. In this way x is accepted as prime if and only if $x < V[j]\ \forall\ j$.

It has been proved by Kumar that if p_j, p_{j+1}, p_{j+2} are 3 successive primes in increasing order, then an odd natural number x, such that $p_j p_{j+1} < x < p_{j+1} p_{j+2}$ and $x \neq p_{j+1}^2$, and $p_k \nmid x$, $k = 2,3,4,...,j$, must be a prime. The program which we have developed for the computation of prime numbers implements the above technique with some modifications tailored in order to avoid the division test. The algorithm is given below.

We accept the only even prime 2. The next 2 primes are given by the relation $p_1 m + a$, where a = 1, for m = 1,2, and primes greater than $p_1 p_2$ upto 30 are given by the relation $m p_1 p_2 + a$ such that a = 1,5 for m = 1,2,3 and a = 5 for m = 4. For numbers greater than 30, instead of testing all the

odd numbers from 30 to N, we test only those numbers which are relatively prime to 30, i.e., the numbers which we consider for the testing of primality are of the form 30m + a, where a,m $\in$ **N** and (a,30) = 1. Thus we never test odd multiples of 3 and 5 for primality. The composite numbers of the form p_j^2 and p_jp_{j+1} are used for initialisation of S_j (the least multiple of p_j not less than x, where x is that number which we are testing) and for determining the number of primes (k) whose multiples are required for deciding whether a number $x > p_jp_{j+1}$ is a prime or not. S_j is initialised with $p_jp_{j+1} + 2p_j$ when $x = p_jp_{j+1}$ and k is set to k + 1. And we can prove very easily that for the least number $x > p_jp_{j+1}$, S_j is the least odd multiple of p_j not less than x, the required condition for performing the primality test without the division operation. Then x is determined to be a prime or a composite by the procedure described below.

For $4 \leq j \leq k$, where $p_kp_{k+1} < x < p_{k+1}p_{k+2}$,

if $x < S_j$, then S_j is set to $S_j + 2p_j$ and the test is repeated.

And if $x = S_j$, S_j is set to $S_j + 2p_j$ and x is declared to be composite.

Thus we accept x as a prime if $x < S_j \; \forall$ j.

The chosen numbers are of the form 30m + a, because 30 is the largest number which satisfies the condition that any prime greater than it can be written in the form 30m + a where a = 1 or a prime < 30 and 30m + a is not divisible by 2,3,5. The increment $2p_j$ is permissible since $p_j \mid x \Rightarrow S_j = x$

$\Rightarrow S_j$ is an odd multiple of p_j (as x is odd).

If S_j is an odd multiple of p_j, then the least odd multiple of $p_j > S_j$ is $S_j + 2p_j$.

The choice of the numbers of the form 30m + a, a gap of $2p_j$ (in-

stead of p_j) and initialisation of S_j only when $x = p_j p_{j+1}$ (and not when $x = p_j^2$) will reduce the number of comparisons and correspondingly also the number of arithmetic operations considerably. As such the computation of primes with this algorithm will be much faster than the algorithm of [28]. Also the auxiliary storage requirement for primes upto x is k + 10 where $p_k p_{k+1} < x < p_{k+1} p_{k+2}$, which is negligible compared to the sieve algorithms.

We close this chapter with 2 closely related combinatorial problems which perform the generation of all the permutations and combinations of the first n natural numbers taken r at a time. We have found that a control structure for variable number of nested loops is very suitable for solving a large number of combinatorial problems (including the generation of permutations and combinations) and also problems requiring backtracking. As such we have first discussed about such structures and then we have solved the problems using these structures.

1.7 Generalisation of the control structure for a variable number of nested loops

A new control structure with a variable number of nested loops was proposed by Skordalakis and Papakonstantinou [51] and an alternative one by Mckenzie and Takaoka [40]. The control structure can be generalised further so as to start a nest of DO loops from any set of starting values and to break the sequence of values of the indices of the nested loops, if required. This we propose in this section to incorporate in the FORTRAN-type DO loop structure. This generalisation removes the unnecessary repetition of calcu-

lations and avoids the waste of effort in computing those values which are never actually used. It is of practical use in those problems for which the results cannot be obtained in 1 execution and in interrupted combinatorial problems.

In the paper [40] they preferred the option (c), i.e., implementing the new control structure through subroutine calls, out of 3 available options, namely,

(a) modifying the compiler of the host programming language to incorporate the new control structure;

(b) developing a preprocessor to translate the augmented programming language into the host programming language;

(c) implementing the new control structure through subroutine calls.

The alternative procedure suggested by Mckenzie and Takaoka has the disadvantage that we have to physically write all the DO loops; as a result, if a program has 50 nested DO loops, it will be an unnecessarily long program; moreover every time the number of DO loops is changed, we shall have to re-write the whole code and consequently to compile it again. We propose the following changes in the format of the control structure proposed by Skordalakis and Papakonstantinou before generalising it further.

Instead of calling the SUBROUTINE subprograms PRE, CORE AND POST from another SUBROUTINE subprogram NEST, we have adopted the technique of calling the SUBROUTINE subprograms from the main program, because all the SUBROUTINE subprograms may not be required in solving a particular combinatorial problem; e.g., the SUBROUTINE subprogram

called POST is not required in the problem of colouring the vertices of the graph. So it will be a better practice if we leave the calling of SUBROUTINE PRE etc. to the main program. As a result the SUBROUTINE NEST generating the DO loops in stricter sense will remain unchanged in all the cases whether the SUBROUTINE subprograms PRE etc. are required or not. We can call them from the main program depending on their necessity.

1.7.1 Description of the Problems

Often in real life situations it so happens that we have a big nest of many DO loops, say, 4 loops, where I = 1(1)10 in the first loop, J = 1(1)15 in the second one, K = 1(1)20 in the third one and L = 1(1)25 in the fourth one. Then the computer has to make a total of $10\times15\times20\times25 = 75000$ sets of calculations. It is very much possible that the user of the computer may not have so much computer time at his disposal at 1 stretch and he might want to use the computer in different sittings. In such a case if he starts the DO loops again from the same point on every occasion, his purpose will not be served, because every time the computer will print out the same set of answers. This problem has actually arisen during some of our calculations because we had to calculate some physical quantities which depended on many parameters and in 1 sitting we were allowed certain fixed computer time. The calculations of each set were so lengthy and so complicated that the computer could not give out more than 3 or 4 results during the allotted time. A lacuna in the DO loop structure is that it does not take care of such situations. For example, consider the following nest of DO loops:-

```
      DO 10 I = 1, 10
      DO 11 J = 1, 15
      DO 12 K = 1, 20
      (actual calculations)
12    CONTINUE
11    CONTINUE
10    CONTINUE
```

Suppose that during the first sitting we have calculated the results corresponding to (1,1,1),(1,1,2), $\cdots$,(1,1,20),(1,2,1), $\cdots$,(1,2,5), i.e., 25 results in all, and during the next sitting we wish to start our calculations from (1,2,6). Then this is not possible using this structure. At a first sight it appears that there exists a simple solution to this problem. Let us consider the following technique:-

```
      READ * , II,JJ,KK,III,JJJ,KKK
      DO 10 I = II, III
      DO 11 J = JJ, JJJ
      DO 12 K = KK, KKK
      (actual calculations)
12    CONTINUE
11    CONTINUE
10    CONTINUE
```

It appears reasonable that on each sitting we can provide extra data (the values of II,JJ,KK,III,JJJ,KKK) relevant to that sitting, and then the program will work as desired. For example, on the first sitting we can enter the values of II,JJ,KK as 1,1,1 and we can provide suitable values for III,JJJ,KKK. After obtaining 25 results, say, when we want to start from (1,2,6), we can enter the values of II,JJ,KK as 1,2,6, and so on. But if we do that, then after calculating for the sets (1,2,6),(1,2,7), $\cdots$,(1,2,20), the computer will start the calculations from the set (1,3,6) and not from the set

(1,3,1), as we would have wanted it to do. So a number of sets will be left out. This is now clear that this technique is not correct. It can work only if there is a single DO loop and not when there is a nest of n loops where n > 1. Therefore in the DO loop structure we have tried to introduce this condition also, so that we can start executing the nest of DO loops from any point desired, and the computer will execute for all the possible combinations for each set in the correct order, neither missing any set, nor calculating more than once for any set. For this purpose we have introduced a SUBROUTINE called START together with an extra 1-dimensional array called LNET. For example, if in a particular case, N = 3 and the 4 arrays LINV, LTEV, LCRE, LNET are as follows:- LINV = (3,1,9), LTEV = (10,15,14), LCRE = (1,2,1), LNET = (4,3,11), the arrays LINV, LTEV, LCRE, LNET containing the initial values, test values, increments and starting values for the 3 loops respectively, then the computer will start calculating from the set (4,3,11) and then it will calculate for (4,3,12),(4,3,13),(4,3,14),(4,5,9),(4,5,10),(4,5,11),(4,5,12),(4,5,13), (4,5,14),(4,7,9), (4,7,10) $\cdots$ until the time allotted to the user is over. On the next sitting the user can enter fresh values for the elements of the array called LNET and restart the computer. For example, if on the previous sitting he has computed upto (5,7,10), then he can type LNET as (5,7,11) and proceed.

In some uses of the nested DO loops structure it becomes necessary that the sequence of the values of the indices of the DO loops needs to be interrupted. For example, if we have a nest of 4 DO loops, where I = 1(1)5 in the first loop, J = 1(1)10 in the second one, K = 1(1)15 in the third one, and L = 1(1)20 in the fourth one, then it is very much possible that after

the calculations have been carried out for the set (1,2,1,1), no further calculations are required with the starting values (1,2), which is said to be the prohibited sequence. And further calculations should proceed from the next set of possible values (if any) in the correct order. In this case the calculations will re-start from the set (1,3,1,1). We have taken care of this problem by introducing a 1-dimensional array called INTERP, a scalar called NINT and by coding a SUBROUTINE subprogram called POST appropriately. In this case the array INTERP in memory is (1,2,1,1) and the value of NINT is 2.

1.7.2 Implementation

The generalised structure consists of 5 SUBROUTINE subprograms called NEST, PRE, CORE, POST and START instead of 4 in the structure proposed in [51]. The purposes of the different SUBROUTINE subprograms are as follows:-

START: This SUBROUTINE facilitates the starting of the execution from any set of valid starting values.

NEST: A call to this SUBROUTINE returns the required set of values of the indices.

PRE: It determines whether a particular condition is satisfied by the set of values of the indices.

CORE: It performs the calculations using the current set of values.

POST: It breaks the sequence of the values of the indices; in general this SUBROUTINE interrupts the sequence of indices at the kth loop where $1 \leq k \leq n$.

In the FORTRAN program for implementing this new control structure we calculate the set of indices together with their sum. This program is given in the Appendix. In this program we have introduced the feature that we can run a nest of N DO loops, where N is not known in advance; its value is supplied to the program via a READ statement. Moreover we have taken care of all possibilities, namely some loops may run forwards and some backwards. The same program can be used to generate

(i) all permutations

(ii) interrupted permutations [62]

(iii) given any set of permutations, the remaining permutations in the lexicographic order

of the first n natural numbers taken r at a time ($n \geq r$) with appropriate input.

By very simple modifications of the above program we can take care of the second problem, i.e., the generation of all combinations and also of all restricted combinations. Thus it is quite clear that the SUBROUTINE implementation of this generalised control structure for a variable number of nested DO loops is a very efficient tool for handling many of the combinatorial problems. It is better than the already existing FORTRAN-type DO loop structure because it has the additional facilities of starting the execution from any set of indices and interrupting a sequence of indices. Moreover our structure can implement the requirement that the indices of all the DO loops

must be pairwise distinct, if it is necessary in a particular real life situation. For example, we may be required to compute a fraction of the type

$$\frac{\phi(i_1, i_2, i_3, \cdots, i_n)}{(i_1 - i_2)(i_1 - i_3)\cdots(i_2 - i_3)\cdots(i_3 - i_4)\cdots(i_{n-1} - i_n)}$$

where $(i_1, i_2, i_3, \cdots, i_n)$ are the indices of the various DO loops. In such a case we shall certainly require that all the indices must be pairwise distinct, otherwise this fraction will have no meaning.

In the conventional FORTRAN-type nested DO loop structure this requirement of pairwise distinct indices can be met with only in the following 2 ways. In order to fix our ideas let us assume that there are 5 DO loops and each index varies from 1 to 10 with an increment of 1. Then 1 method will be to write the nest of DO loops as follows:-

```
        DO 1 I1 = 1, 10
        DO 2 I2 = 1, 10
        IF (I1-I2) 400,2,400
400     DO 3 I3 = 1, 10
        IF (I1-I3) 401,3,401
401     IF (I2-I3) 402,3,402
402     DO 4 I4 = 1, 10
        IF (I1-I4) 403,4,403
403     IF (I2-I4) 404,4,404
404     IF (I3-I4) 405,4,405
405     DO 5 I5 = 1, 10
        IF (I1-I5) 406,5,406
406     IF (I2-I5) 407,5,407
407     IF (I3-I5) 408,5,408
408     IF (I4-I5) 409,5,409
409     (actual calculations)
  5     CONTINUE
  4     CONTINUE
  3     CONTINUE
  2     CONTINUE
  1     CONTINUE
```

We may observe how clumsy and how cumbersome this coding is. In case there happen to be, say, 50 DO loops, we shall have to write 1 + 2 + 3 + 4 + 5 + ... + 49 = 1225 IF statements and we shall have to use a much bigger number of statement numbers. Another approach available in the FORTRAN-type nested DO loop structure, which is much more elegant, is as follows. We can write our program as follows:-

```
        DIMENSION M(5)
        DO 6 I1 = 1, 10
        M(1) = I1
        DO 7 I2 = 1, 10
        M(2) = I2
        LIMIT = 1
        GO TO 82
510     DO 8 I3 = 1, 10
        M(3) = I3
        LIMIT = 2
        GO TO 82
511     DO 9 I4 = 1, 10
        M(4) = I4
        LIMIT = 3
        GO TO 82
512     DO 10 I5 = 1, 10
        M(5) = I5
        LIMIT = 4
        GO TO 82
513     (actual calculations)
 10     CONTINUE
  9     CONTINUE
  8     CONTINUE
  7     CONTINUE
  6     CONTINUE
 82     DO 67 K = 1, LIMIT
        IF (M(K) - M(LIMIT+1)) 67,96,67
 67     CONTINUE
        GO TO (510,511,512,513), LIMIT
 96     GO TO (7,8,9,10), LIMIT
```

At a first sight it appears that this program will work correctly as intended. But unfortunately this coding does not obey all the rules of the FORTRAN language. In coding this program we have used the idea of 'extended range of a DO loop' but there is a rule that if we are coding a nest of n DO loops, where $n > 1$, only the innermost DO loop can have an extended range. And here it is clear that we are not being able to observe this restriction because of the many GO TO 82 statements. From every loop we are jumping out of the nest and again we are jumping into the nest; of course we are jumping into the same loop from which we had jumped out; but this jumping out and jumping in is legally permitted only from a point which lies within the innermost DO loop. And it is equally clear that we have to write all the loops physically in the program; if the total number of DO loops is large, then it will be very cumbersome.

The incorporation of such a control structure, as we have written, in the current programming languages will make them more flexible and more efficient, specially in languages like FORTRAN, BASIC and C [51].

1.7.3 General solution of the combinatorial problems using the control structure

In many combinatorial problems we have to find the set of all vectors (x_1, x_2, x_3,...,x_p), each component satisfying certain conditions, which can be obtained by using t DO loops with $x_t = I_t(J_t)K_t$, where I_t,J_t,K_t are prespecified natural numbers, which stand for the starting limit, the increment and the ending limit respectively for x_t. This process can be very easily

implemented using the above control structure. To illustrate this idea we have considered 3 most common combinatorial problems, namely (i) permutations, (ii) combinations and (iii) partitions.

We have considered all permutations and combinations of n natural numbers taking r at a time and all partitions of n objects into r mutually disjoint subsets.

Solutions:-

(i) Permutations

(a) $1 \leq x_i \leq n \ \forall i | 1 \leq i \leq r,$

(b) $x_i \neq x_j \ \forall$ j $| i + 1 \leq$ j $\leq$ r.

(ii) Combinations

(a) $1 \leq x_i \leq n \ \forall i | 1 \leq i \leq r,$

(b) $x_j < x_{j+1} \ \forall j | 1 \leq j \leq r - 1.$

(iii) Partitions

(a) $1 \leq x_i \leq n \ \forall i | 1 \leq i \leq r,$

(b) $x_j \leq x_{j+1} \ \forall j | 1 \leq j \leq r - 1$ and $\sum_{k=1}^{r} x_k = n.$

r DO loops with indexing parameters 1(1)n for each loop will take care of the condition (a) in each of the above problems, which can be implemented with the routine NEST.

The condition (b) can be tested very easily in the routine PRE with very minor modifications in code for the particular problem.

The implementation of the SUBROUTINE subprograms called NEST and PRE has been discussed in the preceding section.

For the problems (ii), (iii) even better solutions are possible.

(ii) (a), (b) $\Rightarrow 1 \leq x_1 \leq n+1-r$

and $x_{j-1}+1 \leq x_j \leq n+j-r$, $2 \leq j \leq r$.

So the indexing parameters are i(1)n+i−r for the ith loop and the control variable is such that $x_{i-1}+1 \leq x_i$ is sufficient for combinations; consequently the explicit test for $x_j < x_{j+1}$ is no longer necessary.

(iii) Similarly for partitions (a), (b)

$\Rightarrow 1 \leq x_1 \leq [\frac{n}{r}]$

and $x_{j-1} \leq x_j \leq [\frac{n+1-j}{r+1-j}]$, $2 \leq j \leq r$.

For partitions the indexing parameters are $1(1)[\frac{n+1-i}{r+1-i}]$ for the ith loop and the control variable $x_i \leq x_{i+1}$ will do the trick. Then the only condition necessary to be tested is $\sum_{k=1}^{r} x_k = n$.

CHAPTER 2

POLYNOMIAL EQUATIONS

To find the exact zeros of a polynomial over **Z** (or **Q**), **Z**[i] the set of Gaussian integers and so on, we have used the method of factorisation of polynomials over **Z** (or **Q**, which is essentially equivalent). But this factorisation is quite a difficult problem and leads to an extremely complex algorithm. The algorithm for factorisation described in [10,65] uses factorisation modulo p as a sub-resultant algorithm; it is probabilistic in nature [10]. Therefore we have confined our discussions to the processes of finding the linear and quadratic factors. The method we have described in this chapter works independently of any convergence criteria and is equally efficient for polynomials with or without multiple zeros, unlike the method presented in [59]. For finding linear factors we have used the remainder theorem and for finding quadratic factors we have used a Bairstow-type algorithm. Thus we have calculated all the rational zeros and all the irrational and complex zeros of the form a $\pm\sqrt{b}$, a being either an integer or half an odd integer and b being either an integer or the quarter of an odd integer. Furthermore, either b is negative, or if b is positive, it cannot be a perfect square, thus assuring that a $\pm\sqrt{b}$ is, after all, not rational. We can very easily extend this algorithm, so that it will also be able to calculate all the zeros of the form a$\pm\sqrt{b}$, where a and b are both rational, and either b is negative or if b is positive, it is not the square of any rational number.

As our principal interest in this chapter is to calculate the integer and rational zeros as well as the irrational and complex zeros of the particular form indicated above, so to provide test data for the programs written for the calculation of the zeros of a polynomial, we begin this chapter with programs for generating polynomials over **Z** with given zeros, which may be integer, non-integer rationals, and so on.

2.1 Reconstruction of a polynomial from its zeros

A program for reconstructing a polynomial from its zeros is given in [37]. But the program is very lengthy and complicated. The authors have first calculated all the combinations of m natural numbers taken n at a time, where m is fixed and n = 1,2,3,...,m. The SUBROUTINE subprogram used for calculating these combinations is very lengthy and does its job in a very roundabout way. It quite often unnecessarily redefines some variables. The authors have themselves remarked in their book that this program is useful for a polynomial of small degree, say 10. For a polynomial of large degree, it involves too many multiplications, additions and subtractions, thus introducing appreciable errors (if real arithmetic is used). They have suggested that a very simple and straightforward technique is to write a routine for multiplying an mth degree polynomial ($m \geq 1$) by a linear factor. In our program we have used this technique. For generating a polynomial with integer coefficients having irrational or complex zeros of the form $a \pm \sqrt{b}$ (a and b being of the form indicated on page 45), instead of multiplying an mth degree polynomial by a linear factor, we have multiplied it by a quadratic factor associated with a pair of irrational or complex zeros.

2.1.1 All zeros are integers

First of all we form a linear polynomial of the form $x - a_1$, where a_1 is the first integer zero of the required polynomial. If that is the only zero, then we are through. Otherwise we form another polynomial with the second integer zero. We multiply the previous polynomial by the new polynomial and continue the process of forming a linear polynomial with an integer zero and corresponding multiplication till all the zeros are exhausted. The coefficients of the product of the monic polynomial $P(x) = x^n + p_1x^{n-1} + p_2x^{n-2} + p_3x^{n-3} + ... + p_n$ by a linear monic polynomial $x - b$ are calculated by the relations

$p_{n+1} := - bp_n$,

$p_{n-k+1} := p_{n-k+1} - bp_{n-k}$, for k = 1,2,3,....,n−1,

$p_1 := p_1 - b$.

We have not printed the leading coefficient as the polynomial generated in this case is monic (i.e., its leading coefficient is unity). We have printed only the 2nd and later coefficients of the polynomial.

2.1.2 All zeros are rational

For those rational zeros in which the numerator is zero, the program will set the denominator as unity. If the denominator of any zero is negative, it will change the signs of both the numerator and the denominator. For all non-zero rational zeros the program will compute the H.C.F. of the numerator and the denominator and will divide both of them by that H.C.F. Now we do exactly as we did for integer zeros. The only difference in this case is

that now the linear polynomial corresponding to the rational zero $\frac{b_i}{a_i}$ is of the form $a_i x - b_i$ and we have to multiply $P(x) = p_1 x^n + p_2 x^{n-1} + p_3 x^{n-2} + \ldots + p_{n+1}$ by a linear polynomial of the form $cx - d$. The coefficients of the product of the polynomials are computed by the relations

$p_{n+2} := -dp_{n+1}$,

$p_{n+2-k} := cp_{n+2-k} - dp_{n+1-k}$, for k = 1,2,3,...,n,

$p_1 := cp_1$.

In the end we shall get an nth degree polynomial with n+1 coefficients, all of them being integers and the leading coefficient being positive. The computation of the H.C.F. of all the coefficients is not necessary because we have already taken all the zeros in their lowest terms. It follows that the H.C.F. of all the coefficients must be unity. The program together with the SUBROUTINE subprogram for the multiplication of a polynomial by a linear factor is given in the Appendix.

2.1.3 All the irrational and complex zeros of particular form

Let $x^2 - 2ax + (a^2 - b)$ be a quadratic monic polynomial, where $2a \in \mathbf{Z}$, $a^2 - b \in \mathbf{Z}$, $b \neq 0$, b is not a perfect square if $b > 0$. Then this polynomial corresponds to a pair of zeros of the form indicated on page 45 of this thesis. With the first pair of such zeros, we form a monic quadratic polynomial of the form $x^2 + cx + d$, where $c = -2a$, $d = a^2 - b$. If there exists only 1 pair of such zeros, we are through. Otherwise we form another monic quadratic polynomial of the same form with the second pair of zeros and we multiply the earlier polynomial by the new polynomial. The coefficients of the product

of the polynomial $x^n + p_1x^{n-1} + p_2x^{n-2} + ... + p_n$, $n \geq 2$, by the polynomial $x^2 + cx + d$, can be obtained by using the following relations:-

Initially we set $p_{n+2} = 0$ and $p_{n+1} = 0$.

Then we have

$p_{n+2-k} := p_{n+2-k} + cp_{n+1-k} + dp_{n-k}$ for $k = 0,1,2,...,n-1$;

$p_2 := p_2 + cp_1 + d$,

$p_1 := p_1 + c$.

We repeat the process of multiplication till all the zeros of this particular type are exhausted. In the end we shall get a polynomial of degree 2m, where m is the number of pairs of such zeros. Then we print the 2m coefficients of the resulting polynomial, starting from the 2nd coefficient. We do not have to print the 1st coefficient because the polynomial is monic.

2.2 The integer zeros of a monic polynomial over Z

The integer zeros of a monic polynomial $P(x) \in \mathbf{Z}[x]$ must divide the last non-zero coefficient a_m of the polynomial. For large a_m, we may have to test a large number of different factors of the above coefficient in order to calculate all the integer zeros of the polynomial, if any. (**a**) The number of such trial factors can be considerably reduced by casting out all those factors, which when increased by unity do not divide $P(-1)$ and when decreased by unity do not divide $P(1)$. Even if f is a trial factor and $f+1$ and $f-1$ divide $P(-1)$ and $P(1)$ respectively, we can show that still it is possible that f may not be a zero of the polynomial $P(x)$. (**b**) A monic polynomial $P(x) \in \mathbf{Z}[x]$ cannot have any linear factor if 3 divides none of $P(1)$, $P(0)$ and $P(-1)$. (**a**) and (**b**)

immediately follow from the following very simple facts:-

If $P(x) \in \mathbf{Z}[x]$ has a linear factor of the form $x - h$ ($h \in \mathbf{Z}$),
then $P(x)$ can be expressed in the form
$P(x) \equiv (x-h)Q(x)$, where $Q(x) \in \mathbf{Z}[x]$.
Now $-1-h$, $-h$, $1-h$ are 3 consecutive numbers and at least one of them is divisible by 3.
Therefore $P(x) \equiv (x-h)Q(x) \Rightarrow h+1$, h and $h-1$ divide $P(-1)$, $P(0)$ and $P(1)$ respectively
$\Rightarrow$ 3 must divide at least one of $P(1)$, $P(0)$ and $P(-1)$.

Inclusion of these 2 tests considerably speeds up the calculation of zeros because each trial factor requires n multiplications and an equal number of additions or subtractions for determining whether or not the factor is a zero of the polynomial.

2.2.1 Algorithm

Given $P(x) = x^n + a_1x^{n-1} + \ldots + a_n$, a polynomial over $\mathbf{Z}$, we consider the last non-zero coefficient of the polynomial, if any. If there is no such coefficient, the polynomial becomes just x^n. We print all its zeros as 0. If the last non-zero coefficient is a_1, we print $n-1$ zeros as 0 and the nth zero as $-a_1$. In general if some $a_k \neq 0$, $k < n$, and all $a_p=0$, $k< p \leq n$, we conclude that $n - k$ of its zeros are 0. We compute the values of the polynomial $P(n)$ for $n = 1,-1$. In the process of computing $P(1)$, we check whether $P(1) = 0$. If it is so, we conclude that 1 is a zero of $P(x)$ and consequently we divide $P(x)$ by $x-1$. Similarly we check whether $P(-1) = 0$. If it is

so, we conclude that -1 is a zero of P(x) and consequently we divide P(x) by x+1 and so on. Let K(x) be the polynomial of degree $d > 1$ obtained in the process such that K(1), K(-1) and the constant term of K(x) are non-zero. We terminate the process if 3 divides none of the values K(1), K(0), K(-1). In all other cases we find all possible factors of the last non-zero coefficient k_0 by using a SUBROUTINE subprogram called FAC. Let a particular factor of k_0 be m. We divide K(1) by $m - 1$. If we find that K(1) is not divisible by $m - 1$, we reject m. If we find that K(1) is divisible by $m - 1$, we again divide K(-1) by $m + 1$. This time if we find that K(-1) is not divisible by $m + 1$, we now reject m. If both K(1) and K(-1) turn out to be divisible by $m - 1$ and $m + 1$ respectively, we compute the value of the polynomial K(x) for $x = m$. If K(m) turns out to be zero, then K(x) will have m as 1 of its zeros. We deflate the polynomial and calculate the new polynomial Q(x), where $K(x) = (x - m)\, Q(x)$. We compute the coefficients of Q(x) by synthetic division of K(x) by the linear factor $x - m$. After deflation of K(x) we check the degree of Q(x). If it happens to be unity, then the negative of the constant term of Q(x) is the last zero of the polynomial. If $K(m) \neq 0$, we repeat the process starting from $-m$ and so on. Then we check the other factors of the last non-zero coefficient of K(x). As soon as we get any factor which satisfies the polynomial, we immediately deflate the polynomial as before and continue further. We take particular precaution that if a factor does not satisfy the polynomial, we do not test that factor in the new polynomials which are obtained after deflation of the original polynomial. We only check the subsequent factors of the last non-zero coefficient of the original polynomial K(x) only once.

2.2.2 Implementation

The program for calculation of all integer zeros of a monic polynomial P(x) $\in$ **Z**[x] is completely self-contained and consists of 7 SUBROUTINEs namely TELIN, ZEROS, POLDIV, SQRUT, PRIME, FAC and SORT.

The routine called TELIN determines the existence of integer zeros other than 1,0,−1 and computes the zeros 1,0,−1 with their multiplicities (provided they exist). The routine called ZEROS calculates all the integer zeros other than 1,0,−1 (provided they exist). The SUBROUTINE subprogram called POLDIV deflates a monic polynomial $P(x) = x^n + p_1x^{n-1} + p_2x^{n-2} + \ldots + p_n$ by the method of dividing it by a linear polynomial x − a, provided a is a zero of P(x). The coefficients of the deflated polynomial are obtained by the simple relations

$$p_1 := p_1 + a,$$

$$p_i := p_i + ap_{i-1}, \text{ for } i = 2,3,\ldots,n-1.$$

The SUBROUTINE subprograms SQRUT and PRIME are used for the evaluation of the integer part of the square root of a natural number and for the computation of all the primes from 2 to x (where x is a natural number) respectively. Both the routines are given in the Appendix and their algorithms in the previous chapter. They are applied in the computation of all the factors of a natural number in the routine called FAC. The algorithm for computing all the factors of a natural number by trial and division method is given below.

We begin the process by testing whether the given natural number (say x) is greater than 1. If it is not, then we are through, since the only factor of

1 is 1. Otherwise we try to divide the number x by all the primes, less than or equal to $[\sqrt{x}\,]$, in succession. If x is divisible by a particular prime p, then we accept p as a prime factor of x and set x equal to x/p. We again divide the new x by p to find the largest natural number e such that $p^e \mid x$. We continue the process till all such primes have been tested as possible prime factors of x, or till x becomes equal to 1, whichever happens earlier. This process will always end in either of 2 ways; either x will become equal to 1 or x will become equal to a prime. Then we find all the factors of the given natural number. If in this process the number of all prime factors of x happens to be greater than 1, then we re-arrange all the factors of x in ascending order using the routine SORT. SORT is coded according to the 'Selection and Interchange' method.

2.3 H.C.F. of polynomials over Z

The multiplicity of zeros of a polynomial A(x), can be reduced to 1 simply by dividing the polynomial A(x) by the H.C.F. of A(x) and its differential coefficient. Also in the computation of the quadratic factors of a polynomial and in many other similar computations we need an algorithm for calculating the H.C.F. of polynomials over **Z**.

2.3.1 Algorithm for calculating the H.C.F. of 2 polynomials over Z

We start the process by comparing the degrees of the 2 polynomials. If both the polynomials are of degrees ≥ 1, we make them primitive by dividing all

their coefficients by their respective (numerical) H.C.F.'s. Now we divide the polynomial of the higher degree by the polynomial of the lower degree. If both the polynomials are of equal degrees, we simply take 1 of them as the first polynomial and divide it by the second polynomial. This process of division is not the ordinary division, as we normally understand it, but with some changes, which we shall presently explain. (We shall call it pseudo-division.) The remainder $R(x) = r_{k+1}x^k + ... + r_3x^2 + r_2x + r_1$ $(k < m)$ after this pseudo-division of the polynomial $U(x) = u_{n+1}x^n + ... + u_3x^2 + u_2x + u_1$ by the polynomial $V(x) = v_{m+1}x^m + ... + v_3x^2 + v_2x + v_1$ $(n \geq m)$ is computed using the algorithm given below.

Initially we set $r_t = u_t$ for t = 1,2,3,...,n+1.

Now for j = n−m + 1, we compute

$$r_i := \frac{v_{m+1}}{(r_{m+j}, v_{m+1})}\, r_i \text{ for i = m+j−1,m+j−2 ,...,3,2,1}$$

and $r_i := r_i - \frac{r_{m+j}}{(r_{m+j}, v_{m+1})}\, v_{i+1-j}$ for i = m+j−1, m+j−2,...,j.

Now if $r_i = 0\ \forall\ i \mid 1 \leq i \leq m+j-1$, then the H.C.F. of the 2 polynomials is V(x) itself.

And if $r_i = 0\ \forall\ i \mid 2 \leq i \leq m+j-1$, but $r_1 \neq 0$, then the required H.C.F. is 1.

In both these cases the algorithm ends.

Otherwise we find the degree (say d) of the remainder. Then we make it primitive by dividing all its coefficients by their (numerical) H.C.F. Now in the event $d < m$, the degree of the divisor, the divisor becomes the new dividend and the remainder becomes the new divisor. Then we proceed

from the beginning once again. If d $\geq$ m, we continue the above process of pseudo-division, taking j = n−m, n−m−1, ... till the degree of the remainder becomes < m.

In this algorithm we have taken particular precaution to stop the explosion of the coefficients which generally accompanies this algorithm.

The program for calculating the H.C.F. of 2 polynomials is given in the Appendix.

2.4 Algorithm for calculating the quadratic factors of a monic polynomial over Z

To obtain all the complex zeros of a real polynomial, many well-known methods, such as Bairstow's method, have adopted the technique of finding the real quadratic factors of the real polynomial (if any). This technique follows from the fact that the zeros of the quadratic polynomial $B(x) \equiv b_1x^2 + b_2x + b_3$ are also the zeros of the real polynomial $A(x) \equiv a_1x^n + a_2x^{n-1} + \ldots + a_{n+1}$ if and only if $b_1x^2 + b_2x + b_3$ divides A(x) without any remainder. The advantage of calculating the quadratic factors is that the complex zeros of a real polynomial can be found out without using complex arithmetic.

The algorithm which we have developed for finding the quadratic factors of a monic polynomial over **Z** is quite different from such known algorithms for this purpose. Let $A(x) \equiv x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots + a_n$, $(n \geq 3)$, $A(x) \in \mathbf{Z}[x]$, $B(x) \equiv x^2 - px + q$, where p is an indeterminate and q is a factor of the last non-zero coefficient, say a_k, in A(x), where k > 1. (We are tacitly assuming that such an a_k exists; if not, the given polynomial A(x)

will reduce to simply x^n or to $x^n + a_1x^{n-1}$, and in that case there cannot be any quadratic factor of A(x) other than possibly x^2.)

We can always express A(x) in the form

$A(x) \equiv Q(x)B(x) + C'(p)x^2 + D'(p)x + a_k$,

where

$Q(x) \equiv x^{n-2} + r_1x^{n-3} + r_2x^{n-4} + \ldots + r_{n-3}x$,

$C'(p) = p^{n-2} + c_1p^{n-3} + \ldots + c_{n-2}$,

$D'(p) = d_1p^{n-3} + d_2p^{n-4} + \ldots + d_{n-3}p + d_{n-2}$.

In particular for n=3, $D'(p) = d_1$.

(For example, if $A(x) \equiv x^5 - 10x^4 + 2x^3 - 7x^2 + 9x - 10$,

and $B(x) \equiv x^2 - px + 2$,

then

$Q(x) \equiv x^3 + (p-10)x^2 + p(p-10)x$,

$C'(p) = p^3 - 10p^2 - 2p + 13$,

$D'(p) = -2p^2 + 20p + 9$,

$a_5 = -10$.)

Now if $x^2 - zx + q \in \mathbf{Z}[x]$ is a factor of A(x),

then $A(x) = (x^2 - zx + q)\, Q_1(x)$, where $Q_1(x) = Q(x) + \frac{a_k}{q}$.

This implies that z is an integer zero of both the polynomials $C(p) \equiv C'(p) - \frac{a_k}{q}$ and $D(p) \equiv D'(p) + \frac{a_k}{q}p$,

and therefore also of their H.C.F. (C(p), D(p)), and the number of such quadratic factors of A(x) corresponding to the particular chosen constant term q = the number of integer zeros of the polynomial (C(p), D(p)).

2.4.1 Implementation of this algorithm

We first consider the last non-zero coefficient of the polynomial A(x), if any. If there is no such coefficient, the polynomial A(x) will reduce to x^n and in this case all its quadratic factors will be of the form x^2, provided of course that $n > 1$. In case the last non-zero coefficient is a_1, all its quadratic factors will be again of the form x^2, provided $n > 2$. In general, in an nth degree polynomial A(x), if the last non-zero coefficient is a_k, where $k < n - 1$, the number of quadratic factors of A(x) of the form x^2 will be $[\frac{n-k}{2}]$. In case the last non-zero coefficient is a_2, then apart from the quadratic factors of the form x^2, if any, there will be 1 quadratic factor of the form $x^2 + a_1x + a_2$. In case the last non-zero coefficient is a_1 and $n = 2$, then there will be 1 quadratic factor of the form $x^2 + a_1x$. To calculate all other quadratic factors of A(x), we find all possible factors of the last non-zero coefficient a_k. Fixing a particular factor q, we divide the original polynomial A(x) by the quadratic polynomial $x^2-px + q$, where for the time being, p is an indeterminate. In this process we obtain 2 polynomials C(p) and D(p) of degrees $n - 2$ and $n - 3$ respectively in the indeterminate p, provided that $n > 3$. However, if $n = 3$, both the polynomials C(p) and D(p) turn out to be linear. Then we compute the H.C.F. G(p) of the 2 polynomials C(p) and D(p). If D(p) does not exist, then G(p) = C(p). For example, if we consider the polynomial $x^4+4x^3-4x^2-8x+4$, we find that C(p) exists but there is no D(p), provided we choose $q = -2$.

If G(p) = 1, then there does not exist any quadratic factor of the original polynomial A(x), whose constant term is that particular factor q which we

had fixed earlier. If the degree of G(p) is 1, then the value of the indeterminate p is the negative of the constant term in G(p). Otherwise the number of quadratic factors of the original polynomial, whose constant term is the factor q, fixed earlier, is equal to the number of integer zeros of G(p). If G(p) does not happen to have any integer zeros at all, it means that again there does not exist any quadratic factor of A(x) with constant term q. And the value of the indeterminate p in the case of each quadratic factor will be the value of the corresponding integer zero of G(p). As soon as a quadratic factor of the original polynomial has been found, we immediately deflate the polynomial by dividing it by the quadratic factor just calculated. Then we check the degree of the new polynomial obtained by deflation of the original polynomial. If the new polynomial happens to be of degree less than or equal to 1, it means that we have already found out all the quadratic factors of the original polynomial. If the new polynomial is of degree 2, then the last quadratic factor is the new polynomial itself. If its degree is greater than 2, then we have to search for other quadratic factors, if any. For this purpose we change the sign of q and repeat the process. We continue the same process taking q to be the next factor of the new constant term, and so on, until either all the quadratic factors have been obtained, or it has been proved that no further quadratic factors with integer coefficients exist.

2.4.2 The algorithm for finding C(p) and D(p)

Let $A(x) \equiv x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots + a_n$, $(n \geq 3)$, $A(x) \in \mathbf{Z}[x]$,

$B(x) \equiv x^2 - px + q$, where p is an indeterminate and q is a factor of a_n,

$a_n \neq 0$.

We set $C(p) = p + a_1$; $D(p) = a_2 - q$.

B1. For $3 \leq k \leq n-1$

$C_1(p) = pC(p) + D(p)$,

$D(p) = -qC(p) + a_k$,

$C(p) = C_1(p)$.

B2. The required C(p) and D(p) are given by $C(p) := C(p) - \frac{a_n}{q}$;

$D(p) := D(p) + \frac{a_n}{q}p$.

2.4.3 The coefficients of the deflated polynomial after the division of the nth degree polynomial by the quadratic factor $x^2 - px + q$ just calculated

We calculate the above coefficients by the following algorithm:-

B1. $a_1 := a_1 + p$,

B2. $a_2 := a_2 + pa_1 - q$,

B3. $a_k := a_k + pa_{k-1} - qa_{k-2}$, for $k = 3(1)n-2$.

In case $n = 3$, we omit steps (**B2**) and (**B3**) and in case $n = 4$, we omit step (**B3**).

2.5 Algorithm for reducing a non-monic polynomial with integer coefficients to a monic polynomial with integer coefficients (with the objective of calculating its zeros)

Let $A(x) = a_1x^n + a_2x^{n-1} + a_3x^{n-2} + \ldots + a_nx + a_{n+1}$ be a polynomial over $\mathbf{Z}$ and let the zeros of this polynomial be $x_1, x_2, x_3, \ldots, x_n$. If we multiply the zeros by a natural number k and form another polynomial B(x) whose zeros are $kx_1, kx_2, kx_3, \ldots, kx_n$, then we can easily prove that with a suitable choice of k, the other polynomial B(x) will be monic, still having all integer coefficients.

If in the original polynomial A(x), $a_i \in \mathbf{Z}\ \forall i | 1 \leq i \leq n+1$, $a_1 \neq 0$, then it is easy to see that if we multiply all its zeros by $|a_1|$, then the polynomial B(x) having its zeros as $|a_1|x_1$, $|a_1|x_2$, $|a_1|x_3, \ldots, |a_1|x_n$ will certainly be monic and it will have all its coefficients in $\mathbf{Z}$. Notwithstanding this fact, in most cases we can find a natural number k, much smaller than $|a_1|$, which will do the job equally well. Consequently the phenomenon of explosion of coefficients of the polynomial can be avoided. In fact, if all zeros of a particular polynomial happen to be rational and if we express them in their lowest terms, this number k will be just the L.C.M. of the denominators of all the zeros. But in the general case we cannot make this statement. Still the existence of such a k is beyond doubt.

For example, we consider the polynomial

$$-1296x^4 - 1080x^3 + 252x^2 - 420x + 1297.$$

In this case $|a_1| = |-1296| = 1296$. But instead of multiplying all its zeros by 1296, if we multiply them by 6, we get the polynomial

$$-1296x^4 - 6480x^3 + 9072x^2 - 90720x + 1680912,$$

or, equivalently,

$$x^4 + 5x^3 - 7x^2 + 70x - 1297,$$

which is monic and has all its coefficients in **Z**. So now we develop an algorithm for calculating the least natural number k which will do the trick. (At worst, k can be $|a_1|$.) Let the given polynomial over **Z** be primitive and let it be

$$a_1x^n + a_2x^{n-1} + a_3x^{n-2} + ... + a_nx + a_{n+1}, \tag{1}$$

where $a_1 > 0$.

If $a_1 = 1$, k will be 1 and we are through.

Otherwise let $a_1 = \prod_{i=1}^{r} p_i^{e_i}$, where $r \in \mathbf{N}$, p_i is a prime and $e_i \in \mathbf{N}$ $\forall\, i \mid 1 \leq i \leq r$.

Naturally k will also be of the form

$$k = \prod_{i=1}^{r} p_i^{f_i},$$

where $1 \leq f_i \leq e_i$ $\forall i | 1 \leq i \leq r$.

To fix the ideas, we assume that $r = 1$, so that $a_1 = p^e$, $k = p^f$, $1 \leq f \leq e$, p being a prime. Then, after putting $kx = y$, the given polynomial becomes

$$a_1\left(\frac{y}{k}\right)^n + a_2\left(\frac{y}{k}\right)^{n-1} + a_3\left(\frac{y}{k}\right)^{n-2} + ... + a_n\frac{y}{k} + a_{n+1},$$

which has the same zeros as the polynomial

$$y^n + \frac{a_2k}{a_1}y^{n-1} + \frac{a_3k^2}{a_1}y^{n-2} + ... + \frac{a_nk^{n-1}}{a_1}y + \frac{a_{n+1}k^n}{a_1}. \tag{2}$$

Now assume that $p^{m_1} \parallel a_2, p^{m_2} \parallel a_3, \ldots, p^{m_n} \parallel a_{n+1}$, where $m_i \geq 0$, i=1,2, 3,...,n.

(By $p^t \parallel N$ we mean that $p^t \mid N$ but $p^{t+1} \nmid N$, i.e., $3^4 \parallel 405$ as $3^4 \mid 405$ but $3^5 \nmid 405$.) If all the coefficients of the polynomial (2) are in **Z**, then we must have

$m_1 + f \geq e,\ m_2 + 2f \geq e,\ m_3 + 3f \geq e, \ldots, m_n + nf \geq e$

$\Rightarrow f \geq e - m_1,\ 2f \geq e - m_2,\ 3f \geq e - m_3, \ldots, nf \geq e - m_n$

$\Rightarrow f \geq e - m_1,\ f \geq \frac{e-m_2}{2},\ f \geq \frac{e-m_3}{3}, \ldots, f \geq \frac{e-m_n}{n}$

$\Rightarrow f \geq -\left[\frac{m_i-e}{i}\right]\ \forall\ i,\ i=1,2,3,\ldots,n$

$\Rightarrow f \geq \max_{1 \leq i \leq n} \left\{-\left[\frac{m_i-e}{i}\right]\right\}.$

Thus the least possible value of f is $\max_{1 \leq i \leq n} \left\{-\left[\frac{m_i-e}{i}\right]\right\}$.

If any $m_i \geq e$, we can omit it while taking the maximum, because $f > 0$ is evident. Similarly, if any m_i is such that $e > m_i \geq e - i$, we can omit it also, because in that case we shall get $-\left[\frac{m_i-e}{i}\right] = 1$ and $f \geq 1$ is also otherwise evident. So we should consider only those m_i, if any, which are less than e−i. If there is no such m_i, we can take $f = 1$, $k = p$, otherwise we should take

$$f = \max_{\substack{1 \leq i \leq n \\ m_i < e-i}} \left\{-\left[\frac{m_i - e}{i}\right]\right\}, \quad k = p^f.$$

(As the polynomial is primitive, so the H.C.F. of $a_1, a_2, a_3, \ldots, a_{n+1}$ is unity. Therefore at least 1 m_i will be zero.)

We can repeat this process for every prime p_i and then we can obtain $f_i \in \mathbf{N}\ \forall i | 1 \leq i \leq r$. Then we can write

$$k = \prod_{i=1}^{r} p_i^{f_i}$$

The computation of k can be implemented easily by a nest of 2 DO loops, 1 for r and 1 for n. While we are in the nest of DO loops, we shall ignore all coefficients, if any, which happen to be zero.

The program for finding k is given in the Appendix.

2.6 Polynomial with rational coefficients

Now consider a monic polynomial which is given by

$$P(x) = x^n + \frac{b_1}{a_1}x^{n-1} + \frac{b_2}{a_2}x^{n-2} + \frac{b_3}{a_3}x^{n-3} + ... + \frac{b_n}{a_n},$$

where $b_i, a_i \in \mathbf{Z}$, $a_i \neq 0 \ \forall i | \ 1 \leq i \leq n$. Then P(x) can be transformed into a polynomial with integer coefficients for the purpose of computing its zeros and we can compute k as before. For this purpose, we first reduce each fractional coefficient in P(x) to its lowest terms and then we calculate the L.C.M. of all the denominators. Thereafter we multiply the polynomial P(x) by this L.C.M. and we get a polynomial over **Z**.

2.7 Polynomial with Gaussian coefficients

Given a polynomial $P(x) = x^n + c_1x^{n-1} + c_2x^{n-2} + ... + c_{n-1}x + c_n$, where $c_k = a_k + ib_k$, $a_k, b_k \in \mathbf{Z}$. Then PP^* is a polynomial over **Z** of degree 2n, where P^* is the conjugate of P. And if $p_k + iq_k$ is a zero of the polynomial P(x), then $p_k \pm iq_k$ are zeros of the polynomial PP^*. Then we can compute the zeros $p_k \pm iq_k$ of this integer polynomial by the algorithm of this chapter.

CHAPTER 3

EIGENVALUES AND EIGENVECTORS OF A MATRIX

In the numerical solution of an eigenvalue problem for a square matrix of order n, it is required to find a set of scalars λ_i (i = 1,2,3,...,n), called the eigenvalues, and a set of non-zero vectors x_i (i = 1,2,3,...,k, $1 \leq k \leq n$), called the eigenvectors, such that $Ax_i = \lambda_i x_i$. For symmetric matrices the Jacobi method is found to be highly satisfactory in practice. This method is known to be numerically stable. But for non-symmetric matrices there does not exist any such stable method. To compute the exact eigenvalues of any square matrix, irrespective of any special form or property which this matrix may happen to have, we have used the method of first computing its characteristic polynomial and then calculating the zeros of this polynomial using an algorithm which we have developed for this purpose. This technique of first computing the characteristic polynomial and then obtaining the zeros (accurately) of this polynomial by some method has also been suggested by Krishnamurthy and Sen [37] for the computation of the exact eigenvalues of a matrix. First computing the Hessenberg form of a square matrix and then calculating its characteristic polynomial by a recursive algorithm is considered to be 1 of the best methods (an n^3 process) for this purpose. But if the calculations are carried out in **Z** or **Q**, the phenomenon of coefficient explosion neutralises the advantage of an n^3 method over an n^4

(Leverrier-Faddeev) method. Rao[48] has described an algorithm for error-free computations of the characteristic polynomial using residue arithmetic. However, the process of converting data back and forth between usual positional representation and residue representation can be a bottleneck to the efficiency [1]. Thus we have implemented the Leverrier-Faddeev method[11] for the computation of the characteristic polynomial. We begin this chapter with a discussion about the possible improvement of the fundamental algorithm for computing the characteristic polynomial.

3.1 Characteristic Polynomial

1 of the methods of computing the characteristic polynomial of a square matrix is the method of direct expansion. In this method the determinant det (A $-$ λI) is expanded as an nth degree polynomial

$P(\lambda) \equiv \lambda^n + p_1\lambda^{n-1} + p_2\lambda^{n-2} + \ldots + p_n$, which is called the characteristic polynomial of the matrix A and its coefficients p_1,p_2,p_3,...,p_n are the sums of all the principal minors of orders 1, 2, 3,...,n, multiplied by $-1, (-1)^2, (-1)^3, \ldots, (-1)^n$ respectively.

Thus for a given square matrix A = (a_{ij}) of order 4,
$p_1 = -(a_{11} + a_{22} + a_{33} + a_{44})$ (i.e., Tr A, which is the sum of all the diagonal entries of the matrix A),

$$p_2 = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{14} \\ a_{41} & a_{44} \end{vmatrix} + \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{22} & a_{24} \\ a_{42} & a_{44} \end{vmatrix} + \begin{vmatrix} a_{33} & a_{34} \\ a_{43} & a_{44} \end{vmatrix},$$

which is the sum of all the principal minors of order 2 of the matrix A,

$$p_3 = -\left(\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{12} & a_{14} \\ a_{21} & a_{22} & a_{24} \\ a_{41} & a_{42} & a_{44} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{13} & a_{14} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{vmatrix} + \begin{vmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{vmatrix}\right),$$

which is the sum of all the principal minors of order 3 of the matrix A,

$$p_4 = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix},$$

which is the determinant of the matrix A.

In general the number of all the principal minors of order k of the matrix A is equal to ${}^nC_k$. We can evaluate these minors by using any 1 of the algorithms of (i) ordinary elimination (ii) Gaussian elimination (iii) Gauss-Bareiss elimination, and so on. But the algorithm of the evaluation of determinants by the Gaussian elimination method can be suitably modified so as to reduce the explicit evaluations of the minors. This reduction is based on the following observations:-

Let us denote the minors as follows:-

$$d_{12} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}, d_{13} = \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix}, d_{14} = \begin{vmatrix} a_{11} & a_{14} \\ a_{41} & a_{44} \end{vmatrix},$$

$$d_{23} = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix}, d_{24} = \begin{vmatrix} a_{22} & a_{24} \\ a_{42} & a_{44} \end{vmatrix}, d_{34} = \begin{vmatrix} a_{33} & a_{34} \\ a_{43} & a_{44} \end{vmatrix},$$

$$d_{123} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}, d_{124} = \begin{vmatrix} a_{11} & a_{12} & a_{14} \\ a_{21} & a_{22} & a_{24} \\ a_{41} & a_{42} & a_{44} \end{vmatrix},$$

$$d_{134} = \begin{vmatrix} a_{11} & a_{13} & a_{14} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{vmatrix}, d_{234} = \begin{vmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{vmatrix},$$

and

$$d_{1234} = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix}.$$

To find p_1,p_2,p_3 and p_4, we have to evaluate all the minors of different orders. i.e., in all, we have to evaluate $15(=2^4 - 1)$ minors.

Let us evaluate the determinant d_{1234} by Gaussian elimination method.

We suppose that $a_{11} \neq 0$ and subtract $\frac{a_{j1}}{a_{11}}$ times the 1st row from the jth row, j varying from 2 to 4. We get

$$d_{1234} = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} - \frac{a_{21}}{a_{11}}a_{12} & a_{23} - \frac{a_{21}}{a_{11}}a_{13} & a_{24} - \frac{a_{21}}{a_{11}}a_{14} \\ 0 & a_{32} - \frac{a_{31}}{a_{11}}a_{12} & a_{33} - \frac{a_{31}}{a_{11}}a_{13} & a_{34} - \frac{a_{31}}{a_{11}}a_{14} \\ 0 & a_{42} - \frac{a_{41}}{a_{11}}a_{12} & a_{43} - \frac{a_{41}}{a_{11}}a_{13} & a_{44} - \frac{a_{41}}{a_{11}}a_{14} \end{vmatrix}$$

We designate the entries represented by $a_{jk} - \frac{a_{j1}}{a_{11}}a_{1k}$ in the above determinat again by a_{jk}.

Then $d_{12} = a_{11}a_{22}$.

Next we assume that $a_{22} \neq 0$ and repeat this very process for the entries which now appear in the last 2 rows and the last 2 columns of the above determinant. It means that we must now compute $a_{jk} - \frac{a_{j2}}{a_{22}}a_{2k}$ and designate the result again by a_{jk}, where this time j and k will vary independently from 3 to 4. The determinant now reduces to

$$d_{1234} = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{vmatrix}$$

Correspondingly $d_{123} = a_{11}a_{22}a_{33}$.

Next, assuming that $a_{33} \neq 0$, we replace a_{44} by $a_{44} - \frac{a_{43}}{a_{33}}a_{34}$.

Thus

$$d_{1234} = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{vmatrix}$$

Now $d_{1234} = a_{11}a_{22}a_{33}a_{44}$.

But in the process if a_{11} is 0, $a_{21} \neq 0$, then we interchange the row 1 and the row 2 in which there is a non-zero entry in the column 1. Then we change the signs of all the entries in the row 2. Afterwards we perform the first step of elimination for those rows whose entry in the 1st column is non-zero.

Then d_{12} is the same as before.

However if a_{11} and a_{21} are 0, $a_{31} \neq 0$, then we get $d_{12} = 0$

In general, in order to proceed with the process of evaluation we interchange the row 1 and the 1st row in which there is a non-zero entry in the column 1, change the signs of all the entries in that row, and repeat the first step of elimination for those rows whose entry in the 1st column is non-zero. Similarly if, after the 1st step of elimination, we find that $a_{22}=0$, we search for a non-zero entry in the 2nd column in the 3rd and 4th rows; if there exists such a non-zero entry, then we interchange the corresponding rows, change the signs of all entries in 1 of the said rows, and continue the process

of elimination. If the non-zero entry in the 2nd column appears in the 4th row, then we have $d_{123} = 0$. If there does not exist any such entry, then the values of all such determinants are zero.

Thus together with the evaluation of d_{1234}, the values of the determinants d_{12} and d_{123} get determined and as such d_{12} and d_{123} are not required to be evaluated again.

Now we can generalise this process for a determinant of order n. We must now replace a_{jk} by $a_{jk} - \frac{a_{ji}}{a_{ii}} a_{ik}$ in the ith stage of elimination, where i will vary from 1 to n − 1 and keeping i fixed, j and k will independently vary from i+1 to n. At every stage of elimination we keep track of the largest row number j, if $a_{ii} = 0$, to determine the value of the subminor $d_{123...i+1}$.

In general, in the process of evaluation the subminor

$d_{123...r} = a_{11}\ a_{12}.....a_{rr}$,

or = 0, if at least one a_{ii}=0, $1 \leq i \leq r - 1$ and also a_{ji}=0 $\forall j | i < j \leq r$.

So together with the computation of $d_{123...n}$ by the triangulation method, the values of $d_{123...k}$, k=2,3,...(n−1) are also known, i.e., if the method of Gaussian elimination is applied in finding $d_{123...n}$, then at the cost of $n^3/3$ operations, the values of other (n−2) determinants of orders 2,3,...(n−1) can be calculated.

If we apply the same technique in evaluating the determinants d_{124}, d_{134}, d_{234}, d_{14}, d_{24}, d_{34}, we can avoid evaluation of d_{13} and d_{23}.

Thus in the calculation of the characteristic polynomial of a square matrix of order n, we shall evaluate only those minors corresponding to the combinations whose last entry is n. Correspondingly, instead of evaluating $2^n - 1$ determinants (required for finding the characteristic polynomial) we

have to evaluate only $2^{n-1}-1$ determinants.

3.1.1 Leverrier-Faddeev method of computing the characteristic polynomial

The method of Leverrier is based on Newton's formulas for the sums of the powers of the roots of an algebraic equation and consists of the following.

Let

$$P(\lambda) \equiv \lambda^n + p_1\lambda^{n-1} + p_2\lambda^{n-2} + ... + p_n, \tag{1}$$

be the characteristic polynomial of the square matrix A = (a_{ij}) of order n and let $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$ be the complete set of roots of the polynomial (1).

Let us consider the sum

$S_k = \lambda_1^k + \lambda_2^k + \lambda_3^k + ... + \lambda_n^k$ (=tr A^k) $\forall k | 1 \leq k \leq n$. Then $p_1, p_2, p_3, ..., p_n$ are computed using the Newton's formulas.

$$p_k = -\frac{1}{n}(S_k + p_1S_{k-1} + p_2S_{k-2} + ... + p_{k-1}S_1) \; \forall k | 1 \leq k \leq n. \tag{2}$$

Faddeev's simplification of Leverrier's method consists in calculating the sequence of the matrices A_1, A_2, A_3, ... , A_n, as follows:-

$A_1 = A$, Tr $A_1 = q_1$, $B_1 = A_1 - q_1I$,

$A_k = AB_{k-1}$, $\frac{TrA_k}{k} = q_k$, $B_k = A_k - q_kI$ for k = 2,3,...,n.

and $q_i = -p_i$ for i = 1, 2, 3,...,n.

We give the method as an algorithm:-

Let A = (a_{ij}), B = (b_{ij}) be two square matrices of order n.

3.1.2 Algorithm

C1. Initially we set k = 1, B = A

C2. $p_k = -\frac{\sum_{i=1}^{n} b_{ii}}{k}$

C3. $b_{ii} := b_{ii} + p_k \ \forall i | 1 \leq i \leq n$

C4. k := k + 1

C5. If k $\leq$ n do the following:-

B := AB

go to step (**C2**);

else stop.

In step **C2**, the division will be exact if the matrix under consideration consists of all integer entries.

3.1.3 Matrices with Gaussian entries

Let C=A+iB be a matrix with Gaussian entries, then the n×n eigenvalue problem (A+iB)(x+iy)=λ(x+iy)
is equivalent to

$$\begin{pmatrix} A & -B \\ B & A \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$$

Here

$$M = \begin{pmatrix} A & -B \\ B & A \end{pmatrix}$$

is a 2n×2n integer matrix. And if p_k+iq_k is an eigenvalue of the matrix C, then $p_k \pm q_k$ are the eigenvalues of the matrix M.

We can compute the characteristic polynomials of Gaussian matrices using the algorithm 3.1.2 which can be implemented using integer arithmetic only.

3.2 Eigenvalues and eigenvectors

The eigenvalues of a square matrix are the zeros of its characteristic polynomial obtained by the above process. The zeros are calculated using the algorithms/subroutines which we have described in Chapters 1 and 2. Thus we can calculate the integer, rational, irrational and complex eigenvalues of the particular form indicated on page 45, of matrices with integer, rational, Gaussian entries. The eigenvector corresponding to the eigenvalue λ_i is computed by solving the system of equations Ax=λ_ix by the Gaussian elimination method which is a $\frac{1}{3}n^3$ process.

3.3 Jordan canonical form

It is not true that every square matrix is similar to a diagonal matrix, but every square matrix is necessarily similar to a matrix in the Jordan canonical form; i.e., given a square matrix M of order n, having distinct eigenvalues $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_k$, with algebraic multiplicities a(λ_1),a(λ_2),a(λ_3),...,a(λ_k) and

geometric multiplicities $g(\lambda_1)$, $g(\lambda_2)$,$g(\lambda_3)$,...,$g(\lambda_k)$, there exists a unique matrix $J = \mathrm{diag}(J_r(\lambda_i))$, where

$$J_r(\lambda_i) = \begin{pmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & \lambda_i \end{pmatrix},$$

there being r rows in the matrix

Thus

$$J = \begin{pmatrix} J_{r_1(1)}(\lambda_1) & 0 & 0 & \cdots & \cdots & 0 \\ 0 & J_{r_2(1)}(\lambda_1) & 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & J_{r_{g(\lambda_1)}(1)}(\lambda_1) & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & J_{r_1(k)}(\lambda_k) & 0 & 0 \\ 0 & 0 & 0 & 0 & J_{r_2(k)}(\lambda_k) & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & J_{r_{g(\lambda_k)}(k)}(\lambda_k) \end{pmatrix},$$

where $r_1(1)$, $r_2(1)$, $r_3(1)$, ... ,$r_{g(\lambda_1)}(1)$ are the invariants corresponding to the eigenvalue λ_1, such that

$$r_1(1) \geq r_2(1) \geq r_3(1) \geq \ldots \geq r_{g(\lambda_1)}(1) \qquad (1)$$

and

$$r_1(1) + r_2(1) + r_3(1) + \ldots + r_{g(\lambda_1)}(1) = a(\lambda_1), \qquad (2)$$

the algebraic multiplicity of the eigenvalue λ_1; and similar results hold for every distinct eigenvalue λ_i.

(1) and (2) $\Rightarrow$ the orders of the Jordan blocks corresponding to the eigenvalue λ_i are the invariants corresponding to one of the partitions of $a(\lambda_i)$ into $g(\lambda_i)$ parts; e.g., if $a(\lambda_1)=8$ and $g(\lambda_1)=3$, then the orders of the Jordan blocks corresponding to λ_1 can be 6,1,1 or 5,2,1 or 4,3,1 or 4,2,2 or 3,3,2 because $8=6+1+1 = 5+2+1 = 4+3+1 = 4+2+2 = 3+3+2$ and $\exists$ no other partitions of 8 into 3 parts. We can calculate the Jordan canonical matrix J, corresponding to a given matrix A, by the following methods:-

3.3.1 (i) Trial and error method

We first consider the eigenvalue λ_1. In the example quoted above, we set

$$J_{(\lambda_1)} = \begin{pmatrix} J_6(\lambda_1) & 0 & 0 \\ 0 & J_1(\lambda_1) & 0 \\ 0 & 0 & J_1(\lambda_1) \end{pmatrix} \text{ or } J_{(\lambda_1)} = \begin{pmatrix} J_5(\lambda_1) & 0 & 0 \\ 0 & J_2(\lambda_1) & 0 \\ 0 & 0 & J_1(\lambda_1) \end{pmatrix}$$

$$\text{or } J_{(\lambda_1)} = \begin{pmatrix} J_4(\lambda_1) & 0 & 0 \\ 0 & J_3(\lambda_1) & 0 \\ 0 & 0 & J_1(\lambda_1) \end{pmatrix} \text{ or } J_{(\lambda_1)} = \begin{pmatrix} J_4(\lambda_1) & 0 & 0 \\ 0 & J_2(\lambda_1) & 0 \\ 0 & 0 & J_2(\lambda_1) \end{pmatrix}$$

or

$$J_{(\lambda_1)} = \begin{pmatrix} J_3(\lambda_1) & 0 & 0 \\ 0 & J_3(\lambda_1) & 0 \\ 0 & 0 & J_2(\lambda_1) \end{pmatrix},$$

where, e.g.,

$$J_4(\lambda_1) = \begin{pmatrix} \lambda_1 & 1 & 0 & 0 \\ 0 & \lambda_1 & 1 & 0 \\ 0 & 0 & \lambda_1 & 1 \\ 0 & 0 & 0 & \lambda_1 \end{pmatrix}.$$

We make the same construction for the eigenvalues $\lambda_2, \lambda_3, ..., \lambda_k$ and then we set up a Jordan matrix

$$J = \begin{pmatrix} J_{(\lambda_1)} & 0 & 0 & 0 & 0 \\ 0 & J_{(\lambda_2)} & 0 & 0 & 0 \\ 0 & 0 & J_{(\lambda_3)} & 0 & 0 \\ 0 & 0 & 0 & J_{(\lambda_4)} & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & J_{(\lambda_k)} \end{pmatrix}.$$

Then we solve the set of equation AP=PJ for the entries of the matrix P. If we can find some solution which makes P to be non-singular, then we are through. Otherwise we try all possible combinations by taking different forms for $J_{(\lambda_1)}$, $J_{(\lambda_2)}$, $J_{(\lambda_3)}$,..., $J_{(\lambda_k)}$. In the example just quoted $J_{(\lambda_1)}$ can have 5 forms. Similarly $J_{(\lambda_2)}$, $J_{(\lambda_3)}$,..., $J_{(\lambda_k)}$ can have many forms. In a particular example if the total number of forms corresponding to the different eigenvalues $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_k$ are $\mu_1, \mu_2, \mu_3, ..., \mu_k$ respectively (in the above case μ_1=5), then we shall have to examine all the $\mu_1\mu_2\mu_3...\mu_k$ forms at the most. Out of all these forms exactly one form will turn out, which will make J similar to A by means of the similarity transformation AP=PJ. This particular J will be the Jordan canonical matrix similar to A. For every form, we shall have to solve n^2 equations in n^2 unknowns and then we shall have to check, firstly, whether we obtain a non-trivial solution at all, and secondly, whether the non-trivial solution can make P to be non-singular. Then we shall have to repeat the calculations $\mu_1\mu_2\mu_3...\mu_k$ times. Therefore this process is very laborious and cumbersome.

3.3.2 (ii) Alternative method

We solve the system $(A-\lambda_1 I)x = 0$, where x is a column vector having n components corresponding to λ_1. We shall get $g(\lambda_1)$ l.i. solutions, which are the eigenvectors corresponding to the eigenvalue λ_1. Let us denote these solutions by $x_{1_{(1,1)}}$, $x_{1_{(2,1)}}$, $x_{1_{(3,1)}}$,...,$x_{1_{(g(\lambda_1),1)}}$. Now we shall find the principal vectors corresponding to the first eigenvector $x_{1_{(1,1)}}$. We shall denote them by $x_{1_{(1,2)}}$, $x_{1_{(1,3)}}$,...,$x_{1_{(1,p)}}$, where p denotes the value of the first invariant corresponding to the eigenvalue λ_1. For this purpose we solve

$(A-\lambda_1 I)x_{1_{(1,2)}} = x_{1_{(1,1)}}$,

$(A-\lambda_1 I)x_{1_{(1,3)}} = x_{1_{(1,2)}}$, etc.

At the first stage we shall find that the vector $x_{1_{(1,2)}}$ will not turn out to be unique. The equations set up will be non-homogeneous and so the solution will be of the form

$$y+c_1x_{1_{(1,1)}}+c_2x_{1_{(2,1)}}+c_3x_{1_{(3,1)}}+...+c_{g(\lambda_1)}x_{1_{(g(\lambda_1),1)}}$$

where y is a fixed vector, $c_1,c_2,c_3,...,c_{g(\lambda_1)}$ are arbitrary constants and $x_{1_{(1,1)}}$, $x_{1_{(2,1)}}$, $x_{1_{(3,1)}}$,...,$x_{1_{(g(\lambda_1),1)}}$ are the original eigenvectors. We can give any particular values to these arbitrary constants and then the solution obtained will be called as $x_{1_{(1,2)}}$. We continue the solution in the same way and calculate $x_{1_{(1,3)}}$, $x_{1_{(1,4)}}$ etc. At a certain stage we shall find that it will be possible to calculate $x_{1_{(1,p)}}$ but the calculation of $x_{1_{(1,p+1)}}$ will not be possible, because the equations will become inconsistent. This will mean that the first invariant is p.

Now we take up the equations

$(A-\lambda_1 I)x_{1_{(2,2)}} = x_{1_{(2,1)}}$,

$(A-\lambda_1 I)x_{1_{(2,3)}} = x_{1_{(2,2)}}$, etc.

and so on. In this way we shall get all the invariants corresponding to one particular eigenvalue λ_1. What we have done for λ_1, we can repeat for λ_2, λ_3,...,λ_k in the same way. This will give us the complete sets of invariants for each distinct eigenvalue. However it is not necessary that for a particular eigenvalue, the invariants will be always in non-decreasing order and we rename the principal vectors appropriately. We carry out the same construction for the eigenvalue λ_i $(1 \leq i \leq k)$ separately.

For example, if $a(\lambda_1) = 8$ and $g(\lambda_1) = 3$, then we shall get 3 l.i. solutions of $(A-\lambda_1 I)x = 0$. Let the solutions be $x_{1_{(1,1)}}$, $x_{1_{(2,1)}}$, and $x_{1_{(3,1)}}$. Then corresponding to $x_{1_{(1,1)}}$, $x_{1_{(2,1)}}$, and $x_{1_{(3,1)}}$, we shall get 6,1,1 or 5,2,1 or 4,3,1 or 4,2,2 or 3,3,2 principal vectors (after renaming the starting eigenvectors appropriately).

The eigenvectors and the principal vectors can be computed by implementing the Gauss method for the solution of a system of n linear equations in n unknowns. Then we shall arrange all the principal vectors corresponding to λ_1 as columns of a matrix from left to right, and do the same thing for λ_2, λ_3,...,λ_k also. This will give us a square matrix P which we can call as the modal matrix. This matrix will satisfy the equation AP=PJ, where J is the Jordan matrix, to which we have referred above. Also the matrix P so obtained will necessarily be non-singular, which means that A is similar to J. Since all computations are performed using integer arithmetic, therefore no such modifications as proposed in [35,36] are required.

We may not be able to find all the principal vectors, even if they exist, using the method (ii), if we choose the starting eigenvector arbitrarily.

For example, let us consider the matrix

$$A = \begin{pmatrix} -7 & -21 & -15 \\ 6 & 16 & 10 \\ -3 & -7 & -3 \end{pmatrix}.$$

The only eigenvalue of this matrix is 3, with algebraic multiplicity 3 and geometric multiplicity 2. Starting with the eigenvectors (−7,3,0) and (2,−5,0), we could not find another principal vector. But if we start the process with the eigenvector (3,−2,1), then we succeed in calculating another principal vector using this method. For the first 2 eigenvectors the system of equations $Ax = \lambda x$ becomes trivially inconsistent. While testing the algorithm we have found that we can compute the desired principal vectors in all cases except the trivially inconsistent cases, such as the matrix A with eigenvectors (−7,3,0) and (2,−5,0). But we could not develop a technique which can take care of this problem, i.e., there exist some cases when we know that there exists some set of principal vectors, but it is not possible to find those vectors because of the improper choice of the starting eigenvector.

CHAPTER 4

CONSTRUCTION OF TEST MATRICES

Given a program implementing a particular algorithm, often we have to test the program for probable defects. This testing is a creative challenge to all those programmers who are engaged in the task of computer programming. Algorithms for the numerical solution of eigenvalue problems appear frequently in mathematical literature connected with numerical analysis, e.g.,[5,43,52,53,56]. We have to test these algorithms by providing numerical examples. These numerical examples serve the purpose of assessment of the performance of these programs by the method of statistical testing (randomly generated matrices with known spectra give a fair idea of the performance of these programs under different conditions). The methods of computation of eigenvalues differ considerably according to different special forms and different kinds of matrices. For example, we cannot diagonalise every matrix using a similarity transformation, but every symmetric matrix is diagonalisable; and so the eigenvalues of a symmetric matrix can always be computed by diagonalising the matrix with the help of a similarity transformation. Thus the routine for the computation of the eigenvalues of a symmetric matrix cannot be used for the computation of the eigenvalues of a general matrix. It has been proved [58] that for a non-derogatory matrix A, there always exists a symmetric matrix X such that B = XA is symmetric; and the eigenvalues of A can be computed by solving the generalised

eigenvalue problem B = λX. To provide numerical data to all such routines, we have generated matrices having different properties, such as derogatory, diagonalisable, non-diagonalisable and so on.

It is a general practice [16,33,34,56] to consider the test matrices over the base ring **Z** in order to avoid unnecessary propagation of round-off errors. Matrices over **Z** can be generated exactly, which is imperative to test the accuracy of the eigenvalue routines. Apart from the fact that non-singular matrices with integer entries have wide applications in integer programming [49], the process of accurately generating matrices over the principal ideal domain **Z** is comparatively faster and requires less storage than constructing these matrices over the field **Q**. Thus we have given much emphasis on generating matrices with integer entries. Then we have extended these results in order to generate matrices over the fields **Q**, **R**, **C** in non-trivial cases with special forms, such as symmetric, positive, positive- symmetric, etc.

The method of constructing matrices with assigned spectra consists of carrying out similarity transformations of the form PDP^{-1}. It is well-known that similar matrices have the same eigenvalues and PDP^{-1} is similar to D. The straightforward implementation of the similarity transformation PDP^{-1} requires $2n^3$ multiplications with known P and known P^{-1}. For avoiding the more costly operations of multiplication and for minimising round-off errors in case of the imprecise fields **R** and **C**, we have tried to construct matrices with only addition/subtraction operations in possible cases using a particular modal matrix P with a known inverse.

4.1 Matrices with given determinants

If we arbitrarily write a non-trivial square matrix with integer entries, even having a very small order like 3, almost always its determinant will never turn out to be $+1$ or -1. Therefore even if the matrix is non-singular, its inverse matrix can never have all integer entries. We begin this section with algorithms for generating uni-modular matrices $M \in GL_n(\mathbf{Z})$, and then we extend these algorithms for generating matrices with given determinants.

4.1.1 Algorithm for constructing a uni-modular matrix with all integer entries

(i) We generate such a matrix by the method of bordering. It is very easy to write a square matrix of order 2 with unit determinant. We first write such a matrix and then we fill up the first 2 entries of the 3rd row and the first 2 entries of the 3rd column with arbitrary integers. Then by the help of the FUNCTION subprogram for calculating the determinant of a matrix, we calculate m_{33} in such a way that the sub-determinant formed by the first 3 rows and the first 3 columns will be unity. It is easy to see that m_{33} will turn out to be an integer. Then we again fill up the first 3 entries of the 4th row and the first 3 entries of the 4th column with arbitrary integers, and compute m_{44} using the determinant subprogram, and so on. Continuing in this way, it is always possible for us to generate a square matrix of arbitrarily big order such that its determinant is unity, and therefore its inverse matrix

will also have all integer entries. In general,

$$m_{ii} = 1 + \sum_{j=1}^{i-1} (-1)^{i+j+1} m_{ij} M_{ij},\ i > 2,$$

where M_{ij} is the minor of m_{ij}. The Gaussian method of evaluation of a determinant is a $\frac{1}{3}n^3$ process and so this method of generating a unimodular matrix is a $\frac{1}{15}n^5$ process. Explicit evaluation of determinants can be avoided by the following modifications of algorithm (i).

(ii) Instead of filling up the first 2 entries of the 3rd row and the 3rd column arbitrarily, we fill up either the first 2 entries of the 3rd row (or the 3rd column) arbitrarily, and the first 2 entries of the 3rd column (or the 3rd row) by multiples of entries of any of the existing columns (or rows), and then the entry m_{33} can be calculated by using a simple relation. Continuing in this way, we can always generate a matrix $M \in GL_n(\mathbf{Z})$. We have $m_{ii} = 1 + xm_{ij}$, $i > 2$, for a fixed j, the first $i - 1$ entries of the ith column being x times the corresponding entries of the jth column.

Although the second method is less arbitrary than the first, but it does not require explicit evaluation of any determinant, and the entry m_{ii} is simply obtained using a single multiplication followed by a single addition. In the unimodular matrix so constructed, the values of all the principal sub-determinants will be 1 (except perhaps the principal sub-determinant of order 1). But these conditions can be very easily changed, i.e., the values of the principal minors can be other than unity simply by interchanging the rows and columns. All the integer eigenvalues of this matrix, if any, will be $+1$ or -1.

4.1.2 Matrices with given determinant

We can generate a matrix, all of whose entries are integers except perhaps 1 single entry, with determinant x, where x is any given number. In order to obtain such a matrix M, first we generate a unimodular matrix $M = (m_{ij})$, $m_{ij} \in \mathbf{Z}$, by one of the methods already described. Then we compute m_{nn} ($n \neq 2$) using the relation $m_{nn} := m_{nn} + (x - 1)$. The matrix M so obtained will have determinant x and all of its entries will be integers except perhaps the entry m_{nn}.

For $n = 2$, either we choose 3 integer entries of the matrix A arbitrarily, such that $a_{11} \neq 0$, and then we compute a_{22}, the 4th entry of the matrix, so that $\det A = x$. Alternatively we multiply all the elements of a row or a column of a unimodular matrix of order 2 by x (in this process at most 2 entries of the matrix may not be integers.)

4.2 Integer matrices with pre-assigned integer spectra

Let B be a square matrix with integer entries and let $P \in GL_n(\mathbf{Z})$, the group of unimodular matrices with integer entries. Then it is quite evident that if we call the matrix PBP^{-1} as A, then all entries of A are also integers. If the matrix B is of some suitably chosen simple form, such as a diagonal matrix, a Jordan canonical matrix or a triangular matrix, then with given eigenvalues as the diagonal entries of B and all other entries as integers, the matrix A will be an integer matrix with a prescribed integer spectrum.

The matrix P can be obtained using either the algorithm (i) or (ii) of

subsection 4.1.1. The generation of the matrix A is a $\frac{1}{15}n^5$ process if P is constructed using the algorithm (i), whereas it becomes a $3n^3$ process if we use algorithm (ii). The matrix A can be more simply generated by addition/subtraction operations with the help of a particular modal matrix which we shall discuss in the next section.

In the similarity transformation of the form $A = PBP^{-1}$, if B = J, a Jordan matrix, then the properties of the matrix A will depend on the properties of the Jordan matrix J.

Given $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_j$ as the distinct integer eigenvalues of J, such that λ_i is repeated k_i times ($k_i \geq 1\ \forall i \mid 1 \leq i \leq j$ and $\sum_{i=1}^{J} k_i = n$), the characteristic polynomial of A will be $\prod_{i=1}^{J}(x - \lambda_i)^{k_i}$, where x is an indeterminate. The properties of A will be

(i) If each block of J is of order 1, then A will be diagonalisable and the eigenvectors corresponding to the eigenvalue λ_i will be given by the t_eth columns ($1 \leq e \leq k_i$) of P, where

$$t_e = \sum_{f=1}^{i-1} k_f + e.$$

If $\exists$ at least 1 block in J which has order > 1, then A will be non-diagonalisable.

(ii) If $\exists$ only 1 block in J corresponding to each distinct eigenvalue, then A will be non-derogatory; otherwise A will be derogatory.

4.3 The modal matrix

Now we shall define that particular modal matrix to which we have referred earlier in the preceding section. Throughout this chapter whenever we shall refer to the modal matrix, we shall always mean this particular matrix. This matrix P = (p_{ij}) is a lower triangular matrix, such that p_{ij} = 1, i $\leq$ j and p_{ij} = 0, i > j $\forall i, j | 1 \leq i \leq n$, 1 $\leq$ j $\leq$ n. If we define the matrix P as above, then its inverse P^{-1} will be given by P^{-1} = (x_{ij}), such that x_{ii} = 1 $\forall$ i and $x_{k,k-1} = -1$ $\forall$ k | 2$\leq$ k $\leq$ n, while all other elements of P^{-1} are zero. We can trivially verify that $PP^{-1} = P^{-1}P = I$. If we use this particular modal matrix P in similarity transformations, then we can calculate similar matrices by performing addition/subtraction operations only; we can verify this fact easily.

It immediately follows that all the eigenvalues of this matrix are 1. The geometric mulplicity of the eigenvalue 1 is 1 and the algebraic multiplicity is n. The modal matrix P is similar to a Jordan matrix, which has only one block of order n, the order of P. The modal matrix has many interesting properties. For example, the symmetric matrix P^tP is

$$\begin{pmatrix} n & n-1 & \cdots & 3 & 2 & 1 \\ n-1 & n-1 & \cdots & 3 & 2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & 1 & \cdots & 1 & 1 & 1 \end{pmatrix},$$

the inverse of which is a tri-diagonal symmetric matrix of the form

$$\begin{pmatrix} 1 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}.$$

The eigenvalues $\mu_1, \mu_2, \mu_3, ..., \mu_n$ of the above tri-diagonal matrix are

$\mu_i = 2(1 - cos\frac{(2i-1)\pi}{2n+1})$.

It is given in [16](Ex 4.14, p.74) that the eigenvalues $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$ of the integer matrix P^tP are

$\lambda_i = \frac{1}{2}(1 - cos\frac{(2i-1)\pi}{2n+1})^{-1}$.

It can be easily proved that the determinant of the matrix

$$B = \begin{pmatrix} nx_1 & (n-1)x_1 & \cdots & 3x_1 & 2x_1 & x_1 \\ (n-1)x_2 & (n-1)x_2 & \cdots & 3x_2 & 2x_2 & x_2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 2x_{n-1} & 2x_{n-1} & \cdots & 2x_{n-1} & 2x_{n-1} & x_{n-1} \\ x_n & x_n & \cdots & x_n & x_n & x_n \end{pmatrix},$$

is $x_1x_2x_3, ..., x_n$, where B=DP^tP and D = diag$(x_1, x_2, x_3, ..., x_n)$, P is the modal matrix. B is an integer matrix if $x_i \in \mathbf{Z}$,

and the inverse of the matrix B (provided $x_i \neq 0\ \forall i$) is the tri-diagonal matrix of the form

$$\begin{pmatrix} \frac{1}{x_1} & -\frac{1}{x_2} & 0 & \cdots & 0 & 0 \\ -\frac{1}{x_1} & 2\frac{1}{x_2} & -\frac{1}{x_3} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2\frac{1}{x_{n-1}} & -\frac{1}{x_n} \\ 0 & 0 & 0 & \cdots & -\frac{1}{x_{n-1}} & 2\frac{1}{x_n} \end{pmatrix}.$$

If T=(t_{ij}) is an upper triangular matrix and P is the modal matrix, the elements of A(=PTP^{-1}) are computed by using the following algorithm:-

4.3.1 Algorithm

D1. $a_{1j} = t_{1j} - t_{1,j+1}\ \forall j | 1 \leq j \leq n-1$

D2. For i = 2, 3,...,n

$a_{i,i-1} = a_{i-1,i-1} - t_{ii}$

$a_{m,i-1} = a_{i,i-1}\ \forall m | i+1 \leq m \leq n$

$a_{ij} = a_{i-1,j} + t_{ij} - t_{i,j+1}\ \forall j | i \leq j \leq n-1$

D3. $a_{1n} = t_{1n}$

$a_{in} = a_{i-1,n} + t_{in}\ \forall i | 2 \leq i \leq n$

It is easy to generate matrices with known spectra (which are the diagonal elements of the triangular matrices from which we start) having exact entries using this particular similarity transformation since it involves only addition/subtraction operations.

4.3.2 Simple similarity transformations

(i) The repetitions of elements(below the diagonal) in the preceding process i.e. $a_{m,i-1} = a_{i,i-1}\ \forall i \geq 2$ and $\forall m | i+1 \leq m \leq n$ can be avoided by applying further the similarity transformation of interchanging the ith and the jth rows and the ith and the jth columns.

For example given a matrix

$$B = \begin{pmatrix} 5 & 7 & 6 & 9 \\ 1 & 4 & 8 & 10 \\ 1 & 2 & 9 & 12 \\ 1 & 2 & 3 & 15 \end{pmatrix},$$

by applying the similarity transformation of interchanging the 1st and the 3rd rows and then the 1st and the 3rd columns, we have

$$B = \begin{pmatrix} 9 & 2 & 1 & 12 \\ 8 & 4 & 1 & 10 \\ 6 & 7 & 5 & 9 \\ 3 & 2 & 1 & 15 \end{pmatrix}.$$

Or (ii) we can avoid this repetition by applying a simple similarity transformation of the form DBD^{-1}, where D=diag$(d_{11}, d_{22}, d_{33}, ..., d_{nn})$. The diagonal element d_{ii} is a factor of the HCF of all entries in the ith column except the diagonal element of the matrix B and this is true $\forall i | 1 \leq i \leq n$. If we apply this transformation, we can be sure that

B $\in M_n(\mathbf{Z}) \Rightarrow DBD^{-1} \in M_n(\mathbf{Z})$.

For example given a matrix

$$B = \begin{pmatrix} 5 & 6 & 4 \\ 7 & 2 & 6 \\ 7 & 3 & 5 \end{pmatrix}, \; D = \begin{pmatrix} 7 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

then

$$DBD^{-1} = \begin{pmatrix} 5 & 42 & 14 \\ 1 & 2 & 3 \\ 2 & 6 & 5 \end{pmatrix}.$$

4.4 Integer matrices with complex and surd eigenvalues

For construction of test matrices with complex and surd eigenvalues of the form $\lambda_i = \frac{1}{2}(a_i \pm \sqrt{b_i})$, where a_i,b_i, $\frac{a_i^2 - b_i}{4} \in \mathbf{Z}$ and b_i, if positive, is not a perfect square, $i \in \{1,2,3,...,n\}$, using the similarity transformation, the preferred simple form of the matrix D is a block diagonal matrix. This matrix D consists of a 2×2 block corresponding to the pair of eigenvalues of the form $\frac{1}{2}(a_i \pm \sqrt{b_i})$. In order to construct a non-trivial integer matrix having the same eigenvalues, now all we have to do is to apply a similarity transformation to this matrix, using a uni-modular matrix with all integer entries. The matrix so generated will be of order 2n×2n.

4.4.1 Real matrices with complex eigenvalues

Real matrices with complex eigenvalues may be constructed by taking a 2×2 block diagonal matrix corresponding to every pair of complex eigenvalues. Alternatively, we can proceed in the following way. Suppose that we would like to construct a real matrix with eigenvalues $a_j \pm ib_j$, $j \in \{1,2,3,...,n\}$. Then we construct a real diagonal matrix A with eigenvalues a_i and another real diagonal matrix B with eigenvalues b_i. Then a real matrix with complex eigenvalues will be

$$\begin{pmatrix} A & B \\ -B & A \end{pmatrix}$$

In order to construct a non-trivial real matrix having the same eigenvalues, now all we have to do is to apply a similarity transformation to this

matrix, using a modal matrix P such that $|P| = 1$. This will give us our required matrix of order 2n.

4.5 Integer matrices with generalised eigenvalues

Many eigenpackages, e.g., [17,32], deal with the generalised eigenvalue problem. In this problem given 2 square matrices A and B, we have to find a set of scalars λ and a non-zero vector x such that $Ax = \lambda Bx$. It has been found that most of the methods of computation of generalised eigenvalues are less than satisfactory, when the matrix B is ill-conditioned, with the exception of the QZ method for real A and B [32]. The integer matrices A and B generated by the methods described earlier may be ill-conditioned; as such these matrices will be very useful numerical data for justifying the advantages of the QZ algorithm over other algorithms which purport to solve this problem. In certain applications it is required that the matrices A and B should be symmetric. It has been proved by Taussky and Zassenhaus [58] that if A is a non-derogatory matrix, then $\exists$ a symmetric matrix X, called the symmetriser, such that XA is symmetric. The knowledge of the symmetriser makes it possible to reduce an apparently non-symmetric eigenvalue problem of the form $Ay = \lambda y$ to a symmetric eigenvalue problem of the form $By = \lambda Xy$, where B (= XA) and X are symmetric and at least 1 of X and B is positive definite [37]. Given a non-derogatory integer matrix A with known eigenvalues, we can find a symmetriser X of A, and the eigenvalues of A are the generalised eigenvalues satisfying the equation $By = \lambda Xy$.

4.6 Hessenberg matrix

It is well-known that any matrix can be reduced to Hessenberg form by a similarity transformation and its characteristic polynomial can be calculated by a simple recursive algorithm. But the entries of the Hessenberg matrix so obtained are usually non-integral but rational, even if all the entries of the original matrix are integers. In this section we have given an algorithm which produces a non-trivial integer Hessenberg matrix from a triangular matrix with known eigenvalues.

4.6.1 The algorithm

A lower Hessenberg matrix H is generated by taking the matrix P to be a lower triangular matrix. A lower triangular matrix with integer entries and unit determinant can easily be generated by taking all its diagonal entries as 1 and choosing all its other entries (below the diagonal) as arbitrary integers. Then P^{-1} will also be a lower triangular matrix with integer entries and unit determinant. We choose a matrix T with all the given eigenvalues as its diagonal entries and arbitrary non-zero super-diagonal entries, all the remaining entries of T being zero. Then PTP^{-1} will be a lower Hessenberg matrix. Similarly we can generate an upper Hessenberg matrix by replacing the matrix P by an upper triangular matrix and in this case P^{-1} will also be an upper triangular matrix with integer entries. In this case we replace the matrix T by a matrix having arbitrary non-zero sub-diagonal entries instead of non-zero super-diagonal entries.

If we again apply another similarity transformation to the lower/upper

Hessenberg matrix H, choosing P this time to be a uni-modular matrix with integer entries, other than a triangular matrix, then the new matrix A (= PHP^{-1}) now generated will be an arbitrary integer matrix whose Hessenberg form is the matrix H with given eigenvalues.

4.7 The positive matrix

The theory of non-negative matrices is a rapidly growing branch of pure mathematics. The positive matrix (all elements are positive) is a special case of a non-negative matrix, which has wide applications in diverse areas of applied mathematics, such as probability theory, numerical analysis, demography, mathematical economics and dynamic programming [50]. The positive matrix is found to be a very useful test matrix for testing the numerical stability of the routines for computation of the dominant eigenvalues of a matrix.

The algorithm presented by Hall and Porsching [22] for constructing positive matrices with known spectra is based on the following facts:-

(4.7.0.1) If

$$T(=t_{ij}) = \begin{pmatrix} \lambda_n & 0 & 0 & \cdots & \lambda_1 \\ 0 & \lambda_{n-1} & 0 & \cdots & \lambda_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_1 \end{pmatrix},$$

and $P = I + \epsilon\, J_{nn}$ ($J_{nn}=(1,1,\cdots,1)^t(1,1,\cdots,1)$) where $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \ldots\lambda_n > 0$ and $0 < \epsilon \leq \min\left\{ \frac{\lambda_n}{2\sum_{i,j} t_{ij}}, \frac{(\lambda_1-\lambda_2)}{\sum_{i,j} t_{ij}} \right\}$, then $P^{-1}TP$ is a positive matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$.

It is evident that $T \in M_n(\mathbf{Z}) \not\Rightarrow P^{-1}TP \in M_n(\mathbf{Z})$, i.e., it is not possible to generate positive integer matrices with integer spectra using this algorithm. In general the entries of the generated matrix will be non-integral rational numbers, having large numerators and large denominators, even with integer spectra, provided ϵ has been chosen rational.

4.7.1 Sufficient conditions for a matrix to be similar to a positive matrix

We can prove that the matrix $A = (a_{ij})$ is similar to a positive matrix if

(i) $\sum_{j=1}^{i} a_{jn} > 0 \ \forall i | 1 \leq i \leq n,$

(ii) $\sum_{k=1}^{i} a_{kj} > \sum_{k=1}^{i} a_{k,j+1} \ \forall i | 1 \leq i \leq n$ and $\forall j | 1 \leq j \leq n-1.$

If we choose the entries of the matrix A in such a way that the following conditions are satisfied, then we can immediately observe that the conditions written above will also be satisfied and therefore our matrix A will be similar to a positive matrix. As such the following conditions can also be said to be sufficient conditions for A to be similar to a positive matrix. However we must make it clear that the following conditions are stronger than the preceding conditions and the preceding conditions do not imply the following conditions.

(iii) $\sum_{j=1}^{i} a_{jn} > 0 \ \forall i | 1 \leq i \leq n,$

(iv) $a_{1j} > a_{1,j+1} \ \forall j | 1 \leq j \leq n-1,$

(v) $a_{ij} \geq a_{i,j+1}\ \forall i | 2 \leq i \leq n$ and $\forall j | 1 \leq j \leq n-1$.

We can verify that if the conditions (iii), (iv), (v) are satisfied, then PAP^{-1} is a positive matrix, where P is the modal matrix referred to in section 4.3 of this chapter.

Similarly we can argue that the upper triangular matrix $T = (t_{ij})$ is similar to a positive matrix if

(vi) $\sum_{j=1}^{i} t_{jn} > 0\ \forall i | 1 \leq i \leq n$,

(vii) $\sum_{k=1}^{j} t_{ki} > \sum_{k=1}^{j} t_{k,i+1}\ \forall i | 1 \leq i \leq n-1$ and $\forall j | 1 \leq j \leq i$,

(viii) $\sum_{k=1}^{i} t_{ki} > \sum_{k=1}^{i+1} t_{k,i+1}\ \forall i | 1 \leq i \leq n-1$;

or if

(ix) $\sum_{j=1}^{i} t_{jn} > 0\ \forall i | 1 \leq i \leq n$,

(x) (a) $t_{1j} > t_{1,j+1}\ \forall j | 1 \leq j \leq n-1$,

(b) $t_{11} > t_{12} + t_{22}$,

(xi) (a) $t_{ij} \geq t_{i,j+1}\ \forall i | 2 \leq i \leq n-1$ and $\forall j | i \leq j \leq n-1$,

(b) $t_{ii} \geq t_{i,i+1} + t_{i+1,i+1}\ \forall i | 2 \leq i \leq n-1$.

(4.7.1.1) If $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n \in \mathbf{Z}$ satisfy the conditions

(xii) $\lambda_1 + \sum_{i=p+1}^{n} \lambda_i \geq n$ where $\lambda_i < 0\ \forall i \geq p+1$

(xiii) $\lambda_1 \geq n + \lambda_2$

(xiv) $\lambda_i \geq \lambda_{i+1}$ $\forall i | 2 \leq i \leq n-1$, then there exists a positive integer matrix of order n with eigenvalues $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$.

If all the eigenvalues are non-negative, then the condition (xii) reduces to $\lambda_1 \geq$ n, where n is the order of the matrix.

To prove the existence of such matrices, it is enough to prove that there exists an upper triangular matrix satisfying the conditions (ix), (x), (xi).

In order to produce these conditions we have chosen T=(t_{ij}) of the form

$$T = \begin{pmatrix} \lambda_1 & \mu_1 - 1 & \mu_1 - 2 & \cdots & \cdots & \mu_1 - (n-1) \\ 0 & \lambda_2 & \mu_2 & \cdots & \cdots & \mu_2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \lambda_p & \cdots & \cdots & \mu_p \\ 0 & 0 & 0 & \lambda_{p+1} & \cdots & \mu_{p+1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & \lambda_n \end{pmatrix},$$

where $\mu_i = \lambda_i - \lambda_{i+1}$ if $\lambda_{i+1} \geq 0$,

and $\mu_i = \lambda_i$ if λ_{i+1} does not exist or if $\lambda_{i+1} < 0$. Thus in either case $\mu_n = \lambda_n$. It follows from the definition of μ_i and (xiv) that $\mu_i < 0$ if and only if $\lambda_i < 0$. Therefore $t_{in} < 0$ $\forall i \geq p+1$ and correspondingly $t_{in} \geq 0$ $\forall i | 2 \leq i \leq p$. (1)
(It is given that $t_{in} = \mu_i$ $\forall i \geq 2$.)

Proof:-

Without loss of generality, we can assume that $\exists$ (n−p) negative eigenvalues, where $1 \leq p \leq n$. If p=n, it means that no eigenvalue is negative.

(ix) $\lambda_1 + \sum_{i=p+1}^{n} \lambda_i \geq n$ where $\lambda_i < 0$ $\forall i \geq p+1$,

$\Rightarrow \lambda_1 + \lambda_{p+1} + \lambda_{p+2} + ... + \lambda_n \geq$ n

$\Rightarrow (\lambda_1 - \lambda_2) + (\lambda_2 - \lambda_3) + ... + (\lambda_{p-1} - \lambda_p) + \lambda_p + \lambda_{p+1} + ... + \lambda_n \geq$ n

$\Rightarrow \mu_1 + \mu_2 + ... + \mu_{p-1} + \mu_p + \mu_{p+1} + ... + \mu_n \geq$ n

$$\Rightarrow \mu_1 + \sum_{i=2}^{p} \mu_i + \sum_{i=p+1}^{n} \mu_i \geq \text{n}$$

$$\Rightarrow \sum_{i=2}^{p} \mu_i + \mu_1 - n \geq - \sum_{i=p+1}^{n} \mu_i$$

$$\Rightarrow \sum_{i=2}^{p} \mu_i + \mu_1 - (n-1) > - \sum_{i=p+1}^{n} \mu_i$$

$$\Rightarrow \sum_{i=2}^{p} \mu_i + \mu_1 - (n-1) + \sum_{i=p+1}^{n} \mu_i > 0$$

$$\Rightarrow \sum_{i=2}^{p} t_{in} + t_{1n} + \sum_{i=p+1}^{n} t_{in} > 0 \text{ (the given matrix T} = (t_{ij}))$$

$$\Rightarrow \sum_{i=1}^{n} t_{in} > 0. \qquad (2)$$

It is evident that either $\mu_1 = \lambda_1$, or $\mu_1 = \lambda_1 - \lambda_2$.

If $\mu_1 = \lambda_1$, then (xii) $\Rightarrow \mu_1 \geq n$.

Otherwise (xiii) $\Rightarrow \mu_1 \geq n$.

Therefore $\mu_1 - (n-1) > 0$ i.e., $t_{1n} > 0$. (3)

Thus (1),(3) $\Rightarrow t_{in} \geq 0 \ \forall i | 1 \leq i \leq p$ and $t_{in} < 0 \ \forall i \geq p+1$. (4)

It follows from (2) and (4) that $\sum_{j=1}^{i} t_{jn} > 0 \ \forall i | 1 \leq i \leq n$.

(x)(a) $\mu_1 - (n-1) > 0$

$\Rightarrow \exists \ \mu_1 - 1, \mu_1 - 2, \mu_1 - 3, ..., \mu_1 - (n-1)$, (n−1) distinct natural numbers in decreasing sequence.

and so $t_{1i} > t_{1,i+1}\ \forall i | 1 \leq i \leq (n-1)$.

(x)(b) If $\lambda_2 < 0,\ \mu_1 = \lambda_1$

$\Rightarrow \lambda_1 > \mu_1 - 1$

$\Rightarrow \lambda_1 > \lambda_2 + \mu_1 - 1$.

If $\lambda_2 \geq 0,\ \mu_1 = \lambda_1 - \lambda_2$

$\Rightarrow \lambda_1 > \lambda_2 + \mu_1 - 1$.

Hence in both the cases, we have $t_{11} > t_{12} + t_{22}$.

(xi)(a) From the definition of μ_i, we have

$\mu_i = \lambda_i - \lambda_{i+1}$ if $\lambda_{i+1} \geq 0$

i.e., $\lambda_i = \mu_i + \lambda_{i+1}$ (5)

$\Rightarrow \lambda_i \geq \mu_i$

and

$\mu_i = \lambda_i$ if $\lambda_{i+1} < 0$

$\Rightarrow \lambda_i \geq \mu_i + \lambda_{i+1}$ (6)

Furthermore $t_{ij} = \mu_i\ \forall i | 2 \leq i \leq n-1$ and $\forall j | i+1 \leq j \leq n-1$,

therefore $t_{ij} \geq t_{i,j+1}\ \forall i | 2 \leq i \leq n-1$ and $\forall j | i \leq j \leq n-1$.

(xi)(b) (5) and (6) $\Rightarrow t_{ii} \geq t_{i,i+1} + t_{i+1,i+1}\ \forall i | 2 \leq i \leq n-1$.

However we must mention that this particular form is not always the only possible upper triangular form for generating these matrices. For example, in order to construct a positive integer matrix with eigenvalues 15,−2,−1, the upper triangular matrix may be P instead of Q, where

$$P = \begin{pmatrix} 15 & 10 & 4 \\ 0 & -2 & 2 \\ 0 & 0 & -1 \end{pmatrix},\ Q = \begin{pmatrix} 15 & 14 & 13 \\ 0 & -2 & -2 \\ 0 & 0 & -1 \end{pmatrix}.$$

If the entries of the positive matrix belong to **Q(or R)** then the conditions can be relaxed further.

(4.7.1.2) If $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n \in$ **Q(or R)**, satisfy the conditions

(xv) $\lambda_1 + \sum_{i=p+1}^{n} \lambda_i > 0$ where $\lambda_i < 0 \; \forall i \geq p+1$,

(xvi) $\lambda_1 > \lambda_2$,

(xvii) $\lambda_i \geq \lambda_{i+1} \; \forall i \; |2 \leq i \leq n-1$, then there exists a positive rational(or real) matrix of order n with eigenvalues $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$.

If all the eigenvalues are non-negative, then the condition (xv) reduces to $\lambda_1 > 0$.

To prove the existence of such a matrix, the preferred upper triangular matrix is of the same form as chosen for proving 4.7.1.1 except for the elements of the 1st row,

i.e., the upper triangular matrix, which we consider, is of the form

$$T(=t_{ij}) = \begin{pmatrix} \lambda_1 & \mu_1 - \frac{q}{n-2} & \mu_1 - \frac{q}{n-3} & \cdots & \cdots & \mu_1 - q \\ 0 & \lambda_2 & \mu_2 & \cdots & \cdots & \mu_2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \lambda_p & \cdots & \cdots & \mu_p \\ 0 & 0 & 0 & \lambda_{p+1} & \cdots & \mu_{p+1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & \lambda_n \end{pmatrix}.$$

We shall choose q > 0 so that q < min (q_1, q_2) where q_1 and q_2 are arbitrary rational (or real) numbers such that

$$\lambda_1 + \sum_{i=p+1}^{n} \lambda_i > q_1,\ \lambda_i < 0\ \forall i \geq p+1, \tag{1}$$

$$\lambda_1 > \lambda_2 + q_2, \tag{2}$$

which immediately follows from the fact that

$$\lambda_1 + \sum_{i=p+1}^{n} \lambda_i > 0,\ \lambda_i < 0\ \forall i \geq p+1;$$

$$\lambda_1 > \lambda_2$$

$$\Rightarrow \exists\ q_1, q_2 \in \mathbf{Q}(\text{ or } \mathbf{R}) \mid q_1, q_2 > 0,$$

$$\text{s.t. } \lambda_1 + \sum_{i=p+1}^{n} \lambda_i > q_1,\ \lambda_i < 0\ \forall i \geq p+1;$$

$$\text{and } \lambda_1 > \lambda_2 + q_2.$$

$$(1) \text{ and } (2) \Rightarrow \lambda_1 + \sum_{i=p+1}^{n} \lambda_i > q,\ \lambda_i < 0\ \forall i \geq p+1;$$

$$\text{and } \lambda_1 > \lambda_2 + q.$$

Thus $\mu_1 - q > 0$ and we can argue as we did in the preceding proof to show that the entries of the upper triangular matrix satisfy the conditions (ix), (x), (xi).

Thus it is no longer necessary for all the eigenvalues to be positive, as is required in the method of Hall and Porsching, for generating a positive matrix with known spectra.

4.8 Symmetric matrices

It is well-known that a matrix is symmetric if and only if it is orthogonally similar to a diagonal matrix. Real symmetric matrices have many interesting

results associated with them, for example, the algebraic multiplicity corresponding to each eigenvalue is equal to the corresponding geometric multiplicity; and all the eigenvalues are real. In many applications real symmetric matrices play an important role. However there does not always exist a non-trivial integer symmetric matrix with a prescribed integer spectrum. For example, there does not exist any non-trivial integer matrix with eigenvalues 2 and 3. In the case of the imprecise field **R** it is difficult to generate such a matrix exactly with the help of an orthogonal similarity transformation. Thus we have given sufficient conditions on the spectrum, which if satisfied, the generated matrix will have all integer entries (for the integer spectrum) and it will be exact (even if the spectrum consists of non-integral rational numbers).

These conditions are:-

(4.8.0.1) If D = diag $(\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n)$ and Q = $\frac{1}{d}(q_{ij})$, Q being an orthogonal matrix, λ_i,d^2,$q_{ij} \in \mathbf{Z}$, then $QDQ^t \in M_n(\mathbf{Z})$ if $\lambda_i - \lambda_{i+1} = m_i d^2$ $\forall i | 1 \leq i \leq n-1$, $m_i \in \mathbf{Z}$ $\forall i$.

Proof:-

The ith diagonal element of QDQ^t (=X) is given by

$$\begin{aligned}
x_{ii} &= \tfrac{1}{d^2}(\lambda_1 {q_{i1}}^2 + \lambda_2 {q_{i2}}^2 + \lambda_3 {q_{i3}}^2 + ... + \lambda_n {q_{in}}^2) \\
&= \tfrac{1}{d^2}\{\lambda_1 {q_{i1}}^2 + (\lambda_1 - m_1 d^2){q_{i2}}^2 + (\lambda_1 - m_1 d^2 - m_2 d^2){q_{i3}}^2 \\
&\quad + ... + (\lambda_1 - m_1 d^2 - m_2 d^2 - ... - m_{n-1} d^2){q_{in}}^2\} \\
&= \tfrac{1}{d^2}\{\lambda_1({q_{i1}}^2 + {q_{i2}}^2 + {q_{i3}}^2 + ... + {q_{in}}^2) - m_1 d^2({q_{i2}}^2 + {q_{i3}}^2 + ... + {q_{in}}^2) \\
&\quad - m_2 d^2({q_{i3}}^2 + ... + {q_{in}}^2) - ... - m_{n-1} d^2 {q_{in}}^2\} \\
&= \lambda_1 - m_1({q_{i2}}^2 + {q_{i3}}^2 + ... + {q_{in}}^2) - m_2({q_{i3}}^2 + ... + {q_{in}}^2) - ... - m_{n-1}{q_{in}}^2.
\end{aligned}$$

The non-diagonal element x_{ij} of X is given by

$$\begin{aligned}
x_{ij} &= \tfrac{1}{d^2}(\lambda_1 q_{i1}q_{j1} + \lambda_2 q_{i2}q_{j2} + \lambda_3 q_{i3}q_{j3} + \ldots + \lambda_n q_{in}q_{jn}) \\
&= \tfrac{1}{d^2}\{\lambda_1 q_{i1}q_{j1} + (\lambda_1 - m_1 d^2)q_{i2}q_{j2} + (\lambda_1 - m_1 d^2 - m_2 d^2)q_{i3}q_{j3} \\
&\quad + \ldots + (\lambda_1 - m_1 d^2 - m_2 d^2 - \ldots - m_{n-1} d^2)q_{in}q_{jn}\} \\
&= \tfrac{1}{d^2}\{\lambda_1(q_{i1}q_{j1} + q_{i2}q_{j2} + q_{i3}q_{j3} + \ldots + q_{in}q_{jn}) - m_1 d^2(q_{i2}q_{j2} + q_{i3}q_{j3} \\
&\quad + \ldots + q_{in}q_{jn}) - m_2 d^2(q_{i3}q_{j3} + \ldots + q_{in}q_{jn}) - \ldots - m_{n-1} d^2 q_{in}q_{jn}\} \\
&= -m_1(q_{i2}q_{j2} + q_{i3}q_{j3} + \ldots + q_{in}q_{jn}) - m_2(q_{i3}q_{j3} + \ldots + q_{in}q_{jn}) \\
&\quad - \ldots - m_{n-1} q_{in}q_{jn}.
\end{aligned}$$

Now $\lambda_1, m_i, q_{ij} \in \mathbf{Z} \Rightarrow x_{ij} \in \mathbf{Z}\ \forall i, j | 1 \leq i, j \leq n.$

Hence $QDQ^t \in M_n(\mathbf{Z})$.

(i) In the algorithm for the generation of test matrices with certain sign patterns by orthogonal transformation Schneider [55] has observed that the particular orthogonal matrix T may be of the form $T = \frac{1}{d}(t_{ij})$, where $d, t_{ij} \in \mathbf{Z}$, so that the matrices with prescribed integer/rational spectra may have rational entries.

(ii) Ortega [44] has also presented an algorithm for constructing an orthogonal matrix of the particular form $I - 2VV^t$, where $V^t = [n^{-\frac{1}{2}}, n^{-\frac{1}{2}}, n^{-\frac{1}{2}}, \ldots, n^{-\frac{1}{2}}]$, which he has used for constructing a symmetric matrix $A = (a_{ij})$, where $a_{ij} = n^{-1}(nd_i\delta_{ij} - 2d_i - 2d_j + 4n^{-1}\sum_{k=1}^{n} d_k)$.

In the Schneider method, if the spectrum satisfies the condition which we have deduced for generating an integer symmetric matrix, the generated matrix will have certain sign patterns; and in (ii) the division will be exact; as such the generated matrix will also have integer entries.

4.8.1 Sufficient conditions for a matrix to be similar to a symmetric matrix

We can verify that if A = (a_{ij}) is any matrix and P is the modal matrix referred to earlier, then PAP^{-1} is a symmetric matrix if the following conditions are satisfied:-

(i) $\sum_{k=1}^{j} a_{ki} - \sum_{k=1}^{j} a_{k,i+1} = \sum_{m=1}^{i} a_{mj} - \sum_{m=1}^{i} a_{m,j+1}$ $\forall i | 1 \leq i \leq n-1$ and $\forall j | i+1 \leq j \leq n-1$

(ii) $\sum_{i=1}^{n} a_{ij} - \sum_{i=1}^{n} a_{i,j+1} = \sum_{m=1}^{j} a_{mn}$ $\forall j | 1 \leq j \leq n-1.$

Similarly we can verify that if T = (t_{ij}) is any upper triangular matrix and P is the same modal matrix as before, then PTP^{-1} is a symmetric matrix if the following condition is satisfied:-

(iii) $t_{in} - (t_{i,i+1} + t_{i+1,i+1}) = t_{ij} - t_{i,j+1} = t_{in}$ $\forall i | 1 \leq i \leq n-1$ and $\forall j | i+1 \leq j \leq n-1.$

(4.8.1.1) It follows from the condition (iii) that if we can arrange $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$ such that $\lambda_i - \lambda_{i+1} = (n+1-i)x_i$ $\forall i | 1 \leq i \leq n-1$, $\lambda_i, x_i \in \mathbf{Z}$ $\forall$i, then there exists a non-trivial integer symmetric matrix of order n with eigenvalues $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$.

In order to produce this condition we have chosen T=(t_{ij}) of the form

$$T = \begin{pmatrix} \lambda_1 & (n-1)x_1 & (n-2)x_1 & \cdots & x_1 \\ 0 & \lambda_2 & (n-2)x_2 & \cdots & x_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_n \end{pmatrix}.$$

However, if the eigenvalues $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n \in \mathbf{Q}$ (or $\mathbf{R}$), we can always generate a non-trivial symmetric matrix with rational (or real) entries with the given eigenvalues using only addition/subtraction operations.

The conditions which we have deduced for an integer symmetric matrix with known spectrum can simply be tested by taking all possible permutations of the given eigenvalues; the algorithm for the computation of all permutations of {1,2,3,...,n} has been presented in Chapter 1, section 1.7.

4.9 The positive symmetric matrix

In section 4.8 we have deduced the sufficient conditions for the matrix A=(a_{ij}) to be similar to a symmetric matrix. We can further verify that if A = (a_{ij}) is any matrix and P is the modal matrix referred to in section 4.3 of this chapter, then PAP^{-1} is a positive symmetric matrix if the following conditions are satisfied:-

$$\text{(i)} \sum_{k=1}^{j} a_{ki} - \sum_{k=1}^{j} a_{k,i+1} = \sum_{m=1}^{i} a_{mj} - \sum_{m=1}^{i} a_{m,j+1} > 0 \ \forall i | 1 \leq i \leq n-1, \text{ and } \forall j | i+1 \leq j \leq n-1,$$

$$\text{(ii)} \sum_{i=1}^{n} a_{ij} - \sum_{i=1}^{n} a_{i,j+1} = \sum_{m=1}^{j} a_{mn} > 0 \ \forall j | 1 \leq j \leq n-1.$$

Similarly for an upper triangular matrix T = (t_{ij}), PTP^{-1} is a positive symmetric matrix, if the following conditions are satisfied:-

$$\text{(iv)} \ t_{ii} - (t_{i,i+1} + t_{i+1,i+1}) = t_{ij} - t_{i,j+1} = t_{in} \ \forall i | 1 \leq i \leq n-1, \text{ and } \forall j | i+1 \leq j \leq n-1,$$

(v) $\sum_{j=1}^{i} t_{jn} > 0 \ \forall i | 1 \leq i \leq n,$

(vi) (a) $t_{11} - t_{12} > 0,$

(b) $t_{ii} - t_{i,i+1} \geq 0 \ \forall i | 2 \leq i \leq n-1.$

(4.9.0.1) It follows from the conditions (iv), (v), (vi) that if we can arrange $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$ such that

(vii) $\lambda_i - \lambda_{i+1} = (\mathrm{n+1-i})\mathrm{x}_i \ \forall i | 1 \leq i \leq n-1$ and $x_n = \lambda_n,$

(viii) $\sum_{j=1}^{i} x_j > 0 \ \forall i | 1 \leq i \leq n,$

(ix) (a) $\lambda_1 - (n-1)x_1 > 0,$

(b) $\lambda_i - (n+1-i)x_i \geq 0 \ \forall i | 2 \leq i \leq n-1,$

$x_i, \lambda_i \in \mathbf{Z} \ \forall i$, then there exists a non-trivial positive integer symmetric matrix of order n with eigenvalues $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$.

In order to produce conditions (iv), (v), (vi) we have chosen T= (t_{ij}) of the form

$$T = \begin{pmatrix} \lambda_1 & (n-1)x_1 & (n-2)x_1 & \cdots & x_1 \\ 0 & \lambda_2 & (n-2)x_2 & \cdots & x_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_n \end{pmatrix}.$$

We can easily verify that if we choose T as above, the required conditions will be satisfied.

4.10 Hermitian, skew-symmetric, skew-Hermitian

The algorithm given in [44] for generation of Hermitian matrices may also be used for generating skew-symmetric and skew-Hermitian matrices. Let us consider the matrix M = PBP, where P is an orthogonal symmetric matrix and B is a block diagonal matrix.

It is easy to see that

(i) If B is a skew-symmetric matrix, then M is also skew-symmetric;

(ii) If B is Hermitian, then M is Hermitian;

(iii) If B is skew-Hermitian, then M is skew-Hermitian.

The block diagonal matrix B with given eigenvalues and any of these forms can be very easily constructed. For example, a matrix B with given eigenvalues $\pm 2i$, 0 is

$$\begin{pmatrix} 0 & 2 & 0 \\ -2 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Thus

$$M = \frac{1}{9}\begin{pmatrix} 1 & -2 & -2 \\ -2 & 1 & -2 \\ -2 & -2 & 1 \end{pmatrix}\begin{pmatrix} 0 & 2 & 0 \\ -2 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & -2 & -2 \\ -2 & 1 & -2 \\ -2 & -2 & 1 \end{pmatrix}$$

$$= \frac{2}{3}\begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & 2 \\ -2 & -2 & 0 \end{pmatrix}.$$

Thus M is a non-trivial skew-symmetric matrix with given eigenvalues.

4.11 Discussion

For generating (**a**) positive (**b**) symmetric (**c**) positive symmetric matrix with known eigenvalues satisfying certain conditions we have chosen a particular upper triangular matrix. The entries of these matrices may be computed by using the algorithm 4.3.1. It is easy to generate matrices (**a**),(**b**),(**c**), with known spectra (which are the diagonal elements of the triangular matrix from which we start) having exact entries, since the algorithm involves only addition/subtraction operations. Moreover there is no unnecessary explosion of the entries of the generated matrices which generally accompany such algorithms.

The pattern of elements in these matrices may be changed by applying simple similarity transformations discussed in subsection 4.3.2. The transformation (i) of this subsection may be applied to all these matrices (**a**),(**b**),(**c**), whereas the transformation (ii) may be applied to matrices (**a**).

While developing algorithms for generating matrices with pre-assigned spectra we have paid particular attention to the matrices with integer entries and having certain known properties and special forms. The usefulness of such matrices in testing the eigenvalue routines has been amply demonstrated in [33,34,66,67]. These matrices, apart from providing numerical examples to test the eigenvalue routines, will also serve the basic task of providing suitable numerical examples to illustrate known theorems and, in many cases, to make up or to support conjectures which may lead to the discovery of new algorithms.

References

1. Alia, G.; Martinelli, E. On the lower bound to the VLSI complexity of number conversion from weighted to residue representation. IEEE Trans. Computers **42** (1993) 962-967.

2. Brenner, J. L. A set of test matrices for testing computer programs. CACM **5** (1962), 443-444.

3. Bailey, D. H. Algorithm 719, Multiprecision translation and execution of FORTRAN programs. ACM Trans. Math. Softw. **19** (1993), 288-319.

4. Bai Z.; Demmel, J.; Mckenney, A. On computing condition numbers for the nonsymmetric eigenproblem. ACM Trans. Math. Softw. **19** (1993), 202-223.

5. Bini, D.; Pisa; Pan, V. Practical improvement of the divide-and-conquer eigenvalue algorithms. Computing **48** (1992) 109-123.

6. Chase, P. J. Algorithm 382, Combinations of M out of N objects. CACM **13** (1970), 368.

7. Chan, S. P.; Feldman, R.; Parlett, B. N. Algorithm 517, A program for computing the condition numbers of matrix eigenvalues without computing the eigenvectors. ACM Trans. Math. Softw. **3** (1977), 186-203.

8. Czopik, J. The simultaneous determination of all zeros of a polynomial. Computing **45** (1990), 79-91.

9. Charmonman, S. Eigenvectors of a 2nx2n matrix. CACM **10** (1967), 800-801.

10. Cohen, H. A Course in Computational Algebraic Number Theory. Springer-Verlag, Berlin, 1993.

11. Danilina, N. I.; Dubrovskaya, N. S.; Kvasha, O. P.; Smirnov, G. L. Computational Mathematics. Mir publishers, Moscow, 1988.

12. Davis; Hoffmann, FORTRAN 77: A Structured, Disciplined Style. 2nd ed., Mcgraw-Hill, 1983.

13. Eldridge, S. E.; Walter, C. D. Hardware implementation of Montgomery's modular multiplication algorithm. IEEE Trans. Computers **42** (1993), 693-699.

14. Flannery, W. H.; Tenkolsky, B. P.; Vetterling, W. T. Numerical Recipes. Cambridge University Press, 1986.

15. Fenner, T. I.; Loizou, G. A binary tree representation and related algorithms for generating integer partitions. The Computer J. **23** (1980) 332-337.

16. Gregory, R. T.; Karney, D. L. A Collection of Test Matrices for Testing Computational Algorithms. Wiley-Interscience, New York, 1969.

17. Garbow, B. S. The QZ algorithm to solve the generalized eigenvalue problem for complex matrices. ACM Trans. Math. Softw. **4** (1978), 404-410.

18. Golub, G. H.; Robertson, T. N. A generalized Bairstow algorithm. CACM **10** (1967), 371-373.

19. Grad, J.; Brebner, M. A. Algorithm 343, Eigenvalues and eigenvectors of a real general matrix. CACM **11** (1968), 820-826.

20. Gear, C. W. A simple set of test matrices for eigenvalue programs. Math. Comput. **23** (1969), 119-125.

21. Hashemian, R. Square rooting algorithms for integer and floating-point numbers. IEEE Trans. Computers **39** (1990), 1025-1029.

22. Hall, C. A.; Porsching, T. A. Generation of positive test matrices with known positive spectra. CACM **11** (1968), 559-560.

23. Hardy, G. H.; Wright, E. M. An Introduction to the Theory of Numbers. Clarendon Press, Oxford, 1954.

24. Horowitz, E.; Sahni, S. Fundamentals of Computer Algorithms. Galgotia, 1993.

25. Jenkins, M. A.; Traub, J. F. Principles for testing polynomial zerofinding programs. ACM Trans. Math. Softw. **1** (1975), 26-34.

26. Jenkins, M. A. Algorithm 493, Zeros of a real polynomial. ACM Trans. Math. Softw. **1** (1975), 178-189.

27. Kumar, V. Computer Programming in FORTRAN IV, Vol. I: Language Fundamentals. Pragati Prakashan, Meerut.

28. Kumar, V. Some efficient FORTRAN IV computer programs for calculating prime numbers. Proc. of the workshop on Comp. Appl. University of Roorkee (1986) 253-260.

29. Kumar, V. Computer Programming in FORTRAN IV, Vol. II: Selected Algorithms. (Due to be published)

30. Knuth, D. E. The Art of Computer Programming, Vol. 1: Fundamental Algorithms. Addison-Wesley, Reading, Mass., 1968.

31. Knuth, D. E. The Art of Computer Programming, 2nd Ed., Vol. 2: Seminumerical Algorithms. Addison-Wesley, Reading, Mass., 1981.

32. Kaufman, L. Some thoughts on the QZ algorithm for solving the generalized eigenvalue problem. ACM Trans. Math. Softw. **3** (1977), 63-75.

33. Knoble, H. D. Certification of Algorithm 343, Eigenvalues and eigenvectors of a real general matrix. CACM **13** (1970), 122-124.

34. Knight, W.; Mersereau W. Remark on Algorithm 343, Eigenvalues and eigenvectors of a real general matrix. CACM **13** (1970), 694-695.

35. Kågström, B.; Ruhe, A. An algorithm for numerical computation of the Jordan normal form of a complex matrix. ACM Trans. Math. Softw. **6** (1980), 398-419.

36. Kågström, B.; Ruhe, A. An algorithm for numerical computation of the Jordan normal form of a complex matrix. ACM Trans. Math. Softw. **6** (1980), 437-443.

37. Krishnamurthy, E. V.; Sen, S. K. Numerical Algorithms. East-West Press, 1986.

38. Lewis, D. M. An architecture for addition and subtraction of long word length numbers in the logarithmic number system. IEEE Trans. Computers **39** (1990), 1325-1336.

39. Luo, X. A practical sieve algorithm for finding prime numbers. CACM **32** (1989) 344-346 [Carroll, T. D. CACM **32** (1989) 1367.]

40. Mckenzie, B. J.; Takaoka, T. A control structure for a variable number of nested loops. The Computer J. **26** (1983), 282-283.

41. Mcnamee, J. M. A sparse matrix package–part ii: special cases. ACM Trans. Math. Softw. **9** (1983), 340-343.

42. Newman, M. Integral Matrices. Academic Press, New York, 1972.

43. Muthiyalu, N; Usha, S. Eigenvalues of centrosymmetric matrices. Computing **48** (1992) 213-218.

44. Ortega, J. M. Generation of test matrices by similarity transformations. CACM **7** (1964), 377-378.

45. Prado, J.; Alcantara, R. A fast square-rooting algorithm using a digital signal processor. Proc. IEEE , **75** (1987) 262-264.

46. Pritchard, P. Some negative results concerning prime number generators. CACM **27** (1984), 53-57.

47. Pinkert, J. R. An exact method for finding the roots of a complex polynomial. ACM Trans. Math. Softw. **2** (1976) 351-363.

48. Rao, T. M. Error-free computation of characteristic polynomial of a matrix. Computers and Math. with Applications, **4** (1978) 61-65.

49. Rayward-Smith, V. J. On computing Smith normal form of an integer matrix. ACM Trans. Math. Softw. **5** (1979), 451-456.

50. Seneta, E. Non-negative Matrices and Markov Chain 2nd ed., Springer-Verlag, New York, 1980.

51. Skordalakis, E.; Papakonstantinou, G. A control structure for a variable number of nested loops. The Computer J. **25** (1982), 48-51.

52. Stewart, G. W. Algorithm 506, HQR3 and EXCHANG: FORTRAN subroutine for calculating and ordering the eigenvalues of a real upper Hessenberg matrix. ACM Trans. Math. Softw. **2** (1976), 275-280.

53. Stewart, G. W. Algorithm 384, Eigenvalues and eigenvectors of a real symmetric matrix. CACM **13** (1970), 369-371.

54. Stewart, G. W. Introduction to Matrix Computations. Academic Press, New York, 1973.

55. Schneider, A. J. Generation of test matrices having certain sign patterns and prescribed positive spectra. CACM **12** (1969), 378-379.

56. Sentance, W. A. ; Cliff, I. P. The determination of eigenvalues of symmetric quindiagonal matrices. The Computer J. **24** (1981), 177-179.

57. Stoer, J.; Bulirsch, R. Introduction to Numerical Analysis. Springer-Verlag, New York, 1980.

58. Taussky, O.; Zassenhaus, H. On the similarity transformation between a matrix and its transpose. Pacific J. Math. **9** (1959), 893-896.

59. Tessler, L.; Eisenberg, L. A new algorithm for factoring polynomials. Proc. IEEE **60** (1972) 737-738.

60. Topor, R. W. Functional programs for generating permutations. The Computer J. **25** (1982) 257-263.

61. Vuillemin, J. E. Exact real computer arithmetic with continued fractions. IEEE Trans. Computers **39** (1990), 1087-1105.

62. Wilson, J. M. Algorithm 114, Interrupted permutations in lexicographic order. The Computer J. **26** (1983), 92.

63. Wilkinson, J. H. The Algebraic Eigenvalue Problem. Oxford University Press, Oxford, 1965.

64. Wichmann, B. A. A note on the use of floating point in critical systems. The Computer J. **35** (1992) 41-44.

65. Weinberger, P. J.; Rothschild, L. P. Factoring polynomials over algebraic number fields. ACM Trans. Math. Softw. **2** (1976), 335-350.

66. Welsch, J. H. Certification of Algorithm 253, Eigenvalues of a real symmetric matrix by the QR method. CACM **10** (1967), 376.

67. Welsch, J. H. Certification of Algorithm 254, Eigenvalues and eigenvectors of a real symmetric matrix by the QR method. CACM **10** (1967), 376.

68. Zimmer, H. G. Computational Problems, Methods, and Results in Algebraic Number Theory, Lectures Notes in Mathematics No. 262. Springer-Verlag, New York, 1972.

APPENDIX

THE PROGRAMS

All the programs given on the succeeding pages have been successfully run in VMS environment using VAX FORTRAN 77 compiler and also in DOS using SOFTEK FORTRAN 77 compiler. Systematic testing of the programs has been accomplished in part with the aid of programs which reconstruct polynomials from their known zeros and generate matrices with known spectra and certain known properties. Detailed description of the individual subroutines and the meanings of the parameters are described in the initial comments of the subroutines. The dimensions in the programs as given in the listings have been chosen small to facilitate execution in machines with small memory. However these can be changed simply by changing the DIMENSION statements. All the integer variables are declared to be INTEGER * 4, so that, if necessary, they may take up values which are numerically as large as possible for 32-bit machines. We have not physically joined our different programs with the programs for arithmetic operations on multi-precision integers to maintain the aesthetic beauty of coding different algorithms. But whenever occasion arises, we can do it immediately and thus we can get programs which will work satisfactorily in all possible situations. In the programs for multi-precision integers we have taken each piece to be of maximum 4 decimal digits. However larger numbers may be taken by systematically changing the PARAMETER statements and correspondingly

modifying the coding of the PRINT subroutines which have been developed using dynamic formatting. In this thesis we have not included the codes of the algorithms which can be obtained from the given listings by very simple routine modifications. For example, the subroutines for the computation of complex and surd eigenvalues of particular types of matrices with integer entries may be obtained by joining the subroutines for the computation of the characteristic polynomial of an integer matrix with the subroutines for the computation of complex and surd zeros of a particular type of integer polynomials. Also the subroutines presented in this thesis for the computation of eigenvectors corresponding to integer eigenvalues may be used for the computation of eigenvectors corresponding to rational eigenvalues by making necessary routine changes. Although we have used the integer mode in all programs, still many of the programs, such as reconstruction of polynomials from their zeros, computation of the characteristic polynomial of a matrix, etc. will work satisfactorily in real mode also, provided that we make the necessary changes in the type statements. The contents of the Appendix have been given in the end.

5.1 The program for addition of two non-negative m, n-piece integers

```
      PARAMETER (ND = 20)
      DIMENSION L (ND), M (ND), N (ND)
10    WRITE (*,'('' TYPE  NUMBERS OF PIECES IN THE 2 INTEGERS''
     1,1X,''TO BE ADDED IN FORMAT (2I2)'')')
20    READ (4, '(2I2)') I, J
      WRITE (6, '('' THE NUMBERS OF PIECES ARE'', 2I4)') I, J
      IF (I .EQ. 0 .AND. J .EQ. 0) STOP
      IF (I .LE. 0 .OR. J .LE. 0 .OR. I .GT. ND .OR. J .GT. ND) THEN
         IF (I .LE. 0 .OR. J .LE. 0) THEN
            WRITE (6, '('' DATA ILLEGAL, PLEASE TYPE AGAIN'')')
         ELSEIF (I .GT. ND .AND. J .GT. ND) THEN
            WRITE (6,'('' BOTH INTEGERS TOO LARGE, PLEASE TYPE AGAIN'')'
         ELSEIF (I .GT. ND) THEN
            WRITE (6,'('' FIRST INTEGER TOO LARGE, PLEASE TYPE AGAIN'')'
         ELSE
            WRITE (6,'('' SECOND INTEGER TOO LARGE, PLEASE TYPE AGAIN'')
         ENDIF
         GO TO 20
      ENDIF
      WRITE (*, '('' TYPE THE FIRST INTEGER IN FORMAT (20I4)'')')
      READ (4, '(20I4)') (L (K), K = I, 1, -1)
      WRITE (*, '('' TYPE THE SECOND INTEGER IN FORMAT (20I4)'')')
      READ (4, '(20I4)') (M (K), K = J, 1, -1)
      WRITE (6, '('' THE INTEGERS ARE AS FOLLOWS'')')
      CALL PRINT (L, I, 0)
      CALL PRINT (M , J, 0)
      IND = 0
      CALL BIGADD (L, I, M, J, N, KK, IND)
      IF (IND .EQ. 1) THEN
         WRITE (6, '('' THE SUM HAS MORE THAN 80 DIGITS'')')
      ELSE
         WRITE (6, '('' THE SUM IS'')')
         CALL PRINT (N, KK, 0)
      ENDIF
      GO TO 10
      END
C PURPOSE - TO FIND THE SUM OF TWO NON-NEGATIVE LARGE INTEGERS
C INPUT PARAMETERS
C L  : LINEAR ARRAY, THE FIRST INTEGER
C LL : NUMBER OF PIECES IN THE FIRST INTEGER
C M  : LINEAR ARRAY, THE SECOND INTEGER
C MM : NUMBER OF PIECES IN THE SECOND INTEGER
C OUTPUT PARAMETERS
C N  : LINEAR ARRAY, THE SUM OF THE TWO INTEGERS
```

```
C NN : NUMBER OF PIECES IN THE SUM
C IND = 0 INITIALLY
C     = 1 WHEN THE SUM CONTAINS MORE THAN 80 DIGITS
      SUBROUTINE BIGADD (L, LL, M, MM, N, NN, IND)
      PARAMETER (ND = 20, NNINE=9999)
      DIMENSION L (LL), M (MM), N (ND)
      ICARY = 0
C     Compare the numbers of pieces in the 2 numbers
      IF (LL .GT. MM) THEN
         LIMIT = MM
C        Store the leftmost pieces of L or M in N corresponding to
C        which there are no pieces in the other addend
         DO 11 II = MM + 1, LL
            N (II) = L (II)
11       CONTINUE
         NN = LL
      ELSEIF (LL .EQ. MM) THEN
         LIMIT = MM
         NN = LIMIT
      ELSE
         LIMIT = LL
         DO 22 II = LL + 1, MM
            N (II) = M (II)
22       CONTINUE
         NN = MM
      ENDIF
C     Add the numbers piece by piece, and if any piece exceeds NNINE
C     take a CARRY to the next pair of pieces
      DO 33 I = 1, LIMIT
         KTEST = NNINE - M (I) - ICARY
         IF (L (I) .GT. KTEST) THEN
            N (I) = L (I) - KTEST - 1
            ICARY = 1
         ELSE
            N (I) = NNINE - (KTEST - L (I))
            ICARY = 0
         ENDIF
33    CONTINUE
      IF (ICARY .EQ. 1) THEN
C     Further calculation for ICARY=1
         IF (NN .NE. LIMIT) THEN
            DO 44 I = LIMIT + 1, NN
               IF (N (I) .EQ. NNINE) THEN
                  N (I) = 0
               ELSE
                  N (I) = N (I) + 1
```

```
                  RETURN
               ENDIF
44          CONTINUE
         ENDIF
         NN = NN + 1
C        IND=1 if the number of the pieces exceeds the specified limit
         IF (NN .GT. ND) THEN
            IND = 1
         ELSE
            N (NN) = 1
         ENDIF
      ENDIF
      RETURN
      END

C PURPOSE - GIVEN AN N-PIECE INTEGER THE SUBROUTINE
C              PRINTS THE INTEGER, IN ORDINARY POSITIONAL
C              NOTATION IN WHICH INTEGERS ARE EXPRESSED
C  INPUT PARAMETERS
C  NUMB : LINEAR ARRAY, THE INTEGER
C  J    : NUMBER OF PIECES IN THE INTEGER
      SUBROUTINE PRINT (NUMB, J, INDEX)
      INTEGER P1
      PARAMETER (ND = 20, P1 = 185)
      DIMENSION NUMB (ND)
      CHARACTER*1 FORM (P1)
      DATA (FORM(I), I = 1, 4) /'(', '1', 'X', ','/
      IF (INDEX .EQ. 1) THEN
         FORM (5) ='1'
         FORM (6) ='H'
         FORM (7) = '-'
         FORM (8) = ','
         FORM (9) = 'I'
         NP = 10
      ELSE
         FORM (5) = 'I'
         NP = 6
      ENDIF
      IF (NUMB (J) .LT. 10) THEN
         FORM (NP) = '1'
      ELSEIF (NUMB (J) .LT. 100) THEN
         FORM (NP) = '2'
      ELSEIF (NUMB (J) .LT. 1000) THEN
         FORM (NP) = '3'
      ELSE
         FORM (NP) = '4'
```

```
      ENDIF
      IF (J .GT. 1) THEN
         DO 44 K = 2, J
            NP = NP + 1
            FORM (NP) = ','
            IF (NUMB (J + 1 - K) .GE. 1000) THEN
               FORM (NP + 2) = '4'
            ELSE
               NP = NP + 2
               FORM (NP) = 'H'
               IF (NUMB (J + 1 - K) .LT. 10) THEN
                  FORM (NP - 1) = '3'
                  DO 11 I = NP + 1, NP + 3
                     FORM (I) = '0'
11                CONTINUE
                  NP = NP + 4
                  FORM (NP + 2) = '1'
               ELSEIF (NUMB (J + 1 - K) .LT. 100) THEN
                  FORM (NP - 1) = '2'
                  DO 22 I = NP + 1, NP + 2
                     FORM (I) = '0'
22                CONTINUE
                  NP = NP + 3
                  FORM (NP + 2) = '2'
               ELSE
                  FORM (NP - 1) = '1'
                  DO 33 I = NP + 1, NP + 1
                     FORM (I) = '0'
33                CONTINUE
                  NP = NP + 2
                  FORM (NP + 2) = '3'
               ENDIF
               FORM (NP) = ','
            ENDIF
            FORM (NP + 1) = 'I'
            NP = NP + 2
44       CONTINUE
      ENDIF
      FORM (NP + 1) = ','
      DO 55 I = NP + 2, P1 - 1
         FORM (I) = ' '
55    CONTINUE
      FORM (P1) = ')'
      WRITE (6, FORM) (NUMB (J + 1 - K), K = 1, J)
      RETURN
      END
```

```
THE NUMBERS OF PIECES ARE   1   1
THE INTEGERS ARE AS FOLLOWS
654
208
THE SUM IS
862
THE NUMBERS OF PIECES ARE   2   3
THE INTEGERS ARE AS FOLLOWS
4264827
27694278406
THE SUM IS
27698543233
THE NUMBERS OF PIECES ARE  10   8
THE INTEGERS ARE AS FOLLOWS
89657843452675436885000087638043049829O
8965784345188994612215464925064
THE SUM IS
896578443492538714038995488595895423354
THE NUMBERS OF PIECES ARE  10  10
THE INTEGERS ARE AS FOLLOWS
80100000400000500000200060000070000060
40002000020000004000100004000000800O0
THE SUM IS
12010200042000050400030006400007080060
THE NUMBERS OF PIECES ARE   5  15
THE INTEGERS ARE AS FOLLOWS
2657890657939270525
6848964532789654328965838007880870769628361097790150495O632
THE SUM IS
684896453278965432896583800788087076962862688685594442211S7
THE NUMBERS OF PIECES ARE  15  15
THE INTEGERS ARE AS FOLLOWS
89657843452675436885265789065793927052500080000000000000000
870769628361097790150495632684896453278965432896583800700SS
THE SUM IS
1767348062887852159003153523342835723803966232896583800700SS
THE NUMBERS OF PIECES ARE  21  20
FIRST INTEGER TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  20  21
SECOND INTEGER TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  21  21
BOTH INTEGERS TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   2   0
DATA ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   0   4
DATA ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  -1   2
DATA ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   5  -6
```

```
 DATA ILLEGAL, PLEASE TYPE AGAIN
 THE NUMBERS OF PIECES ARE   0   0
```

5.2 The program for subtracting a non-negative m-piece integer from a non-negative n-piece integer

```
      PARAMETER (ND = 20)
      DIMENSION L (ND), M (ND), N (ND)
10    WRITE (*,'('' TYPE THE NUMBERS OF PIECES IN THE MINUEND''
     1,1X,''AND THE SUBTRAHEND IN FORMAT (2I2)'')')
20    READ (4, '(2I2)') I, J
      WRITE (6,'('' THE NUMBERS OF PIECES ARE'',2I4)') I, J
      IF (I .EQ. 0 .AND. J .EQ. 0) STOP
      IF (I .LE. 0 .OR. J .LE. 0 .OR. I .GT. ND .OR. J .GT. ND) THEN
         IF (I .LE. 0 .OR. J .LE. 0) THEN
            WRITE (6, '('' DATA ILLEGAL, PLEASE TYPE AGAIN'')')
         ELSEIF (I .GT. ND .AND. J .GT. ND) THEN
            WRITE (6,'('' BOTH INTEGERS TOO LARGE,PLEASE TYPE AGAIN'')')
         ELSEIF (I .GT. ND) THEN
            WRITE (6, '('' MINUEND TOO LARGE, PLEASE TYPE AGAIN'')')
         ELSE
            WRITE (6, '('' SUBTRAHEND TOO LARGE, PLEASE TYPE AGAIN'')')
         ENDIF
         GO TO 20
      ENDIF
      WRITE (*, '('' TYPE THE MINUEND IN FORMAT (20I4)'')')
      READ (4, '(20I4)') (L (K), K = I, 1, -1)
      WRITE (*, '('' TYPE THE SUBTRAHEND IN FORMAT (20I4)'')')
      READ (4, '(20I4)') (M (K), K = J, 1, -1)
      WRITE (6, '('' THE MINUEND AND SUBTRAHEND ARE AS FOLLOWS'')')
      CALL PRINT (L, I, 0)
      CALL PRINT (M, J, 0)
      CALL BIGSUB (L, I, M, J, N, NN, INDEX)
      WRITE (6, '('' THE DIFFERENCE IS'')')
      CALL PRINT (N, NN, INDEX)
      GO TO 10
      END

C PURPOSE - THIS SUBROUTINE SUBTRACTS ONE LARGE NON-NEGATIVE
C           INTEGER FROM ANOTHER
C INPUT PARAMETERS
C L  : LINEAR ARRAY, THE MINUEND
C LL : NUMBER OF PIECES IN THE MINUEND
C M  : LINEAR ARRAY, THE SUBTRAHEND
C MM : NUMBER OF PIECES IN THE SUBTRAHEND
C OUTPUT PARAMETERS
C N  : LINEAR ARRAY, THE REMAINDER
```

```
C NN : NUMBER OF PIECES IN THE REMAINDER
C INDEX = 0 IF THE MINUEND IS GREATER THAN OR EQUAL TO THE SUBTRAHEND
C       = 1 IF THE SUBTRAHEND IS GREATER THAN THE MINUEND
      SUBROUTINE BIGSUB (L, LL, M, MM, N, NN, INDEX)
      PARAMETER (ND = 20, NNINE=9999, NTHOUS=1000)
      DIMENSION L (LL), N1 (ND), M (MM), N (ND)
      IBOROW = 0
C     Compare the numbers of pieces in the 2 numbers
      IF (LL .GT. MM) THEN
         INDEX = 0
      ELSEIF (LL .LT. MM) THEN
         INDEX = 1
      ELSE
C        For equal number of pieces compare pieces starting from left
         DO 11 I = MM, 1, -1
            IF (L (I) .NE. M (I)) THEN
               IF (L (I) .GT. M (I)) THEN
                  INDEX = 0
               ELSE
                  INDEX = 1
               ENDIF
               GO TO 20
            ENDIF
11       CONTINUE
         INDEX = 0
         NN = 1
         N (1) = 0
         RETURN
      ENDIF
C     For SUBTRAHEND > MINUEND, put INDEX=1, otherwise INDEX=0
20    IF (INDEX .EQ. 0) THEN
         DO 33 I = 1, MM
            N1 (I) = M (I)
33       CONTINUE
         DO 44 I = 1, LL
            N (I) = L (I)
44       CONTINUE
         NN = LL
         KK = MM
      ELSE
         DO 55 I = 1, MM
            N (I) = M (I)
55       CONTINUE
         DO 66 I = 1, LL
            N1 (I) = L (I)
66       CONTINUE
```

```
         KK = LL
         NN = MM
      ENDIF
C     Subtract the corresponding pieces, taking care of borrowed
C     figures from the previous piece
      DO 77 I = 1, KK
         IDIF = N (I) - N1 (I)
         IF (IDIF .GT. 0) THEN
            N (I) = IDIF - IBOROW
            IBOROW = 0
         ELSEIF (IDIF .EQ. 0 .AND. IBOROW .EQ. 0) THEN
            N (I) = 0
         ELSE
            N (I) = IDIF + 1 - IBOROW + NNINE
            IBOROW = 1
         ENDIF
77    CONTINUE
      IF (KK .NE. NN) THEN
         IF (IBOROW .EQ. 0) RETURN
         IF (KK + 1 .LT. NN) THEN
            DO 88 I = KK + 1, NN - 1
               IF (N (I) .NE. 0) THEN
                  N (I) = N (I) - 1
                  RETURN
               ENDIF
               N(I) = NNINE
88          CONTINUE
         ENDIF
         N (NN) = N (NN) - 1
      ENDIF
      IF (NN .GT. 1) THEN
         MN1 = NN - 1
C        Test whether leftmost pieces are zero
         DO 99 I = 1, MN1
            IF (N (NN) .NE. 0) RETURN
            NN = NN - 1
99       CONTINUE
      ENDIF
      RETURN
      END
 THE NUMBERS OF PIECES ARE   1   1
 THE MINUEND AND SUBTRAHEND ARE AS FOLLOWS
 6549
 8630
 THE DIFFERENCE IS
 -2081
```

```
THE NUMBERS OF PIECES ARE   2   3
THE MINUEND AND SUBTRAHEND ARE AS FOLLOWS
42648278
276985432345
THE DIFFERENCE IS
-276942784067
THE NUMBERS OF PIECES ARE  10   8
THE MINUEND AND SUBTRAHEND ARE AS FOLLOWS
896578434526754368850000876380430498290l
78549076284543838739340230320
THE DIFFERENCE IS
89657843451889946122154649250649647525 81
THE NUMBERS OF PIECES ARE  10  10
THE MINUEND AND SUBTRAHEND ARE AS FOLLOWS
800965784345263456000087638043049876982 9
34700854907628963894543838983082773934 02
THE DIFFERENCE IS
45395723526897381705464924821222213764 27
THE NUMBERS OF PIECES ARE   5  15
THE MINUEND AND SUBTRAHEND ARE AS FOLLOWS
26578906579392705258
6848964532789654328965838007880870769654940004369543200890
THE DIFFERENCE IS
-6848964532789654328965838007880870769628361097790150495632
THE NUMBERS OF PIECES ARE  15  15
THE MINUEND AND SUBTRAHEND ARE AS FOLLOWS
900000000000200000000000000050000000000000020000000000000
900000000000200000000000000040000000000000010000000000000
THE DIFFERENCE IS
100000000000000100000000000000
THE NUMBERS OF PIECES ARE  21  20
MINUEND TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  20  21
SUBTRAHEND TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  21  21
BOTH INTEGERS TOO LARGE,PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   2   0
DATA ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   0   4
DATA ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  -1   2
DATA ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   5  -6
DATA ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   0   0
```

5.3 The program for multiplcation of two non-negative m, n-piece integers

```
      PARAMETER (ND = 20)
      DIMENSION L (ND), M (ND), N (ND)
10    WRITE (*,'('' TYPE NUMBERS OF PIECES OF THE MULTIPLICAND''
     1,1X,''AND THE MULTIPLIER IN FORMAT (2I2)'')')
20    READ (4, '(2I2)') I, J
      WRITE (6,'('' THE NUMBERS OF PIECES ARE'',2I4)') I, J
      IF (I .EQ. 0 .AND. J .EQ. 0) STOP
      IF (I .LE. 0 .OR. J .LE. 0 .OR. I .GT. ND .OR. J .GT. ND) THEN
         IF (I .LE. 0 .OR. J .LE. 0) THEN
            WRITE (6, '('' DATA  ILLEGAL, PLEASE TYPE AGAIN'')')
         ELSEIF (I .GT. ND .AND. J .GT. ND) THEN
            WRITE (6,'('' BOTH INTEGERS TOO LARGE,PLEASE TYPE AGAIN'')')
         ELSEIF (I .LE. ND) THEN
            WRITE (6, '('' MULTIPLICAND TOO LARGE, PLEASE TYPE AGAIN'')'
         ELSE
            WRITE (6, '('' MULTIPLIER TOO LARGE, PLEASE TYPE AGAIN'')')
         ENDIF
         GO TO 20
      ENDIF
      WRITE (*, '('' TYPE THE MULTIPLICAND IN FORMAT (20I4)'')')
      READ (4, '(20I4)') (L (K), K = I, 1, -1)
      WRITE (*, '('' TYPE THE MULTIPLIER IN FORMAT (20I4)'')')
      READ (4, '(20I4)') (M (K), K = J, 1, -1)
      IND = 0
      WRITE (6,'('' MULTIPLICAND AND MULTIPLIER ARE AS FOLLOWS'')')
      CALL PRINT (L, I, 0)
      CALL PRINT (M, J, 0)
      CALL BIGMUL (L, I, M, J, N, NN, IND)
      IF (IND .EQ. 0) THEN
         WRITE (6, '('' THE PRODUCT IS'')')
         CALL PRINT (N, NN, 0)
      ELSE
         WRITE (6, '('' THE PRODUCT HAS MORE THAN 80 DIGITS'')')
      ENDIF
      GO TO 10
      END

C PURPOSE - MULTIPLICATION OF TWO LARGE INTEGERS
C INPUT PARAMETERS
C A : LINEAR ARRAY, THE MULTIPLICAND
C I : NUMBER OF PIECES IN THE MULTIPLICAND
C B : LINEAR ARRAY, THE MULTIPLIER
C J : NUMBER OF PIECES OF IN THE MULTIPLIER
C OUTPUT PARAMETERS
```

```
C C : LINEAR ARRAY, THE PRODUCT
C M : NUMBER OF PIECES OF THE PRODUCT
      SUBROUTINE BIGMUL (A, I, B, J, C, M, IND)
      INTEGER P1
      PARAMETER (ND = 20,  P1 = 9, NDIG=4)
      INTEGER A (I), IPROD (ND), B (J), C (ND), D (80),
     1INP (P1), ISP (P1, ND), C1 (ND)
      IS = - 1
      C (1) = 0
      M = 1
      DO 11 K = 2, 9
         INP (K) = 0
11    CONTINUE
C     Digitise the multiplier
      CALL INTEG (B, J, D, IDN)
       DO 66 MM = 1, IDN
          IS = IS + 1
C         Special treatment for the digits 0 and 1
          IF (D (MM) .NE. 0) THEN
             IF (D (MM) .GT. 1) THEN
C               Multiply by a particular digit only once
                IF (INP (D (MM)) .EQ. 0) THEN
                   CALL MULT (A, I, D (MM), IPROD, N1, IND)
                   IF (IND .EQ. 1) RETURN
C                  Make a table of 8 multipliers of the multiplicand
                   DO 22 K = 1, N1
                      ISP (D (MM), K) = IPROD (K)
22                 CONTINUE
                   INP (D (MM)) = N1
C                  If the multiplicand has already been multiplied by a
C                  particular digit do not multiply it again
                ELSE
                   N1 = INP (D (MM))
                   DO 33 K = 1, N1
                      IPROD (K) = ISP (D (MM), K)
33                 CONTINUE
                ENDIF
             ELSE
C               For the digit 1 the multiplicand is the product
                DO 44 K = 1, I
                   IPROD (K) = A (K)
44              CONTINUE
                N1 = I
             ENDIF
C            Decide whether shifting is required
```

```
            IF (N1 .NE. 1 .OR. IPROD (1) .NE. 0) THEN
               CALL SHIFT (IPROD, N1, IS, IND)
               IF (IND .EQ. 1) RETURN
               CALL BIGADD (C, M, IPROD, N1, C1, M1, IND)
               IF (IND .EQ. 1) RETURN
               M = M1
C              Transfer the product into the original array
               DO 55 MK = 1, M
                  C (MK) = C1 (MK)
55             CONTINUE
            ENDIF
         ENDIF
66    CONTINUE
      RETURN
      END

C PURPOSE - TO MULTIPLY A NUMBER BY A SINGLE DIGIT
C INPUT PARAMETERS
C J   : LINEAR ARRAY, THE MULTIPLICAND
C N   : NUMBER OF PIECES IN THE MULTIPLICAND
C K   : THE DIGIT WITH WHICH IT IS TO BE MULTIPLIED
C OUTPUT PARAMETERS
C IPROD  : LINEAR ARRAY, PRODUCT OF MULTIPLICAND WITH
C          A SINGLE DIGIT
C N1     : NUMBER OF PIECES IN THE PRODUCT
      SUBROUTINE MULT (J, N, K, IPROD, N1, IND)
      PARAMETER (ND = 20, NNINE=9999, NTHOUS=1000)
      DIMENSION J (ND), IPROD (ND)
      IPROD (1) = 0
C     Special treatment for the digits 0 and 1
      IF (K .EQ. 0) THEN
         N1 = 1
      ELSEIF (K .EQ. 1) THEN
         DO 10 I = 1, N
            IPROD (I) = J (I)
10       CONTINUE
         N1 = N
      ELSE
C        Multiply by the digit
         DO 22 I = 1, N
            IQ = J (I) / NTHOUS
            M = (J (I) - NTHOUS * IQ) * K + IPROD (I)
            L = IQ * K
            IQ = L / 10
            ITEST = M - (NNINE - NTHOUS * (L - 10 * IQ))
            IF (ITEST .GT. 0)  THEN
               IPROD (I) = ITEST - 1
```

```
                  IQ = IQ + 1
               ELSE
                  IPROD (I) = ITEST + NNINE
               ENDIF
               IF (I .NE. N) THEN
                  IPROD (I + 1) = IQ
               ENDIF
22          CONTINUE
            N1 = N
            IF (IQ .NE. 0) THEN
               N1 = N1 + 1
C              IND=1 when number of pieces crosses specified limit
               IF (N1 .GT. ND) THEN
                  IND = 1
               ELSE
                  IPROD (N1) = IQ
               ENDIF
            ENDIF
         ENDIF
         RETURN
         END

C  PURPOSE - THIS ROUTINE SHIFTS AN N-PIECE POSITIVE
C             INTEGER BY A FIXED NUMBER OF POSITIONS TO THE LEFT
C  INPUT PARAMETERS
C  IPR  : LINEAR ARRAY, THE INTEGER BEFORE SHIFTING
C  IAS  : NUMBER OF PIECES IN THE INTEGER BEFORE SHIFTING
C  MP   : THE NUMBER POSTIONS BY WHICH SHIFTING IS REQUIRED
C  OUTPUT PARAMETERS
C  IPR  : LINEAR ARRAY, THE INTEGER AFTER SHIFTING
C  IAS   : NUMBER OF PIECES IN THE INTEGER AFTER SHIFTING
         SUBROUTINE SHIFT (IPR, IAS, MP, IND)
         PARAMETER (ND = 20, NDIG=4)
         DIMENSION IPR (ND)
C        Check whether shift is required
         IF (MP .EQ. 0) RETURN
         IP = MP
         INTR = 0
C        Test whether shift is possible within the specified limit
         IF (IP .GE. NDIG) THEN
            ID = IP / NDIG
            IF (IAS + ID .GT. ND) THEN
               IND = 1
               RETURN
            ENDIF
C           Shift each piece by the required number of pieces
            DO 11 I = IAS, 1, -1
```

```
            IPR (I + ID) = IPR (I)
11          CONTINUE
            IAS = IAS + ID
            DO 22 I = 1, ID
               IPR (I) = 0
22          CONTINUE
C           Determine the number of positions by which the digits
C           are to be shifted
            IP = IP - NDIG * ID
            IF (IP .EQ. 0) RETURN
            KK = ID + 1
         ELSE
            KK = 1
         ENDIF
         K = 10 ** (NDIG- IP)
C        Shift required number of digits
         DO 44 I = KK, IAS
            IHIN = IPR (I) / K
            IPR (I) = (IPR (I) - IHIN * K) * 10 ** IP + INTR
            INTR = IHIN
44       CONTINUE
C        The number of pieces are to be increased
         IF (INTR .NE. 0) THEN
            IAS = IAS + 1
            IPR (IAS) = INTR
         ENDIF
         RETURN
         END

C  PURPOSE - THE SUBROUTINE INTEG DIGITISES A LARGE INTEGER
C            INTO ITS DIGITS
C  INPUT PARAMETERS
C  B   : LINEAR ARRAY, THE INTEGER
C  J   : NUMBER OF PIECES IN THE GIVEN INTEGER
C  OUTPUT PARAMETERS
C  D   : LINEAR ARRAY, THE DIGITS
C  IDN: THE NUMBER OF DIGITS
         SUBROUTINE INTEG (B, J, D, IDN)
         PARAMETER (NDIG=4)
         INTEGER B (J), D (80)
C        Set the initial values
         IDN = 0
         NG = NDIG
         DO 44 II = 1, J
            IDIV = B (II)
            IF (II .EQ. J) THEN
C              Find the number of digits in the leftmost piece
```

```
            DO 11 JJ = 1, NDIG - 1
               IF (IDIV .LT. 10 ** JJ) THEN
                  NG = JJ
                  GO TO 20
               ENDIF
11          CONTINUE
         ENDIF
C        Find the digits of each piece
20       DO 33 KK = 1, NG
            IDN = IDN + 1
            D (IDN) = MOD (IDIV, 10)
            IF (KK .NE. NG) THEN
               IDIV = IDIV / 10
            ENDIF
33       CONTINUE
44    CONTINUE
      RETURN
      END
```

```
 THE NUMBERS OF PIECES ARE   1   1
 MULTIPLICAND AND MULTIPLIER ARE AS FOLLOWS
 6549
 8630
 THE PRODUCT IS
 56517870
 THE NUMBERS OF PIECES ARE   2   1
 MULTIPLICAND AND MULTIPLIER ARE AS FOLLOWS
 67894521
 2000
 THE PRODUCT IS
 135789042000
 THE NUMBERS OF PIECES ARE   2   3
 MULTIPLICAND AND MULTIPLIER ARE AS FOLLOWS
 42648278
 276985432345
 THE PRODUCT IS
 11812951720599751910
 THE NUMBERS OF PIECES ARE   5   5
 MULTIPLICAND AND MULTIPLIER ARE AS FOLLOWS
 690804000012000044
 78900280000560078
 THE PRODUCT IS
 54504629026453710954331040960643432
 THE NUMBERS OF PIECES ARE  10   8
 MULTIPLICAND AND MULTIPLIER ARE AS FOLLOWS
 89657843452675436885000087638043049829O1
 78549076284543838739340230320
```

```
THE PRODUCT IS
70425407848718922443281659517675551059184058498626823950354900175832O
THE NUMBERS OF PIECES ARE  15  15
MULTIPLICAND AND MULTIPLIER ARE AS FOLLOWS
900000000000200000000000000050000000000000020000000000000
900000000000200000000000000040000000000000010000000000000
THE PRODUCT HAS MORE THAN 80 DIGITS
THE NUMBERS OF PIECES ARE  21  20
MULTIPLIER TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  20  21
MULTIPLICAND TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  21  21
BOTH INTEGERS TOO LARGE,PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   2   0
DATA  ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   0   4
DATA  ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  -1   2
DATA  ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   5  -6
DATA  ILLEGAL, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   0   0
```

5.4 Program for division of an m-piece non-negative integer by an n-piece positive integer

```
      PARAMETER (ND = 20, NDIG=4, NTHOUS=1000)
      INTEGER A (ND), A1 (ND), B (ND), IGIT (80), IQT (ND)
10    WRITE (*,'('' TYPE THE NUMBERS OF PIECES OF DIVIDEND''
     1,1X,'' DIVISOR IN FORMAT (2I2)'')')
20    READ (4, '(2I2)') I, J
      WRITE (6,'('' THE NUMBERS OF PIECES ARE'', 2I4)') I, J
      IF (I .EQ. 0 .AND. J .EQ. 0) STOP
      IF (I .LE. 0 .OR. J .LE. 0 .OR. I .GT. ND .OR. J .GT. ND) THEN
         IF (I .LE. 0 .OR. J .LE. 0) THEN
            WRITE (6, '('' DATA  ILLEGAL, PLEASE TYPE AGAIN'')')
         ELSEIF (I .GT. ND .AND. J .GT. ND) THEN
            WRITE (6,'('' BOTH NUMBERS TOO LARGE, PLEASE TYPE AGAIN'
         ELSEIF (I .GT. ND) THEN
            WRITE (6, '('' DIVIDEND TOO LARGE, PLEASE TYPE AGAIN'')'
         ELSE
            WRITE (6, '('' DIVISOR TOO LARGE, PLEASE TYPE AGAIN'')')
         ENDIF
         GO TO 20
      ENDIF
      WRITE (*, '('' TYPE THE DIVIDEND IN FORMAT (20I4)'')')
```

```
      READ (4, '(20I4)') (A (K), K = I, 1, -1)
      WRITE (*, '('' TYPE THE DIVISOR IN FORMAT (20I4)'')')
      READ (4, '(20I4)') (B (LL), LL = J, 1, -1)
      IF (J .EQ. 1 .AND. B (1) .EQ. 0) THEN
         WRITE (6, '('' DIVISION BY ZERO IS NOT DEFINED'')')
         GO TO 10
      ENDIF
      CALL BIGDIV (A, I, B, J, IQT, M, A1, MT)
      WRITE (6, '('' THE DIVIDEND AND DIVISOR ARE AS FOLLOWS'')')
      CALL PRINT (A, I, 0)
      CALL PRINT (B, J, 0)
      WRITE (6, '('' THE QUOTIENT IS'')')
      CALL PRINT (IQT, M, 0)
      WRITE (6, '('' THE REMAINDER IS'')')
      CALL PRINT (A1, MT, 0)
      GO TO 10
      END

C  PURPOSE - PROGRAM FOR DIVISION OF A M-PIECE NON-NEGATIVE
C            INTEGER BY AN N-PIECE NON-NEGATIVE INTEGER
C  INPUT PARAMETERS
C  A   : LINEAR ARRAY, THE DIVIDEND
C  I   : THE NUMBER PIECES OF THE DIVIDEND
C  B   : LINEAR ARRAY, THE DIVISOR
C  J   : THE NUMBER OF PIECES IN THE DIVISOR
C  OUTPUT PARAMETERS
C  IQT : LINEAR ARRAY, THE QUOTIENT
C  M   : NUMBER OF PIECES IN THE QUOTIENT
C  A1  : LINEAR ARRAY, THE REMAINDER
C  MT  : NUMBER OF PIECES IN THE REMAINDER
      SUBROUTINE BIGDIV (A, I, B, J, IQT, M, A1, MT)
      PARAMETER (ND = 20, NDIG=4, NTHOUS=1000)
      INTEGER A (I), A1 (I), B (J), IGIT (80), IQT (ND)
      IND = 0
      M = 1
      IQT (1) = 0
      IF (I .LT. J) THEN
         MT = I
C        The remainder, when I < J
         DO 33 II = I, 1, -1
            A1 (II) = A (II)
33       CONTINUE
      ELSE
         IF (I .GT. J) THEN
            LM = I - J
C           Digitise the rightmost LM pieces of the dividend
            CALL INTEG (A, LM, IGIT, NDI)
            IC = NDIG * LM
```

```
            IF (NDI .NE. IC) THEN
               NDI = NDI + 1
               DO 44 II= NDI, IC
                  IGIT (II) = 0
44             CONTINUE
               NDI = IC
            ENDIF
         ELSE
            NDI = 0
         ENDIF
         MT = J
C        Form partial dividend with left most J pieces of dividend
         IM = I - J
         DO 55 II = J, 1, -1
            A1 (II) = A (II + IM)
55       CONTINUE
C        Find the number of digits in the leftmost pieces of
C        the dividend and the divisor
         IDEND = A1 (J)
         ISOR = B (J)
         DO 66 I1 = 1, NDIG - 1
            IF (IDEND .LT. 10 ** I1) GO TO 70
66       CONTINUE
         I1 = NDIG
70       DO 88 J1 = 1, NDIG - 1
            IF (ISOR .LT. 10 ** J1) GO TO 90
88       CONTINUE
         J1 = NDIG
90       DO 111 II = J, 1, -1
            IF (A1 (II) .GT. B (II)) GO TO 120
            IF (A1 (II) .LT. B (II)) GO TO 150
111      CONTINUE
120      IF (J1 .LT. NDIG .AND. J .NE. 1) THEN
C           Form the first partial dividend with required number of digit
            CALL PARDEF (A1, B, J, I1, J1, IGIT, NDI, ISOR, IDEND)
         ENDIF
         IF (IDEND .LT. ISOR) THEN
            NISOR = ISOR / 10
         ELSE
            NISOR = ISOR
         ENDIF
130      IW = IDEND / NISOR
C        Find the actual digits of the quotient
         CALL EXACDQ (A1, MT, B, J, IW, IND)
C        Form the quotient by appending digit(s) to the right
140      CALL APPEND (IQT, M, IW, IND)
```

```
C        Form the partial dividend by appending digit(s) to the right
150      IF (NDI .GT. 0) THEN
            CALL APPEND (A1, MT, IGIT (NDI), IND)
            NDI = NDI - 1
            IF (MT .GT. J) THEN
               IDEND = A1 (MT) * NTHOUS + A1 (MT - 1) / 10
               NISOR = ISOR / 10
               GO TO 130
            ELSE
               IF (IQT (M) .EQ. 0) THEN
                  I1 = I1 + 1
                  GO TO 90
               ENDIF
               IF (MT .LT. J) THEN
                  IW = 0
               ELSE
                  DO 166 II = J, 1, -1
                     IF (A1 (II) .NE. B (II)) THEN
                        IF (A1 (II) .LT. B (II)) THEN
                           IW = 0
                           GO TO 140
                        ELSE
                           IDEND = A1 (J)
                           NISOR = ISOR
                           IF (J1 .LT. NDIG .AND. J .NE. 1) THEN
C                             Initialize IDEND with right most 4 digits of
C                             the partial dividend and ISOR with right mos
C                             3 digits of the divisor
                              IF (IDEND .GE. 10 ** J1) THEN
                                 IDEND = A1 (J) * 10 ** (NDIG - (J1+1))
     1                           + (A1 (J - 1)/10 ** (J1 + 1))
                                 NISOR = ISOR/10
                              ELSE
                                 IDEND = A1 (J) * 10 ** (NDIG - J1) +
     1                           (A1 (J - 1)/10 ** J1)
                              ENDIF
                           ENDIF
                        ENDIF
                        GO TO 130
                     ENDIF
166               CONTINUE
                  MT = 1
                  A1 (1) = 0
                  IW = 1
               ENDIF
            ENDIF
```

```
            GO TO 140
         ENDIF
      ENDIF
      RETURN
      END

C  PURPOSE - TO OBTAIN THE FIRST PARTIAL DIVIDEND WITH
C            THE REQUIRED NUMBER OF DIGITS
C  INPUT PARAMETERS
C  A1   : LINEAR ARRAY, THE FIRST TRIAL DIVIDEND
C  B    : LINEAR ARRAY, THE DIVISOR
C  J    : THE NUMBER OF PIECES IN THE DIVISOR
C  I1   : THE NUMBER OF DIGITS IN THE LEFTMOST PIECE OF
C         THE DIVIDEND
C  J1   : THE NUMBER OF DIGITS IN THE LEFTMOST PIECE OF
C         THE DIVISOR
C  IGIT: LINEAR ARRAY, THE REMAINING DIGITS OF THE DIVIDEND
C  NDI  : THE NUMBER OF THE REMAINING DIGITS
C  IDEND: THE LEFTMOST PIECE OF THE DIVIDEND
C  ISOR: THE LEFTMOST PIECE OF THE DIVISOR
C  OUTPUT PARAMETERS
C  A1   : LINEAR ARRAY, THE FIRST PARTIAL DIVIDEND
C  IGIT: LINEAR ARRAY, REMAINING DIGITS OF THE DIVIDEND
C  NDI  : THE NUMBER OF THE REMAINING DIGITS
C  IDEND: THE LEFTMOST NDIG DIGITS NUMBER OF THE DIVIDEND
C  ISOR: THE LEFTMOST NDIG DIGITS NUMBER OF THE DIVISOR
      SUBROUTINE PARDEF (A1, B, J, I1, J1, IGIT, NDI, ISOR, IDEND)
      PARAMETER (ND = 20, NDIG=4)
      INTEGER  A1 (ND), IGIT (80),  B (ND)
C     Set IDEND and ISOR equal to the leftmost NDIG digits of the
C     dividend and divisor
      IF (I1 .NE. NDIG) THEN
         IDEND = A1 (J) * 10 ** (NDIG - I1) + (A1 (J - 1)/10
     1   ** I1)
      ENDIF
      ISOR = B (J) * 10 ** (NDIG - J1) + (B (J - 1)/10
     1** J1)
      IF (IDEND .LT. ISOR) THEN
         ICUT = I1 - (J1 + 1)
      ELSE
         ICUT = I1 - J1
      ENDIF
      IF (ICUT .EQ. 0) RETURN
C     Digitise the excess digits of rightmost piece of the
C     first partial dividend
      MICUT = NDIG - ICUT
      ITEM = MOD (A1 (1), 10 ** ICUT)
      DO 11 I = 1, ICUT
         IGIT (NDI + I) = MOD (ITEM, 10)
```

```
         ITEM = ITEM / 10
11     CONTINUE
       NDI = NDI + ICUT
C      Form partial dividend with left most required number of digits
       DO 22 II = 1, J - 1
          ITEMP = MOD (A1 (II + 1), 10 ** ICUT)
          A1 (II) = ITEMP * 10 ** (NDIG - ICUT) + A1 (II) / 10 ** ICUT
22     CONTINUE
       A1 (J) = A1 (J) / 10 ** ICUT
       RETURN
       END

C  PURPOSE - THIS SUBROUTINE FORMRS NEW NUMBER BY
C            APPENDING DIGIT(S) TO THE RIGHT
C  INPUT PARAMETERS
C  A1   : LINEAR ARRAY, EXISTING INTEGER
C  MT   : NUMBER OF PIECES IN THE EXISTING INTEGER
C  OUTPUT PARAMETERS
C  A1   : LINEAR ARRAY, THE INTEGER AFTER APPENDING
C  MT   : NUMBER OF PIECES IN THE INTEGER AFTER APPENDING
       SUBROUTINE APPEND (A1, MT, IW, IND)
       PARAMETER (ND = 20)
       INTEGER A1 (ND)
       IF (A1 (MT) .NE. 0) THEN
          CALL SHIFT (A1, MT, 1, IND)
          A1 (1) = A1 (1) + IW
       ELSE
          IF (IW .NE. 0) THEN
             MT = 1
             A1 (1) = IW
          ENDIF
       ENDIF
       RETURN
       END

C  PURPOSE - TO FIND THE EXACT DIGIT(S) OF THE QUOTIENT AND
C            REMAINING PARTIAL DIVIDEND
C  INPUT PARAMETERS
C  B  : LINEAR ARRAY, THE DIVISOR
C  J  : NUMBER OF PIECES IN THE DIVISOR
C  A1 : LINEAR ARRAY, THE PARTIAL DIVIDEND
C  MT : NUMBER OF PIECES IN PARITAL DIVIDEND
C  IW : THE TRIAL DIGIT(S) OF  QUOTIENT
C  OUTPUT PARAMETERS
C  A1 : LINEAR ARRAY, REMAINING PARTIAL DIVIDEND
C  MT : NUMBER OF PIECES IN REMAINING PARITAL DIVIDEND
C  IW : THE REQUIRED DIGIT(S) OF THE QUOTIENT
       SUBROUTINE EXACDQ (A1, MT, B, J, IW, IND)
```

```
      PARAMETER (ND = 20)
      INTEGER B (J), A1 (MT), D1 (ND), C (1), C1 (ND)
10    C (1) = IW
C     Multiply the divisor by the trial digit(s) of the quotient
      CALL BIGMUL (B, J, C, 1, D1, NN, IND)
      IF (IND .EQ. 1) THEN
         IND = 0
      ELSE
C        Compare the product of trial digit(s) and divisor with
C        partial dividend
         IF (NN .LE. MT) THEN
            IF (NN .EQ. MT) THEN
               DO 22 KK = NN, 1, -1
                  IF (D1 (KK) .LT. A1 (KK)) GO TO 30
                  IF (D1 (KK) .GT. A1 (KK)) GO TO 50
22             CONTINUE
               A1 (1) = 0
               MT = 1
               RETURN
            ENDIF
30          CALL BIGSUB (A1, MT, D1, NN, C1, MM, INDEX)
            DO 44 II = 1, MM
               A1 (II) = C1 (II)
44          CONTINUE
            MT = MM
            RETURN
         ENDIF
      ENDIF
C     Reduce the trial digit(s) of quotient by unity, if required
50    IW = IW - 1
      GO TO 10
      END
```

```
 THE NUMBERS OF PIECES ARE   2   1
 THE DIVIDEND AND DIVISOR ARE AS FOLLOWS
 67894521
 2000
 THE QUOTIENT IS
 33947
 THE REMAINDER IS
 521
 THE NUMBERS OF PIECES ARE   2   2
 THE DIVIDEND AND DIVISOR ARE AS FOLLOWS
 65490000
 86300000
 THE QUOTIENT IS
 0
```

```
THE REMAINDER IS
65490000
THE NUMBERS OF PIECES ARE   3   1
THE DIVIDEND AND DIVISOR ARE AS FOLLOWS
199900000000
19
THE QUOTIENT IS
10521052631
THE REMAINDER IS
11
THE NUMBERS OF PIECES ARE   5   5
THE DIVIDEND AND DIVISOR ARE AS FOLLOWS
690804000012000044
78900280000560078
THE QUOTIENT IS
8
THE REMAINDER IS
59601760007519420
THE NUMBERS OF PIECES ARE  10   2
THE DIVIDEND AND DIVISOR ARE AS FOLLOWS
8965784345267543688500008763804304982901
78549
THE QUOTIENT IS
1141425650901672037645292589823461l4
THE REMAINDER IS
74315
THE NUMBERS OF PIECES ARE  15   1
THE DIVIDEND AND DIVISOR ARE AS FOLLOWS
6848964532789654328965838007880870769654940004369543200890
2657
THE QUOTIENT IS
257770588362425830973497855019980081658070756656738547 2
THE REMAINDER IS
1786
THE NUMBERS OF PIECES ARE  15  15
THE DIVIDEND AND DIVISOR ARE AS FOLLOWS
900000000000200000000000000050000000000000020000000000000
900000000000200000000000000040000000000000010000000000000
THE QUOTIENT IS
1
THE REMAINDER IS
100000000000000100000000000000
THE NUMBERS OF PIECES ARE  21  20
DIVIDEND TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  20  21
DIVISOR TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE  21  21
BOTH NUMBERS TOO LARGE, PLEASE TYPE AGAIN
THE NUMBERS OF PIECES ARE   2   0
```

```
 DATA  ILLEGAL, PLEASE TYPE AGAIN
 THE NUMBERS OF PIECES ARE   0   4
 DATA  ILLEGAL, PLEASE TYPE AGAIN
 THE NUMBERS OF PIECES ARE  -1   2
 DATA  ILLEGAL, PLEASE TYPE AGAIN
 THE NUMBERS OF PIECES ARE   5  -6
 DATA  ILLEGAL, PLEASE TYPE AGAIN
 THE NUMBERS OF PIECES ARE   0   0
```

5.5 Given a positive integer this program computes the exact integer part of its square root

```
      IMPLICIT INTEGER*4 (A - Z)
10    READ (4, '(I12)') NUMBER
      IF (NUMBER .EQ. 0) STOP
      CALL SQRUT (NUMBER, ISQRT, IGUESV, IORIGU, NOP)
      NS = ISQRT * ISQRT
      NSP1 = (ISQRT + 1) * (ISQRT + 1)
      WRITE (6,'('' THE NUMBER'',I10,'' ROOT'',I6,'' GUES'',I7,
     1'' ORIGUE'',I7)') NUMBER, ISQRT, IGUESV, IORIGU
      WRITE (6,'('' ITERATIONS'',I2,'' SQ OF ROOT'',I10,
     1'' SQ OF ROOT+1'',I11)') NOP, NS, NSP1
      WRITE (6, *)
      GO TO 10
      END
```

When the subroutine SQRUT is called from the main program (the preceding one), it has the following arguments:- NUMBER, ISQRT, IGUESV, IORGU, NOP i.e. it is of the form
SUBROUTINE SQRUT (NUMBER, ISQRT, IGUESV, IORGU, NOP)
instead of
SUBROUTINE SQRUT (NUMBER, ISQRT)

```
C PURPOSE - GIVEN A POSITIVE INTEGER THE PROGRAM FIRST COMPUTE
C           THE CLOSE INTEGER INITIAL ESTIMATE OF THE SQUARE
C           ROOT AND THEN THE EXACT INTEGER PART OF THE
C           SQUARE ROOT USING VARIANT OF NEWTON-RAPHSON METHOD

      SUBROUTINE SQRUT (NUMBER, ISQRT)
      IMPLICIT INTEGER*4 (A - Z)
      CALL SIMPGV (NUMBER, IGUESV, NB)
      IORGU = IGUESV
C     Improve the guess value for odd bits number
      IF (NB / 2 * 2 .NE. NB .AND. NB .GT. 2) THEN
         IGUESV =  (IGUESV + 1) * 71 / 100
      ENDIF
      CALL TONRAP (IGUESV, NUMBER, ISQRT, NOP)
```

```
      RETURN
      END

C  PURPOSE - TO COMPUTE THE INITIAL ESTIMATE OF THE SQUARE ROOT
C  INPUT PARAMETER
C  NUMBER : NUMBER SQUARE ROOT OF WHICH IS TO BE EVALUATED
C  OUTPUT PARAMETER
C  IGUESV : THE FIRST APPROXIMATION OF THE SQUARE ROOT
C  NB     : THE NUMBER OF BITS OF THE GIVEN NUMBER
      SUBROUTINE SIMPGV (NUMBER, IGUESV, NB)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 30)
      DIMENSION LTAB (ND)
      KK = NUMBER
      IDETER = 2
      NB = 1
      LTAB (1) = 2
C     Determine the number of bits
10    IF (NUMBER .GE. IDETER) THEN
         IDETER = IDETER + IDETER
         NB = NB + 1
         LTAB (NB) = IDETER
         GO TO 10
      ENDIF
      IF (NB .GT. 2) THEN
         IHALF = NB / 2
C        For odd number of bits
         IF (NB .NE. 2 * IHALF) THEN
            IHALF = IHALF + 1
            KK = 2 * NUMBER
         ENDIF
         IGUESV = KK / LTAB (IHALF + 1)
         IGUESV = IGUESV +  LTAB (IHALF - 1)
      ELSE
C        The nuber of bits is less than equal to 2
         IGUESV = 1
      ENDIF
      RETURN
      END

C PURPOSE - TO EVALUATE THE SQUARE ROOT BY NEWTON-RAPHSON METHOD
C  INPUT PARAMETERS
C  IGUESV : THE INITIAL ESTIMATE OF THE SQUARE ROOT
C  NUMBER : NUMBER THE SQUARE ROOT OF WHICH IS TO BE EVALUATED
C  OUTPUT PARAMETERS
C  IX     : THE REQUIRED SQUARE ROOT
C  NOP    : THE NUMBER OF ITERATIONS
```

```
      SUBROUTINE TONRAP (IGUESV, NUMBER, IX,  NOP)
      IMPLICIT INTEGER*4 (A - Z)
      NOP = 0
      IX = IGUESV
10    IY =  (IX + NUMBER / IX) / 2
      IF (IY .GE. IX) RETURN
      IX = IY
      NOP = NOP + 1
      GO TO 10
      END
```

```
 THE NUMBER          1 ROOT     1 GUES       1 ORIGUE       1
 ITERATIONS 0 SQ OF ROOT          1 SQ OF ROOT+1           4
 THE NUMBER          4 ROOT     2 GUES       2 ORIGUE       3
 ITERATIONS 0 SQ OF ROOT          4 SQ OF ROOT+1           9
 THE NUMBER          6 ROOT     2 GUES       2 ORIGUE       3
 ITERATIONS 0 SQ OF ROOT          4 SQ OF ROOT+1           9
 THE NUMBER         11 ROOT     3 GUES       3 ORIGUE       3
 ITERATIONS 0 SQ OF ROOT          9 SQ OF ROOT+1          16
 THE NUMBER         16 ROOT     4 GUES       4 ORIGUE       6
 ITERATIONS 0 SQ OF ROOT         16 SQ OF ROOT+1          25
 THE NUMBER         25 ROOT     5 GUES       5 ORIGUE       7
 ITERATIONS 0 SQ OF ROOT         25 SQ OF ROOT+1          36
 THE NUMBER        256 ROOT    16 GUES      17 ORIGUE      24
 ITERATIONS 1 SQ OF ROOT        256 SQ OF ROOT+1         289
 THE NUMBER       1324 ROOT    36 GUES      37 ORIGUE      52
 ITERATIONS 1 SQ OF ROOT       1296 SQ OF ROOT+1        1369
 THE NUMBER       7068 ROOT    84 GUES      85 ORIGUE     119
 ITERATIONS 1 SQ OF ROOT       7056 SQ OF ROOT+1        7225
 THE NUMBER       4096 ROOT    64 GUES      68 ORIGUE      96
 ITERATIONS 1 SQ OF ROOT       4096 SQ OF ROOT+1        4225
 THE NUMBER       9999 ROOT    99 GUES     103 ORIGUE     103
 ITERATIONS 2 SQ OF ROOT       9801 SQ OF ROOT+1       10000
 THE NUMBER      16396 ROOT   128 GUES     137 ORIGUE     192
 ITERATIONS 1 SQ OF ROOT      16384 SQ OF ROOT+1       16641
 THE NUMBER      99999 ROOT   316 GUES     320 ORIGUE     451
 ITERATIONS 1 SQ OF ROOT      99856 SQ OF ROOT+1      100489
 THE NUMBER     100111 ROOT   316 GUES     320 ORIGUE     451
 ITERATIONS 1 SQ OF ROOT      99856 SQ OF ROOT+1      100489
 THE NUMBER    1558888 ROOT  1248 GUES    1268 ORIGUE    1785
 ITERATIONS 1 SQ OF ROOT    1557504 SQ OF ROOT+1     1560001
 THE NUMBER  16999935 ROOT  4123 GUES    4382 ORIGUE    6171
 ITERATIONS 2 SQ OF ROOT  16999129 SQ OF ROOT+1    17007376
 THE NUMBER 999111189 ROOT 31608 GUES  31629 ORIGUE  31629
 ITERATIONS 1 SQ OF ROOT 999065664 SQ OF ROOT+1  999128881
 THE NUMBER 121000000 ROOT 11000 GUES  11060 ORIGUE  15577
 ITERATIONS 1 SQ OF ROOT 121000000 SQ OF ROOT+1  121022001
```

5.6 The program to generate all the prime numbers

```
      PARAMETER (ND = 26379)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION IPRIM (ND)
      COMMON/A1/IPRIM
10    WRITE (6, '('' TYPE THE LIMIT IN FORMAT I8'')')
      READ (4, '(I8)') LIMIT
      WRITE (6, '(1X, I8)' ) LIMIT
      IF (LIMIT .EQ. 0) STOP
      IF (LIMIT .LT. 0) THEN
         WRITE (6, '('' LIMIT CANNOT BE NEGATIVE'')')
         GO TO 10
      ENDIF
      CALL PRIME (LIMIT, NP)
      WRITE (6,'('' PRIME NUMBERS ARE AS FOLLOWS''/(1X,10I8))')
     1(IPRIM (II), II = 1, NP)
      WRITE (6, '('' TOTAL NUMBER OF PRIMES = '', I5)') NP
      GO TO 10
      END
```

When the subroutine PRIME is called from the main program (the preceding one), it has the following arguments:- LIMIT, NP, i.e. it is of the form
SUBROUTINE PRIME (LIMIT, NP)
instead of
SUBROUTINE PRIME (LIMIT)

```
C  PURPOSE :- TO GENERATE ALL THE PRIME NUMBERS UPTO A
C             CERTAIN UPPER LIMIT

      SUBROUTINE PRIME (LIMIT)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION S (100), IPRIM (26379), IREP (8), INTV (2)
      COMMON/A1/IPRIM
      CALL UPTHIR (LIMIT, IPRIM, NP, INDEX)
C     Set the initial values
      NOFT = 0
      IF (INDEX .EQ. 0) THEN
         IND = 1
         INTV (1) = 49
         INTV (2) = 77
         IREP (1) = 1
         DO 22 I = 2, 8
            IREP (I) = IPRIM (I + 2)
22       CONTINUE
         IND = 1
         NSTOR = 8
```

```
         M = 3
      ENDIF
30    NOFT = NOFT + 30
C     Check whether all possible numbers are tested for primality
      IF (NOFT + 29 .GT. LIMIT) THEN
         IF (NOFT + 1 .GT. LIMIT) RETURN
         DO 44 II = NSTOR - 1, 2, -1
            IF (NOFT + IREP (II) .LE. LIMIT) THEN
               NSTOR = II
               GO TO 50
           ENDIF
44       CONTINUE
         NSTOR = 1
      ENDIF
C     Test whether the number lies between P[j] square and P[j+1]P[j+2]
50    DO 88 JJ = 1, NSTOR
         N = NOFT + IREP (JJ)
         IF (N .EQ. INTV (IND)) THEN
            IF (IND .EQ. 1) THEN
               IND = 2
            ELSE
C              Initialize for N=P[j+1]P[j+2]
               IND = 1
               M = M + 1
               S (M) = 2 * IPRIM (M) + INTV (2)
               PMP1 =  IPRIM (M + 1)
               INTV (1) = PMP1 * PMP1
               INTV (2) = PMP1 *  IPRIM (M + 2)
            ENDIF
         ELSE
            IF (M .GT. 3) THEN
C              Carry out the primality test
               DO 77 J = 4, M
60                IF (N .GE. S (J)) THEN
                  IF (N .EQ. S (J)) THEN
                        S (J) = S (J) + 2 * IPRIM (J)
                        GO TO 88
                     ENDIF
                     S (J) = S (J) + 2 * IPRIM (J)
                     GO TO 60
                   ENDIF
77             CONTINUE
            ENDIF
            NP = NP + 1
            IPRIM (NP) = N
         ENDIF
```

```
88    CONTINUE
      GO TO 30
      END

C  PURPOSE :- THIS SUBROUTINE COMPUTES ALL THE PRIME NUMBERS FROM 2
C              TO A GIVEN NATURAL NUMBER LESS THAN EQUAL TO 30
C  INPUT PARAMETER
C  LIMIT : THE NUMBER UPTO WHICH PRIME IS REQUIRED
C  OUTPUT PARAMETER
C  IPRIM  : ONE-DIMENSIONAL ARRAY, CONTAINS ALL THE PRIME
C           NUMBERS UPTO A CERTAIN LIMIT
C  NP     : THE NUMBER OF PRIMES
C  INDEX  : 0 IF THE NUMBER IS GREATER THAN 29
C         : 1 IF THE NUMBER IS LESS THAN 29
      SUBROUTINE UPTHIR (LIMIT, IPRIM, NP, INDEX)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION IPRIM (10), IREP (8)
C     Set the initial values
      IPRIM (1) = 2
      NP = 1
      IREP (1) = 1
      NOFT = 0
      NSTOR = 1
      NTIME = 2
      IADD = 2
10    DO 33 J = 1, NTIME
         NOFT = NOFT + IADD
         DO 22 I = 1, NSTOR
            N = NOFT + IREP (I)
C           Check whether all the numbers are tested for primality
            IF (N .GT. LIMIT) THEN
                INDEX = 1
                RETURN
            ENDIF
C           The only composite number
            IF (N .NE. 25) THEN
               NP = NP + 1
               IPRIM (NP) = N
            ENDIF
22       CONTINUE
33    CONTINUE
C     Determine primes upto 30 is computed
      IF (NOFT .EQ. 24) THEN
         INDEX = 0
         RETURN
      ENDIF
C     Initialization for numbers greater than 5
```

```
      IREP (2) = IPRIM (3)
      NSTOR = 2
      NTIME = 4
      NOFT = 0
      IADD = 6
      GO TO 10
      END

TYPE THE LIMIT IN FORMAT I8
      11
PRIME NUMBERS ARE AS FOLLOWS
       2       3       5       7      11
TOTAL NUMBER OF PRIMES =     5
TYPE THE LIMIT IN FORMAT I8
      24
PRIME NUMBERS ARE AS FOLLOWS
       2       3       5       7      11      13      17      19      23
TOTAL NUMBER OF PRIMES =     9
TYPE THE LIMIT IN FORMAT I8
      25
PRIME NUMBERS ARE AS FOLLOWS
       2       3       5       7      11      13      17      19      23
TOTAL NUMBER OF PRIMES =     9
TYPE THE LIMIT IN FORMAT I8
      30
PRIME NUMBERS ARE AS FOLLOWS
       2       3       5       7      11      13      17      19      23
TOTAL NUMBER OF PRIMES =    10
TYPE THE LIMIT IN FORMAT I8
      31
PRIME NUMBERS ARE AS FOLLOWS
       2       3       5       7      11      13      17      19      23
      31
TOTAL NUMBER OF PRIMES =    11
TYPE THE LIMIT IN FORMAT I8
      59
PRIME NUMBERS ARE AS FOLLOWS
       2       3       5       7      11      13      17      19      23

      31      37      41      43      47      53      59
TOTAL NUMBER OF PRIMES =    17
TYPE THE LIMIT IN FORMAT I8
      60
PRIME NUMBERS ARE AS FOLLOWS
       2       3       5       7      11      13      17      19      23
```

```
      31      37      41      43      47      53      59
 TOTAL NUMBER OF PRIMES =    17
 TYPE THE LIMIT IN FORMAT I8
     100
 PRIME NUMBERS ARE AS FOLLOWS
       2       3       5       7      11      13      17      19      23      29

      31      37      41      43      47      53      59      61      67      71

      73      79      83      89      97
 TOTAL NUMBER OF PRIMES =    25
 TYPE THE LIMIT IN FORMAT I8
       0
```

5.7 This program creates a nest of DO loops and finds the distinct indices of the loops and their sum

```
      INTEGER P
      PARAMETER (P = 20)
      DIMENSION LINV (P), LTEV (P), LCRE (P), LDEX (P), INTERP(P)
10    WRITE (6, '(1X, ''TYPE THE NUMBER OF DO LOOPS IN FORMAT (I2)'')')
20    READ (4, '(I2)') ND
      WRITE (6,'(1X,I2)') ND
      IF (ND .LT. 0) THEN
         WRITE (6, '(1X, ''DATA ILLEGAL, KINDLY TYPE IT AGAIN'')')
         GO TO 20
      ELSEIF (ND .GT. P) THEN
         WRITE (6, '(1X, ''THIS NUMBER WILL CROSS THE DIMENSION,'',1X,
     1             ''CHANGE THE DIMENSION'')')
      ELSEIF (ND .NE. 0) THEN
         WRITE (6,'(1X,''TYPE INITIAL, TEST VALUES AND INCREMENTS OF''
     1         /1X,''DO LOOPS IN FORMAT (20I2), ONE SET PER LINE'')')
         DO 44  J = 1, ND
30          READ (4, '(3I2)') LINV (J), LTEV (J), LCRE (J)
            WRITE (6,'(1X, 3I2)') LINV (J), LTEV (J), LCRE (J)
            IF (LCRE (J) .EQ. 0) THEN
               WRITE (6, '(1X, ''NO INCREMENT CAN BE ZERO, KINDLY'',1X,
     1                     ''TYPE THIS SET AGAIN'')')
               GO TO 30
            ENDIF
44       CONTINUE
         WRITE (6,'(1X,''TYPE THE NUMBER OF ELEMENTS IN THE SET OF''
     1           ,1X,''PROHIBITED SEQUENCE IN FORMAT (I2)'')')
45       READ (4, '(I2)') NINT
```

```
          WRITE (6, '(1X, I2)') NINT
          IF (NINT .EQ. 0) THEN
             IOVER = 1
          ELSEIF (NINT .GT. ND) THEN
             WRITE (6,'(1X,''NUMBER OF ELEMENTS CANNOT EXCEED'',1X,
     1               ''NUMBER OF LOOPS, KINDLY TYPE AGAIN'')')
             GO TO 45
          ELSEIF (NINT .LT. 0) THEN
             WRITE (6,'(1X,''NUMBER OF ELEMENTS CANNOT NEGATIVE,''
     1             ,1X,''KINDLY TYPE AGAIN'')')
             GO TO 45
          ELSE
             WRITE (6,'(1X,''TYPE THE SET OF VALUES OF THE'',1X,
     1               ''PROHIBITED SEQUENCE IN FORMAT (20I2)'')')
             READ (4, '(20I2)') (INTERP (K), K = 1, ND)
             WRITE (6,'(1X, 20I2)') (INTERP (K), K = 1, ND)
             IOVER = 0
          ENDIF
C         Starting set of values of the DO loops
          CALL START (LDEX, ND,  LINV, LTEV, LCRE)
          LP = 1
          INDEX = 0
          ICHEK = 0
50        CALL PRE (LDEX, ND, INDEX, LP)
          IF (INDEX .EQ. 0) GO TO 70
C         Change the values of the indices of the DO loops
60        CALL NEST (LDEX, ND, INDEX, LP, LINV, LTEV, LCRE)
          IF (INDEX .EQ. 0 .AND. ND .GT. 1) GO TO 50
C         To perform calculation using distinct indices
70        CALL CORE (LDEX, ND, INDEX, ICHEK)
          IF (INDEX .EQ. 1) GO TO 10
          IF (IOVER .EQ. 1) GO TO 60
C         To interrupt a sequence at a particular position
          CALL POST (LDEX, LTEV,  ND, INTERP, NINT, IOVER)
          GO TO 60
       ENDIF
       STOP
       END

C  PURPOSE : TO START DO LOOPS FROM ANY SET OF STARTING VALUES
C  INPUT PARAMETERS
C  LINV : LINEAR ARRAY, CONTAINS INITIAL VALUES
C  LTEV : LINEAR ARRAY, CONTAINS TEST VALUES
C  LCRE : LINEAR ARRAY, CONTAINS INCREMENTS
C    ND : DENOTES THE NUMBER OF DO LOOPS
C  OUTPUT PARAMETERS:
C  LDEX : LINEAR ARRAY, CONTAINS STARTING VALUES
```

```
      SUBROUTINE START (LDEX, ND, LINV, LTEV, LCRE)
      INTEGER P
      PARAMETER (P = 20)
      DIMENSION LINV (ND),LTEV (ND),LCRE (ND),LDEX (ND),LNET (P)
      WRITE (6,'(1X,''TYPE THE SET OF STARTING VALUES,'',1X,
     1       ''ONE VALUE PER LINE IN FORMAT (I2)'')')
C     Validate the data
      DO 22 J = 1, ND
10       READ (4, '(I2)') LNET (J)
         WRITE (6, '(1X, I2)') LNET (J)
         IF (LINV (J) .NE. LNET (J) ) THEN
            IF ((LCRE (J) .LT. 0 .AND. (LINV (J) .LT. LNET (J) .OR.
     1           LNET (J) .LT. LTEV (J)))
     1          .OR. (LCRE (J) .GT. 0 .AND. (LINV (J) .GT. LNET (J) .OR.
     1          LNET (J) .GT. LTEV (J)))
     1          .OR. (MOD (LNET (J) - LINV (J),  LCRE(J)) .NE. 0)) THEN
                WRITE (6,'(1X,''STARTING VALUE OF LOOP NUMBER'',1X,I2,1X,
     1               ''IS WRONG, PLEASE TYPE IT AGAIN'')') J
                GO TO 10
            ENDIF
         ENDIF
22    CONTINUE
C     Put indices of all DO loops equal to the starting values
      DO 33 K = 1, ND
         LDEX (K) = LNET (K)
33    CONTINUE
      RETURN
      END

C  PURPOSE -  TO FIND OUT WHETHER THE INDICES OF THE
C             DO LOOPS ARE DISTINCT
C  INPUT PARAMETERS
C  LDEX : LINEAR ARRAY, THE VALUES OF THE INDICES OF THE DO LOOPS
C    ND : NUMBER OF DO LOOPS
C    LP : THE LOOP NUMBER
C INDEX : = 1 IF THE CONDITON IS NOT SATISFIED
C         = 0 OTHERWISE
      SUBROUTINE PRE (LDEX, ND, INDEX, LP)
      DIMENSION LDEX (ND)
      IF (ND .GT. 1) THEN
         IF (LP .EQ. 1) LP = 2
         DO 22  K = LP, ND
            IPV = LDEX (K)
C           Compare the indices of the DO loops
            DO 11 J = 1, K - 1
               IF (LDEX (J) .EQ. IPV) THEN
C              If all indices are not distinct
```

```
                  INDEX = 1
                  GO TO 30
               ENDIF
11           CONTINUE
22        CONTINUE
      ENDIF
      RETURN
30    LP = K
      RETURN
      END

C  PURPOSE - TO CHANGE VALUES OF INDICES OF DO LOOPS
C  INPUT PARAMETERS
C  LDEX : LINEAR ARRAY, CONTAINS THE STARTING VALUES
C  LINV : LINEAR ARRAY, CONTAINS THE INITIAL VALUES
C  LTEV : LINEAR ARRAY, CONTAINS THE TEST VALUES
C  LCRE : LINEAR ARRAY, CONTAINS THE INCREMENTS
C    ND : DENOTES NUMBER OF LOOPS
C  INDEX = 0 IF INCREMENT OF A PARTICULAR LOOP IS POSSIBLE
C          = 1 OTHERWISE
C    LP : IS A PARTICULAR LOOP NUMBER
C  OUTPUT PARAMETER
C  LDEX : LINEAR ARRAY, CONTAINS AUGMENTED VALUES
C         INDICES OF DO LOOPS
      SUBROUTINE NEST (LDEX, ND, INDEX, LP, LINV, LTEV, LCRE)
      DIMENSION LDEX (ND), LINV (ND), LTEV (ND), LCRE (ND)
      IF (INDEX .EQ. 1) THEN
         INDEX = 0
      ELSEIF (LP .LT. ND) THEN
         LP = ND
      ENDIF
      M = LP
C     Check whether a particular index can be increased or decreased
      DO 11 J = 1, M
         LDEX (LP) = LDEX (LP) + LCRE (LP)
         IF ((LCRE (LP) .LT. 0 .AND. LDEX (LP) .GE. LTEV (LP))
     1   .OR. (LCRE (LP) .GT. 0 .AND. LDEX (LP) .LE. LTEV (LP))) RETURN
         LDEX (LP) = LINV (LP)
         LP = LP - 1
11    CONTINUE
      INDEX = 1
      RETURN
      END

C  PURPOSE : TO PERFORM THE CALCULATION, USING A DISTINCT
C            SET OF INDICES
C  INPUT PARAMETERS
C  LDEX : LINEAR ARRAY, IS THE SET OF DISTINCT VALUES OF THE INDICES
```

```
C    ND : THE NUMBER OF LOOPS
C    LP : THE LOOP NUMBER
C  INDEX : = 1, NO MORE DISTINCT SET OF INDICES EXISTS
C         = 0, DISTINCT SET OF INDICES EXISTS
C  ICHEK : = 0, IF NO SET OF VALUES IS PRINTED
C         = 1, IF ATLEAST ONE SET OF VALUES IS PRINTED
      SUBROUTINE CORE (LDEX, ND, INDEX, ICHEK)
      DIMENSION LDEX (ND)
      IF (INDEX .EQ. 1) THEN
         IF (ICHEK .EQ. 1) RETURN
         WRITE (6,'(1X,''NO SET OF DISTINCT VALUES EXISTS'')')
         RETURN
      ELSE
         IF (ICHEK .EQ. 0) THEN
            ICHEK = 1
            WRITE (6,'(1X,''SET OF DISTINCT VALUES/PERMUTATIONS''
     1             ,1X,''AND THEIR SUM'')')
         ENDIF
         NSUM = 0
         DO 11 I = 1, ND
            NSUM = NSUM + LDEX (I)
11       CONTINUE
         WRITE (6, '(1X, 16I5)') (LDEX (J), J = 1, ND), NSUM
      ENDIF
      RETURN
      END

C  PURPOSE : TO INTERRUPT A SEQUENCE AT A POSITION K (1 ≤ K < ND)
C  INPUT PARAMETERS
C  LDEX : LINEAR ARRAY, CONTAINS VALUES OF INDICES
C         OF THE DO LOOPS
C  LTEV : LINEAR ARRAY, CONTAINS TEST VALUES
C  INTERP: LINEAR ARRAY, CONTAINS SET OF VALUES,
C          AFTER WHICH INTERRUPTION IS NECESSARY
C  ND    : NUMBER OF DO LOOPS
C  K     : THE POSITION NUMBER OF THE INTERRUPTION
C  IOVER: 0 IF INTERRUPTION IS NOT CARRIED OUT
C       : 1 IF INTERRUPTION IS CARRIED OUT
C  OUTPUT PARAMETERS
C  LDEX :  LAST (ND - (K+1)) CONTAINS TEST VALUES OF RESPECTIVE LOOPS,
C         AND THE REST REMAIN UNCHANGED
      SUBROUTINE POST (LDEX, LTEV,  ND, INTERP, K, IOVER)
      DIMENSION LDEX (ND),  LTEV (ND),  INTERP (ND)
      IF (ND .GT. K) THEN
         DO 11 I = 1, ND
            IF (LDEX (I) .NE. INTERP (I) ) RETURN
11       CONTINUE
         IOVER = 1
```

```
         DO 22 J = K + 1, ND, 1
            LDEX (J) = LTEV (J)
22       CONTINUE
      ENDIF
      RETURN
      END
```

```
 TYPE THE NUMBER OF DO LOOPS IN FORMAT (I2)
  2
 TYPE INITIAL, TEST VAlUES AND INCREMENTS OF
 DO LOOPS IN FORMAT (20I2), ONE SET PER LINE
  5 6 1
  5 9 1
 TYPE THE NUMBER OF ELEMENTS IN THE SET OF PROHIBITED SEQUENCE IN FORMAT (I2)
  1
 TYPE THE SET OF VALUES OF THE PROHIBITED SEQUENCE IN FORMAT (20I2)
  5 7
 TYPE THE SET OF STARTING VALUES, ONE VALUE PER LINE IN FORMAT (I2)
  5
  5
 SET OF DISTINCT VALUES/PERMUTATIONS AND THEIR SUM
     5    6   11
     5    7   12
     6    5   11
     6    7   13
     6    8   14
     6    9   15
 TYPE THE NUMBER OF DO LOOPS IN FORMAT (I2)
  3
 TYPE INITIAL, TEST VAlUES AND INCREMENTS OF
 DO LOOPS IN FORMAT (20I2), ONE SET PER LINE
  6 1-2
  4 5 1
  3 3-1
 TYPE THE NUMBER OF ELEMENTS IN THE SET OF PROHIBITED SEQUENCE IN FORMAT (I2)
  3
 TYPE THE SET OF VALUES OF THE PROHIBITED SEQUENCE IN FORMAT (20I2)
  2 3 4
 TYPE THE SET OF STARTING VALUES, ONE VALUE PER LINE IN FORMAT (I2)
  4
  5
  4
 STARTING VALUE OF LOOP NUMBER  3 IS WRONG, PLEASE TYPE IT AGAIN
  3
 SET OF DISTINCT VALUES/PERMUTATIONS AND THEIR SUM
     4    5    3   12
     2    4    3    9
     2    5    3   10
```

```
TYPE THE NUMBER OF DO LOOPS IN FORMAT (I2)
 3
TYPE INITIAL, TEST VAlUES AND INCREMENTS OF
DO LOOPS IN FORMAT (20I2), ONE SET PER LINE
 8 8 7
 8 9 6
1215 5
TYPE THE NUMBER OF ELEMENTS IN THE SET OF PROHIBITED SEQUENCE IN FORMAT (I2)
 0
TYPE THE SET OF STARTING VALUES, ONE VALUE PER LINE IN FORMAT (I2)
 8
 8
12
NO SET OF DISTINCT VALUES EXISTS
TYPE THE NUMBER OF DO LOOPS IN FORMAT (I2)
 2
TYPE INITIAL, TEST VAlUES AND INCREMENTS OF
DO LOOPS IN FORMAT (20I2), ONE SET PER LINE
 2 3 1
10 8-2
TYPE THE NUMBER OF ELEMENTS IN THE SET OF PROHIBITED SEQUENCE IN FORMAT (I2)
 1
TYPE THE SET OF VALUES OF THE PROHIBITED SEQUENCE IN FORMAT (20I2)
 310
TYPE THE SET OF STARTING VALUES, ONE VALUE PER LINE IN FORMAT (I2)
 2
 4
STARTING VALUE OF LOOP NUMBER  2 IS WRONG, PLEASE TYPE IT AGAIN
 8
SET OF DISTINCT VALUES/PERMUTATIONS AND THEIR SUM
    2    8   10
    3   10   13
TYPE THE NUMBER OF DO LOOPS IN FORMAT (I2)
 2
TYPE INITIAL, TEST VAlUES AND INCREMENTS OF
DO LOOPS IN FORMAT (20I2), ONE SET PER LINE
 2 7 0
NO INCREMENT CAN BE ZERO, KINDLY TYPE THIS SET AGAIN
 5 5 4
 1 3 2
TYPE THE NUMBER OF ELEMENTS IN THE SET OF PROHIBITED SEQUENCE IN FORMAT (I2)
 3
NUMBER OF ELEMENTS CANNOT EXCEED NUMBER OF LOOPS, KINDLY TYPE AGAIN
 2
TYPE THE SET OF VALUES OF THE PROHIBITED SEQUENCE IN FORMAT (20I2)
 1 2
TYPE THE SET OF STARTING VALUES, ONE VALUE PER LINE IN FORMAT (I2)
 5
 3
```

```
SET OF DISTINCT VALUES/PERMUTATIONS AND THEIR SUM
    5    3    8
TYPE THE NUMBER OF DO LOOPS IN FORMAT (I2)
-8
DATA ILLEGAL, KINDLY TYPE IT AGAIN
 1
TYPE INITIAL, TEST VALUES AND INCREMENTS OF
DO LOOPS IN FORMAT (20I2), ONE SET PER LINE
505050
TYPE THE NUMBER OF ELEMENTS IN THE SET OF PROHIBITED SEQUENCE IN FORMAT (I2)
 0
TYPE THE SET OF STARTING VALUES, ONE VALUE PER LINE IN FORMAT (I2)
50
SET OF DISTINCT VALUES/PERMUTATIONS AND THEIR SUM
   50   50
TYPE THE NUMBER OF DO LOOPS IN FORMAT (I2)
21
THIS NUMBER WILL CROSS THE DIMENSION, CHANGE THE DIMENSION
```

5.8 Program to reconstruct a polynomial from its integer zeros

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION ITZ (ND), ICOF (ND)
10    WRITE (6, '('' TYPE THE NUMBER OF ZEROS IN FORMAT (I2)''
     1      /,1X,''FOR STOPPING PLEASE TYPE : 0'')')
20    READ (4, '(I2)') NZ
      WRITE (6,'(1X, I2)') NZ
      IF (NZ .EQ. 0) STOP
      IF (NZ .LT. 0) THEN
         WRITE (6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
         GO TO 20
      ELSEIF (NZ .GT. ND) THEN
         WRITE (6, '('' THE GIVEN NUMBER WILL CROSS THE DIMENSION,''
     1,1X,''CHANGE DIMENSION'')')
         STOP
      ENDIF
      WRITE (6, '('' TYPE ALL ZEROS IN FORMAT (10I5)'')')
      READ (4, '(15I5)') (ITZ (K), K = 1, NZ)
      WRITE (6, '(1X, 10I5)') (ITZ (K), K = 1, NZ)
      CALL RECONZ (ITZ, NZ, ICOF)
      WRITE (6, '('' COEFFICIENTS OF THE REQUIRED MONIC POLYNOMIAL,''
     1/,1X,''STARTING FROM THE 2ND COEFFICIENT, ARE AS FOLLOWS'')')
      WRITE (6, '(1X, 10(I7, 1X))') (ICOF (K), K = 1, NZ)
      GO TO 10
```

```
      END

C  PURPOSE - RECONSTRUCT A POLYNOMIAL FROM ITS INTEGER ZEROS
C  INPUT PARAMETERS
C  ITZ  : LINEAR ARRAY, THE ZEROS OF THE POLYNOMIAL
C   NZ  : THE NUMBER OF ZEROS
C  ICOF : LINEAR ARRAY, COEFFICIENTS OF THE POLYNOMIAL
C         TO BE MULTIPLIED
C  OUTPUT PARAMETER
C  ICOF : THE REQUIRED COEFFICIENTS OF THE POLYNOMIAL

      SUBROUTINE RECONZ (ITZ, NZ, ICOF)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION ITZ (NZ), ICOF (NZ)
C     Form the linear polynomial with the first zero
      ICOF (1) = - ITZ (1)
      M = 1
      IF (NZ .GT. 1) THEN
C        Multiply the existing polynomial by linear polynomial
         DO 33 I = 2, NZ
            CALL LINZ (ITZ (I), ICOF, M)
33       CONTINUE
      ENDIF
      RETURN
      END

C  PURPOSE - MULTIPLY A POLYNOMIAL BY A LINEAR FACTOR
C  INPUT PARAMETERS
C  MLZ  : THE CONSTANT TERM OF THE LINEAR POLYNOMIAL
C  ICOF : LINEAR ARRAY, COEFFICIENTS OF THE POLYNOMIAL
C         TO BE MULTIPLIED
C  M    : THE DEGREE OF THE POLYNOMIAL
C  OUTPUT PARAMETERS
C  ICOF : COEFFICIENTS OF THE POLYNOMIAL AFTER MULTIPLICATION
C  M    : NEW DEGREE OF THE POLYNOMIAL

      SUBROUTINE LINZ (MLZ, ICOF, M)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION ICOF (M)
      M = M + 1
      ICOF (M) = - MLZ * ICOF (M - 1)
      IF (M .NE. 2) THEN
         DO 11 K = 1, M - 2
            ICOF (M - K) = ICOF (M - K) - MLZ * ICOF (M - K - 1)
11       CONTINUE
      ENDIF
      ICOF (1) = ICOF (1) - MLZ
      RETURN
      END
```

```
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 1
TYPE ALL ZEROS IN FORMAT (10I5)
42745
COEFFICIENTS OF THE REQUIRED MONIC POLYNOMIAL,
STARTING FROM THE 2ND COEFFICIENT, ARE AS FOLLOWS
 -42745
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 2
TYPE ALL ZEROS IN FORMAT (10I5)
  256  421
COEFFICIENTS OF THE REQUIRED MONIC POLYNOMIAL,
STARTING FROM THE 2ND COEFFICIENT, ARE AS FOLLOWS
   -677  107776
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 3
TYPE ALL ZEROS IN FORMAT (10I5)
   11   22   33
COEFFICIENTS OF THE REQUIRED MONIC POLYNOMIAL,
STARTING FROM THE 2ND COEFFICIENT, ARE AS FOLLOWS
    -66    1331   -7986
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 4
TYPE ALL ZEROS IN FORMAT (10I5)
   10   20   30   40
COEFFICIENTS OF THE REQUIRED MONIC POLYNOMIAL,
STARTING FROM THE 2ND COEFFICIENT, ARE AS FOLLOWS
   -100    3500  -50000  240000
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 5
TYPE ALL ZEROS IN FORMAT (10I5)
    1    2    3    4    5
COEFFICIENTS OF THE REQUIRED MONIC POLYNOMIAL,
STARTING FROM THE 2ND COEFFICIENT, ARE AS FOLLOWS
    -15      85    -225     274    -120
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 8
TYPE ALL ZEROS IN FORMAT (10I5)
    1    2    3    4    5    6    7    8
COEFFICIENTS OF THE REQUIRED MONIC POLYNOMIAL,
STARTING FROM THE 2ND COEFFICIENT, ARE AS FOLLOWS
    -36     546   -4536   22449  -67284  118124 -109584   40320
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
```

```
 FOR STOPPING PLEASE TYPE : 0
 -2
 DATA ILLEGAL, KINDLY TYPE AGAIN
 30
 THE GIVEN NUMBER WILL CROSS THE DIMENSION, CHANGE DIMENSION
```

5.9 Program for the reconstruction of a polynomial over Z from its rational zeros

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION NUM (ND), IDEN (ND), ICOF (11)
10    WRITE (6, '('' TYPE THE NUMBER OF ZEROS IN FORMAT (I2)''
     1/,1X, ''FOR STOPPING PLEASE TYPE : 0'')')
20    READ (4, '(I2)') NZ
      WRITE (6, '(1X, I2)') NZ
      IF (NZ .EQ. 0) STOP
      IF (NZ .LT. 0) THEN
         WRITE (6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
         GO TO 20
      ELSEIF (NZ .GT. ND) THEN
         WRITE (6, '('' THE GIVEN NUMBER WILL CROSS THE DIMENSION,''
     1,1X,''CHANGE DIMENSION'')')
         STOP
      ENDIF
      WRITE (6, '('' TYPE ALL NUMERATOR OF ZEROS IN FORMAT (10I5)'')')
      READ (4, '(10I5)') (NUM (K), K = 1, NZ)
      WRITE (6, '(1X, 10I5)') (NUM (K), K = 1, NZ)
      WRITE (6, '('' TYPE ALL DENOMINATOR OF ZEROS IN FORMAT (10I5)'')')
30    READ (4, '(15I5)') (IDEN (K), K = 1, NZ)
      WRITE (6, '(1X, 10I5)') (IDEN (K), K = 1, NZ)
      DO 44 K = 1, NZ
         IF (IDEN (K) .EQ. 0) THEN
            WRITE (6, '('' NO DENOMINATORS CAN BE ZERO, KINDLY TYPE''
     1                ,1X,''ALL THE DENOMINATORS AGAIN'')')
         GO TO 30
         ENDIF
44    CONTINUE
      CALL RECONQ (NUM, IDEN, NZ, ICOF, M)
      WRITE (6, '('' COEFFICIENTS OF THE REQUIRED POLYNOMIAL,''
     1          ,1X,''ARE AS FOLLOWS'')')
      WRITE (6, '(1X, 10(I7, 1X))') (ICOF (K), K = 1, M)
      GO TO 10
      END

C  PURPOSE - RECONSTRUCTION OF A POLYNOMIAL OVER Z FROM ITS
```

```
C              RATIONAL ZEROS
C  INPUT PARAMETERS
C  NUM :  LINEAR ARRAY, THE NUMERATOR OF THE ZEROS
C  IDEN:  LINEAR ARRAY, THE DENOMINATOR OF THE ZEROS
C  NZ  :  THE NUMBER OF ZEROS
C  OUTPUT PARAMETERS
C  ICOF : COEFFICIENTS OF THE REQUIRED POLYNOMIAL
C  M    : DEGREE OF THE POLYNOMIAL
       SUBROUTINE RECONQ (NUM, IDEN, NZ, ICOF, M)
       IMPLICIT INTEGER*4 (A - Z)
       DIMENSION NUM (NZ), IDEN (NZ), ICOF (11)
C      Validate the data
       DO 55 K = 1, NZ
          ICN = NUM (K)
          ICD = IDEN (K)
          IF (ICN .EQ. 0) THEN
             ICD = 1
          ELSE
             IF (ICD .LT. 0) THEN
                ICD = - ICD
                ICN = - ICN
             ENDIF
             IHCF = HCF (ICN, ICD)
             IF (IHCF .NE. 1) THEN
                ICN = ICN / IHCF
                ICD = ICD / IHCF
             ENDIF
          ENDIF
          IF (K .NE. 1) THEN
             CALL LINQ (ICN, ICD, ICOF, M)
          ELSE
C            Form a linear polynomial with the first zero
             ICOF (1) = ICD
             ICOF (2) = - ICN
             M = 2
          ENDIF
55     CONTINUE
       RETURN
       END

C  PURPOSE - MULTIPLY A POLYNOMIAL BY A LINEAR FACTOR
C  INPUT PARAMETERS
C  MLNZ : THE NUMERATOR OF THE CONSTANT TERM OF THE
C         LINEAR POLYNOMIAL
C  MLDZ : THE DENOMINATOR OF THE CONSTANT TERM OF THE
C         LINEAR POLYNOMIAL
C  ICOF : COEFFICIENT OF THE POLYNOMIAL TO BE MULTIPLIED
C  M    : DEGREE OF THE POLYNOMIAL
```

```
C  OUTPUT PARAMETERS
C  ICOF : COEFFICIENTS OF THE POLYNOMIAL, AFTER MULTIPLICATION
C  M    : NEW DEGREE OF THE POLYNOMIAL
      SUBROUTINE LINQ (MLNZ, MLDZ, ICOF, M)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION ICOF (M)
      M = M + 1
      ICOF (M) = - MLNZ * ICOF (M - 1)
      IF (M .NE. 2) THEN
         MN2 = M - 2
         DO 11 K = 1, MN2
            ICOF (M - K) = MLDZ * ICOF (M - K) - MLNZ * ICOF (M - K - 1)
11       CONTINUE
      ENDIF
      ICOF (1) = MLDZ * ICOF (1)
      RETURN
      END

 TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  1
 TYPE ALL NUMERATOR OF ZEROS IN FORMAT (10I5)
 21245
 TYPE ALL DENOMINATOR OF ZEROS IN FORMAT (10I5)
 42182
 COEFFICIENTS OF THE REQUIRED POLYNOMIAL, ARE AS FOLLOWS
    6026   -3035
 TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  2
 TYPE ALL NUMERATOR OF ZEROS IN FORMAT (10I5)
   652  100
 TYPE ALL DENOMINATOR OF ZEROS IN FORMAT (10I5)
   267  200
 COEFFICIENTS OF THE REQUIRED POLYNOMIAL, ARE AS FOLLOWS
     534   -1571     652
 TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  3
 TYPE ALL NUMERATOR OF ZEROS IN FORMAT (10I5)
    25   50   60
 TYPE ALL DENOMINATOR OF ZEROS IN FORMAT (10I5)
    32   24   56
 COEFFICIENTS OF THE REQUIRED POLYNOMIAL, ARE AS FOLLOWS
    5376  -21160   25250   -9375
 TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  4
```

```
TYPE ALL NUMERATOR OF ZEROS IN FORMAT (10I5)
   10    0   30   40
TYPE ALL DENOMINATOR OF ZEROS IN FORMAT (10I5)
  -55   66   77   88
COEFFICIENTS OF THE REQUIRED POLYNOMIAL, ARE AS FOLLOWS
   9317   -6171     220     300       0
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 8
TYPE ALL NUMERATOR OF ZEROS IN FORMAT (10I5)
    1    2    3    4    5    6    7    8
TYPE ALL DENOMINATOR OF ZEROS IN FORMAT (10I5)
    0    1    0    1    1    1    1    1
NO DENOMINATORS CAN BE ZERO, KINDLY TYPE ALL THE DENOMINATORS AGAIN
    1    1    1    1    1    1    1    1
COEFFICIENTS OF THE REQUIRED POLYNOMIAL, ARE AS FOLLOWS
      1     -36     546   -4536   22449  -67284  118124 -109584   40320
TYPE THE NUMBER OF ZEROS IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
-2
DATA ILLEGAL, KINDLY TYPE AGAIN
30
THE GIVEN NUMBER WILL CROSS THE DIMENSION, CHANGE DIMENSION
```

5.10 Program for the reconstruction of polynomial over Z from its complex and surd zeros

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 20)
      DIMENSION A (ND), B (ND), ICOF (ND)
10    WRITE (6, '('' TYPE THE NUMBER OF ZEROS IN FORMAT (I2)''
     1      /1X,''FOR STOPPING PLEASE TYPE : 0'')')
20    READ (4, '(I2)') NZ
      WRITE (6,'(1X, I2)') NZ
      IF (NZ .EQ. 0) STOP
      IF (NZ .LT. 0) THEN
         WRITE (6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
         GO TO 20
      ELSEIF (NZ .GT. ND) THEN
         WRITE (6, '('' THE GIVEN NUMBER WILL CROSS THE DIMENSION,''
     1,1X,''CHANGE DIMENSION'')')
         STOP
      ENDIF
      WRITE (6, '('' TYPE ALL ZEROS IN FORMAT (10I4)'')')
      READ (4, '(20I4)') (A (K), B (K), K = 1, NZ)
      WRITE (6, '(1X, 20I4)') (A (K), B (K), K = 1, NZ)
```

```
      CALL RCONSC (A, B, NZ, ICOF, N)
      WRITE (6, '('' COEFFICIENTS OF THE REQUIRED MONIC POLYNOMIAL,''
     1/1X,''STARTING FROM THE 2ND COEFFICIENT, ARE AS FOLLOWS'')')
      WRITE (6, '(1X, 10(I7, 1X))') (ICOF (K), K = 1, N)
           IF (NZ .EQ. 8) STOP
      GO TO 10
      END

C  PURPOSE - RECONSTRUCTION OF POLYNOMIAL OVER Z FROM ITS
C            COMPLEX AND SURDS ZEROS
C  INPUT PARAMETERS
C  A   : LINEAR ARRAY, THE INTEGER PART OF THE ZEROS
C  B   : LINEAR ARRAY, THE IMAGINARY/IRRATIONAL PART OF THE ZEROS
C  NZ  : THE NUMBER OF ZEROS
C  OUTPUT PARAMETER
C  ICOF : LINEAR ARRAY, THE COEFFICIENTS OF THE POLYNOMIAL

      SUBROUTINE RCONSC (A, B, NZ, ICOF, N)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION A (NZ), B (NZ), ICOF (N)
C     Form the quadratic polynomial with the first pair of zeros
      ICOF (1) = -  2 * A (1)
      ICOF (2) = (A (1)) ** 2 - B (1)
      N = 2
      IF (NZ .GT. 1) THEN
C        Multiply the existing polynomial by quadratic polynomial
         DO 33 I = 2, NZ
            C = -  2 * A (I)
            D = (A (I)) ** 2 - B (I)
            CALL QMUL (ICOF, N, C, D)
33       CONTINUE
      ENDIF
      RETURN
      END

C  PURPOSE - MULTIPLY A POLYNOMIAL BY A QUADRATIC FACTOR
C  INPUT PARAMETERS
C  P    : LINEAR ARRAY, COEFFICIENTS OF THE POLYNOMIAL
C         TO BE MULTIPLIED
C  N    : THE DEGREE OF THE POLYNOMIAL
C  C    : THE SECOND COEFFICIENT OF THE QUADRATIC POLYNOMIA,
C  D    : THE CONSTANT TERM OF THE QUADRATIC POLYNOMIAL
C  OUPUT PARAMETERS
C  ICOF : COEFFICIENTS OF THE POLYNOMIAL AFTER MULTIPLICATION
C  N    : NEW DEGREE OF THE POLYNOMIAL

      SUBROUTINE QMUL (P, N, C, D)
      IMPLICIT INTEGER*4 (A-Z)
      DIMENSION P (N)
      P (N + 2) = 0
```

```
      P (N + 1) = 0
      DO 11 K = 0, N - 1
         P (N+2-K) = P (N+2-K) +  C * P (N+1-K) + D * P (N - K)
11    CONTINUE
      P (2) = P (2) +  C * P (1) + D
      P (1) = P (1) +  C
      N = N + 2
      RETURN
      END
```

5.11 Given a monic polynomial over Z, this program computes its integer zeros

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER  (ND = 10)
      DIMENSION PR (26379), ICOF (ND), ITZ (ND)
      COMMON /A1/PR
      CALL PRIME (304678)
10    WRITE(6,'('' TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)''/
     11X,''FOR STOPPING PLEASE TYPE : 0'')')
20    READ(4, '(I2)') M
      WRITE (6, '(1X, I2)') M
      IF (M .EQ. 0) STOP
      IF (M .LT. 0) THEN
          WRITE(6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
          GO TO 20
      ELSEIF (M .GT. ND) THEN
         WRITE (6,'('' THE GIVEN NUMBER WILL CROSS THE DIMENSION,''
     1   ,1X,''CHANGE THE DIMENSION'')')
         STOP
      ENDIF
      WRITE(6,'('' TYPE THE COEFFICIENTS OF THE POLYNOMIAL FROM 2ND''
     1/1X,''ON WARDS IN FORMAT (10I8)'')')
      READ (4, '(10I8)') (ICOF (I), I = 1, M)
      WRITE(6, '(1X, 10I8)') (ICOF (I), I = 1, M)
      NC = M
      CALL ZEROS (ICOF, NC, ITZ, NZ)
      IF (NZ .EQ. 0) THEN
         WRITE(6,'('' THIS POLYNOMIAL HAS NO INTEGRAL ROOTS'')')
      ELSEIF (NZ .LT. M) THEN
         NUMZ = M - NZ
         WRITE (6,'('' THIS POLYNOMIAL HAS '',I2,'' ROOTS''
     1   ,1X,''WHICH ARE NOT INTEGERS'')') NUMZ
         WRITE (6,'('' THE '',I2,'' INTEGRAL ROOTS OF THIS POLYNOMIAL''
```

```
     1   ,1X,''ARE''/1X,10I7)') NZ, (ITZ(KK), KK = 1, NZ)
         ELSE
            WRITE(6,'('' ALL THE ROOTS OF THIS POLYNOMIAL ARE''
     1   ,1X,'' INTEGERS AND ARE''/1X,10I7)')(ITZ(KK), KK=1,M)
         ENDIF
         GO TO 10
         END

C  PURPOSE - TO FIND INTEGER ZEROS (OTHER THAN 1,0,-1) OF A MONIC
C            POLYNOMIAL OVER THE SET OF INTEGERS
C  INPUT PARAMETERS
C  ICOF : ONE-DIMENSIONAL ARRAY, COEFFICIENTS OF THE POLYNOMIAL
C  NC   : THE DEGREE OF THE POLYNOMIAL
C  OUTPUT PARAMETERS
C  ITZ  : ONE-DIMENSIONAL ARRAY, CONTAINS ALL INTEGER ZEROS OF
C         THE POLYNOMIAL
C  NZ   : NUMBER OF INTEGER ZEROS OF THE POLYNOMIAL
       SUBROUTINE ZEROS (ICOF, NC, ITZ, NZ)
       IMPLICIT INTEGER*4 (A - Z)
       PARAMETER (ND = 20)
       DIMENSION ICOF (NC), ITZ (ND), IFAC (500), IVAL (3)
       COMMON /B1/IFAC, NF /D1/KK
       IND = 0
       NZ = 0
       CALL TELIN (ICOF, NC, ITZ, NZ, IVAL, IND)
       IF (IND .EQ. 1) RETURN
       IF (NC .NE. 1) THEN
C         Compute zeros of nonlinear polynomial
          NEFC = 1
          KK = IABS (ICOF (NC))
C         Find all the factors of the last non-zero coefficient
          CALL FAC
10        NEFC = NEFC + 1
          IF (NEFC .GT. NF) RETURN
          IC = IFAC (NEFC)
20        ISUM = 1
          IF (IVAL (2) / (IC + 1) * (IC + 1) .EQ. IVAL (2)) THEN
             IF (IVAL (1) / (IC - 1) * (IC - 1) .EQ. IVAL (1)) THEN
C               Test a particular factor satisfies the polynomial
                DO 33 K = 1, NC
                   ISUM = ISUM * IC + ICOF (K)
33              CONTINUE
             ENDIF
          ENDIF
          IF (ISUM .NE. 0) THEN
C            Determine the factor is positive or negative
             IF (IC .LT. 0) GO TO 10
```

```
            IC = - IC
            GO TO 20
         ENDIF
         NZ = NZ + 1
C        The integer zeros of the polynomial
         ITZ (NZ) = IC
         CALL POLDIV (ICOF, NC, IC)
         IF (NC .NE. 1) GO TO 20
      ENDIF
C     For linear polynomial
      NZ = NZ + 1
      ITZ (NZ) = - ICOF (1)
      RETURN
      END

C  PURPOSE - TO FIND THE INTEGER ZEROS 1,0,-1 TOGETHER WITH THEIR
C            MULTIPLICITIES IF THEY EXIST AND TO DETERMINE THE
C            INTEGER ZEROS OTHER THAN THESE
C  INPUT PARAMETERS
C  ICOF : ONE DIMENSIONAL ARRAY, COEFFICIENTS OF THE POLYNOMIAL
C  ID   : THE DEGREE OF THE POLYNOMIAL
C  OUTPUT PARAMETERS
C  ICOF : ONE DIMENSIONAL ARRAY, COEFFICIENTS OF THE POLYNOMIAL
C         AFTER REMOVING THE ZEROS 1,0,1 (IF THEY EXIST)
C  ID   : THE THE DEGREE OF THE POLYNOMIAL AFTER REMOVING THE ZEROS 1,0,-1
C  ITZ  : ONE DIMENSIONAL ARRAY, CONTAINS ZEROS 1,0,-1 (IF THEY EXIST)
C         OF THE POLYNOMIAL
C  NZ   : NUMBER OF ZEROS 1,0,-1 OF THE POLYNOMIAL
C  IVAL : ONE DIMENSIONAL ARRAY, THE NON-ZERO VALUES OF THE DEFLATED
C         POLYNOMIAL (AFTER REMOVING THE ZEROS 1,0,-1) AT 1,0,-1
C  IND  : 0 INITIALLY
C       : 1 IF THE POLYNOMIAL DOES NOT HAVE ANY INTEGER ZEROS
C           OTHER THAN 1,0,-1
      SUBROUTINE TELIN (ICOF, ID, ITZ, NZ, IVAL, IND)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 20)
      DIMENSION ICOF (ID), IVAL (3), IFAC (500), ITZ (ND)
C     Test whether 0 is one of the zeros of the polynomial
      NC = ID
      DO 11 J = 1, ID
         IF (ICOF (NC) .NE. 0) GO TO 20
         NZ = NZ + 1
         ITZ (NZ) = 0
         NC = NC - 1
11    CONTINUE
      IND = 1
      RETURN
20    IC = 1
```

```
30     ID = NC
C      Test 1 and - 1 satisfy the polynomial
       DO 55 I = 1, ID
          ISUM = 1
          DO 44 K = 1, NC
             ISUM = ISUM * IC + ICOF (K)
44        CONTINUE
          IF (ISUM .NE. 0) GO TO 60
          CALL POLDIV (ICOF, NC, IC)
          NZ = NZ + 1
          ITZ (NZ) = IC
55     CONTINUE
       IND = 1
       RETURN
60     IF (IC .EQ. 1) THEN
          IC = - 1
          GO TO 30
       ENDIF
       ID = NC
C      Compute the values of the polynomial at -1,1,0
       IVAL (2) = ISUM
       ISUM = 1
       DO 77 K = 1, ID
          ISUM = ISUM + ICOF (K)
77     CONTINUE
       IVAL (1) = ISUM
       IVAL (3) = ICOF (ID)
       DO 88 I = 1, 3
          IF (IVAL (I) .EQ. IVAL (I) / 3 * 3) RETURN
88     CONTINUE
C      Linear factors do not exist
       IND = 1
       RETURN
       END

C PURPOSE -  DEFLATION OF THE MONIC POLYNOMIAL BY A LINEAR FACTOR
C  INPUT PARAMETERS
C  ICOF   : ONE DIMENSIONAL ARRAY, CONTAINS THE COEFFICIENTS OF
C           THE POLYNOMIAL TO BE DEFLATED
C  ID     : THE DEGREE OF THE POLYNOMIAL TO BE DEFLATED
C  IC     : THE CONSTANT TERM OF THE LINEAR FACTOR OF THE POLYNOMIAL
C  OUTPUT PARAMETERS
C  ICOF   : ONE-DIMENSIONAL ARRAY, COEFFICIENTS OF THE
C           DEFLATED POLYNOMIAL
C  ID     : THE DEGREE OF THE DEFLATED POLYNOMIAL

       SUBROUTINE POLDIV (ICOF, ID, IC)
       IMPLICIT INTEGER*4 (A - Z)
```

```
      DIMENSION ICOF (ID)
C     Reduce the degree of the polynomial by unity
      ID = ID - 1
      ICOF (1) = ICOF (1) + IC
      IF (ID .GT. 1) THEN
C        Synthetic division of the polynomial by the linear factor
         DO 10 I = 2, ID
            ICOF (I) = ICOF (I) + IC * ICOF (I-1)
10       CONTINUE
      ENDIF
      RETURN
      END

C  PURPOSE - THE SUBROUTINE CALCULATES ALL FACTORS OF
C            A POSITIVE INTEGER
C  INPUT PARAMETER
C  NUMBER : THE POSITIVE INTEGER, FACTORS OF WHICH ARE REQUIRED
C  OUTPUT PARAMETERS
C  FACTOR : ONE-DIMENSIONAL ARRAY, CONTAINS ALL THE FACTORS
C  N      : NUMBER OF FACTORS
      SUBROUTINE FAC
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION PRIME (26379), FACTOR (500), FACT (8), POWER (8)
      COMMON/A1/PRIME /B1/ FACTOR, N /C1/FACT, POWER, NOF /D1/NUMBER
C     Special treatment for unity
      IF (NUMBER .EQ. 1) THEN
         N = 1
         FACTOR (1) = 1
      ELSE
C        Find all prime factors of an integer>1 with their powers
         N1 = NUMBER
         J = 1
         NOF = 0
         DIV = 2
10       I = 1
         CALL SQRUT (N1, LIMIT)
20       IF (DIV .GT. LIMIT) THEN
            NOF = NOF + 1
            POWER (NOF) = 1
            FACT (NOF) = N1
         ELSE
30          QUOT = N1 / DIV
            IF (N1 .GT. QUOT * DIV) THEN
               J = J + 1
               DIV = PRIME (J)
           IF (I .EQ. 2) GO TO 10
```

```
               GO TO 20
            ENDIF
            IF (I .EQ. 2) THEN
               POWER (NOF) = POWER (NOF) + 1
            ELSE
               NOF = NOF + 1
               POWER (NOF) = 1
               FACT (NOF) = DIV
               I = 2
            ENDIF
            N1 = QUOT
C           Test for unity
            IF (N1 .GT. 1) GO TO 30
         ENDIF
         M = 1
         N = 1
         FACTOR (1) = 1
         PROD = 1
C        Find all the factors of the integer
40       FACTM = FACT (M)
         POWERM = POWER (M)
         LIMIT = PROD * POWERM
         DO 55 J1 = 1, LIMIT
            N = N + 1
            FACTOR (N) = FACTOR (N - PROD) * FACTM
55       CONTINUE
         IF (M .LT. NOF) THEN
            PROD = PROD + LIMIT
            M = M + 1
            GO TO 40
         ENDIF
C        Arrange the factors in ascending order
         IF (NOF .GT. 1) CALL SORT (FACTOR, N)
      ENDIF
      RETURN
      END

C  PURPOSE :- TO ARRANGE A GIVEN SET OF NUMBERS IN ASCENDING
C             ORDER
C  INPUT AND OUTPUT PARAMETER
C  FAC  INTEGER, ONE DIMENSIONAL ARRAY CONTAINS THE NUMBERS
C  L    THE NUMBER OF NUMBERS

    SUBROUTINE SORT (FAC, L)
    IMPLICIT INTEGER*4 (A-Z)
    DIMENSION FAC (500)
    LM1 = L-1
```

```
C   Compare and rearange the numbers if required
    DO 22 I = 1,LM1
       IP1 = I + 1
       NSMALL = IP1
       SMALL = FAC (NSMALL)
C      Check last but one number is reached
       IF (I .NE. LM1) THEN
          DO 11 K = I + 2,L
             IF (SMALL .GT. FAC (K)) THEN
                NSMALL = K
                SMALL = FAC (NSMALL)
             ENDIF
11        CONTINUE
C         Compare and rearrange if required the last two numbers
       ELSEIF (SMALL .LT. FAC(I)) THEN
          COPY = FAC(I)
          FAC (I) = SMALL
          FAC (NSMALL) = COPY
       ENDIF
22  CONTINUE
    RETURN
    END
```

```
 TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  1
 TYPE THE COEFFICIENTS OF THE POLYNOMIAL FROM 2ND
 ON WARDS IN FORMAT (10I8)
       -1
 ALL THE ROOTS OF THIS POLYNOMIAL ARE  INTEGERS AND ARE
       1
 TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  2
 TYPE THE COEFFICIENTS OF THE POLYNOMIAL FROM 2ND
 ON WARDS IN FORMAT (10I8)
       -4       4
 ALL THE ROOTS OF THIS POLYNOMIAL ARE  INTEGERS AND ARE
       2      2
 TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  3
 TYPE THE COEFFICIENTS OF THE POLYNOMIAL FROM 2ND
 ON WARDS IN FORMAT (10I8)
       -6      11      -6
 ALL THE ROOTS OF THIS POLYNOMIAL ARE  INTEGERS AND ARE
       1      2      3
```

```
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 4
TYPE THE COEFFICIENTS OF THE POLYNOMIAL FROM 2ND
ON WARDS IN FORMAT (10I8)
     -10      35     -50      24
ALL THE ROOTS OF THIS POLYNOMIAL ARE  INTEGERS AND ARE
      1      2      4      3
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 5
TYPE THE COEFFICIENTS OF THE POLYNOMIAL FROM 2ND
ON WARDS IN FORMAT (10I8)
     -15      85    -225     274    -120
ALL THE ROOTS OF THIS POLYNOMIAL ARE  INTEGERS AND ARE
      1      2      4      3      5
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 5
TYPE THE COEFFICIENTS OF THE POLYNOMIAL FROM 2ND
ON WARDS IN FORMAT (10I8)
       2      -3     -15      32     -12
THIS POLYNOMIAL HAS NO INTEGRAL ROOTS
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 8
TYPE THE COEFFICIENTS OF THE POLYNOMIAL FROM 2ND
ON WARDS IN FORMAT (10I8)
      -2      -4       8       5     -14      -2      16       8
THIS POLYNOMIAL HAS  2 ROOTS WHICH ARE NOT INTEGERS
THE  6 INTEGRAL ROOTS OF THIS POLYNOMIAL ARE
     -1     -1     -1     -1      2      2
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
-2
DATA ILLEGAL, KINDLY TYPE AGAIN
25
THE GIVEN NUMBER WILL CROSS THE DIMENSION, CHANGE THE DIMENSION
```

5.12 The program to calculate the H.C.F. of two polynomials over the unique factorization domain Z

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION U (ND), V (ND), POHCF (ND)
10    WRITE (6, '('' TYPE THE DEGREES OF THE TWO POLYNOMIALS''
```

```
     1,1X, ''IN FORMAT (2I2)'')')
      READ (4, '(2I2)') NCF, NCS
      WRITE (6, '(1X, 2I2)') NCF, NCS
      NCF = NCF + 1
      NCS = NCS + 1
      IF (NCF .EQ. 0 .AND. NCS .EQ. 0) STOP
      IF (NCF .LE. 0 .OR. NCS .LE. 0) THEN
         WRITE (6, '('' THE DEGREES OF THE POLYNOMIALS CANNOT''
     1   ,1X, ''BE NEGATIVE'')')
         GO TO 10
      ELSEIF (NCF .GE. 10 .OR. NCS .GE. 10) THEN
         WRITE (6, '( '' THE DEGREE(S) OF SOME POLYNOMIAL(S)''
     1   ,1X, ''EXCEED(S) THE GIVEN DIMENSION'')')
         STOP
      ELSE
20       WRITE (6, '('' TYPE THE COEFFICIENTS OF THE FIRST''
     1   ,1X,''POLYNOMIAL IN FORMAT (10I8)'')')
         READ (4, '(10I8)') (U (I), I = NCF, 1, -1)
         WRITE (6, '(1X, 10I8)') (U (I), I = NCF, 1, -1)
         IF (U (NCF) .EQ. 0) THEN
            WRITE (6, '('' THE LEADING COEFFICIENT OF THE''
     1      ,1X,''FIRST POLYNOMIAL CANNOT BE ZERO'')')
            GO TO 20
         ENDIF
30       WRITE (6, '('' TYPE THE COEFFICIENTS OF THE SECOND''
     1   ,1X, ''POLYNOMIAL IN FORMAT (10I8)'')')
         READ (4, '(10I8)') (V (I), I = NCS, 1, -1)
         WRITE (6, '(1X, 10I8)') (V (I), I = NCS, 1, -1)
         IF (V (NCS) .EQ. 0) THEN
            WRITE (6, '('' THE LEADING COEFFICIENT OF THE''
     1      ,1X,''SECOND POLYNOMIAL CANNOT BE ZERO'')')
            GO TO 30
         ENDIF
      ENDIF
      CALL POLHCF (U, NCF, V, NCS, POHCF, ICS2)
      WRITE (6, '('' HCF OF THE POLYNOMIALS IS'')')
      WRITE (6, '(1X, 10I8)') (POHCF (I), I = ICS2, 1, -1)
      GO TO 10
      END

C  PURPOSE - THIS PROGRAM CALCULATES THE HCF OF TWO POLYNOMIALS OVER
C            THE UNIQUE FACTORIZATION DOMAIN Z
C  INPUT PARAMETERS
C  U  : LINEAR ARRAY, THE FIRST POLYNOMIAL
C  N  : NUMBER OF COEFFICIENTS IN THE FIRST POLYNOMIAL
C  V  : LINEAR ARRAY, THE SECOND POLYNOMIAL
C  NV : NUMBER OF COEFFICIENTS IN THE SECOND POLYNOMIAL
```

```
C  OUTPUT PARAMETES
C  POHCF : LINEAR ARRAY, THE HCF OF THE POLYNOMIALS
C  ICS2  : THE NUMBER OF COEFFCIENTS OF THE HCF OF THE POLYNOMIALS
      SUBROUTINE POLHCF (U, N, V, NV, POHCF, ICS2)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION U (N), V (NV), R (ND), POHCF (ND), DEND (ND)
      IF (NV .GT. N) THEN
         DO 11 I = 1, NV
            DEND (I) = V (I)
11       CONTINUE
         ICF1 = NV
         DO 22 I = 1, N
            POHCF (I) = U (I)
22       CONTINUE
         ICS2 = N
      ELSE
         DO 44 I = 1, N
            DEND (I) = U (I)
44       CONTINUE
         ICF1 = N
         DO 55 I = 1, NV
            POHCF (I) = V (I)
55       CONTINUE
         ICS2 = NV
      ENDIF
      CALL REMCOM ( POHCF, ICS2)
      CALL REMCOM (DEND, ICF1)
60    CALL POLDI (DEND, ICF1, POHCF, ICS2, R, ICR)
      IF (ICR .GT. 1) THEN
C        Change the divisor and dividend
         DO 77 I = ICS2, 1, -1
            DEND (I) = POHCF (I)
77       CONTINUE
         DO 88 I =  ICR, 1, -1
            POHCF (I) = R (I)
88       CONTINUE
         ICF1 = ICS2
         ICS2 = ICR
         GO TO 60
      ELSEIF (ICR .EQ. 1) THEN
         POHCF (1) = 1
         ICS2 = 1
      ENDIF
      RETURN
      END
```

```
C PURPOSE - THIS SUBROUTINE CALCULATES THE REMAINDER POLYNOMIAL BY
C           PSEUDO-DIVISION OF A POLYNOMIAL BY ANOTHER POLYNOMIAL
C  INPUT PARAMETERS
C  U  : LINEAR ARRAY, THE DIVIDEND
C  N  : NUMBER OF COEFFICIENTS IN THE DIVIDEND
C  V  : LINEAR ARRAY, THE DIVISOR
C  NV : NUMBER OF COEFFICIENTS IN THE DIVISOR
C  OUTPUT PARAMETES
C  R  : LINEAR ARRAY, THE REMAINDER AFTER THE PSEUDO-DIVISION
C  NR : THE NUMBER OF COEFFICIENTS IN THE REMAINDER
      SUBROUTINE POLDI (U, N, V, NV, R, NR)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION U (N), V (NV), R (N)
      DO 11 J = 1, N
         R (J) = U (J)
11    CONTINUE
      DO 44 K = N - NV, 0, -1
         NVPK = NV + K
C        Stop explosion of the coefficients
         IHCF = HCF (R (NVPK), V (NV))
         LCDEND = R (NVPK) / IHCF
         LCISOR = V (NV) / IHCF
         DO 22 I = NVPK - 1, 1, -1
            R (I) = LCISOR * R (I)
22       CONTINUE
         DO 33 J = NVPK - 1, K + 1, -1
            R (J) = R (J) - LCDEND * V (J - K)
33       CONTINUE
         DO 77 I = NVPK - 1, 1, -1
            IF (R (I) .NE. 0)  THEN
               CALL REMCOM (R, I)
               IF (I .LT. NV) THEN
                  NR = I
                  RETURN
               ENDIF
               GO TO 44
            ENDIF
77       CONTINUE
         NR = 0
         RETURN
44    CONTINUE
      RETURN
      END

 TYPE THE DEGREES OF THE TWO POLYNOMIALS IN FORMAT (2I2)
  1 1
 TYPE THE COEFFICIENTS OF THE FIRST POLYNOMIAL IN FORMAT (10I8)
```

```
     520    1560
TYPE THE COEFFICIENTS OF THE SECOND POLYNOMIAL IN FORMAT (10I8)
     724    2172
HCF OF THE POLYNOMIALS IS
       1      3
TYPE THE DEGREES OF THE TWO POLYNOMIALS IN FORMAT (2I2)
 2 1
TYPE THE COEFFICIENTS OF THE FIRST POLYNOMIAL IN FORMAT (10I8)
     112     560     672
TYPE THE COEFFICIENTS OF THE SECOND POLYNOMIAL IN FORMAT (10I8)
     172     344
HCF OF THE POLYNOMIALS IS
       1       2
TYPE THE DEGREES OF THE TWO POLYNOMIALS IN FORMAT (2I2)
 4 3
TYPE THE COEFFICIENTS OF THE FIRST POLYNOMIAL IN FORMAT (10I8)
       2     -40     300   -1000    1250
TYPE THE COEFFICIENTS OF THE SECOND POLYNOMIAL IN FORMAT (10I8)
       5     -75     375    -625
HCF OF THE POLYNOMIALS IS
       1     -15      75    -125
TYPE THE DEGREES OF THE TWO POLYNOMIALS IN FORMAT (2I2)
 3 4
TYPE THE COEFFICIENTS OF THE FIRST POLYNOMIAL IN FORMAT (10I8)
       5     -75     375    -625
TYPE THE COEFFICIENTS OF THE SECOND POLYNOMIAL IN FORMAT (10I8)
       2     -40     300   -1000    1250
HCF OF THE POLYNOMIALS IS
       1     -15      75    -125
TYPE THE DEGREES OF THE TWO POLYNOMIALS IN FORMAT (2I2)
 3 8
TYPE THE COEFFICIENTS OF THE FIRST POLYNOMIAL IN FORMAT (10I8)
       1       0       2      -2
TYPE THE COEFFICIENTS OF THE SECOND POLYNOMIAL IN FORMAT (10I8)
       1      -2      -4       0     -15      -4       9       2       5
HCF OF THE POLYNOMIALS IS
       1
TYPE THE DEGREES OF THE TWO POLYNOMIALS IN FORMAT (2I2)
 8 4
TYPE THE COEFFICIENTS OF THE FIRST POLYNOMIAL IN FORMAT (10I8)
       8     -16     -32      64      40    -112     -16     128      64
TYPE THE COEFFICIENTS OF THE SECOND POLYNOMIAL IN FORMAT (10I8)
      12     -48      36      48     -48
HCF OF THE POLYNOMIALS IS
       1      -3       0       4
TYPE THE DEGREES OF THE TWO POLYNOMIALS IN FORMAT (2I2)
-2-1
THE DEGREES OF THE POLYNOMIALS CANNOT BE NEGATIVE
TYPE THE DEGREES OF THE TWO POLYNOMIALS IN FORMAT (2I2)
```

```
 25 5
 THE DEGREE(S) OF SOME POLYNOMIAL(S) EXCEED(S) THE GIVEN DIMENSION
```

5.13 Program to find the quadratic factors of a polynomial over Z

```
      IMPLICIT INTEGER*4 (A-Z)
      PARAMETER  (ND = 10)
      DIMENSION PR (26379), ICOF (ND), ICQF (20)
      COMMON /A1/PR
      CALL PRIME (304678)
10    WRITE(6,'('' TYPE THE DIGREE OF THE POLYNOMIAL IN FORMAT (I2)''
     1,/,1X,''FOR STOPPING PLEASE TYPE : 0'')')
20    READ(4, '(I2)') N
      WRITE (6, '(1X, I2)') N
      IF (N .EQ. 0) STOP
      IF (N .LT. 0) THEN
          WRITE(6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
          GO TO 20
      ELSEIF (N .GT. ND) THEN
         WRITE (6, '('' THE GIVEN NUMBER WILL CROSS DIMENSION,''
     1   ,1X,''CHANGE THE DIMENSION'')')
      STOP
      END IF
      READ (4, '(10I7)') (ICOF (I), I = 1, N)
      WRITE(6, '('' THE POLYNOMIAL IS GIVEN BELOW'')')
      WRITE(6, '(1X, 10I8)') (ICOF (I), I = 1, N)
      CALL REIMA (ICOF, N, ICQF, NQF)
      IF (NQF .EQ. 0) THEN
         WRITE (6, '('' THE POLYNOMIAL DOES NOT HAVE A''
     1   ,1X,''QUADRATIC FACTOR'')')
      ELSE
         WRITE (6,  '('' THE COEFFICIENTS OF THE QUADRATIC FACTORS''/
     1   1X,''STARTING FROM THE 2ND ARE GIVEN BELOW'')')
         DO 33 I = 1, NQF, 2
            WRITE (6,  '(1X, 10I4/)')  ICQF (I), ICQF (I+1)
33       CONTINUE
      ENDIF
      GO TO 10
      END

C  PURPOSE- COMPUTATION OF QUADRATIC FACTOR OF A POLYNOMIAL
C  INPUT PARAMETERS
C  ICOF : LINEAR ARRAY, THE COEFFICIENT OF THE POLYNOMIAL
```

```
C  N    : THE DEGREE OF THE POLYNOMIAL
C  OUTPUT PARAMETERS
C  ICQF : LINEAR ARRAY, THE 2ND AND 3RD COEFFICIENTS OF ALL THE
C       : QUADRATIC FACTORS
C  NQF  : THE TOTAL NUMBER OF 2ND AND 3RD COEFFICIENTS
      SUBROUTINE REIMA (ICOF, N, ICQF, NQF)
      IMPLICIT INTEGER*4 (A-Z)
      PARAMETER (ND = 10)
      DIMENSION IFAC (500), ICOF (N), ITZ (ND), ICO (ND), ICQF (20),
     1IQUOT (ND), IREM (ND)
      COMMON /B1/IFAC, NF /D1/KK1
      KK1 =IABS (ICOF (N))
      CALL FAC
      IQ = IFAC (1)
      JJ =  1
      NQF = 0
      DO 11 I = 1, N
         ICO (I) = ICOF (I)
11    CONTINUE
      NC = N
C     Test the necessacity of computing the polynomials in
C     another indeterminant
20    IF (NC .GE. 3) THEN
         CALL CPDP (ICO, NC, IQ, IQUOT, MINUS1, IREM, MINUS2)
C        Check whether all the coefficients of a particular
C        polynomial obtained in the above process are zero
         DO 22 J = 1, MINUS2
            IF (IREM (J) .NE. 0) GO TO 40
22       CONTINUE
         DO 33 J = 1, MINUS1
            IREM (J) = IQUOT (J)
33       CONTINUE
         MINUS2 =  MINUS1
         GO TO 50
40       CALL POLHCF (IQUOT, MINUS1, IREM, MINUS2, IREM, MINUS2)
C        Determine wheather the HCF of the polynomials is other
C        than the constant polynomial
50       IF (MINUS2 .GT. 1) THEN
            IF (MINUS2 .EQ. 2) THEN
               ITZ (1) = - IREM (2) / IREM (1)
               NZ = 1
            ELSE
               MINUS2 = MINUS2 - 1
               DO 66 J = 1,  MINUS2
                  IREM (J) = IREM (J + 1)
66             CONTINUE
```

```
               CALL ZEROS (IREM, MINUS2, ITZ, NZ)
            ENDIF
            IF (NZ .GE. 1) THEN
C              The  2nd and 3rd coefficients of the quadratic
C              factor (if exists)
               DO 77 I = 1, NZ
                  IP = - ITZ (I)
                  ICQF (NQF + 1) = IP
                  NQF = NQF + 2
                  ICQF (NQF) = IQ
                  CALL DIVPMQ (ICO, NC, IP, IQ)
                  IF (NC .LT. 3) GO TO 80
77             CONTINUE
            ENDIF
         ENDIF
         IF (JJ .LE. NF) THEN
            IF (IQ .GT. 0) THEN
               IQ = - IFAC (JJ)
               JJ = JJ + 1
            ELSE
                IQ = IFAC (JJ)
            ENDIF
            GO TO 20
         ENDIF
         RETURN
      ENDIF
80    IF (NC .EQ. 2) THEN
C        The 2nd and 3rd coefficients of the last
C        quadratic factor
         ICQF (NQF + 1) = ICO (1)
         NQF = NQF + 2
         ICQF (NQF) = ICO (2)
      ENDIF
      RETURN
      END

C  PURPOSE- DIVISION OF A POLYNOMIAL BY A MONIC QUADRATIC
C           FACTOR
C  INPUT PARAMETERS
C  ICO   : LINEAR ARRAY, THE COEFFICIENT OF THE POLYNOMIAL
C  NC    : THE NUMBER OF THE COEFFICIENT OF THE POLYNOMIAL
C  IP    : THE 2ND COEFFICIENT OF THE QUADRATIC FACTOR
C  IQ    : THE LAST COEFFICIENT OF THE QUADRATIC FACTOR
C  OUTPUT PARAMETERS
C  ICO   : LINEAR ARRAY, THE COEFFICIENT OF THE POLYNOMIAL
C          AFTER DIVISION
C  NC   : THE NUMBER OF THE COEFFICIENT OF THE POLYNOMIAL
C         AFTER DIVISION
```

```
      SUBROUTINE DIVPMQ (ICO, NC, IP, IQ)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION ICO (NC)
      ICO (1) = ICO (1) - IP
      ICO (2) = ICO (2) - IQ - ICO (1) * IP
      DO 11 I = 3, NC - 2
         ICO(I) = - ICO(I-2)*IQ - ICO(I-1)*IP + ICO(I)
11    CONTINUE
      NC = NC - 2
      RETURN
      END

C  PURPOSE -  THIS PROGRAM READS IN ALL THE COEFFICIENTS OF A MONIC
C             POLYNOMIAL OF ANY DEGREE > 2 AND < N+1, DIVIDES IT BY
C             A QUADRATIC POLYNOMIAL OF THE FORM X**2 - P*X + Q (Q
C             BEING A FACTOR OF THE CONSTANT TERM OF THE ORIGINAL
C             POLYNOMIAL), AND PRINTS OUT ALL THE COEFFICIENTS OF THE
C             2 POLYNOMIALS IN P, WHOSE VALUES MUST SIMULTANEOUSLY
C             VANISH FOR THOSE VALUES OF P FOR WHICH THE ABOVE DIVISION
C             LEAVES NO REMAINDER
C  INPUT PARAMETERS
C  ICO  : LINEAR ARRAY, THE COEFFICIENT OF THE POLYNOMIAL
C  NC   : NUMBER OF COEFFICIENT OF THE POLYNOMIAL
C  LAST : THE FACTOR OF THE LAST NON-ZERO COEFFICIENT
C  OUTPUT PARAMETERS
C  IQUOT  : LINEAR ARRAY, THE COEFFICIENTS OF THE POLYNOMIAL C (P)
C  MINUS1 : NUMBER OF COEFFICIENTS OF THE POLYNOMIAL C (P)
C  IREM   : LINEAR ARRAY, THE COEFFICIENTS OF THE POLYNOMIAL D (P)
C  MINUS2 : NUMBER OF COEFFICIENTS OF THE POLYNOMIAL D (P)
      SUBROUTINE CPDP (ICO, NC, LAST, IQUOT, MINUS1, IREM, MINUS2)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION ICO (NC), IQUOT (NC), IREM (NC)
      ICARE = ICO (NC) / LAST
      IQUOT (1) = ICO (1)
      IF (NC .EQ. 3) THEN
         IREM (1) = ICARE
         IREM (2) = - LAST + ICO (2)
         MINUS2 = NC - 1
      ELSE
         IQUOT (1) = ICO (1)
         MINUS2 = NC - 2
         IREM (1) = - LAST
         IQUOT (2) = - LAST + ICO (2)
         IREM (2) = - LAST * ICO (1) + ICO (3)
         DO 22 K = 1 , NC - 4
            IREM (K + 2) = - LAST * IQUOT (K + 1) + ICO (K + 3)
            IQUOT (K + 2) = IREM (K + 1)
```

```
            IF (K .EQ. 1) THEN
               IREM (2) = - LAST * IQUOT (1)
            ELSE
               IF (K .GE. 3) THEN
                  DO 11 J = K , 3 , - 1
                     IREM (J + 1) = - LAST * IQUOT (J)
                     IQUOT (J + 1) = IQUOT (J + 1) + IREM (J)
11                CONTINUE
               ENDIF
               IQUOT (3) = IQUOT (3) + IREM (2)
               IREM (3) = - LAST * IQUOT (2)
            ENDIF
            IQUOT (2) = IQUOT (2) + IREM (1)
22       CONTINUE
         IREM (NC - 3) = IREM (NC - 3) + ICARE
      ENDIF
      IQUOT (NC - 2) = IQUOT (NC - 2) - ICARE
      MINUS1 = NC - 1
      DO 33 I = MINUS1 - 1, 1, -1
         IQUOT (I + 1) = IQUOT (I)
33    CONTINUE
      IQUOT (1) = 1
      RETURN
      END
```

```
 TYPE THE DIGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  1
 THE POLYNOMIAL IS GIVEN BELOW
       2
 THE POLYNOMIAL DOES NOT HAVE A QUADRATIC FACTOR
 TYPE THE DIGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  2
 THE POLYNOMIAL IS GIVEN BELOW
       2       4
 THE COEFFICIENTS OF THE QUADRATIC FACTORS
 STARTING FROM THE 2ND ARE GIVEN BELOW
    2   4
 TYPE THE DIGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  3
 THE POLYNOMIAL IS GIVEN BELOW
       0      61      24
 THE POLYNOMIAL DOES NOT HAVE A QUADRATIC FACTOR
 TYPE THE DIGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
```

```
 5
THE POLYNOMIAL IS GIVEN BELOW
      4       6       1       4       6
THE COEFFICIENTS OF THE QUADRATIC FACTORS
STARTING FROM THE 2ND ARE GIVEN BELOW
  -1   1
   4   6
TYPE THE DIGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 6
THE POLYNOMIAL IS GIVEN BELOW
       0     -18      16      28     -32       8
THE COEFFICIENTS OF THE QUADRATIC FACTORS
STARTING FROM THE 2ND ARE GIVEN BELOW
  -4   2
   0  -2
   4  -2
TYPE THE DIGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 8
THE POLYNOMIAL IS GIVEN BELOW
      -2      -4       8       5     -14      -2      16       8
THE COEFFICIENTS OF THE QUADRATIC FACTORS
STARTING FROM THE 2ND ARE GIVEN BELOW
   2   1
   2   1
  -2   2
  -4   4
TYPE THE DIGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
15
THE GIVEN NUMBER WILL CROSS DIMENSION, CHANGE THE DIMENSION
```

5.14 This program computes the rational zeros of a polynomial over Z

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER  (ND = 10)
      DIMENSION PR (26379), ICOF (ND), ITZNUM (ND), ITZDEN (ND)
      COMMON /A1/PR
      CALL PRIME (304678)
10    WRITE(6,'('' TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)''/
     11X,''FOR STOPPING PLEASE TYPE : 0'')')
20    READ(4, '(I2)') M
      WRITE (6, '(1X, I2)') M
      IF (M .EQ. 0) STOP
      IF (M .LT. 0) THEN
```

```
          WRITE(6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
          GO TO 20
       ELSEIF (M + 1 .GT. ND) THEN
          WRITE (6,'('' THE GIVEN NUMBER WILL CROSS THE DIMENSION,''
      1   ,1X,''CHANGE THE DIMENSION'')')
          STOP
       ENDIF
       NC = M + 1
       WRITE(6,'('' TYPE THE COEFFICIENTS OF THE POLYNOMIAL''
      1,1X,''IN FORMAT (10I8)'')')
30     READ (4, '(10I8)') (ICOF (I), I = 1, NC)
       WRITE(6, '(1X, 10I8)') (ICOF (I), I = 1, NC)
       IF (ICOF (1) .EQ. 0) THEN
          WRITE(6,'('' THE FIRST COEFFICIENT OF THE POLYNOMIAL''
      1   ,1X,'' CANNOT BE ZERO''/1X,''TYPE THE COEFFICIENT OF THE''
      1   ,1X,''POLYNOMIAL AGAIN'')')
          GO TO 30
       ENDIF
       CALL ZEROSQ (ICOF, NC, ITZNUM, ITZDEN, NZ)
       IF (NZ .EQ. 0) THEN
          WRITE(6,'('' THIS POLYNOMIAL HAS NO RATIONAL ZEROS'')')
       ELSE
          IF (NZ .LT. M) THEN
             WRITE (6,'('' ONLY'',I2,'' ZEROS OF THE POLYNOMIAL''
      1      ,1X,''ARE RATIONAL'')') NZ
          ELSE
             WRITE (6,'('' ALL ZEROS OF THE POLYNOMIAL''
      1      ,1X,''ARE RATIONAL'')')
          ENDIF
          WRITE (6,'('' THE NUMERATOR OF THE ZEROS''
      1   ,1X,''ARE''/1X,10I8)')  (ITZNUM (KK), KK = 1, NZ)
          WRITE (6,'('' THE DENOMINATOR OF THE ZEROS''
      1   ,1X,''ARE''/1X,10I8)') (ITZDEN (KK), KK = 1, NZ)
       ENDIF
       GO TO 10
       END

C PURPOSE - THIS PROGRAM COMPUTES THE RATIONAL ZEROS OF THE
C           POLYNOMIAL
C  INPUT PARAMETERS
C  ICOF : LINEAR ARRAY, COEFFICIENTS OF THE POLYNOMIAL
C  NC   : THE NUMBER OF THE COEFFICIENT OF THE POLYNOMIAL
C  OUTPUT PARAMETERS
C  ITZNUM :  LINEAR ARRAY, THE NUMERATOR OF THE ZEROS
C  ITZDEN :  LINEAR ARRAY, THE DENOMINATOR OF THE ZEROS
C  NZ     :  THE NUMBER OF ZEROS
       SUBROUTINE ZEROSQ (ICOF, NC, ITZNUM, ITZDEN, NZ)
```

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER  (ND = 10)
      DIMENSION ICOF (NC), ITZNUM (NC), ITZDEN (NC), NEWCOF (ND)
      CALL LEASTK (ICOF, NC, LN)
      DO 11 I = 2, NC
         ICOF (I) = ICOF (I) * LN ** (I - 1)
11    CONTINUE
      KDIV = ICOF (1)
      DO 22 I = 1, NC - 1
         NEWCOF (I) = ICOF (I + 1) / KDIV
22    CONTINUE
      CALL ZEROS (NEWCOF, NC-1, ITZNUM, NZ)
      IF (NZ .EQ. 0) RETURN
      DO 33 I = 1, NZ
         NHCF = HCF (ITZNUM (I), LN)
         ITZNUM (I) = ITZNUM (I) / NHCF
         ITZDEN (I) = LN / NHCF
33    CONTINUE
      RETURN
      END

C PURPOSE - THE PROGRAM FINDS THE LEAST NATURAL NUMBER BY
C           WHICH ALL ZEROS OF A POLYNOMIAL SHOULD BE MULTIPLIED
C           SO THAT THE POLYNOMIAL RECONSTRUCTED FROM NEW ZEROS
C           MIGHT BE MONIC
C  INPUT PARAMETERS
C  ICOF : LINEAR ARRAY, COEFFICIENTS OF THE POLYNOMIAL
C  NC   : THE NUMBER OF THE COEFFICIENT OF THE POLYNOMIAL
C  OUTPUT PARAMETER
C  LN   :  THE REQUIRED LEAST NUMBER
      SUBROUTINE LEASTK (ICOF, NC, LN)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION PR (26379), ICOF (NC), INCO (ND), IFAC (8), POWER (8)
      COMMON /A1/PR /C1/IFAC, POWER, NT /D1/KK
      LN = 1
C     If the leading coefficient is 1 or - 1, then LN = 1
      IF (IABS (ICOF (1)) .EQ. 1) RETURN
      DO 11 J = 1, NC
         INCO (J) = ICOF (J)
11    CONTINUE
      CALL REMCOM (INCO,  NC)
      IF ( 1 .EQ. IABS (INCO(1))) RETURN
      KK = IABS (INCO (1))
      CALL FAC
C     For a particular prime factor, calculate the least power
      DO 44 K = 1, NT
```

```
         KF = 1
         IP = IFAC (K)
         IE = POWER (K)
         DO 22 I = 2, NC
            ICAR = I - 1
            IF (ICAR .EQ. IE) GO TO 30
            IF (INCO (I) .NE. 0) THEN
               M = 0
               ITEST = INCO (I)
10             ICHEK = ITEST / IP
               IF (ITEST .NE. IP * ICHEK) THEN
C                 Calculate the highest power of the current
C                 prime factor
                  L = (IE - M) / (I - 1)
                  IF (L * (I - 1) .NE. IE - M) L = L + 1
                  IF (L .GT. KF) KF = L
               ELSE
                  ITEST = ICHEK
                  M = M + 1
C                 Check whether the coefficient plus the number of
C                 times it is already divisible is equal to the power
C                 of the factor
                  IF (M + ICAR .NE. IE) GO TO 10
               ENDIF
            ENDIF
22       CONTINUE
30       LN = LN * IP ** KF
44    CONTINUE
      RETURN
      END
```

```
 TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  2
 TYPE THE COEFFICIENTS OF THE POLYNOMIAL IN FORMAT (10I8)
       60    -414     156
 ALL ZEROS OF THE POLYNOMIAL ARE RATIONAL
 THE NUMERATOR OF THE ZEROS ARE
        2      13
 THE DENOMINATOR OF THE ZEROS ARE
        5       2
 TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
 FOR STOPPING PLEASE TYPE : 0
  3
 TYPE THE COEFFICIENTS OF THE POLYNOMIAL IN FORMAT (10I8)
       50     175     175     125
 ONLY 1 ZEROS OF THE POLYNOMIAL ARE RATIONAL
```

```
THE NUMERATOR OF THE ZEROS ARE
      -5
THE DENOMINATOR OF THE ZEROS ARE
       2
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 4
TYPE THE COEFFICIENTS OF THE POLYNOMIAL IN FORMAT (10I8)
      24      14     -56      24       0
ALL ZEROS OF THE POLYNOMIAL ARE RATIONAL
THE NUMERATOR OF THE ZEROS ARE
       0       2      -2       3
THE DENOMINATOR OF THE ZEROS ARE
       1       3       1       4
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 5
TYPE THE COEFFICIENTS OF THE POLYNOMIAL IN FORMAT (10I8)
       4     -72      64     112    -128      32
THIS POLYNOMIAL HAS NO RATIONAL ZEROS
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
 7
TYPE THE COEFFICIENTS OF THE POLYNOMIAL IN FORMAT (10I8)
       2      -1     -21       1      55       0     -36       0
ALL ZEROS OF THE POLYNOMIAL ARE RATIONAL
THE NUMERATOR OF THE ZEROS ARE
       0       1      -1      -2      -2       3       3
THE DENOMINATOR OF THE ZEROS ARE
       1       1       1       1       1       2       1
TYPE THE DEGREE OF THE POLYNOMIAL IN FORMAT (I2)
FOR STOPPING PLEASE TYPE : 0
-2
DATA ILLEGAL, KINDLY TYPE AGAIN
25
THE GIVEN NUMBER WILL CROSS THE DIMENSION, CHANGE THE DIMENSION
```

5.15 Given a square matrix, this program computes the integer eigenvalues of the matrix

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION MAT (ND, ND), P (ND), PR (26379),  ITZ (ND)
      COMMON /A1/PR
      CALL PRIME (304678)
10    WRITE(6,'('' TYPE THE ORDER OF THE MATRIX IN FORMAT(I2)''
     1/1X,''FOR STOPPING PLEASE TYPE : 0'')')
```

```
20    READ(4, '(I2)') N
      WRITE (6, '(1X, I2)') N
      IF (N .EQ. 0) STOP
      IF (N .LT. 0) THEN
          WRITE(6,'('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
          GO TO 20
      ELSEIF (N .GT. ND) THEN
         WRITE (6,'('' THE GIVEN NUMBER WILL CROSS THE DIMENSION,''
     1             1X,''CHANGE THE DIMENSION'')')
         STOP
      ENDIF
      WRITE (6,'('' TYPE THE ENTRIES OF THE MATRIX IN FORMAT (10I5)'')')
      DO 33 I = 1, N
         READ (4, '(10I5)') (MAT (I, JJ), JJ = 1, N)
33    CONTINUE
      DO 44 MM = 1, N
         WRITE (6, '(1X, 10I5)') (MAT (MM, KK), KK = 1, N)
44    CONTINUE
      CALL CHPOLY (MAT, N, P)
      WRITE (6,'('' THE COEFFICIENTS OF THE CHARACTERISITC POLYNOMIAL''
     1/1X,''STARTING FROM THE 2ND ARE GIVEN BELOW'')')
      WRITE(6, '(10(1X, I7))') (P (I), I = 1, N)
      M = N
      CALL ZEROS (P, M, ITZ, NZ)
      IF (NZ .EQ. 0) THEN
         WRITE(6,'('' THIS MATRIX HAS NO INTEGRAL EIGENVALUES'')')
      ELSEIF (NZ .LT. N) THEN
         NUMZ = N - NZ
         WRITE (6,'('' THIS MATRIX HAS '',I2,'' EIGENVALUES''
     1   ,1X,''WHICH ARE NOT INTEGERS'')') NUMZ
         WRITE (6,'('' THE '',I2,'' INTEGRAL EIGENVALUES OF THIS''
     1   ,1X,''MATRIX ARE''/1X,10I7)') NZ, (ITZ(KK), KK = 1, NZ)
      ELSE
         WRITE(6,'('' ALL THE EIGENVALUES OF THIS MATRIX ARE''
     1   ,1X,''INTEGERS AND ARE''/1X,10I7)') (ITZ(KK), KK = 1, NZ)
      ENDIF
      GO TO 10
      END

C PURPOSE- GIVEN A SQUARE MATRIX, THIS SUBROUTINE COMPUTES
C          THE CHARACTERISTIC POLYNOMIAL OF THE MATRIX BY
C          LEVERRIER-FADEEV METHOD
C  INPUT PARAMETERS
C  MAT: TWO DIMENSIONAL ARRAY, THE GIVEN MATRIX
C  N  : THE ORDER OF THE MATRIX
C  OUTPUT PARAMETERS
C  P  : ONE DIMENSIONAL ARRAY, THE CHAARACTERISTIC POLYNOMIAL
```

```
      SUBROUTINE CHPOLY (MAT, N, P)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION MAT (ND, ND), B (ND, ND), P (ND)
      DO 22 I = 1, N
         DO 11 J = 1, N
            B (I, J) = MAT (I, J)
11       CONTINUE
22    CONTINUE
      DO 55 K = 1, N
         SIGMA = 0
         DO 33 I = 1, N
            SIGMA = SIGMA - B (I, I)
33       CONTINUE
         P (K) = SIGMA / K
         DO 44 I = 1, N
            B (I, I) = B (I, I) + P (K)
44       CONTINUE
         CALL MULT (MAT, B, N)
55    CONTINUE
      RETURN
      END

C PURPOSE - TO MULTIPLY TWO SQUARE MATRICES
C  INPUT PARAMETERS
C  X  : TWO DIMENSIONAL ARRAY, THE FIRST MATRIX
C  Y  : TWO DIMENSIONAL ARRAY, THE SECOND MATRIX
C  N  : THE ORDER OF THE MATRIX
C  OUTPUT PARAMETER
C  Y  : TWO DIMENSIONAL ARRAY, THE PRODUCT OF THE MATRIX
      SUBROUTINE MULT (X, Y, N)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION X (ND, ND), Y (ND, ND), Z (ND, ND)
      DO 33 I = 1, N
         DO 22 J = 1, N
            SUM = 0
            DO 11 K = 1, N
               SUM = SUM + X (I, K) * Y (K, J)
11          CONTINUE
            Z (I, J) = SUM
22       CONTINUE
33    CONTINUE
      DO 55 I = 1, N
         DO 44 J = 1, N
            Y (I, J) = Z (I, J)
44       CONTINUE
```

```
55    CONTINUE
      RETURN
      END

 TYPE THE ORDER OF THE MATRIX IN FORMAT(I2)
 FOR STOPPING PLEASE TYPE : 0
  1
 TYPE THE ENTRIES OF THE MATRIX IN FORMAT (10I5)
   100
 THE COEFFICIENTS OF THE CHARACTERISITC POLYNOMIAL
 STARTING FROM THE 2ND ARE GIVEN BELOW
    -100
 ALL THE EIGENVALUES OF THIS MATRIX ARE INTEGERS AND ARE
     100
 TYPE THE ORDER OF THE MATRIX IN FORMAT(I2)
 FOR STOPPING PLEASE TYPE : 0
  2
 TYPE THE ENTRIES OF THE MATRIX IN FORMAT (10I5)
     5   -3
     2   10
 THE COEFFICIENTS OF THE CHARACTERISITC POLYNOMIAL
 STARTING FROM THE 2ND ARE GIVEN BELOW
     -15      56
 ALL THE EIGENVALUES OF THIS MATRIX ARE INTEGERS AND ARE
       8      7
 TYPE THE ORDER OF THE MATRIX IN FORMAT(I2)
 FOR STOPPING PLEASE TYPE : 0
  3
 TYPE THE ENTRIES OF THE MATRIX IN FORMAT (10I5)
    18   28    4
     6   18   16
   -20  -44  -26
 THE COEFFICIENTS OF THE CHARACTERISITC POLYNOMIAL
 STARTING FROM THE 2ND ARE GIVEN BELOW
     -10       4     -40
 THIS MATRIX HAS  2 EIGENVALUES WHICH ARE NOT INTEGERS
 THE  1 INTEGRAL EIGENVALUES OF THIS MATRIX ARE
      10
 TYPE THE ORDER OF THE MATRIX IN FORMAT(I2)
 FOR STOPPING PLEASE TYPE : 0
  4
 TYPE THE ENTRIES OF THE MATRIX IN FORMAT (10I5)
     3    6    2    5
     6    3    2    5
     2    2   10    2
     5    5    2    4
 THE COEFFICIENTS OF THE CHARACTERISITC POLYNOMIAL
 STARTING FROM THE 2ND ARE GIVEN BELOW
```

```
    -20      35     440     384
 ALL THE EIGENVALUES OF THIS MATRIX ARE INTEGERS AND ARE
     -1       8      16      -3
 TYPE THE ORDER OF THE MATRIX IN FORMAT(I2)
 FOR STOPPING PLEASE TYPE : 0
  5
 TYPE THE ENTRIES OF THE MATRIX IN FORMAT (10I5)
    7    4    3   -2   -3
    0    5    0    0    0
   -2   -4    2    2    3
    2    4    3    3   -3
   -2   -4   -3    2    8
 THE COEFFICIENTS OF THE CHARACTERISITC POLYNOMIAL
 STARTING FROM THE 2ND ARE GIVEN BELOW
    -25     250   -1250    3125   -3125
 ALL THE EIGENVALUES OF THIS MATRIX ARE INTEGERS AND ARE
      5       5       5       5       5
 TYPE THE ORDER OF THE MATRIX IN FORMAT(I2)
 FOR STOPPING PLEASE TYPE : 0
 -5
 DATA ILLEGAL, KINDLY TYPE AGAIN
  6
 TYPE THE ENTRIES OF THE MATRIX IN FORMAT (10I5)
    0    0    3    4    5    2
    0    0    4    5    1    1
    0    0    1    3    4    4
    4    0    2    4    8    5
    5    0    7    1    3    3
    6    7    1    2    3    2
 THE COEFFICIENTS OF THE CHARACTERISITC POLYNOMIAL
 STARTING FROM THE 2ND ARE GIVEN BELOW
    -10     -90    -570    -997   -1851  -10080
 THIS MATRIX HAS NO INTEGRAL EIGENVALUES
 TYPE THE ORDER OF THE MATRIX IN FORMAT(I2)
 FOR STOPPING PLEASE TYPE : 0
  0
```

5.16 This program computes the eigenvectors of a given matrix

```
      IMPLICIT INTEGER*4 (A-Z)
      PARAMETER (ND = 15)
      DIMENSION B (ND, ND), EV (ND), U (ND), V (ND)
      COMMON A (ND, 16), ADEN (ND, 16), X (ND, ND), XDEN (ND, ND),
     1M, N, P, IND
10    WRITE (6, '('' TYPE THE ORDER OF THE MATRIX AND THE NUMBER''
     1/1X,''OF DISTINCT EIGENVALUES IN FORMAT (2I2)'')')
```

```
      READ (4, '(2I2)') N, NEQ
      WRITE (6, '(1X, 2I2)')  N, NEQ
      IF (N .EQ. 0) STOP
      IF (N .LT. 0) THEN
         WRITE (6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
         GO TO 10
      ELSEIF (N .GT. ND) THEN
         WRITE (6, '('' ORDER OF THE MATRIX, TOO LARGE KINDLY''
     1   1X,''TYPE AGAIN'')')
         GO TO 10
      ENDIF
      M = N
      NS = 0
      NP1 = N + 1
      WRITE (6,'('' TYPE THE COEFFICIENTS OF THE MATRIX IN''
     1,1X,''FORMAT (15I4)'')')
      DO 22 I = 1, N
         READ (4, '(15I4)') (B (I, J), J = 1, N)
22    CONTINUE
      DO 33 I = 1, N
         WRITE (6, '(1X, 15I4)') (B (I, J), J = 1, N)
33    CONTINUE
      WRITE (6,'('' TYPE THE DISTINCT EIGENVALUES IN FORMAT (15I4)'')')
      READ (4, '(15I4)') (EV (I), I = 1, NEQ)
      WRITE (6, '(1X, 15I4)') (EV (I), I = 1, NEQ)
40    NS = NS + 1
      IF (NS .GT. NEQ) GO TO 10
      CALL FECEIV (B, EV, N, NS, A)
      DO 77 I = 1, N
         DO 66 J = 1, N + 1
            ADEN (I, J) = 1
66       CONTINUE
77    CONTINUE
      CALL SOLVE
      IF (IND .EQ. 1) THEN
         WRITE (6, '('' THERE IS NO SOLUTION'')')
      ELSE
         WRITE (6, '('' THE CORRESPONDING EIGENVECTORS ARE'')')
         P = P + 1
         DO 122 K = 2, P
            DO 88 J = 1, N
               U (J) = XDEN (J, K)
               V (J) = X (J, K)
88          CONTINUE
            CALL MULLCM (U, V, N)
            CALL REMCOM (V, N)
```

```
            DO 111 I = 1, N
               IF (V (I) .GT. 0) GO TO 120
               IF (V (I) .LT. 0) THEN
                  DO 99 J = I, N
                     V (J) = - V (J)
99                CONTINUE
                  GO TO 120
               ENDIF
111         CONTINUE
120         WRITE (6, '(1X, 15I4)') (V (I), I = 1, N)
122      CONTINUE
      ENDIF
      GO TO 40
      END

C  PURPOSE- TO FORM AN EQUATION  AX = LX
C  INPUT PARAMETERS
C  B    : TWO-DIMENSIONAL ARRAY, THE GIVEN MATRIX
C  EV   : LINEAR ARRAY,  THE EIGENVALUES
C  N    : THE ORDER OF THE MATRIX
C  NS   : THE NUMBER OF THE EIGENVALUE
C OUTPUT PARAMETER
C A     : TWO-DIMENSIONAL ARRAY, THE COEFFICIENTS OF THE MATRIX
C         EQUATIONS
      SUBROUTINE FECEIV (B, EV, N, NS, A)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 15)
      DIMENSION B (ND, ND), A (ND, 16), EV (ND)
      DO 22 I = 1, N
         DO 11 J = 1, N
            IF (I .EQ. J) THEN
               A (I, J) = B (I, J) - EV (NS)
            ELSE
               A (I, J) = B (I, J)
            ENDIF
11       CONTINUE
22    CONTINUE
      DO 33 I = 1, N
         A (I, N + 1) = 0
33    CONTINUE
      RETURN
      END

C PURPOSE - TO SOLVE A SYSTEM OF M LINEAR EQUATIONS IN N UNKNOWNS
C  INPUT PARAMETERS
C  A    : TWO-DIMENSIONAL ARRAY, NUMERATOR OF THE COEFFICIENTS
C  ADEN : TWO-DIMENSIONAL ARRAY, DENOMINATOR OF THE COEFFICIENTS
C  M    : THE NUMBER OF EQUATIONS
```

```
C  N    : THE NUMBER OF UNKNOWNS
C  OUTPUT PARAMETERS
C  X    : TWO-DIMENSIONAL ARRAY, THE NUMERATOR OF THE SOLUTIONS
C  XDEN : TWO-DIMENSIONAL ARRAY, THE DENOMINATOR OF THE SOLUTIONS
C  P    = N - R; R BEING THE RANK OF THE  MxN MATRIX
C  IND = 1, IF THERE IS NO SOLUTION
C      = 0, IF THERE EXIST A SOLUTION
       SUBROUTINE SOLVE
       IMPLICIT INTEGER*4 (A-Z)
       PARAMETER (ND = 15)
       COMMON A (ND, 16), ADEN (ND, 16), X (ND, ND), XDEN (ND, ND),
      1M, N, P, IND
       DIMENSION LOC (ND), ROW (ND), COL (ND), T (ND)
       DO 22 II=1, ND
          LOC (II) = 0
22     CONTINUE
       NP1 = N + 1
       RANK = 0
       P = 0
C      Test consistency condition by determining rank
       DO 88 I = 1, NP1
          DO 33 L = 1, M
             IF (LOC (L) .NE. 1) THEN
                IF (A (L, I) .NE. 0) GO TO 40
             ENDIF
33        CONTINUE
          IF (I .EQ. NP1) GO TO 120
          P = P + 1
          T (P) = I
          GO TO 88
40        IF (I .EQ. NP1) THEN
C            The system is not consistent
             IND = 1
             RETURN
          ENDIF
          RANK = RANK + 1
          ROW (RANK) = L
          COL (RANK) = I
          LOC (L) = 1
          CALL RECIP (A(L,I), ADEN(L,I), RECNUM, RECDEN)
          IP1 = I + 1
          DO 55 K = IP1, NP1
             IF (A (L, K) .NE. 0) THEN
                CALL MULTN (A (L, K), ADEN (L, K), RECNUM, RECDEN,
      1         A (L, K), ADEN (L, K))
             ENDIF
```

```
55        CONTINUE
C         The system is consitent, find the solution
          IF (RANK .EQ. M) GO TO 90
          DO 77 J = L+1, M
             IF (LOC (J) .NE. 1 .AND. A(J, I) .NE. 0) THEN
                DO 66 K = IP1, NP1
                   IF (A (L, K) .NE. 0) THEN
                      CALL MULTN (A (J, I), ADEN (J, I), A (L, K),
     1                ADEN (L,K), TEMP1, TEMP2)
                      CALL SUBTN (A (J,K), ADEN(J,K), TEMP1, TEMP2,
     1                A (J, K), ADEN (J, K))
                   ENDIF
66              CONTINUE
             ENDIF
77        CONTINUE
88     CONTINUE
90     IF (I .NE. N) THEN
C         Initilize the identity matrix
          DO 111 INDEX = IP1, N
             P = P + 1
             T (P) = INDEX
111       CONTINUE
       ENDIF
120    IND = 0
       IF (RANK .EQ. 0) THEN
          DO 144 I = 1, N
             X (I, I) = 1
             XDEN (I, I) = 1
             IF (N .NE. 1) THEN
                DO 133  J = 1, N
                   IF (J .NE. I) THEN
                      X (J, I) = 0
                      XDEN (J, I) = 1
                   ENDIF
133             CONTINUE
             ENDIF
144       CONTINUE
          RETURN
       ENDIF
       DO 166 I = 1, RANK
          NMI = RANK + 1 - I
          L = ROW (NMI)
          Y = COL (NMI)
          IF (I .NE. 1) THEN
             IM1 = I - 1
             DO 155 K = 1,  IM1
```

```
                Z = COL (NMI + K)
                IF (A (L, Z) .NE. 0 .AND. X(Z, 1) .NE. 0) THEN
                   CALL MULTN (A (L, Z), ADEN (L, Z), X (Z, 1),
     1             XDEN (Z, 1), TEMP1, TEMP2)
                   CALL SUBTN (A (L, NP1), ADEN (L, NP1), TEMP1,
     1             TEMP2, A (L, NP1), ADEN (L, NP1))
                ENDIF
155          CONTINUE
          ENDIF
          X (Y, 1) = A (L, NP1)
          XDEN (Y, 1) = ADEN (L, NP1)
166    CONTINUE
C      Test for the existence of unique solution
       IF (P .EQ. 0) RETURN
       DO 177 I = 1, P
          X (T (I), 1) = 0
          XDEN (T (I), 1) = 1
177    CONTINUE
       DO 222 INDEX = 1, P
          INDP1 = INDEX + 1
          R = T (INDEX)
          DO 199 I =1, RANK
             NMI = RANK + 1 -I
             L = ROW (NMI)
             Y = COL (NMI)
             IF (Y .GE. R) THEN
                X (Y, INDP1) = 0
                XDEN (Y, INDP1) = 1
                GO TO 199
             ENDIF
             IF (I .NE. 1) THEN
                IM1 = I - 1
                DO 188 K = 1, IM1
                   Z = COL (NMI + K)
                   IF (A (L, Z) .NE. 0 .AND. X (Z, INDP1) .NE. 0) THEN
                      CALL MULTN (A (L, Z), ADEN (L, Z), X (Z, INDP1),
     1                XDEN (Z, INDP1), TEMP1, TEMP2)
                      CALL SUBTN (A (L, R), ADEN (L, R), -TEMP1, TEMP2,
     1                A (L, R), ADEN (L, R))
                   ENDIF
188             CONTINUE
             ENDIF
             X (Y, INDP1) = - A (L, R)
             XDEN (Y, INDP1) = ADEN (L, R)
199       CONTINUE
          X (R, INDP1) = 1
```

```
         XDEN (R, INDP1) = 1
         IF (P .NE. 1) THEN
            DO 211 J = 1, P
               IF (J .NE. INDEX) THEN
                  X (T(J), INDP1) = 0
                  XDEN (T(J), INDP1) = 1
               ENDIF
211         CONTINUE
         ENDIF
222   CONTINUE
      RETURN
      END

C  PURPOSE - TO REDUCE A VECTOR WITH RATIONAL COEFFICIENTS
C            TO A VECTOR WITH INTEGER COEFFICIENTS
C  INPUT PARAMETERS
C  U    : LINEAR ARRAY, THE RATIONAL VECTOR
C  NCF : THE NUMBER OF COMPONENTS OF THE VECTOR
C OUTPUT PARAMETER
C V     : LINEAR ARRAY, THE INTEGER VECTOR
      SUBROUTINE MULLCM (U, V, NCF)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION U (NCF), V (NCF)
      ICOHCF = IABS (U (1))
      LCM = U (1)
      DO 22 I = 2, NCF
         ICOHCF = HCF (ICOHCF, U (I))
         LCM = IABS (LCM * U (I) / ICOHCF)
22    CONTINUE
      DO 33 I = 1, NCF
         V (I) = V (I) * LCM / U (I)
33    CONTINUE
      RETURN
      END

 TYPE THE ORDER OF THE MATRIX AND THE NUMBER
 OF DISTINCT EIGENVALUES IN FORMAT (2I2)
  2 1
 TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
    8   9
   -4  -4
 TYPE THE DISTINCT EIGENVALUES IN FORMAT (15I4)
    2
 THE CORRESPONDING EIGENVECTORS ARE
    3  -2
 TYPE THE ORDER OF THE MATRIX AND THE NUMBER
 OF DISTINCT EIGENVALUES IN FORMAT (2I2)
  3 2
```

```
TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
   6  -2   2
  -2   3  -1
   2  -1   3
TYPE THE DISTINCT EIGENVALUES IN FORMAT (15I4)
   2   8
THE CORRESPONDING EIGENVECTORS ARE
   1   2   0
   1   0  -2
THE CORRESPONDING EIGENVECTORS ARE
   2  -1   1
TYPE THE ORDER OF THE MATRIX AND THE NUMBER
OF DISTINCT EIGENVALUES IN FORMAT (2I2)
 4 4
TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
  10   3   5   2
   8   3   4   5
   8   2   1   9
   8   2   1   9
TYPE THE DISTINCT EIGENVALUES IN FORMAT (15I4)
  20   2   1   0
THE CORRESPONDING EIGENVECTORS ARE
   1   1   1   1
THE CORRESPONDING EIGENVECTORS ARE
   5  -4  -4  -4
THE CORRESPONDING EIGENVECTORS ARE
  13 -25  -6  -6
THE CORRESPONDING EIGENVECTORS ARE
   1  51 -29  -9
TYPE THE ORDER OF THE MATRIX AND THE NUMBER
OF DISTINCT EIGENVALUES IN FORMAT (2I2)
21 1
ORDER OF THE MATRIX, TOO LARGE KINDLY TYPE AGAIN
TYPE THE ORDER OF THE MATRIX AND THE NUMBER
OF DISTINCT EIGENVALUES IN FORMAT (2I2)
 5 1
TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
   7   4   3  -2  -3
   0   5   0   0   0
  -2  -4   2   2   3
   2   4   3   3  -3
  -2  -4  -3   2   8
TYPE THE DISTINCT EIGENVALUES IN FORMAT (15I4)
   5
THE CORRESPONDING EIGENVECTORS ARE
   2  -1   0   0   0
   3   0  -2   0   0
   1   0   0   1   0
   3   0   0   0   2
```

```
 TYPE THE ORDER OF THE MATRIX AND THE NUMBER
 OF DISTINCT EIGENVALUES IN FORMAT (2I2)
  0 0
```

5.17 The program computes the Jordan canonical form of a given matrix

```
      IMPLICIT INTEGER*4 (A-Z)
      PARAMETER (ND = 15)
      DIMENSION B (ND, ND),  U (ND), V (ND)
      COMMON A (ND, 16), ADEN (ND, 16), X (ND, ND), XDEN (ND, ND),
     1M, N, P, IND
10    WRITE (6, '('' TYPE THE ORDER OF THE MATRIX AND THE NUMBER''
     1/1X,''OF EIGENVECTOR IN FORMAT (2I2)'')')
      READ (4, '(2I2)') N, NEQ
      WRITE (6, '(1X, 2I2)')  N, NEQ
      IF (N .EQ. 0) STOP
      IF (N .LT. 0) THEN
         WRITE (6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
         GO TO 10
      ELSEIF (N .GT. ND) THEN
         WRITE (6, '('' ORDER OF THE MATRIX, TOO LARGE KINDLY''
     1   ,1X,''TYPE AGAIN'')')
         GO TO 10
      ENDIF
      M = N
      NS = 0
      NP1 = N + 1
      WRITE (6,'('' TYPE THE COEFFICIENTS OF THE MATRIX IN''
     1,1X,''FORMAT (15I4)'')')
      DO 22 I = 1, N
         READ (4, '(20I4)') (B (I, J), J = 1, N)
22    CONTINUE
      DO 33 I = 1, N
         WRITE (6, '(1X, 20I4)') (B (I, J), J = 1, N)
33    CONTINUE
40    NS = NS + 1
      IF (NS .GT. NEQ) GO TO 10
      WRITE (6, '('' TYPE THE EIGENVALUE IN FORMAT (I4)'')')
      READ (4, '(I4)') EV
      WRITE (6, '(1X, I4)') EV
      WRITE (6, '('' TYPE THE CORRESPONDING EIGENVECTOR''
     1,1X,''IN FORMAT (15I4)'')')
      READ (4, '(15I4)') (A (I, NP1), I = 1, N)
      WRITE (6, '(1X, 15I4)') (A (I, NP1), I = 1, N)
```

```
      DO 55 I = 1, N
         ADEN (I, N + 1) = 1
55    CONTINUE
60    CALL FEJORD (B, N, EV, A)
      DO 88 I = 1, N
         DO 77 J = 1, N + 1
            ADEN (I, J) = 1
77       CONTINUE
88    CONTINUE
      M = N
      CALL SOLVE
      IF (IND .EQ. 1) THEN
         WRITE (6, '('' THERE IS NO SOLUTION'')')
         GO TO 40
      ELSE
         WRITE (6, '('' THE PRINCIPAL VECTOR IS'')')
         WRITE (6, '(1X, 10(I4,''/'',I2))') (X (I, 1),
     1   XDEN (I, 1),  I = 1, N)
      ENDIF
      DO 99 I = 1, N
          A (I, N+1) = X (I, 1)
          ADEN (I, N+1) = XDEN (I, 1)
99    CONTINUE
      GO TO 60
      END

C  PURPOSE- TO FORM AN EQUATION  AX = LX
C  INPUT PARAMETERS
C  B    : TWO DIMENSIONAL ARRAY, THE GIVEN MATRIX
C  EV   : THE EIGENVALUE
C  N    : THE ORDER OF THE MATRIX
C OUTPUT PARAMETER
C A     : TWO DIMENSIONAL ARRAY, THE COEFFICIENTS OF THE MATRIX
C         EQUATIONS
      SUBROUTINE FEJORD (B, N, EV, A)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 15)
      DIMENSION B (ND, ND), A (ND, 15)
      DO 22 I = 1, N
         DO 11 J = 1, N
            IF (I .EQ. J) THEN
               A (I, J) = B (I, J) - EV
            ELSE
               A (I, J) = B (I, J)
            ENDIF
11       CONTINUE
22    CONTINUE
```

```
      RETURN
      END

TYPE THE ORDER OF THE MATRIX AND THE NUMBER
OF EIGENVECTOR IN FORMAT (2I2)
 3 3
TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
  -7 -21 -15
   6  16  10
  -3  -7  -3
TYPE THE EIGENVALUE IN FORMAT (I4)
   2
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
  -7   3   0
THERE IS NO SOLUTION
TYPE THE EIGENVALUE IN FORMAT (I4)
   2
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
  -5   0   3
THERE IS NO SOLUTION
TYPE THE EIGENVALUE IN FORMAT (I4)
   2
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
   3  -2   1
THE PRINCIPAL VECTOR IS
  -1/ 3   0/ 1   0/ 1
THERE IS NO SOLUTION
TYPE THE ORDER OF THE MATRIX AND THE NUMBER
OF EIGENVECTOR IN FORMAT (2I2)
 4 2
TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
   2  -1  -2   1
  -4  -3   2   1
  17  -3 -10   2
  -6   1   2  -5
TYPE THE EIGENVALUE IN FORMAT (I4)
  -4
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
   0  -2   1   0
THE PRINCIPAL VECTOR IS
  -1/ 1  -6/ 1   0/ 1   0/ 1
THERE IS NO SOLUTION
TYPE THE EIGENVALUE IN FORMAT (I4)
  -4
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
  -1  -5   0   1
THE PRINCIPAL VECTOR IS
  -3/ 1 -17/ 1   0/ 1   0/ 1
THERE IS NO SOLUTION
```

```
TYPE THE ORDER OF THE MATRIX AND THE NUMBER
OF EIGENVECTOR IN FORMAT (2I2)
 4 3
TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
  10  -1  -2   0
  -8   7   3  -1
  18  -4  -2   1
  -5   1   2   5
TYPE THE EIGENVALUE IN FORMAT (I4)
   5
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
   1   1   2   0
THERE IS NO SOLUTION
TYPE THE EIGENVALUE IN FORMAT (I4)
   5
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
   1   5   0   2
THERE IS NO SOLUTION
TYPE THE EIGENVALUE IN FORMAT (I4)
   5
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
   1  -1   3  -1
THE PRINCIPAL VECTOR IS
   1/ 2   3/ 2   0/ 1   0/ 1
THERE IS NO SOLUTION
TYPE THE ORDER OF THE MATRIX AND THE NUMBER
OF EIGENVECTOR IN FORMAT (2I2)
 5 3
TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
   5   8   2   4   2
   2   7   4   2   4
   3   6   6   6   9
   1   2   1   3   4
   2   4   2   4   9
TYPE THE EIGENVALUE IN FORMAT (I4)
   3
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
 -60   8  12   4   8
THE PRINCIPAL VECTOR IS
  38/ 1 -17/ 1   0/ 1   0/ 1   0/ 1
THE PRINCIPAL VECTOR IS
 -21/ 2  19/ 2 -17/ 2   0/ 1   0/ 1
THERE IS NO SOLUTION
TYPE THE EIGENVALUE IN FORMAT (I4)
  20
TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
   4   4   6   2   4
THERE IS NO SOLUTION
TYPE THE EIGENVALUE IN FORMAT (I4)
```

```
   1
 TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
   84 -30  12 -34   8
 THERE IS NO SOLUTION
 TYPE THE ORDER OF THE MATRIX AND THE NUMBER
 OF EIGENVECTOR IN FORMAT (2I2)
  4 2
 TYPE THE COEFFICIENTS OF THE MATRIX IN FORMAT (15I4)
    9  -1  -2   0
   -8   6   3  -1
   18  -4  -3   1
   -5   1   2   4
 TYPE THE EIGENVALUE IN FORMAT (I4)
    4
 TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
    1   1   2   0
 THERE IS NO SOLUTION
 TYPE THE EIGENVALUE IN FORMAT (I4)
    4
 TYPE THE CORRESPONDING EIGENVECTOR IN FORMAT (15I4)
    1   5   0   2
 THERE IS NO SOLUTION
 TYPE THE ORDER OF THE MATRIX AND THE NUMBER
 OF EIGENVECTOR IN FORMAT (2I2)
  0 0
```

5.18 Program to find the entries of a matrix with given determinant

```
      PARAMETER (ND = 10)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION MAT (ND, ND)
      OPEN (4, FILE = 'ZGDET.IN', STATUS= 'OLD')
10    WRITE (6, '('' TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)'')')
      READ (4, '(I2)') N
      WRITE (6, '(1X, I2)') N
      IF (N .EQ. 0) STOP
      IF (N .LT. 0) THEN
         WRITE (6, '('' ORDER OF THE MATRIX CANNOT BE NEGATIVE'')')
         GO TO 10
      ELSEIF (N .GT. ND) THEN
         WRITE (6, '('' DESIRED MATRIX TOO LARGE, TYPE'',1X,
     1   ''A SMALLER NUMBER''/'' AS THE ORDER OF THE MATRIX'')')
         GO TO 10
      ELSEIF (N .EQ. 1) THEN
         IO = 1
      ELSE
```

```
         IO = 2
      ENDIF
      WRITE (6, '('' TYPE THE INTEGER ENTRIES OF THE MATRIX''
     1,1X,''OF ORDER'',1X,I2,1X,''WITH DETERMINANT 1'')') IO
20    DO 33 I = 1, IO
         READ (4, '(10I6)') (MAT (I, J), J = 1, IO)
33    CONTINUE
      DO 44 I = 1, IO
         WRITE (6, '(1X, 10I6)') (MAT (I, J), J = 1, IO)
44    CONTINUE
      IND = 1
50    CALL MUD (MAT, IO, IND)
      IF (IND .EQ. 2) THEN
         WRITE (6, '('' ELEMENTS ARE WRONG, KINDLY TYPE AGAIN'')')
         GO TO 20
      ENDIF
      IF (IO .LT. N) THEN
         IO = IO + 1
         WRITE (6,'('' TYPE ANY'',I3,1X,''INTEGERS IN''
     1   ,1X,''FORMAT (10I6)'')') IO - 1
         READ (4, '(10I6)') (MAT (IO, LL-1), LL = 2, IO)
         WRITE (6, '(1X, 10I6)') (MAT (IO, LL-1), LL = 2, IO)
         WRITE (6, '('' AGAIN TYPE ANY'',I3,1X,''INTEGERS IN''
     1   ,1X,''FORMAT (10I6)'')') IO - 1
         READ (4, '(10I6)') (MAT (LL-1, IO), LL = 2, IO)
         WRITE (6, '(1X, 10I6)') (MAT (LL-1, IO), LL = 2, IO)
         GO TO 50
      ENDIF
      WRITE (6,'('' TYPE THE DETERMINANT OF THE MATRIX''
     1,1X,''IN FORMAT (I6)'')')
      READ (4, '(I6)') IDETR
      WRITE (6, '(1X, I6)') IDETR
C     Special treatment for order 1 and 2
      IF (N .LE. 2) THEN
         DO 55 I = 1, N
            MAT (1, I) = IDETR * MAT (1, I)
55       CONTINUE
      ELSE
         MAT (N, N) = MAT (N, N) + (IDETR - 1)
      ENDIF
      WRITE (6, '('' REQUIRED MATRIX IS'')')
      DO 66 I = 1, N
         WRITE (6, '(1X, 10I6)') (MAT (I, J), J = 1, N)
66    CONTINUE
      GO TO 10
      END
```

```
C  PURPOSE :- TO  GENERATE A MATRIX WITH UNIT DETERMINANT AND INTEGER
C              ENTRIES
C  INPUT PARAMETERS
C  MAT :  TWO DIMENSIONAL ARRAY, NON DIAGONAL ENTRIES OF THE MATRIX
C         TO BE GENERATED
C  N   :  ORDER OF THE MATRIX
C  IND  = 1 INITIALLY
C       = 2 MATRIX WITH UNIT DETERMINAT IS NOT POSSIBLE
C  OUTPUT PARAMETER
C  MAT : TWO DIMENSIONAL ARRAY, INTEGER MATRIX WITH UNIT DETERMINANT
       SUBROUTINE MUD (MAT, N, IND)
       IMPLICIT INTEGER*4 (A - Z)
       PARAMETER (ND = 10)
       DIMENSION MAT (ND, ND), NMAT (ND, ND)
       IF (N .LE. 2) THEN
C         Special treatment for orders 1 and 2
          IF (N .EQ. 1) THEN
             IF (MAT (1, 1) .NE. 1) IND = 2
          ELSE
             IF (MAT(1, 1)*MAT (2,2) - MAT(1,2) * MAT(2,1).NE. 1) IND=2
          ENDIF
          RETURN
       ELSE
C         Calculate the entry MAT (N, N), N>=3
          MN1 = N - 1
          NEW = 1
          DO 44 JJ = 1, MN1
             IF (JJ .EQ. 1) THEN
C               The minor of an element
                DO 22 I = 1, MN1
                   DO 11 K = 1, MN1
                      NMAT (I, K) = MAT (I, K + 1)
11                 CONTINUE
22              CONTINUE
             ENDIF
C            Calculation of the new entry
             NEW = NEW + (-1)**(JJ+N+1)*MAT (N, JJ) * LDET (NMAT,MN1)
             IF (JJ .NE. MN1) THEN
                DO 33 I = 1, MN1
                   NMAT (I, JJ) = MAT (I, JJ)
33              CONTINUE
             ENDIF
44        CONTINUE
C         The required element of the matrix
          MAT (N, N) = NEW
       ENDIF
       RETURN
```

```
      END

C  PURPOSE - TO EVALUATE DETERMINANT USING GAUSSIAN
C             ELIMINATION METHOD
C  INPUT PARAMETERS
C  A : TWO DIMENSIONAL ARRAY, ELEMENTS OF THE DETERMINANT
C  N : ORDER OF THE DETERMINANT
      FUNCTION LDET (A, N)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      INTEGER A (ND, ND), COPYA (ND, ND), COPYD (ND, ND)
C     Initialize the arrays
      DO 22 MM=1,N
         DO 11 NN=1,N
            COPYA (MM, NN) = A (MM, NN)
            COPYD (MM, NN) = 1
11       CONTINUE
22    CONTINUE
C     Special treatment for order 1
      IF (N .NE. 1) THEN
         DO 88 I = 1, N-1
            IP1 = I + 1
            IF (COPYA (I,I) .EQ. 0) THEN
C              Search for non-zero entry in the column
               DO 33 J = IP1, N
                  IF (COPYA (J,I) .NE. 0) GO TO 40
33             CONTINUE
C              The above condition is true when det is 0
               LDET = 0
               RETURN
C              Interchange the rows
40             DO 55 K = I, N
                  EXTRA =COPYA (I, K)
                  COPYA (I, K) = COPYA (J, K)
                  COPYA (J, K) = - EXTRA
                  EXTRA = COPYD (I, K)
                  COPYD (I, K) = COPYD (J, K)
                  COPYD (J, K) = EXTRA
55             CONTINUE
            ENDIF
C           Carry out the process of elimination
            CALL RECIP (COPYA (I, I),COPYD (I, I), RECNUM, RECDEN)
            DO 77 J = IP1, N
               IF (COPYA (J, I) .NE. 0) THEN
                  CALL MULTN (COPYA (J, I),COPYD (J, I),RECNUM,RECDEN,
     1            FACNUM,FACDEN)
```

```
C                  Replace a (j,k) by a (j,k)-a (j,i)/a (i,i)*a (i,k)
                   DO 66 K = IP1, N
                      IF (COPYA (I, K) .NE. 0) THEN
                         CALL MULTN (FACNUM, FACDEN,COPYA (I, K),
     1                   COPYD (I, K), TEMP1, TEMP2)
                         CALL SUBTN (COPYA (J, K),COPYD (J,K),
     1                   TEMP1, TEMP2, COPYA (J,K), COPYD (J,K))
                      ENDIF
66                 CONTINUE
                ENDIF
77           CONTINUE
88        CONTINUE
       ENDIF
       NUM = 1
       DEN = 1
C      Multiply all the diagonal elements
       DO 99 I = 1, N
      CALL MULTN (NUM, DEN, COPYA (I, I),COPYD (I, I), NUM, DEN)
99     CONTINUE
       IF (DEN .EQ. 1) THEN
          LDET = NUM
       ELSE
          WRITE (*, *) ' THERE IS SOME MISTAKE IN THE CALCULATIONS '
       ENDIF
       RETURN
       END

 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  1
 TYPE THE INTEGER ENTRIES OF THE MATRIX OF ORDER  1 WITH DETERMINANT 1
      1
 TYPE THE DETERMINANT OF THE MATRIX IN FORMAT (I6)
 452046
 REQUIRED MATRIX IS
 452046
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  2
 TYPE THE INTEGER ENTRIES OF THE MATRIX OF ORDER  2 WITH DETERMINANT 1
      1    -1
     -3     4
 TYPE THE DETERMINANT OF THE MATRIX IN FORMAT (I6)
   1000
 REQUIRED MATRIX IS
   1000 -1000
     -3     4
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 25
```

```
DESIRED MATRIX TOO LARGE, TYPE A SMALLER NUMBER
AS THE ORDER OF THE MATRIX
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 5
TYPE THE INTEGER ENTRIES OF THE MATRIX OF ORDER  2 WITH DETERMINANT 1
    24     1
     1     1
ELEMENTS ARE WRONG, KINDLY TYPE AGAIN
    -2     1
     1    -1
TYPE ANY  2 INTEGERS IN FORMAT (10I6)
     0    -1
AGAIN TYPE ANY  2 INTEGERS IN FORMAT (10I6)
    -4     2
TYPE ANY  3 INTEGERS IN FORMAT (10I6)
     1     4    -1
AGAIN TYPE ANY  3 INTEGERS IN FORMAT (10I6)
     2    -2    -2
TYPE ANY  4 INTEGERS IN FORMAT (10I6)
     1     2     3    -1
AGAIN TYPE ANY  4 INTEGERS IN FORMAT (10I6)
    -4     2     0     2
TYPE THE DETERMINANT OF THE MATRIX IN FORMAT (I6)
   105
REQUIRED MATRIX IS
    -2     1    -4     2    -4
     1    -1     2    -2     2
     0    -1     1    -2     0
     1     4    -1     9     2
     1     2     3    -1   107
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 0
```

5.19 Program for generation of integer matrices (diagonalisable) with pre-assigned integer spectra

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION MAT (ND, ND), INVM (ND, ND), IMT (ND, ND), NMAT (ND, ND)
     1,EV (ND)
10    WRITE (6, '('' TYPE THE ORDER OF THE MATRIX''
     1,1X,''IN FORMAT (I2)'')')
      READ (4, '(I2)') N
      WRITE (6, '(1X, I2)') N
      IF (N .EQ. 0) STOP
      IF (N .LT. 0) THEN
         WRITE (6,'('' ORDER OF THE MATRIX CANNOT BE NEGATIVE'')')
```

```
         GO TO 10
        ELSEIF (N .GT. ND) THEN
           WRITE (6, '('' ORDER OF THE DESIRED MATRIX TOO LARGE, TYPE''
       1   ,1X,''A SMALLER NUMBER''/'' AS THE ORDER OF THE MATRIX'')')
           GO TO 10
        ENDIF
        DO 44 I = 1, N
           READ (4, '(10I6)') (MAT (I, J), J = 1, N)
44      CONTINUE
        WRITE (6, '('' TYPE THE INTEGER EIGENVALUES IN FORMAT (10I6)'')')
        READ (4, '(10I6)') (EV (I), I = 1, N)
        WRITE (6, '(1X, 10I6)') (EV (I), I = 1, N)
        WRITE (6, '('' TYPE A MATRIX WITH UNIT DETERMINANT IN''
       1,1X,''FORMAT (10I6)'')')
        DO 55 I = 1, N
           WRITE (6, '(1X, 10I6)') (MAT (I, J), J = 1, N)
55      CONTINUE
        CALL MATINV (MAT, INVM, N, IND)
C       IND = 1, if the matrix is singular
        IF (IND .EQ. 1) GO TO 10
        WRITE (6, '('' INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS'')')
        DO 66 I = 1, N
           WRITE (6, '(1X, 10I7)') (INVM (I,J), J = 1, N)
66      CONTINUE
        CALL DMM (MAT, EV, NMAT, N)
        CALL MATMUL (NMAT, INVM, IMT, N)
        WRITE (6, '('' THE REQUIRED MATRIX IS'')')
        DO 77 I = 1, N
           WRITE (6, '(1X, 10I7)') (IMT (I, J), J = 1, N)
77      CONTINUE
        GO TO 10
        END

C  PURPOSE - TO FIND THE PRODUCT OF A MATRIX WITH A DIAGONAL
C            MATRIX
C  INPUT PARAMETERS
C  EV   : ONE DIMENSIONAL ARRAY, CONTAINS THE DIAGONAL ELEMENTS
C  KMAT: TWO DIMENSIONAL ARRAY, CONTAINS THE MATRIX
C  N    : THE ORDER OF THE MATRIX
C  OUTPUT PARAMETERS
C  NMAT: TWO DIMENSIONAL ARRAY, CONTAINS THE PRODUCT MATRIX
        SUBROUTINE DMM (KMAT, EV, NMAT, N)
        IMPLICIT INTEGER*4 (A - Z)
        PARAMETER (ND = 10)
        DIMENSION KMAT (ND, ND), EV (N), NMAT (ND, ND)
        DO 22 J = 1, N
           DO 11 I = 1, N
```

```
            NMAT (J, I) = KMAT (J, I) * EV (I)
11        CONTINUE
22     CONTINUE
       RETURN
       END

C  PURPOSE - TO MULTIPLY TWO SQUAR MATRICES
C  INPUT PARAMETERS
C  F   : TWO DIMENSIONAL ARRAY, CONTAINS THE FIRST MATRIX
C  G   : TWO DIMENSIONAL ARRAY, CONTAINS THE SECOND MATRIX
C  K   : ORDER OF THE MATRIX
C  OUTPUT PARAMETERS
C  H   : TWO DIMENSIONAL ARRAY, CONTAINS THE PRODUCT MATRIX
       SUBROUTINE MATMUL (F, G, H, K)
       IMPLICIT INTEGER*4 (A - Z)
       PARAMETER (ND = 10)
       DIMENSION F (ND, ND),  G (ND, ND),  H (ND, ND)
       DO 33 I = 1, K
          DO 22 J = 1, K
             NSUM = 0
             DO 11 L = 1, K
                NSUM = NSUM +  F (I, L) *  G (L, J)
11           CONTINUE
             H (I, J) = NSUM
22        CONTINUE
33     CONTINUE
       RETURN
       END

C  PURPOSE - TO FIND THE INVERSE OF A MATRIX
C  INPUT PARAMETERS
C  A1 : TWO DIMENSIONAL ARRAY, CONTAINS THE MATRIX
C  N  : ORDER OF THE MATRIX
C  OUTPUT PARAMETERS
C  A  : TWO DIMENSIONAL ARRAY, CONTAINS THE INVERSE MATRIX
C  IND = 0 INITIALLY
C      = 1 THE MATRIX IS SINGULAR
       SUBROUTINE MATINV (A1, A, N, IND)
       IMPLICIT INTEGER*4 (A - Z)
       PARAMETER (ND = 10)
       DIMENSION  A1 (ND, ND),  ADEN (ND, ND),  INT1 (ND),
      1INT2 (ND), A (ND, ND)
C      Intialize the arrays
       DO 22 I = 1, N
          DO 11 J = 1, N
             A (I, J) = A1 (I, J)
             ADEN (I, J) = 1
11        CONTINUE
```

```
22    CONTINUE
      L = 0
      DO 99 I= 1, N
C        Search for the pivot
         IF (A (I, I) .EQ. 0) THEN
            IF (I .NE. N) THEN
               DO 33 J = I+1, N
                  IF (A (J, I) .NE. 0) GO TO 40
33             CONTINUE
            ENDIF
C           The matrix is singular
            IND = 1
            RETURN
40          L = L + 1
            INT1 (L) = I
            INT2 (L) = J
C           Interchange the rows
            DO 44 K = 1, N
               EXTRA = A (I, K)
               A (I, K) = A (J, K)
               A (J, K) = EXTRA
               EXTRA = ADEN (I, K)
               ADEN (I, K) = ADEN (J, K)
               ADEN (J, K) = EXTRA
44          CONTINUE
         ENDIF
C        Carry out elimination
         CALL RECIP (A (I, I), ADEN (I, I), A(I, I), ADEN(I, I))
         DO 55 K = 1, N
            IF (K .NE. I .AND. A (I, K) .NE. 0) THEN
               CALL MULTN (A (I, K), ADEN (I, K), A (I, I),
     1         ADEN (I, I), A (I, k), ADEN (I, K))
            ENDIF
55       CONTINUE
C        Replace a(j,k) by a(j,k) - a(j,i)/a(i,i) * a(i,k)
         DO 77 J = 1, N
            IF (J .NE. I .AND. A (J, I) .NE. 0) THEN
            DO 66 K = 1, N
               IF (K .NE. I .AND. A (I, K) .NE. 0) THEN
                  CALL MULTN (A (J, I), ADEN (J, I),
     1            A (I, K),  ADEN (I, K), TEMP1, TEMP2)
                  CALL SUBTN (A (J, K), ADEN (J, K),
     1            TEMP1, TEMP2,  A (J, K), ADEN (J, K))
               ENDIF
66          CONTINUE
         ENDIF
```

```
77    CONTINUE
      DO 88 J = 1, N
         IF (J .NE. I .AND. A (J, I) .NE. 0) THEN
            TEMP = - A (I, I)
            CALL MULTN (A (J, I), ADEN (J, I), TEMP,
     1      ADEN (I, I), A (J, I), ADEN (J, I))
         ENDIF
88    CONTINUE
99  CONTINUE
    IND= 0
    IF (L .EQ. 0) RETURN
C   Inter change the columns corresponding to the rows
C   allready interchanged, but in the reverse order
    DO 122 I = 1, L
       LMI = L + 1 - I
       K1 = INT1 (LMI)
       K2 = INT2 (LMI)
       DO 111 J = 1, N
          EXTRA = A (J, K1)
          A (J, K1) = A (J, K2)
          A (J, K2) = EXTRA
          EXTRA = ADEN (J, K1)
          ADEN (J, K1) = ADEN (J, K2)
          ADEN (J, K2) = EXTRA
111    CONTINUE
122   CONTINUE
      RETURN
      END

 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  1
 TYPE THE INTEGER EIGENVALUES IN FORMAT (10I6)
   5000
 TYPE A MATRIX WITH UNIT DETERMINANT IN FORMAT (10I6)
      1
 INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS
       1
 THE REQUIRED MATRIX IS
    5000
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  3
 TYPE THE INTEGER EIGENVALUES IN FORMAT (10I6)
     10     4     1
 TYPE A MATRIX WITH UNIT DETERMINANT IN FORMAT (10I6)
      1    -1     2
     -3     4     3
      1    -1     3
```

```
 INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS
      15       1    -11
      12       1     -9
      -1       0      1
 THE REQUIRED MATRIX IS
     100       6    -72
    -261     -14    189
      99       6    -71
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 15
 ORDER OF THE DESIRED MATRIX TOO LARGE, TYPE A SMALLER NUMBER
 AS THE ORDER OF THE MATRIX
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  5
 TYPE THE INTEGER EIGENVALUES IN FORMAT (10I6)
     -8     7    -5     4     2
 TYPE A MATRIX WITH UNIT DETERMINANT IN FORMAT (10I6)
     -2     1    -4     2    -4
      1    -1     2    -2     2
      0    -1     1    -2     0
      1     4    -1     9     2
      1     2     3    -1     3
 INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS
     -27    -41    -30    -10     -2
      -5     -8     -6     -2      0
      -1     -2      1      0      0
       2      3      3      1      0
      14     22     14      5      1
 THE REQUIRED MATRIX IS
    -583   -904   -590   -206    -40
     301    468    304    106     20
      24     42     13      6      0
     199    290    241     80     20
     237    366    213     78     22
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  0
```

5.20 Program for generation of integer matrices starting from block-diagonal matrices

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION MAT (ND, ND), INVM (ND, ND),  NMAT (ND, ND),
     1BLOM (ND, ND), MKEI (ND, ND), P (ND)
10    WRITE (6, '('' TYPE THE ORDER OF THE MATRIX IN''
     1,1X,''FORMAT (I2)'')')
      READ (4, '(I2)') N
```

```
      WRITE (6, '(1X, I2)') N
      IF (N .EQ. 0) STOP
      IF (N .LT. 0) THEN
         WRITE (6, '('' ORDER OF THE MATRIX CANNOT BE NEGATIVE,''
     1   /'' KINDLY TYPE AGAIN'')')
         GO TO 10
      ELSEIF (N .GT. ND) THEN
         WRITE (6, '('' ORDER OF THE DESIRED MATRIX TOO LARGE, TYPE''
     1   1X,''A SMALLER NUMBER''/'' AS THE ORDER OF THE MATRIX'')')
         GO TO 10
      ENDIF
      WRITE (6, '('' TYPE A MATRIX WITH UNIT DETERMINANT IN''
     1,1X,''FORMAT (10I6)'')')
      DO 22 I = 1, N
         READ (4, '(10I6)') (MAT (I, J), J = 1, N)
22    CONTINUE
      DO 33 I = 1, N
         WRITE (6, '(1X, 10I6)') (MAT (I, J), J = 1, N)
33    CONTINUE
      CALL MATINV (MAT, INVM, N, IND)
C     IND = 1 if the matrix is singular
      IF (IND .EQ. 1) GO TO 10
      WRITE (6, '('' INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS'')')
      DO 44 I = 1, N
         WRITE (6, '(1X, 10I6)') (INVM (I, J), J = 1, N)
44    CONTINUE
      WRITE (6, '('' TYPE A BLOCK DIAGONAL MATRIX IN''
     1,1X,''FORMAT (10I6)'')')
      DO 55 I = 1, N
         READ (4, '(10I6)') (BLOM (I, J), J = 1, N)
55    CONTINUE
      DO 66 I = 1, N
         WRITE (6, '(1X, 10I6)') (BLOM (I, J), J = 1, N)
66    CONTINUE
      CALL MATMUL (MAT, BLOM, NMAT, N)
      CALL MATMUL (NMAT, INVM, MKEI, N)
      WRITE (6, '('' THE REQUIRED MATRIX IS'')')
      DO 77 I = 1,N
         WRITE (6, '(1X, 10I7)') (MKEI (I, J), J = 1, N)
77    CONTINUE
      CALL CHPOLY (MKEI, N, P)
      WRITE (6,'('' THE COEFFICIENTS OF CHARACTERISTIC POLYNOMIAL''/
     11X,''OF THE MATRIX, STARTING FROM THE 2ND ARE GIVEN BELOW''/
     11X, 10I8)') (P (I), I = 1, N)
      GO TO 10
      END
```

```
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 1
TYPE A MATRIX WITH UNIT DETERMINANT IN FORMAT (10I6)
     1
INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS
     1
TYPE A BLOCK DIAGONAL MATRIX IN FORMAT (10I6)
    50
THE REQUIRED MATRIX IS
     50
THE COEFFICIENTS OF CHARACTERISTIC POLYNOMIAL
OF THE MATRIX, STARTING FROM THE 2ND ARE GIVEN BELOW
     -50
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 2
TYPE A MATRIX WITH UNIT DETERMINANT IN FORMAT (10I6)
     3    -1
    -2     1
INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS
     1     1
     2     3
TYPE A BLOCK DIAGONAL MATRIX IN FORMAT (10I6)
     2     1
     0     2
THE REQUIRED MATRIX IS
      8      9
     -4     -4
THE COEFFICIENTS OF CHARACTERISTIC POLYNOMIAL
OF THE MATRIX, STARTING FROM THE 2ND ARE GIVEN BELOW
      -4       4
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
20
ORDER OF THE DESIRED MATRIX TOO LARGE, TYPE A SMALLER NUMBER
AS THE ORDER OF THE MATRIX
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 3
TYPE A MATRIX WITH UNIT DETERMINANT IN FORMAT (10I6)
     3    -1     4
    -2     1    -1
     1    -1    -1
INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS
    -2    -5    -3
    -3    -7    -5
     1     2     1
TYPE A BLOCK DIAGONAL MATRIX IN FORMAT (10I6)
     2     1     0
     0     2     1
     0     0     2
THE REQUIRED MATRIX IS
```

```
     -8    -23    -16
      7     18     11
     -4     -9     -4
 THE COEFFICIENTS OF CHARACTERISTIC POLYNOMIAL
 OF THE MATRIX, STARTING FROM THE 2ND ARE GIVEN BELOW
      -6      12      -8
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  4
 TYPE A MATRIX WITH UNIT DETERMINANT IN FORMAT (10I6)
      1     0     1     1
     -1     1     1     1
      3    -1     2     2
     -1     0    -1     0
 INVERSE OF THE MATRIX WITH UNIT DETERMINANT IS
      3    -1    -1     0
      5    -1    -2     0
     -3     1     1    -1
      1     0     0     1
 TYPE A BLOCK DIAGONAL MATRIX IN FORMAT (10I6)
      3     1     0     0
      0     3     0     0
      0     0    -4     1
      0     0     0    -4
 THE REQUIRED MATRIX IS
      23     -8     -9      1
      10     -3     -5      1
      45    -17    -17      2
     -27      8      9     -5
 THE COEFFICIENTS OF CHARACTERISTIC POLYNOMIAL
 OF THE MATRIX, STARTING FROM THE 2ND ARE GIVEN BELOW
       2     -23     -24     144
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  0
```

5.21 Program to generate a matrix with addition and subtraction operations only

```
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER  (ND = 10)
      DIMENSION MAT (ND, ND), ITRIM (ND, ND)
10    WRITE (6, '('' TYPE THE ORDER OF THE MATRIX IN''
     1,1X,''FORMAT (I2)'')')
      READ (4, '(I2)') N
      WRITE (6, '(1X, I2)') N
      IF (N .EQ. 0) STOP
      IF (N .LT. 0) THEN
         WRITE (6, '('' ORDER OF THE MATRIX CANNOT BE NEGATIVE''
```

```
     1   /'' KINDLY TYPE AGAIN''))')
         GO TO 10
        ELSEIF (N .GT. ND) THEN
         WRITE (6, '('' ORDER OF DESIRED MATRIX IS TOO LARGE, TYPE''
     1   ,'' A SMALLER NUMBER''/'' AS THE ORDER OF THE MATRIX''))')
         GO TO 10
        ENDIF
        WRITE (6, '('' TYPE THE ENTIRES OF THE UPPER TRIANGULAR''
     1,''MATRIX IN FORMAT (10I6)''))')
        DO 22 I = 1, N
         READ (4, '(10I6)') (ITRIM(I, J), J = 1, N)
22      CONTINUE
        DO 33 I = 1, N
         WRITE (6, '(1X, 10I6)') (ITRIM (I, J), J = 1, N)
33      CONTINUE
        CALL GEMAS (ITRIM, MAT, N)
        WRITE (6, '('' THE REQUIRED MATRIX IS'')')
        DO 44 I = 1, N
         WRITE (6, '(1X, 10I7)') (MAT (I, J), J = 1, N)
44      CONTINUE
        CALL INTERC (MAT, N)
        GO TO 10
        END

C  PURPOSE - TO GENERATE A MATRIX WITH ADDITION AND SUBTRACTION
C            OPERATIONS ONLY
C  INPUT PARAMETERS
C  F   : TWO DIMENSIONAL ARRAY, CONTAINS THE FIRST MATRIX
C  G   : TWO DIMENSIONAL ARRAY, CONTAINS THE SECOND MATRIX
C  K   : ORDER OF THE MATRIX
C  OUTPUT PARAMETERS
C  H   : TWO DIMENSIONAL ARRAY, CONTAINS THE GENERATED MATRIX

        SUBROUTINE GEMAS (F, H, K)
        PARAMETER (ND = 10)
        INTEGER F (ND, ND),  H (ND, ND)
        IF (K .GT. 1) THEN
         DO 11 J = 1, K - 1
            H (1, J) = F (1, J) - F (1, J+1)
11       CONTINUE
         DO 44 I = 2, K
           H (I, I-1) = H (I-1, I-1) - F (I, I)
           ITEMP = H (I, I-1)
           IF (I .LT. K) THEN
            DO 22 M = I+1, K
              H (M, I-1) = ITEMP
22          CONTINUE
            DO 33 J = I, K - 1
```

```
                  H (I, J) = H (I-1, J) +  F (I, J) - F (I, J+1)
33              CONTINUE
              ENDIF
44          CONTINUE
            H (1, K) = F (1, K)
            DO 55 I = 2, K
               H (I, K) = H (I-1, K) + F (I, K)
55          CONTINUE
         ELSE
            H (1, 1) = F (1, 1)
         ENDIF
         RETURN
         END

C  PURPOSE - TO APPLY SIMILARITY TRANSFORMATION OF
C             INTERCHANGING ROW AND COLUMN
C  INPUT PARAMETERS
C  COPYA: TWO-DIMENSIONAL ARRAY, THE MATRIX
C  N    : ORDER OF THE MATRIX
C  OUTPUT PARAMETER
C  COPYA: TWO-DIMENSIONAL ARRAY, THE MATRIX OBTAINED
C         BY APPLYING SIMILARITY TRANSFORMATION
         SUBROUTINE INTERC (COPYA, N)
         IMPLICIT INTEGER*4 (A - Z)
         PARAMETER (ND = 10)
         DIMENSION COPYA (ND, ND)
         WRITE (6, '('' TYPE THE NUMBER OF SIMILARITY TRANSFORMATION''
        1,1X,''IN FORMAT (I2)'')')
         READ (4, '(I2)') NS
         WRITE (6, '(1X, I2)') NS
         IF (NS .LE. 0) RETURN
         D = 0
10       WRITE (6, '('' TYPE THE ROW AND COLUMN NUMBER''
        1,1X,''IN FORMAT (2I2)'')')
         READ (4, '(2I2)') I, J
         WRITE (6, '(1X, 2I2)') I, J
         IF (I .LE. 0 .OR. J .LE. 0 .OR. I .GT. N .OR. J .GT. N) THEN
            WRITE (6, '('' DATA ILLEGAL, KINDLY TYPE AGAIN'')')
            GO TO 10
         ENDIF
         DO 22 K = 1, N
            EXTRA = COPYA (I, K)
            COPYA (I, K) = COPYA (J, K)
            COPYA (J, K) =  EXTRA
22       CONTINUE
         DO 33 K = 1, N
            EXTRA = COPYA (K, I)
```

```
         COPYA (K, I) = COPYA (K, J)
         COPYA (K, J) =  EXTRA
33     CONTINUE
       WRITE (6, '('' THE CHANGED MATRIX IS'')')
       DO 44 II = 1, N
         WRITE (6, '(1X, 10I7)') (COPYA (II, JJ), JJ = 1, N)
44     CONTINUE
       D = D + 1
       IF (D .EQ. NS) RETURN
       GO TO 10
       END
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  1
 TYPE THE ENTIRES OF THE UPPER TRIANGULAR MATRIX IN FORMAT (10I6)
    100
 THE REQUIRED MATRIX IS
     100
 TYPE THE NUMBER OF SIMILARITY TRANSFORMATION IN FORMAT (I2)
  1
 TYPE THE ROW AND COLUMN NUMBER IN FORMAT (2I2)
  1 1
 THE CHANGED MATRIX IS
     100
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  2
 TYPE THE ENTIRES OF THE UPPER TRIANGULAR MATRIX IN FORMAT (10I6)
     25     8
      0    15
 THE REQUIRED MATRIX IS
      17      8
       2     23
 TYPE THE NUMBER OF SIMILARITY TRANSFORMATION IN FORMAT (I2)
  1
 TYPE THE ROW AND COLUMN NUMBER IN FORMAT (2I2)
  1 2
 THE CHANGED MATRIX IS
      23      2
       8     17
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  3
 TYPE THE ENTIRES OF THE UPPER TRIANGULAR MATRIX IN FORMAT (10I6)
     25    28   -42
      0     4     3
      0     0    60
 THE REQUIRED MATRIX IS
      -3     70    -42
      -7     71    -39
```

```
     -7     11     21
TYPE THE NUMBER OF SIMILARITY TRANSFORMATION IN FORMAT (I2)
 1
TYPE THE ROW AND COLUMN NUMBER IN FORMAT (2I2)
 1 3
THE CHANGED MATRIX IS
     21     11     -7
    -39     71     -7
    -42     70     -3
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 4
TYPE THE ENTIRES OF THE UPPER TRIANGULAR MATRIX IN FORMAT (10I6)
    16     6     4     2
     0     8     6     3
     0     0    -1     1
     0     0     0    -3
THE REQUIRED MATRIX IS
     10      2      2      2
      2      4      5      5
      2      5      3      6
      2      5      6      3
TYPE THE NUMBER OF SIMILARITY TRANSFORMATION IN FORMAT (I2)
 2
TYPE THE ROW AND COLUMN NUMBER IN FORMAT (2I2)
 2 4
THE CHANGED MATRIX IS
     10      2      2      2
      2      3      6      5
      2      6      3      5
      2      5      5      4
TYPE THE ROW AND COLUMN NUMBER IN FORMAT (2I2)
 0 1
DATA ILLEGAL, KINDLY TYPE AGAIN
TYPE THE ROW AND COLUMN NUMBER IN FORMAT (2I2)
 7 8
DATA ILLEGAL, KINDLY TYPE AGAIN
TYPE THE ROW AND COLUMN NUMBER IN FORMAT (2I2)
 2-1
DATA ILLEGAL, KINDLY TYPE AGAIN
TYPE THE ROW AND COLUMN NUMBER IN FORMAT (2I2)
 1 3
THE CHANGED MATRIX IS
      3      6      2      5
      6      3      2      5
      2      2     10      2
      5      5      2      4
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 0
```

```
C  PURPOSE - TO CHANGE THE ELEMENTS OF THE MATRIX BY APPLYING
C            A SIMILARITY TRANSFORMATION OF PARTICULAR TYPE
C  INPUT PARAMETERS
C  MAT: TWO-DIMENSIONAL ARRAY, CONTAINS THE MATRIX
C  N  : THE ORDER OF THE MATRIX
C  OUTPUT-PARAMETER
C  MAT: TWO-DIMENSIONAL ARRAY, CONTAINS THE CHANGED MATRIX
      SUBROUTINE CEMAT (MAT, N)
      IMPLICIT INTEGER*4 (A - Z)
      PARAMETER (ND = 10)
      DIMENSION MAT (ND, ND), MUP (ND), IHCF (ND)
      IF (N .EQ. 1) THEN
         IHCF (1) = MAT (1, 1)
      ELSE
         DO 22 I = 1, N
            IF (I .EQ. 1) THEN
               IHCF (I) = MAT (2, I)
            ELSE
               IHCF (I) = MAT (1, I)
            ENDIF
            DO 11 J = 2, N
               IF (I .NE. J) THEN
                  IHCF (I) = HCF (MAT (J, I), IHCF (I))
               ENDIF
11          CONTINUE
22       CONTINUE
      ENDIF
      WRITE (6,'('' MULTIPLIERS MAY BE THE FACTORS OF''
     1/10(1X,I4))')(IHCF (I), I = 1, N)
      WRITE (6, '('' TYPE THE MULTIPLIERS IN FORMAT (10I6)'')')
      READ (4, '(10I6)') (MUP (I), I = 1, N)
      WRITE (6, '(1X, 10I6)') (MUP (I), I = 1, N)
      DO 44 J = 1, N
         DO 33 I = 1, N
            IF (I .NE. J) THEN
               ITEMP = MAT (I, J) * MUP (I) / MUP (J)
               IF (ITEMP * MUP(J) .NE. MAT(I, J) * MUP(I)) THEN
                  WRITE (6,'(1X,I6,'' CANNOT BE A MULTIPLIER OF''
     1            ,1X,''COLUMN'',I2)') MUP(J), J
                  RETURN
               ENDIF
            ENDIF
33       CONTINUE
44    CONTINUE
      DO 66 J = 1, N
         DO 55 I = 1, N
```

```
            MAT (I, J) = MAT (I, J) * MUP (I) / MUP (J)
55        CONTINUE
66     CONTINUE
       WRITE (6, '('' THE MATRIX WITH CHANGED ELEMENTS IS'')')
       DO 77 I = 1, N
          WRITE (6, '(1X, 10I6)') (MAT (I, J), J = 1, N)
77     CONTINUE
       RETURN
       END
```

The following output is generated using the subroutine GEMAS; and then we have applied another simple similarity transformation using the routine CEMAT, instead of applying the similarity transformation of interchanging rows and columns.

```
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  2
 TYPE AN UPPER TRIANGULAR MATRIX IN IN FORMAT (10I6)
     16    11
      0    -4
 THE POSITIVE MATRIX IS
      5    11
      9     7
 MULTIPLIERS MAY BE THE FACTORS OF
    9   11
 TYPE THE MULTIPLIERS IN FORMAT (10I6)
      1     1
 THE MATRIX WITH CHANGED ELEMENTS IS
      5    11
      9     7
 TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
  3
 TYPE AN UPPER TRIANGULAR MATRIX IN IN FORMAT (10I6)
     15    10     4
      0    -2     2
      0     2    -1
 THE ABOVE MATRIX IS NOT AN UPPER TRIANGULAR  MATRIX
 TYPE AN UPPER TRIANGULAR MATRIX IN IN FORMAT (10I6)
     15    10     4
      0    -2     2
      0     0    -1
 THE POSITIVE MATRIX IS
      5     6     4
      7     2     6
      7     3     5
 MULTIPLIERS MAY BE THE FACTORS OF
    7    3    2
 TYPE THE MULTIPLIERS IN FORMAT (10I6)
      7     1     2
```

```
THE MATRIX WITH CHANGED ELEMENTS IS
     5    42    14
     1     2     3
     2     6     5
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 4
TYPE AN UPPER TRIANGULAR MATRIX IN IN FORMAT (10I6)
    20    10     7     2
     0     2     2     3
     0     0     1     4
     0     0     0     0
THE POSITIVE MATRIX IS
    10     3     5     2
     8     3     4     5
     8     2     1     9
     8     2     1     9
MULTIPLIERS MAY BE THE FACTORS OF
   8    1    1    1
TYPE THE MULTIPLIERS IN FORMAT (10I6)
     1     2     2     2
     2 CANNOT BE A MULTIPLIER OF COLUMN 2
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 5
TYPE AN UPPER TRIANGULAR MATRIX IN IN FORMAT (10I6)
    20    15     7     4     2
     0     3     4     1     2
     0     0     3     3     2
     0     0     0     3     2
     0     0     0     0     1
THE POSITIVE MATRIX IS
     5     8     3     2     2
     2     7     6     1     4
     2     4     6     2     6
     2     4     3     3     8
     2     4     3     2     9
MULTIPLIERS MAY BE THE FACTORS OF
   2    4    3    1    2
TYPE THE MULTIPLIERS IN FORMAT (10I6)
     2     2     3     1     2
THE MATRIX WITH CHANGED ELEMENTS IS
     5     8     2     4     2
     2     7     4     2     4
     3     6     6     6     9
     1     2     1     3     4
     2     4     2     4     9
TYPE THE ORDER OF THE MATRIX IN FORMAT (I2)
 0

C  PURPOSE - TO SUBTRACT ONE VULGAR FRACTION FROM ANOTHER
```

```
C  INPUT PARAMETERS
C  I : THE NUMERATOR OF THE MINUEND
C  J : THE DENOMINATOR OF THE MINUEND
C  K : THE NUMERATOR OF THE SUBTRAHEND
C  L : THE DENOMINATOR OF THE SUBTRAHEND
C  OUTPUT PARAMETERS
C  M : THE NUMERATOR OF THE DIFFERENCE
C  N : THE DENOMINATOR OF THE DIFFERENCE
      SUBROUTINE SUBTN (I, J, K, L, M, N)
      IMPLICIT INTEGER*4 (A - Z)
      DEN = J / HCF (J, L) * L
      NUM = DEN / J*I - DEN / L * K
      FAC = HCF (NUM, DEN)
      M = NUM / FAC
      N = DEN / FAC
      RETURN
      END
C  PURPOSE - TO MULTIPLY TWO VULGAR FRACTION
C  INPUT PARAMETERS
C  I : THE NUMERATOR OF THE MULTIPLICAND
C  J : THE DENOMINATOR OF THE MULTIPLICAND
C  K : THE NUMERATOR OF THE MULTIPLIER
C  L : THE DENOMINATOR OF THE MULTIPLIER
C  OUTPUT PARAMETERS
C  M : THE NUMERATOR OF THE PRODUCT
C  N : THE DENOMINATOR OF THE PRODUCT
      SUBROUTINE MULTN (I, J, K, L, M, N)
      IMPLICIT INTEGER*4 (A - Z)
      HCF1 = HCF (I, L)
      HCF2 = HCF (J, K)
      M = I / HCF1 * (K / HCF2)
      N = J / HCF2 * (L / HCF1)
      RETURN
      END
C  PURPOSE - TO FIND THE RECIPROCAL OF A VULGAR FRACTION
C  INPUT PARAMETERS
C  I : THE NUMERATOR OF THE VULGAR FRACTION
C  J : THE DENOMINATOR OF THE VULGAR FRACTION
C  OUTPUT PARAMETERS
C  K : THE NUMERATOR OF THE RECIPROCAL FRACTION
C  L : THE DENOMINATOR OF THE RECIPROCAL FRACTION
      SUBROUTINE RECIP (I, J, K, L)
      IMPLICIT INTEGER*4 (A - Z)
      ITEMP=I
      K = ISIGN (J, I)
      L = IABS (ITEMP)
      RETURN
```

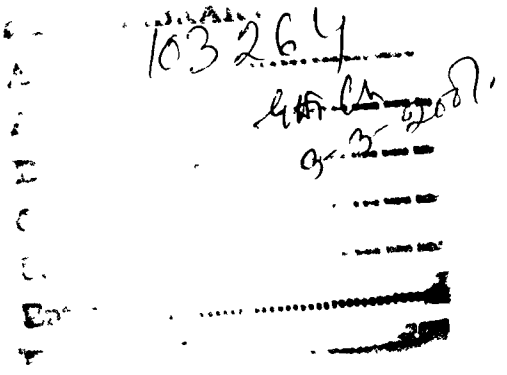
```
      END

C  PURPOSE - TO REDUCE THE GIVEN POLYNOMIAL TO A PRIMITIVE POLYNOMIAL
C  INPUT PARAMETERS
C  U : LINEAR ARRAY, THE COEFFICIENTS OF THE GIVEN POLYNOMIAL
C  NCF : THE NUMBER OF COEFFICIENTS OF THE POLYNOMIAL
C  OUTPUT PARAMETERS
C  U : LINEAR ARRAY, COEFFICIENTS OF THE PRIMITIVE POLYNOMIAL

      SUBROUTINE REMCOM (U, NCF)
      IMPLICIT INTEGER*4 (A - Z)
      DIMENSION U (NCF)
      ICOHCF = IABS (U (1))
      DO 22 I = 2, NCF
         ICOHCF = HCF (ICOHCF, U (I))
22    CONTINUE
      DO 33 I = 1, NCF
         U (I) = U (I) / ICOHCF
33    CONTINUE
      RETURN
      END

C  PURPOSE - TO FIND THE HCF OF TWO INTEGERS
C  INPUT PARAMETERS
C  I : THE FIRST NUMBER
C  J : THE SECOND NUMBER

      INTEGER FUNCTION HCF (I, J)
      IMPLICIT INTEGER*4 (A - Z)
      IF (I .EQ. 0) THEN
         HCF = IABS (J)
      ELSEIF (J .EQ. 0) THEN
        HCF = IABS (I)
      ELSE
         KOPY1 = IABS (I)
         KOPY2 = IABS (J)
10       REM = MOD (KOPY1, KOPY2)
         IF (REM .NE. 0) THEN
            KOPY1 = KOPY2
            KOPY2 = REM
            GO TO 10
         ELSE
            HCF = KOPY2
         ENDIF
      ENDIF
      RETURN
      END
```



CONTENTS OF THE APPENDIX

CONTENTS OF THE APPENDIX

CONTENTS OF THE APPENDIX

CONTENTS OF THE APPENDIX