C.P. No. 1124

C.P. No. 1124

MINISTRY OF TECHNOLOGY

AERONAUTICAL RESEARCH COUNCIL

CURRENT PAPERS

ALGOL Programmes for the Response Analysis of Linear Systems with Deterministic or Random Inputs

by

L. J. Hazlewood and E. Huntley

Aerodynamics Dept., R.A.E., Bedford

LONDON: HER MAJESTY'S STATIONERY OFFICE

1970

PRICE *£1 0s 0d* NET

U.D.C. 518.5 : 621.374.32 : 519.283 : 512.831 : 519.242 : 517.511

C.P. No. 1124*
July 1969

ALGOL PROGRAMMES FOR THE RESPONSE ANALYSIS OF LINEAR SYSTEMS WITH DETERMINISTIC OR RANDOM INPUTS

by

L. J. Hazlewood
E. Huntley
Aerodynamics Dept., R.A.E., Bedford

SUMMARY

In previous publications the so-called serial/matrix technique has been developed for the response analysis of systems defined by time-invariant ordinary differential equations. One paper describes how an explicit formulation for the output function may be easily obtained when the input function is deterministic. A second gives the output autocorrelation function and output mean square value when the input is a stationary random process.

This paper gives computer programmes in ALGOL which implement these ideas. The programmes are described primarily from the point of view of the user with illustrative examples to demonstrate the use of prepared data sheets but sufficient information is included to enable users to develop the programmes further if required.

*Replaces R.A.E. Technical Report 69140 - A.R.C. 31816

CONTENTS

INTRODUCTION

In previous papers methods were developed to simplify the response analysis of time-invariant linear systems when subjected to deterministic[1,2] or random inputs[3]. It was pointed out that although they could be employed in relatively small scale desk-type calculations these methods would be particularly useful when programmed for a digital computer.

The major part of this programming work has now been done, the programmes having been written in ALGOL. As at present constituted they are explicitly for the Elliott 503 computer but require only minor modifications of input and output instructions in order to be usable on any moderately sized computer with an ALGOL compiler. They could also be used, with small modifications indicated herein, on a large multipurpose computer such as ATLAS which includes an Elliott 503 ALGOL compiler in its range.

This paper describes the programmes and the way in which they are to be used together with illustrative examples. Everything relating to the deterministic input programme is discussed in Part I. The extension to random inputs is dealt with in Part II, but, to minimize repetition, reference is made to sections of Part I. The paper is meant to be used in conjunction with Refs.2 and 3.

PART I - DETERMINISTIC INPUT FUNCTIONS

2 BACKGROUND INFORMATION AND THE SCOPE OF PROGRAMME I

Many problems in engineering are formulated as a set of time-invariant linear ordinary differential equations. Usually, such equations are solved by the use of Laplace transform techniques and at some stage in the process a transfer function relating the required response variable and the input variable is obtained. Engineers working on control systems may build up a composite transfer function for a complicated system from the known transfer functions of simple elements. The next stage in the conventional application of Laplace transforms is to transform the input, express the resulting function for the transformed response variable in partial fraction form and then perform the inverse transformation. The serial/matrix method described in Ref.2, eliminates this tedious last stage. Once the factorised transfer function is specified, together with the input function, an explicit formulation of the output response may be obtained by precise matrix operations without having to resort to partial fraction expansions and inverse transformations.

The construction and mode of operation of the programme written to perform these operations together with detailed instructions for its use are described in this first part of the paper. The programme may be described as a basic programme with its starting point at the transfer function stage. Extensions are envisaged[4] which will increase its usefulness but these have not yet been programmed.

We assume then that the problem is formulated in the form of a system transfer function and deterministic input function of time which belongs to a large class of functions to be described shortly. The numerator and denominator of the transfer function have each to be factorised into linear or quadratic factors, thus avoiding the use of complex roots. (This factorisation may be done as convenient depending upon the programmes available but for the testing work on the programme we have been using the programme TRIPLE LENGTH BAIRSTOW-QUADRATIC FACTORS OF A POLYNOMIAL written by Wilkinson for Deuce; Ref.494 (RP04 T/1).) To each factor there corresponds a physically realisable filter and the overall transfer function is represented by a sequence of these filters. The input function is in effect passed through each of these elementary filters in turn.

The elementary filters which have to be allowed for in the programme are:

$$\text{constant gain } K,\ s,\ s + k,\ s^2 + 2ns + m^2 \text{ with } n < m$$

and

$$1/s,\ 1/(s + k),\ 1/(s^2 + 2ns + m^2) \text{ with } n < m \quad .$$

The input function is assumed to be any linear combination of functions of the following types:

(i) generalised functions, $u_1(t), \ldots, u_4(t)$ ($u_1(t)$ is a unit impulse),

(ii) unit step function $u_0(t)$, t, t^2, t^3,

(iii) e^{-at}, te^{-at}, t^2e^{-at},

(iv) $\sin \omega t$, $\cos \omega t$,

(v) $e^{-at}\sin \omega t$, $e^{-at}\cos \omega t$.

(The parameter 'a' is, of course, generally not the same in both groups (iii) and (v).)

There may be more than one group of each of the last three types, differing in their values of a and/or ω, but if any one member of a group occurs in the input it is assumed that the other member(s) also occur(s) but with zero multiplying coefficients. The input function is then of the form

$$X_1(t) = \underline{a}\ \underline{x}(t)$$

where $\underline{a}$ is row vector of coefficients and $\underline{x}(t)$ is a column vector of functions making up the input function set. Now, with each of the six possible types of factor occurring in the system transfer function, and for the particular input function vector $\underline{x}(t)$, can be associated a so-called response matrix.

The analogue of passing an input function through the elementary filters in turn becomes that of successive multiplications of the input coefficient vector $\underline{a}$ by the response matrices associated with the various filters. The programme is therefore concerned primarily with classification of the input function, the construction of the response matrices of appropriate order and the successive matrix multiplications.

3 MODE OF OPERATION

This section contains a brief discussion of the major processes in the programme. It is primarily for the programmer wishing to extend the programme in some way or to alter it so that it can be run on a computer other than the Elliott 503.

The programme contains identifiable chapters labelled by the transfer functions of the elementary filters s, $1/(s + k)$ etc. Since the programme was too large to be run whole on the Elliott 503 computer at Westcott it was split into two parts. The first tape contains the input chapter, the $1/(s + k)$ and $1/(s^2 + 2ns + m^2)$ chapters; the second tape contains the s, $s + k$, $s^2 + 2ns + m^2$ and $1/s$ chapters together with a chapter for the output of results. The output tape from the first part of the programme is used as the input data tape for the second part.

The two parts of the programme are shown separately in Appendix A sections (b) and (c) and their corresponding flow diagrams in Fig.1. The alterations which would be required in order to run the programme whole on a larger computer are discussed in section 3.4.

3.1 Input chapter

The input chapter is that shown in the first column of Appendix A section (b). In addition to putting in input function and transfer function data it serves to set up the input coefficient vector and to determine the order of the final coefficient vector. The input coefficient vector is augmented by the addition of zero elements so as to be of the same order. By this device all the response matrices can be set up as square matrices of order $q[12] \times q[12]$.

In accordance with the normal structure of an ALGOL programme, various procedures used in the programme are also stored in this chapter. The most important of these are 'mxprod', 'convert', 'testc' and 'normalise'. The purpose of these will be explained as we come to them in this discussion.

For nomenclature, the array 'b' is used to store the values of a and ω occurring in the components of the input function vector; arrays 'c' and 'e' represent input and output coefficient vectors respectively and array 'd' is used to represent the response matrix.

3.2 Response matrices

The six types of response matrix, corresponding to the six possible types of filter in the transfer function, are contained in quite distinct chapters of the programme. During execution a particular chapter is not entered if its corresponding type of filter is not present in the transfer function (as indicated by the value of $y[i]$). These chapters are all very similar in structure so we shall discuss, as a typical one, the $1/(s^2 + 2ns + m^2)$ chapter. This is shown in the first half of the third column of Appendix A section (a) following the heading:

<u>comment</u> $1/(s^2 + 2ns + m^2)$ Response Section;

See also the second half of the first column of the flow diagram, (Fig.1).

The various processes involved are as follows. Firstly, all the elements of response matrix 'd' are set to zero by use of the procedure 'set zero'. The elements of the response matrix corresponding to the 'standard' input functions $u_4(t), \ldots, t^3$ are set up. If any one of these functions is absent (i.e. has a zero coefficient) the corresponding elements in the response matrix are left as zeros. The remaining functions are dealt with similarly, with a block of elements set up in the appropriate part of the matrix for each independent pair of values of a and ω.

When the response matrix is complete it is multiplied by the input coefficient vector 'c' and the resulting output coefficient vector is stored in array 'e'. This is done by the procedure 'mxprod(e,c,d)'. The elements of the array 'e' are then transferred back to the array 'c' using the 'convert' procedure and the elements of the array 'c' are made of order unity by dividing them all by their average value, using the procedure 'normalise'. The normalising factor then multiplies the content of store 'cg', which at the start of the calculation contains the constant gain of the transfer function.

It frequently happens during this sequence of operations that, owing to inaccuracies in the computation, certain coefficients of the output coefficient vector appear as small numbers of the order of 10^{-8} to 10^{-6} when they should in fact, be zero. The programme sets any coefficient to zero by means of the 'testc' procedure if after normalisation of the vector the coefficient is less than 5×10^{-6}.

A $1/(s^2 + 2ns + m^2)$ filter with $0 < n < m$ will generate exponential-trigonometric functions. On output from a filter of this type, the coefficients of such generated functions are stored in the coefficient vector directly below those of existing input functions (and thus overwriting zeros inserted at the input stage). These generated functions are then incorporated fully into the input function format by modifying the input coefficient vector 'c' and the array 'b'. That completes the cycle for one such filter and the complete process is repeated for further filters of the same type.

The chapters for other types of filter are basically the same as that just described but with minor modifications. The 'numerator' filters s, $s + k$, $s^2 + 2ns + m^2$ do not generate new functions in the manner described above so the last stage for incorporating generated functions into the input vector is not required.

The $1/(s + k)$ filter produces exponential-polynomial functions. The processes of setting up the response matrix and for dealing with the generated functions differ in some respects from those described above since we allow for the possibility of the filter $1/(s + k)$ being up to three-fold repeated and, as a consequence, the possibility of an exponential-polynomial input group with the same parameter 'k' as the filter.

The $1/s$ filter chapter also differs from the standard case in that we allow functions of the type t^n, with $n > 3$, to be generated by the filter. Although, such functions do not fit into the standard input function format, their introduction at this stage presents no difficulties since this 1/s chapter is the last to be executed before output.

When all these matrix calculations are complete the current coefficient vector is multiplied by the current constant gain to give the final output coefficient vector.

Should an error occur at any stage, use is made of the two procedures 'write(string)' and 'outerror'. The first gives a print-out of the error indication (discussed in section 4.2), the second reads in any remaining data for the case that has failed. Any further cases on the data tape can then be run using the standard Elliott procedure 'restart'.

3.3 Output of results

The relevant part of the programme is shown in the third column of Appendix A section (c) and consists of three main parts.

The first gives a printout of the output function in explicit algebraic form following the programme heading and data title. The second part is headed:

<u>comment</u> Output to Programme II (Mean Square Programme);

This section is relevant only when the input is a random process and is discussed in Part II. When the problem concerns only deterministic input functions the parameter SAF has to be set to zero. This is done automatically by the use of the data sheet which contains the necessary zero immediately above the tabulation section.

The third part concerns tabulation of the output function. If no tabulation is required 'del' should be set to zero and this section is not entered during execution. Otherwise the tabulation data are read in and the output function is computed for $t_o(h_1)t_f$, the values of t_o, h_1 and t_f being reassigned for each change of interval of the tabulation.

3.4 Comments on the two-part programme

Since the existing programme is in two parts an explanation is now given of how the data is passed from the first half of the programme to the second.

By means of the first programme tape, the data title is read in and reproduced on the output tape but without the right hand string quote (the ? character). The input function and transfer function data are read in and the response calculations for the $1/(s + k)$ and $1/(s^2 + 2ns + m^2)$ filters are performed. If these calculations are completed without error the right hand string quote is punched on the output tape, followed by a 1, the remaining transfer function data, the current input function data and finally the tabulation data. If an error does occur the appropriate error indication is punched out followed by a right hand string quote and a zero.

When the output tape from the first part is read in as input data to the second part of the programme the programme title is punched out together with the information between string quotes on the data tape. If no error occurred in the first part the next character on the tape is a 1; the computer takes this as an indication that the computation is correct so far, continues to read in the remaining data and proceeds with the calculation. If, on the other hand, the next character is a zero the string already reproduced contains the error indication and the computer performs no further calculations on that case.

When several cases are to be dealt with at the same time they should all be punched on the same data tape. Part I calculations are performed on all the cases and then followed by all the Part II calculations.

It should be possible to run the programme in one piece on an Elliott 503 computer with more than 12K words of storage. The few alterations which would have to be made to the existing tapes before joining them together are listed in Appendix A section (d).

If the programme is to be used on a computer which will not accept Elliott 503 ALGOL (ATLAS is one computer which does have such a compiler), some alterations will have to be made to the programme. The computer should be one having an on-line teleprinter and two other fast output devices, (in the existing programmes these are referred to as punch (3) and punches (1) and (2) respectively).

Those sections of the programme which are most likely to require alteration are indicated by a vertical line at the side of the printed programme in Appendix A sections (a) and (b). The more obvious alterations are the switch lists, not required on most compilers, the input and output procedures and their associated setting procedures.

The existing programme also contains the following Elliott software procedures - all of which will have to be altered: 'elliot', 'restart', 'location', 'address', 'size' and 'range'. The 'elliot' procedure is used in the boolean procedure 'key(n)'. This allows the user, if he wishes, to control various steps in the computation by switching on appropriate key(s) on the computer console. 'restart' has already been mentioned and the remaining procedures are used in the 'mxprod' procedure. A matrix multiplication procedure could of course, have been written in standard ALGOL but using the above procedures helps to cut down the operating time.

Finally, many machines are capable of converting a programme in a given code to one in a different code, so it is possible that the necessary alterations could be done by the computer.

3.5 Further facilities on the Elliott 503 computer

By running the programme with key(1) on, the data title of each case is printed on the teleprinter together with any error indications and the word 'NXDATA' when each case has been completed. This makes it possible to keep a watch on the progress of the computations. However, it is advisable to use this facility sparingly since the teleprinter operates so slowly (c 10 characters/sec).

If the coefficient vector needs to be examined before and after every filter of the transfer function this can be achieved by running the programme with key(2) on, whereupon the coefficient vectors are all fed to punch (2). This facility is useful for checking any results obviously incorrect but which do not throw up any error indications (possibly due to faulty data punching).

4 THE USE OF THE PROGRAMME

In this section is contained all the information needed in order to be able to use the programme. It is therefore concerned primarily with the preparation of the data sheet, the format of the results produced by the computer, and possible causes of failure of the programme.

4.1 Preparation of the data sheet and output format

A copy of a blank data sheet is given in Fig.2. It may be seen to divide into the three main sections:- transfer function, input function and tabulation of results.

4.1.1 The title

The first piece of information to be punched on the data tape is the title. Each set of data run on the computer must have a title containing not more than thirty characters.

The opening character of the title must be a £, and the closing character a ?. The title may not contain any other £ or ? characters.

4.1.2 Transfer function

All information relating to the transfer function goes into the appropriate part of the left hand column.

The first parameter to be entered is the constant gain. There follows a block of constants y[1] to y[6] which dictate the structure of the transfer function by indicating the number of factors of each of the six types which may be present (see section 3). If any type is absent, the appropriate y[i] should be set equal to zero.

Taking them in order, y[1] is the number of $1/(s + k)$ factors present. This includes repeated factors $1/(s + k)^r$ where r is restricted in the present programme to be not greater than three. As an example, the transfer function $1/(s + 1)\ (s + 6.1)^2\ (s + 3.9)^3$ would have y[1] entered as 6.

y[2] is the number of quadratic factors of the form $1/(s^2 + 2ns + m^2)$ where $n < m$ and m is not zero. Repeated factors of this form have not been allowed for in the programme. y[3] is the number of s factors, i.e. the index r of s^r. y[4] is the number of $(s + k)$ factors present, including repeated factors $(s + k)^r$, in the same way as y[1] but, generally speaking, there is no restriction on the value of r. y[5] and y[6] are obtained in a similar manner of y[2] and y[3]. All the constants y[i] should be written as integers.

The parameters occurring in the transfer function are then entered in the blocks below. If any y[i] is zero, the corresponding block is left blank. When entering the values of k corresponding to $1/(s + k)$ factors, the k's of nonrepeated factors must precede those of any repeated factor. The k's of a repeated factor must be entered in consecutive squares. If there is more than one repeated factor the order in which they are taken is immaterial.

On the data sheet (Fig.2) space for only six factors of a given type has been allowed but extra rows can be added to any block if required.

When the data tape is being prepared, data should be punched in the order indicated by the dotted line, starting at the title, and ending at the label A.

4.1.3 The input function

The section on the upper right hand side of the data sheet is for setting up the input function.

The first block concerning the input functions $u_4(t)$ to t^3 must always be completed. If any one of these functions is present in the input function, the coefficient multiplying it is inserted on the appropriate line of the block; otherwise a zero is inserted.

The three blocks which follow cover the other three classes of function allowed for in the programme. Consider the exponential-polynomial group $e^{-at}(\alpha u_0(t) + \beta t + \gamma t^2)$. As mentioned in section 3 the programme does not allow for functions of higher order in t, such as $t^3 e^{-at}$, etc. When all the exponential terms are grouped so as to comply with this format, the number p[9] is the number of independent groups, i.e. the number of different parameters 'a' occurring in the exponential functions. The value of p[9] must be inserted even if it is zero. For each independent 'a' the block of multiplying coefficients (α,β,γ) is inserted and, in the lower block of the same column, the value of 'a' itself. For each value of 'a', the user has to insert a zero opposite the corresponding value of 'ω'. Precisely p[9] sets of data (α,β,γ) and (a,ω) have to be inserted. The sheet does not allow for p[9] > 3 but the user may add extra blocks if he wishes.

p[10] is the number of functions of the type $(\alpha \sin \omega t + \beta \cos \omega t)$ present in the input. When p[10] $\neq$ 0 the blocks in the second column must be filled in as described for the exponential-polynomial functions. For each independent 'ω', the corresponding 'a' must be written as zero.

p[11] indicates the presence or absence of exponential-trigonometric functions, and the data sheet is filled in in just the same way as for the other functions.

When the data tape is being prepared the punching order is indicated by the dotted line, starting at the label A and ending at the label C.

4.1.4 The output function and tabulation

The section on the lower right hand side of the data sheet is used for defining the format of the output function.

Firstly, the output function is always produced in explicit algebraic form immediately following the heading, thus:

Response of linear systems   Programme I
(title of data).
Output function.

Secondly, should the user require it, the output function will be computed and tabulated at times dictated by the quantities entered in the block headed 'TABULATION'.

The quantities t_0, t_1, t_2, ... , are the times at which the tabulation either starts or finishes, or at which the tabulation interval h_0, h_1 changes in magnitude. Thus, if the user wanted to tabulate for $t = 0(0.1)5$ and $t = 5(0.2)10$ he would enter $t_0 = 0$, $h_0 = 0.1$, $t_1 = 5$, $h_2 = 0.2$, and $t_2 = 10$. In this case the number of large time intervals, 'del', would be two.

The programme has a great deal of flexibility, since the input is computed from an explicit formula. The user may, if he wishes, start the computation at time t_0 not equal to zero. Again, by suitable manipulation of the tabulation blocks he can jump a large time interval without doing any intermediate calculations.

The number of large time intervals, 'del', is the suffix of the final value of t. If no tabulation is required, 'del' should be set equal to zero. The user may add extra rows to the tabulation block to allow for more than 4 large time intervals if required.

The zero following the label C must always be inserted. The significance of this zero is mentioned in section 3.3.

The data for the problem should be punched on Elliott 503, 8 hole paper tape, beginning with a new line, and ending with a new line, each number being separated from the previous one by a new line.

Should the user wish to run more than one set of data on the computer, he should include all the sets of data on one tape allowing, say, three inches of run-out between each set. It should be borne in mind that a case consists of all three ingredients: transfer function, input function and tabulation. The user is advised to give each set of data a different title, otherwise confusion may occur over the sets of results obtained from the computer.

4.1.5 Illustrative example

As an illustration of the way in which the data sheet should be filled in, Fig.3 shows a completed sheet for the following problem. Since it has no particular physical significance, the response has not been computed.

Transfer function:

$$\frac{0.634102(s^2 + 2 \times 8.8949s + 10.3265^2)\ (s^2 - 2 \times 4.7884s + 5.26^2)s}{(s - 0.31401)(s + 5.549)^2\ (s^2 + 2 \times 3.7372s + 6.21034^2)} \quad .$$

Input function:

$$3.507 + 6.25t^2 - 2.9131e^{-2.799t} + (0.5928t + 0.3160t^2)e^{-1.304t}$$
$$+ 5.3857e^{-1.9486t} \cos(7.2619t) \quad .$$

Tabulation:

For $t = 3(0.1)5$ and $6(0.05)10.5$.

4.2 Failure cases

As indicated in section 2, the programme copes with an extensive, but bounded range of input functions. Since it is possible to produce incorrect answers by going beyond the indicated bounds, certain error indications have been built into the programme, and these will now be briefly discussed.

4.2.1 Transfer function numerator errors

These are caused by the attempt to pass higher order generalised functions through differentiating elements. Details of the two error indications in this class will be found in Appendix B section (a). Here a typical one will be considered.

For example, suppose that the problem involved calculating the impulse response of a system with the following transfer function

$$\frac{(s + k_1)\ (s + k_2)\ (s + k_3)\ (s + k_4)\ (s + k_5)}{(s + k_0)\ s^5}$$

and the results tape contained only the programme title, the data title, and the following error indications:

$u_4(t)$ into filter number 6 .

The input function is $u_4(t)$, and the 'filter number' the stage in the computation when the error occurred, which in our example is the sixth filter in the chain, $(s + k_5)$. (When determining the number of the filter, it should be remembered that the order of computation is the same as that indicated when setting up the factors of the transfer function on the data sheet.)

The passing of a $u_4(t)$ function through the $(s + k_5)$ filter would cause a $u_5(t)$ function to be generated. This function lies outside the limits of the input function (see section 2), and would, therefore, not be acceptable to the next filter in the chain.

This fault is therefore seen to arise from the conjunction of an untypical form of transfer function (with the $1/s^5$ term) together with the order in which factors are dealt with in the programme. Nevertheless, even this case could be satisfactorily computed by the device of breaking it up into two stages

$$\xrightarrow{u_1(t)} \boxed{\frac{(s + k_1)\,(s + k_2)\,(s + k_3)}{(s + k_0)\,s^3}} \longrightarrow \boxed{\frac{(s + k_4)\,(s + k_5)}{s^2}} \longrightarrow$$

The problem is then run as two separate cases, the input function for the second one being the output function from the first.

4.2.2 Transfer function denominator errors

These errors occur during the passing of an input function through $1/(s + k)$ or $1/(s^2 + 2ns + m^2)$ filters of the transfer function. Details of these errors will be found in Appendix B section (b), and again we will discuss a typical one.

For example, suppose the problem involved calculating the response of a system with the following transfer function:

$$\frac{s(s^2 + 2ns + m^2)}{(s + k_1)\,(s + k_2)\,(s + k_3)^2}$$

for the input function $te^{-k_3 t}$, and the results tape contained only the programme title, the data title, and the following error indication:

failure case II filter number 3 .

The third filter in the transfer function is the first of the filters $1/(s + k_3)$, and the passing of the function $te^{-k_3 t}$ (which is one of the components of the output vector from the $1/(s + k_2)$ filter) through the filter $1/(s + k_3)^2$ will cause a $t^3e^{-k_3t}$ function to be generated. This function lies outside the bounds of allowable input functions, and would therefore not be acceptable to the next filter in the chain.

$$te^{-k_3t} \longrightarrow \boxed{\frac{1}{(s + k_3)}} \longrightarrow \tfrac{1}{2}t^2e^{-k_3t} \longrightarrow \boxed{\frac{1}{(s + k_3)}} \longrightarrow \tfrac{1}{6}t^3e^{-k_3t}$$

Although the $t^3e^{-k_3t}$ function is not generated until the second filter of the type $1/(s + k_3)$, it should be noted that the filter number given in the error indication will be that of the first filter of this type. This also applies to the failure cases I and III (see Appendix B section (b)).

The only method of dealing with the problem would be to remove the offending filter from the transfer function, and complete the calculation for this filter by hand using as input function the output function from the computer for the 'reduced' transfer function.

4.2.3 Discussion of error cases

It must be stressed that these error indications were built into the programme more as a safeguard than a restriction. Provided that the user has set up his input function and transfer function correctly on the data sheet he should have little trouble from errors, since they are unlikely to occur in most physical problems.

5 DISCUSSION

In a large comprehensive programme such as this there are numerous possibilities for error which have to be rooted out. The programme has been tested in several ways both by comparing results with those obtained by other methods and by a unique self-checking property of the method which will be discussed shortly.

The following is an example run to produce results which could be compared with those obtained another way; it also serves to illustrate the whole computational procedure involved. This example was taken from a paper by Steiglitz[5] for no other reason but that it provided a complicated looking transfer function for which the impulse and step responses were presented.

The transfer function was:

$$\frac{0.02191s^7 + 0.05325s^6 - 2.01s^5 + 11.93s^4 - 35.32s^3 + 59.84s^2 - 56.20s + 23.04}{s^8 + 8.823s^7 + 30.52s^6 + 86.42s^5 + 142.6s^4 + 189.8s^3 + 161.6s^2 + 89.29s + 23.00}$$

The roots of the numerator and denominator polynomials were obtained by the Deuce programme mentioned in section 2, and are tabulated in Fig.5. Each complex pair had to be manipulated to give the constants in a quadratic factor. The data sheet was then filled in as illustrated in Fig.4 for a unit impulse input function, tabulation of the response being requested for $t = 0(0.1)4(0.05)9(0.1)10$. Since we expected the response function to have a peak around $t = 6.3$ sec we arranged for the function to be tabulated at closer intervals in this region.

Fig.5 shows part of the tabulation and a plot of the response. As far as can be ascertained from the graph in Ref.5, our results and those of Steiglitz agree precisely. As a further check the responses to a unit step were also computed and again they agree with those given in Ref.5. According to Laplace transform theory, working from the transfer function, the asymptotic value of the response at $t \to \infty$ should be 23.04/23.00 or 1.00173913. The value given by the programme was, in fact, 1.0017391 and this, it should be remembered, was after several matrix operations.

The self-checking property of the method mentioned above relies on the fact that, if numerator and denominator both contain precisely the same factors, the output function should be identical with the input function. This provides an excellent way both of checking the programme for mistakes and of gaining some idea of the accuracy that is attainable.

An example is provided in Fig.6, which shows a comprehensive input function, incorporating nearly every possible type of component function, into a transfer function with a factor of each type. It can be seen that the coefficients in the output function differ only in the last decimal place from those of the input function, and that functions generated by the denominator filters emerge with zero coefficients (as, of course, they should).

This is a good example with very little error arising. Experience with the programme is, as yet, too limited for us to be able to say that such accuracy will always be attainable, and some recent calculations suggest that accuracy is lost when the denominator constants are small in magnitude in comparison with the other constants in the transfer function. It is too early to draw conclusions on this point but it may turn out that difficult cases will be better dealt with when run on a computer with a longer word length. For example, ATLAS has a word length of 44 BITS (12-13 significant figures) as opposed to 32 BITS (8-9 significant figures) on the Elliott 503.

Turning now to the question of economical computer usage, the time needed to run a case varies, of course, with the computer and its peripheral equipment but the following are approximate times for the first example quoted above when run on the Elliott 503 computer at Westcott.

To run on 1st tape of programme - 20 seconds
To compute response with 1st tape - 10 seconds
To run in 2nd tape of programme - 20 seconds
To compute response with 2nd tape - 10 seconds
To punch out tabulated response (151 points) - 50 seconds

This gives a total of 1 minute 50 seconds. If we allow time for the operators to load the tape readers, reset the computer etc., the total time could be around 2 minutes 15 seconds. However, if several cases had been computed in sequence, 40 seconds per case would have been saved by not having to run in the programme tapes each time. Also, the actual formulation of the output function is obtained in only 20 seconds, the remaining computing time being required for the tabulation of the output which would take at least as long by any other method.

With regard to further applications, a potential user may have a problem in which the input function is not of the required type. He could, however, use the programme to derive the unit impulse response from the transfer function and use this in a convolution programme.

With regard to extensions of the programme which are envisaged, a great deal of work is currently being done on methods of analysis of multivariable systems by the state space approach. It has been shown[4] that the serial/matrix technique could be usefully employed in connection with this; all the necessary algebraic formulations have been worked out and it is intended that they be programmed for computer.

PART II - RANDOM INPUT FUNCTIONS

6 INTRODUCTION AND BACKGROUND INFORMATION

The extension of the method to stationary random processes in linear systems is described in Ref.3. That paper, in two parts, gives two methods of obtaining the output autocorrelation function $\phi_{oo}(\tau)$ of the process when the input autocorrelation function $\phi_{ii}(\tau)$ is a linear combination of certain prescribed autocorrelation functions. Once $\phi_{oo}(\tau)$ is known the output mean square value is given by setting $\tau = 0$.

The system autocorrelation function method is based upon the equation

$$\phi_{oo}(t) = \int_{-\infty}^{\infty} \phi_{hh}(\tau)\, \phi_{ii}(t - \tau)\, d\tau \quad .$$

$\phi_{hh}(\tau)$ is the system autocorrelation function which is derived from the system unit impulse function $h(t)$ and defined by

$$\phi_{hh}(\tau) = \int_{-\infty}^{\infty} h(t)\, h(t + \tau)\, dt \quad .$$

Since $h(t)$ is given analytically as a linear combination of known functions of t by the deterministic input programme, simple matrix operations lead to $\phi_{hh}(\tau)$. Further matrix operations may then give $\phi_{oo}(t)$ by the first equation above.

This particular method was not programmed in full generality; it was decided to take it only as far as the determination of output mean square value using the equation

$$\sigma_o^2 = \phi_{oo}(0) = \int_{-\infty}^{\infty} \phi_{hh}(\tau)\, \phi_{ii}(\tau)\, d\tau \quad ,$$

since $\phi_{ii}(-\tau) = \phi_{ii}(\tau)$.

This Mean Square programme is discussed in section 7 and details are to be found in Appendix C. It is a supplementary programme following on from Programme I and using as input an output tape from that programme.

An allowable input autocorrelation function is any linear combination of functions of the following types:

(i) generalised function $u_1(\tau)$,

(ii) $e^{-a|\tau|}$, $|\tau| e^{-a|\tau|}$, $|\tau|^2 e^{-a|\tau|}$,

(iii) $e^{-a|\tau|} \sin \nu|\tau|$, $e^{-a|\tau|} \cos \nu|\tau|$.

The transfer function of the system giving unit impulse function $h(t)$ is restricted to having no factor $1/s$, i.e. the pure integration of a stationary random process is not considered.

The second method (the 'serial' method) is quite self contained and the programme based on it makes no use of Programmes I or II. In essence it is very similar to the deterministic input method. The input autocorrelation function is written as a linear combination of functions of the types listed above; an autocorrelation response matrix is defined for each of the filters in the filter chain representing the transfer function. The output autocorrelation function is given by multiplying the input coefficient vector by these response matrices in turn.

The Autocorrelation function programme implementing this method is discussed in section 8 and details are given in Appendix D.

7 PROGRAMME II - MEAN SQUARE PROGRAMME

7.1 Mode of operation

A copy of the programme is given in Appendix C and the flow diagram in Fig.7.

Data specifying the system transfer function is used in conjunction with Programme I to give the system unit impulse response function $h(t)$. The parameter S.A.F. is set to the value one thus indicating to the computer that the data has to be prepared for Programme II. The data title, and the analytic expression for $h(t)$ are printed out at the second punch followed by the input autocorrelation function which is reproduced directly from the data tape. This second data tape is then fed in with Programme II to produce the system autocorrelation function and output mean square.

The two procedures key(n) and mxprod are used. With key(1) on, the data title for each case followed by the word 'NXDATA' upon completion of that case are printed out on the on-line teleprinter. mxprod is a matrix product procedure as mentioned in section 3.

The coefficient vector of the system autocorrelation function is determined from the matrix product

$$c = a\,A\,B\ .$$

The programme reads in a, the coefficient vector of $h(t)$, and matrix B is constructed as an assembly of small sub-matrices involving only the components of a. The elements of A are then determined from the variables contained in the component functions of $h(t)$, i.e. the values of a and ω in the exponential and exp-trig functions. Having computed c the computer prints out the system autocorrelation function in a standard format.

The input autocorrelation function is read in, its coefficient vector being denoted by cc. The matrix E is set up, its elements being defined by the variables in the component functions of the S.A.F. and the input autocorrelation function. The programme then produces mean square value s by the calculation

$$s = c\ E\ cc \quad .$$

Having printed out s it goes on to the next case.

7.2 Use of the programme

In this section is contained all the information needed in order to be able to use the programme. It is concerned with the preparation of the data sheet and the format of the results produced by the computer.

A copy of a blank data sheet is shown in Fig.8 and may be seen to divide into two main sections headed transfer function and input autocorrelation function. There is, in addition, a string of 0's and 1's in the upper right hand corner which are used by the computer in conjunction with the transfer function data to produce the system autocorrelation function. This, together with the input autocorrelation function, gives the output mean square value.

The data title and the transfer function data are set up in the manner described in sections 4.1.1 and 4.1.2.

The first block of the input autocorrelation function section contains the coefficient of $u_1(\tau)$. A constant must always be inserted here. If $u_1(\tau)$ is not a component term of the input then the coefficient is set to zero.

The two blocks which follow cover the two classes of function allowed for in the programme. Consider the exponential-polynomial group

$$e^{-a|\tau|}\ (\alpha + \beta|\tau| + \gamma|\tau|^2) \quad .$$

(The programme does not allow for functions of higher order in τ than $\tau^2 e^{-a|\tau|}$.) All the exponential terms are grouped so as to comply with this format, and the number pp1 is the number of independent groups, i.e. the number of different parameters 'a' occurring in the exponential functions. The value of pp1 must be inserted even if it is zero. For each independent 'a', the block of multiplying coefficients (α,β,γ) is inserted, and in the lower block of the same column, the value of 'a' itself. Precisely ppl sets of data (α,β,γ) and (a) are required. The sheet does not allow for ppl greater than three but the user may add extra blocks if necessary.

pp2 is the number of functions of the type $e^{-a|\tau|}(\alpha \sin \omega|\tau| + \beta \cos \omega|\tau|)$ present in the input autocorrelation function. When $pp2 \neq 0$ then the blocks in the second column must be filled in as described for the exponential-polynomial expressions.

When the data tape is being prepared, the punching order is that indicated by the dotted line. The data should be punched on Elliott 503, 8 hole paper tape beginning with a new line, and ending with a new line, each number being separated from the previous one by a new line.

Should the user wish to run more than one set of data on the computer, he should include all the sets of data on one tape, allowing, say, three inches of blank tape between each set and giving each set of data a title.

The results tape will contain:

Title: Response of Linear Systems Programme II.

Data title.

System autocorrelation function in analysic form.

Value of the mean square response.

Since Programme I is used in the computation, if any errors occur the error indications will be those discussed in section 4.2.

8 PROGRAMME III - AUTOCORRELATION FUNCTION PROGRAMME

Because of the similarity between this programme and Programme I, reference will be made to sections 3 and 4 containing the description of Programme I.

8.1 Mode of operation

This section contains a brief discussion of the major processes in the programme. The programme is shown in Appendix D and its corresponding flow diagram in Fig.9.

The programme contains easily identifiable chapters corresponding to input, the elementary filters, output and tabulation. The mode of operation of these chapters is as described for Programme I in section 3, but with the restrictions that there should be no $1/s$ filter in the transfer function, and that the input function should contain only a unit impulse, exponential - polynomial and exponential - trigonometric functions. The names of some of the variables in this programme differ from the corresponding variables used in Programme I; however, they should be easily identified by comparing the lists of variables in Appendix A section (a) and Appendix D section (a). Programme III is all on one tape.

The facility of using the keys on the computer console is available, and depressing the keys has the same effect as that described in section 3.5.

8.2 Use of the programme

A copy of a blank data sheet is given in Fig.10. The information required for the title and transfer function is as described for Programme I in sections 4.1.1 and 4.1.2. For the input function, the coefficient of $u_1(\tau)$ must always be filled in (even if it is zero), and the method for setting up the coefficients of the exponential - polynomial and exponential - trigonometric functions is also the same as for Programme I (section 4.1.3). The information required for the output tabulation, is exactly that described in section 4.1.4.

The failure indications given by the programme are almost identical with those of Programme I (section 4.2) but, for exactness, they are listed in Appendix E.

Appendix A

PROGRAMME I - DETERMINISTIC INPUT FUNCTIONS

(a) List of variables and procedures

a, k, w, n, m	real	variables in I.F. and T.F.
A[0:10]	integer	stores data title
b[1:r[12], 1:2]	real	holds a and w values, b[i,1] ≡ a, b[i,2] ≡ w
cc[1:8]	real	holds first 8 input coefficients
cg	real	constant gain of T.F., later used for normalising factor
count	integer	number of factors dealt with
c[1:1, 1:q[12]]	real	input coefficient vector
d[1·q[12], 1:q[12]]	real	response matrix
e[1:1, 1:q[12]]	real	output coefficient vector
del	integer	number of large tabulation intervals
f1[1:y[1]]	real	holds k's of $1/(s + k)$ factors
f21[1:y[2]]	real	holds n's of $1/(s^2 + 2ns + m^2)$ factors
f22[1:y[2]]	real	holds m's of $1/(s^2 + 2ns + m^2)$ factors
f4[1:y[4]]	real	holds k's of $(s + k)$ factors
f51[1:y[5]]	real	holds n's of $(s^2 + 2ns + m^2)$ factors
f52[1:y[5]]	real	holds m's of $(s^2 + 2ns + m^2)$ factors
iA	integer	constant used in storing data title
ny[0]	integer	not significant, always zero
ny[1], ny[2], etc.	integer	number of factors of each $1/(s + k)^n$ type (i.e. value of index n), excluding n = 1
p[1]	integer	value 1 if $u_4(t)$ present in input, otherwise 0
p[2]	integer	value 1 if $u_3(t)$ present in input, otherwise 0

p[3]	integer	value 1 if $u_2(t)$ present in input, otherwise 0
p[4]	integer	value 1 if $u_1(t)$ present in input, otherwise 0
p[5]	integer	value 1 if $u_0(t)$ present in input, otherwise 0
p[6]	integer	value 1 if t present in input, otherwise 0
p[7]	integer	value 1 if t^2 present in input, otherwise 0
p[8]	integer	value 1 if t^3 present in input, otherwise 0
p[9]	integer	number of exp-poly functions in input
p[10]	integer	number of trig. functions in input
p[11]	integer	number of exp-trig. functions in input
q[9]	integer	position of last coefficient of exp-poly functions in input vector
q[10]	integer	position of last coefficient of trig. functions in input vector
q[11]	integer	position of last coefficient of exp-trig. functions in input vector
q[12]	integer	length of final coefficient vector
r1	integer	used in $1/(s + k)$ chapter when input function is passed through repeated factors
R1	integer	
r2	integer	
R2	integer	
r[9]	integer	position of last a, w values in exp-poly fns in b array
r[10]	integer	position of last a, w values in trig. fns in b array
r[11]	integer	position of last a, w values in exp-trig. fns in b array
r[12]	integer	length of final b array
SAF	integer	value 1 if data present for Programme II, otherwise 0
y[1]	integer	number of $1/(s + k)$ factors
y[2]	integer	number of $1/(s^2 + 2ns + m^2)$ factors
y[3]	integer	number of s factors

y[4]	integer	number of $(s + k)$ factors
y[5]	integer	number of $(s^2 + 2ns + m^2)$ factors
y[6]	integer	number of $1/s$ factors
yy1	integer	total number of factors in $1/(s + k)^n$ terms
yy2	integer	number of terms of the type $1/(s + k)^n$

Variables in $1/(s + k)$ chapter

ak	integer	position of exp-poly input function with parameter k for a single $1/(s + k)$ factor (otherwise zero)
akn	integer	position of exp-poly input function with parameter k for a repeated $1/(s + k)$ factor (otherwise zero)
d1	real	auxiliary variable
g	integer	position of 1st coefficient of generated exp-poly function
m2	integer	value of p[10] + p[11]
m1	integer	used in computation of $1/(s + k)^n$ term; determines the current factor
z[1:3]	real	coefficients of generated exp-poly functions when transferring to correct position in input

Variables in $1/(s^2 + 2ns + m^2)$ chapter

x1	real	$m^2 - n^2$
x2	real	$(m^2 - n^2)^{\frac{1}{2}}$

Variables in 'output to Tape 2' section

co	real	stores to read data into computer and reproduce on output tape
coe	real	

Variable in $(s^2 + 2ns + m^2)$ chapter

x3	real	$m^2 - n^2$

Variable in $1/s$ block

extra	integer	for functions in input of type t^n, $n > 3$, extra is index n minus 3

Variables in tabulation block

B2	integer	number of time increments in one large time interval

fn	real	value of function at time t
h1	real	time increment
m2	integer	count variable
R4	integer	used in tabulation of first value of function
to	real	initial time value in one large time interval
tf	real	final time value in one large time interval
t	real	current value of time

List of procedures

key(n)	takes logical value TRUE or FALSE if the key on the computer console of value n is switched on.
write(string)	prints string on output device, and on teleprinter if key(1) on. String is a set of characters between the £? string quotes.
<u>if</u> tp(n) <u>then</u>	equivalent to: <u>if</u> p[n] $\neq$ 0 <u>then</u>
<u>if</u> ty(n) <u>then</u>	equivalent to: <u>if</u> y[n] $\neq$ 0 <u>then</u>
readr(n,B)	reads in values of the real array B[i] for i taking values from 1 to n.
set(n)	used in the section of programme which determines the $1/(s + k)^n$ terms by testing values of k.
outerror	reads in SAF data (if any) and tabulation data (if any). If key(1) on, prints NXDATA on teleprinter and then restarts the programme. Uses real variable coe to read in the real numbers.
mxprod(A,B,C)	performs matrix operation A: = B $\times$ C
setzero	sets all elements of R.M. to zero.
testc	tests coefficients of 1st 8 fns in the input, and sets corresponding p to zero or 1 as required.
convert	converts elements of e array to corresponding elements in c array.
normalise	normalises input coefficient vector; nn used to accumulate coefficients and find the mean; normalising factor accumulated in cg.

(d) Alterations needed to produce programme on single tape

Alterations to tape 1.

(a) Remove 'Tape 1' from the title of the programme.

(b) Insert the variable 'extra' in the first integer declaration.

(c) The last but one instruction in the 'outerror' procedure should read:

print £ £ 1 2 r 10 ? ? ;

instead of

print £ £ u 1 ? 0 £ 1 r 10 ? ? ;

(d) Before the first call of the procedure 'instring' insert the following:

extra := 0 ;

print £ £ 1 4 ? Response of Linear Systems Programme I £ 1 2 ? ? ;

(e) The line reading:

print ££ 1q ??, sameline, outstring (A,iA) ;

should be replaced by:

outstring (A,iA) ;

(f) The assignment of the variable q[12] should now read:

q[12] := q[11] + 3*(y[1] - yy1 + yy2) + 2*y[2] + y[6] ;

Alterations to tape 2.

(g) The calls of 'write' should be altered. For example, replace

write (£ u4(t) into filter number ?) ;

by

write (£ u4(t) into ?) ;

Having made these alterations the complete programme is obtained by joining the part of the first tape from the title up to, but not including, the **comment** "Output of data for input to Tape II" to that section of the second tape from, and including, the title **comment** s Response Chapter to the end.

Response of Linear Systems Programme 1 Tape 1   E.Huntley L.J.Hazlewood,

```
begin integer array p[1:11],q,r[9:12],y[1:6],A[0:10],ny[0:20];
      integer h,i,j,r1,R1,yy1,yy2,del,count,SAF,iA, real cg;
      boolean procedure key(n),
      value n; integer n,
      begin elliot(7,0,0,0,2,3,n),
            key:=n≠0
      end;
      procedure write(string);
      string string,
      begin print ££l??,string,££s?filter number£s??,sameline,leadzero(£?),count;
            if key(1) then print punch(3),££l??,string,££s?filter number£s??,
            sameline,leadzero(£?),count
      end;
      boolean procedure tp(n);
      value n; integer n;
      tp:=p[n]≠0;
      boolean procedure ty(n),
      value n; integer n;
      ty:=y[n]≠0,
      procedure readr(n,B);
      value n; integer n, array B,
      begin integer i;
            for i:=1 step 1 until n do read B[i]
      end;
      procedure set(n);
      value n; integer n;
      begin j:=j+1; ny[j]:=n
      end,
      procedure outerror;
      begin integer i,j,n; real coe;
            read SAF;
            if SAF≠0 then
            begin read coe,i,j;
                  n:=4*(i+j),
                  if n≠0 then
                  begin for i:=1 step 1 until n do read coe
                  end
            end;
            read del,
            if del≠0 then
            begin for i:=1 step 1 until 2*del+1 do read coe
            end;
            if key(1) then print punch(3),££l2?NXDATA?;
            print ££ul?0£lr10??;
            restart
      end;
      procedure mxprod(A,B,C);
      array A,B,C;
      begin integer aa,ab,ac,ra2,rb2,j,jstop,l,lstop,m,mstart,sa, real sum;
            aa:=address(A); sa:=size(A)+aa-1,
            ab:=address(B), ac:=address(C);
            ra2:=range(A,2); rb2:=range(B,2);
            if aa=ab or aa=ac or range(C,2) ≠ ra2
            or range(C,1) ≠ rb2 or range(A,1) ≠ range(B,1)
            then
            begin write(£mxprod error?),
                  outerror
            end;
            for aa:=aa step ra2 until sa do
            begin jstop:=aa+ra2-1; mstart:=ac-1;
                  for j:=aa step 1 until jstop do
                  begin m:=mstart:=mstart+1;
                        lstop:=ab+rb2-1; sum:=0,
                        for l:=ab step 1 until lstop do
                        begin sum:=sum+location(l)*location(m);
                              elliott(3,0,ra2,0,2,4,m);
                        end;
                        location(j):=sum
                  end,
                  ab:=ab+rb2
            end
      end mxprod;
      count:=iA:=0;
      instring(A,iA);
      iA:=0,
      print ££lq??,sameline,outstring(A,iA);
      if key(2) then
      begin iA:=0;
            print punch(2),££r10l4??,outstring(A,iA),£Tape 1?
      end,
      if key(1) then
      begin iA:=0,
            print punch(3),££l??,outstring(A,iA)
      end;
read cg,y[1],y[2],y[3],y[4],y[5],y[6];
begin array f1[1:y[1]],f21,f22[1:y[2]],f4[1:y[4]],f51,f52[1:y[5]],cc[1:8],
      if ty(1) then readr(y[1],f1),
      if ty(2) then
      begin readr(y[2],f21),
            readr(y[2],f22)
      end;
      if ty(4) then readr(y[4],f4);
      if ty(5) then
      begin readr(y[5],f51),
            readr(y[5],f52)
      end;
      j:=ny[0]:=yy1:=yy2:=0,
      if ty(1) then
      begin if y[1]>1 then
            begin if y[1]=2 and abs(f1[1]-f1[2])<_10-6 then set(2),
                  if y[1]>2 then
                  begin for i:=3 step 1 until y[1] do
                        begin if abs(f1[i]-f1[i-1])<_10-6 and
                              abs(f1[i-1]-f1[i-2])<_10-6 then set(3) else
                              if abs(f1[i-1]-f1[i-2])<_10-6 and ny[j]≠3 then
                              set(2) else
                              if abs(f1[i]-f1[i-1])<_10-6 and i=y[1] then set(2)
                        end
                  end
            end;
            yy2:=j;
            if yy2≠0 then
            begin r1:=1, R1:=ny[1],
                  for i:=1 step 1 until yy2 do yy1:=yy1+ny[i]
            end
            else R1:=y[1]
      end;
      readr(8,cc);
      read p[9],p[10],p[11];
      q[9]:=8+3*p[9]; r[9]:=p[9],
      q[10]:=q[9]+2*p[10]; r[10]:=r[9]+p[10],
      q[11]:=q[10]+2*p[11]; r[11]:=r[10]+p[11];
      q[12]:=q[11]+3*(y[1]-yy1+yy2)+2*y[2];
      r[12]:=r[11]+y[1]-yy1+yy2+y[2],
      begin array e,c[1:1,1:q[12]],d[1:q[12],1:q[12]],b[1:r[12],1:2];
            procedure setzero,
            begin integer i,j,
                  for i:=1 step 1 until q[12] do
                  begin for j:=1 step 1 until q[12] do d[i,j]:=0
                  end
            end setzero,
            procedure testc,
            begin integer i;
                  for i:=1 step 1 until 8 do
                  begin if abs(c[1,i])<5_10-6 then p[i]:=0 else p[i]:=1
                  end
            end testc,
            procedure convert,
            begin integer i;
                  for i:=1 step 1 until q[12] do c[1,i]:=e[1,i]
            end convert;
            procedure normalise,
            begin integer i,j, real nn;
                  nn:=0,
                  for i:=1 step 1 until q[12] do nn:=nn+abs(c[1,i]),
                  nn:=nn/q[12];
                  if key(2) then print punch(2),££l4??,
                  for i:=1 step 1 until q[12] do
                  begin c[1,i]:=c[1,i]/nn;
                        if key(2) then print punch(2),c[1,i]
                  end;
                  cg:=cg*nn;
                  if key(2) then print punch(2),££l2??,cg
            end normalise;
            for i:=1 step 1 until 8 do c[1,i]:=cc[i],
            if q[11]>8 then
            begin for i:=9 step 1 until q[11] do read c[1,i];
                  for i:=1 step 1 until r[11] do read b[i,1],b[i,2]
            end;
            if r[12]>r[11] then
            begin for i:=r[11]+1 step 1 until r[12] do b[i,1]:=b[i,2]:=0
            end,
            if q[12]>q[11] then
            begin for i:=q[11]+1 step 1 until q[12] do c[1,i]:=0
            end;
            normalise,
            testc,
comment 1/(s+k) Response chapter,

if ty(1) then
begin real k,a,w,d1; array z[1:3];
      integer h,i,j,m1,m2,r2,R2,g,ak,akn; switch ss:=L3,L4;
      r2:=ak:=akn:=0; R2:=y[1]-yy1+1;
      for h:=1 step 1 until y[1] do
      begin count:=count+1,
            k:=f1[h];
            if h>R2 then
            begin m1:=m1+1, d[g+m1-1,g+m1]:=1/m1
            end
            else
            begin m1:=0,
                  setzero,
                  g:=q[11]+1
            end,
            if tp(1) then
            begin d[1,2]:=1; d[1,3]:=-k,
                  d[1,4]:=k↑2; d[1,g]:=-k↑3
            end,
            if tp(2) then
            begin d[2,3]:=1; d[2,4]:=-k, d[2,g]:=k↑2
            end;
            if tp(3) then
            begin d[3,4]:=1, d[3,g]:=-k
            end;
            if tp(4) then d[4,g]:=1,
            if tp(5) then
            begin d[5,5]:=1/k; d[5,g]:=-d[5,5]
            end;
            if tp(6) then
            begin d[6,5]:=-1/k↑2; d[6,6]:=1/k; d[6,g]:=-d[6,5]
            end;
            if tp(7) then
            begin d[7,5]:=2/k↑3; d[7,6]:=-2/k↑2,
                  d[7,7]:=1/k; d[7,g]:=-d[7,5]
            end;
            if tp(8) then
            begin d[8,5]:=-6 k↑4, d[8,6]:=6/k↑3;
                  d[8,7]:=-3/k↑2; d[8,8]:=1/k; d[8,g]:=-d[8,5]
            end;
            if h>R2 then goto L3;
            if tp(9) then
            begin for i:=1 step 1 until p[9] do
                  begin a:=b[i,1]; j:=6+3*i,
                        if abs(k-a)<_10-6 then
                        begin if abs(c[1,j+2])>_10-6 then
                              begin write(£failure case I?);
                                    outerror
                              end;
                              if h>y[1]-yy1 then
                              begin if abs(c[1,j+1])>_10-6 then
                                    begin write(£failure case II?);
                                          outerror
                                    end,
                                    if ny[r1]=3 then
                                    begin write(£failure case III?);
                                          outerror
                                    end,
                                    akn:=j
                              end
                              else ak:=j,
                              d[j,j+1]:=1; d[j+1,j+2]:=.5
                        end
                        else
                        begin d[j,j]:=d[j+1,j+1]:=d[j+2,j+2]:=1/(k-a);
                              d[j,g]:=-d[j,j]; d[j+1,j]:=-1/(k-a)↑2;
                              d[j+1,g]:=-d[j+1,j], d[j+2,j]:=2/(k-a)↑3,
                              d[j+2,j+1]:=-2/(k-a)↑2, d[j+2,g]:=-d[j+2,j]
                        end
                  end
            end;
            if tp(10) then
            begin for i:=1 step 1 until p[10] do
                  begin w:=b[r[9]+i,2], j:=q[9]+2*i-1; d1:=k↑2+w↑2;
                        d[j,j]:=d[j+1,j+1]:=k/d1, d[j+1,g]:=-k/d1,
                        d[j,g]:=d[j+1,j]:=w/d1; d[j,j+1]:=-w/d1
                  end
            end,
            if tp(11) then
            begin for i:=1 step 1 until p[11] do
                  begin a:=b[r[10]+i,1], w:=b[r[10]+i,2],
                        j:= [10]+2*i-1; d1:=(k-a)↑2+w↑2,
                        d[j,j]:=d[j+1,j+1]:=(k-a)/d1;
                        d[j+1,g]:=-d[j,j], d[j+1,j]:=d[j,g]:=w/d1;
                        d[j,j+1]:=-d[j+1,j]
                  end
            end;
```

App.A(b) Programme 1, Tape 1

```
L3:     mxprod(e,c,d);
        convert;
        normalise;
        testc;
        begin switch sss:=L5;
              if h ≤ y[1]-yy1 then
              begin if ak≠0 then
                    begin c[1,ak]:=c[1,ak]+c[1,g],
                          c[1,g]:=0, ak:=0, goto L4
                    end
                    else
                    begin z[1]:=c[1,g]; z[2]:=z[3]:=0,
                          c[1,g]:=0; goto L5
                    end
              end;
              if yy2≠0 and y[1]-yy1+R1=h then
              begin if h≠y[1] then
                    begin r1:=r1+1; R1:=R1+ny[r1],
                          r2:=r2+1; R2:=R2+ny[r2]
                    end;
                    if akn≠0 then
                    begin c[1,akn]:=c[1,akn]+c[1,g],
                          c[1,akn+1]:=c[1,akn+1]+c[1,g+1],
                          c[1,g]:=c[1,g+1]:=0;
                          akn:=0; goto L4
                    end;
                    for i:=1 step 1 until 3 do
                    begin z[i]:=c[1,g+i-1]; c[1,g+i-1]:=0
                    end;
                    goto L5
              end;
              goto L4,
        L5:         begin m2:=p[10]+p[11],
                    if m2≠0 then
                    begin for i:=1 step 1 until 2*m2 do
                          begin j:=g+3-i; c[1,j]:=c[1,j-3]
                          end,
                          for i:=1 step 1 until m2 do
                          begin j:=r[11]+2-i,
                                b[j,1]:=b[j-1,1]; b[j,2]:=b[j-1,2]
                          end
                    end;
                    for i:=1 step 1 until 3 do
                    begin j:=8+3*p[9]+i; c[1,j]:=z[i]
                    end;
                    p[9]:=p[9]+1;  [9]:=8+3*p[9]; r[9]:=p[9];
                    b[p[9],1]:=k; b[p[9],2]:=0,
                    for i:=10 step 1 until 11 do
                    begin q[i]:=q[i-1]+2*p[i], r[i]:=r[i-1]+p[i]
                    end
              end
        end;
L4:  end
end;

comment 1/(s↑2+2ns+m↑2) Response chapter;

if ty(2) then
begin real a,w,n,m,x1,x2,z1,z2,d1, integer h,i,j,g;
      for h:=1 step 1 until y[2] do
      begin n:=f21[h]; m:=f22[h], x1:=m↑2-n↑2; count:=count+1;
            if abs(m)<⏨-6 or abs(x1)<⏨-6 or x1<0 then
            begin write(£failure case IV?),
                  outerror
            end,
            x2:=sqrt(x1);
            setzero,
            g:=q[11]+1,
            if tp(1) then
            begin d[1,3]:=1, d[1,4]:=-2*n;
                  d[1,g]:=-n*(n↑2-3*x1)/x2; d[1,g+1]:=3*n↑2-x1
            end;
            if tp(2) then
            begin d[2,4]:=1, d[2,g]:=(n↑2-x1)/x2, d[2,g+1]:=-2*n
            end;
            if tp(3) then
            begin d[3,g]:=-n/x2; d[3,g+1]:=1
            end;
            if tp(4) then d[4,g]:=1/x2;
            if tp(5) then
            begin d[5,5]:=1/m↑2; d[5,g]:=-n/x2/m↑2;
                  d[5,g+1]:=-d[5,5]
            end,
            if tp(6) then
            begin d[6,5]:=-2*n/m↑4, d[6,6]:=1/m↑2,
                  d[6,g]:=(n↑2-x1)/x2/m↑4; d[6,g+1]:=-d[6,5]
            end;
            if tp(7) then
            begin d[7,5]:=2*(3*n↑2-x1)/m↑6; d[7,6]:=-4*n/m↑4;
                  d[7,7]:=1/m↑2; d[7,g]:=-2*n*(n↑2-3*x1)/x2/m↑6;
                  d[7,g+1]:=-d[7,5]
            end;
            if tp(8) then
            begin d[8,5]:=-24*n*(n↑2-x1)/m↑8, d[8,6]:=6*(3*n↑2-x1)/m↑6,
                  d[8,7]:=-6*n/m↑4, d[8,8]:=1/m↑2,
                  d[8,g]:=6*(n↑4-6*n↑2*x1+x1↑2)/x2/m↑8;
                  d[8,g+1]:=-d[8,5]
            end,
            if tp(9) then
            begin for i:=1 step 1 until p[9] do
                  begin a:=b[i,1]; d1:=(a-n)↑2+x1, j:=6+3*i;
                        d[j,j]:=d[j+1,j+1]:=d[j+2,j+2]:=1/d1;
                        d[j,g]:=(a-n)/x2/d1; d[j,g+1]:=-d[j,j];
                        d[j+1,j]:=2*(a-n)/d1↑2;
                        d[j+1,g]:=((a-n)↑2-x1)/x2/d1↑2;
                        d[j+1,g+1]:=-d[j+1,j];
                        d[j+2,j]:=2*(3*(a-n)↑2-x1)/d1↑3;
                        d[j+2,j+1]:=4*(a-n)/d1↑2,
                        d[j+2,g]:=2*(a-n)*((a-n)↑2-3*x1)/x2/d1↑3,
                        d[j+2,g+1]:=-d[j+2,j]
                  end
            end;
            if tp(10) then
            begin for i:=1 step 1 until p[10] do
                  begin w:=b[r[9]+i,2]; d1:=(n↑2+x1-w↑2)↑2+4*n↑2*w↑2;
                        j:=q[9]+2*i-1,
                        d[j,j]:=d[j+1,j+1]:=(n↑2+x1-w↑2)/d1;
                        d[j+1,g+1]:=-d[j,j];
                        d[j+1,j]:=d[j,g+1]:=2*n*w/d1,
                        d[j,j+1]:=-d[j+1,j];
                        d[j,g]:=w*(n↑2-x1+w↑2)/x2/d1,
                        d[j+1,g]:=-n*(n↑2+x1+w↑2)/x2/d1
                  end
            end,
            if tp(11) then
            begin for i:=1 step 1 until p[11] do
                  begin a:=b[r[10]+i,1]; w:=b[r[10]+i,2];
                        if abs(a-n)<⏨-6 and abs(x1-w↑2)<⏨-6 then
                        begin write(£failure case V?);
                              outerror
                        end;
                        j:= [10]+2*i-1;
                        d1:=((a-n)↑2+x1-w↑2)↑2+4*(a-n)↑2*w↑2;
                        d[j,j]:=d[j+1,j+1]:=((a-n)↑2+x1-w↑2)/d1;
                        d[j+1,g+1]:=-d[j,j];
                        d[j+1,j]:=d[j,g+1]:=-2*(a-n)*w/d1,
                        d[j,j+1]:=-d[j+1,j];
                        d[j,g]:=w*((a-n)↑2-x1+w↑2)/x2/d1,
                        d[j+1,g]:=(a-n)*((a-n)↑2+x1+w↑2)/x2/d1
                  end
            end;
            mxprod(e,c,d);
            convert;
            normalise;
            testc,
            if abs(n)>⏨-6 then
            begin p[11]:=p[11]+1, q[11]:= [10]+2*p[11];
                  r[11]:=r[10]+p[11],
                  b[r[11],1]:=n; b[r[11],2]:=x2
            end
            else
            begin z1:=c[1,g]; z2:=c[1,g+1]; c[1,g]:=c[1,g+1]:=0;
                  if p[11]≠0 then
                  begin for i:=1 step 1 until 2*p[11] do
                        begin j:=g+2-i; c[1,j]:=c[1,j-2]
                        end;
                        for i:=1 step 1 until p[11] do
                        begin j:=r[11]+2-i,
                              b[j,1]:=b[j-1,1]; b[j,2]:=b[j-1,2]
                        end
                  end,
                  j:=8+3*p[9]+2*p[10]+1,
                  c[1,j]:=z1; c[1,j+1]:=z2;
                  p[10]:=p[10]+1; q[10]:=q[9]+2*p[10],
                  r[10]:=r[9]+p[10];
                  q[11]:=q[10]+2*p[11]; r[11]:=r[10]+p[11];
                  b[r[10],1]:=0; b[r[10],2]:=x2
            end
      end
end;

comment Output of data for input to tape 2;

print ££ul?1£l??,count;
print scaled(8),cg;
print y[3],y[4],y[5],y[6],
if ty(4) then
begin for i:=1 step 1 until y[4] do print scaled(8),f4[i]
end;
if ty(5) then
begin for i:=1 step 1 until y[5] do print scaled(8),f51[i];
      for i:=1 step 1 until y[5] do print scaled(8),f52[i]
end,
for i:=1 step 1 until 8 do print scaled(8),c[1,i],
print p[9],p[10],p[11];
if q[11]>8 then
begin for i:=9 step 1 until q[11] do print scaled(8),c[1,i];
      for i:=1 step 1 until r[11] do print scaled(8),b[i,1],b[i,2]
end,
read SAF;
print SAF;
if SAF≠0 then
begin real co;
      read co,i,j,
      print co,i,j;
      h:=4*(i+j);
      if h≠0 then
      begin for i:=1 step 1 until h do
            begin read co;
                  print co
            end
      end
end;
read del;
print del;
if del≠0 then
begin real ce;
      for i:=1 step 1 until 2*del+1 do
      begin read ce;
            print ce
      end
end;

print ££l r10??;
if key(1) then print punch(3),££l2?NXDATA?;
restart
end
end
end;
```

```
Response of Linear Systems  Programme I  Tape 2   E.Huntley  L.J.Hazlewood;

begin integer array p[1:11],q,r[9:12],y[3:6],A[0:20],
        integer h,i,j,extra,del,count,SAF,iA, real cg;
        boolean procedure key(n);
        value n; integer n;
        begin elliot(7,0,0,0,2,3,n);
              key:=n≠0
        end;
        procedure write(string);
        string string;
        begin print ££l??,string,££s??,sameline,leadzero(£?),count;
              if key(1) then print punch(3),££l??,string,££s??,
              sameline,leadzero(£?),count
        end;
        boolean procedure tp(n);
        value n, integer n;
        tp:=p[n]≠0;
        boolean procedure ty(n);
        value n, integer n;
        ty:=y[n]≠0;
        procedure outerror;
        begin integer i,j,n; real coe;
              read SAF,
              if SAF≠0 then
              begin read coe,i,j;
                    n:=4*(i+j);
                    if n≠0 then
                    begin for i:=1 step 1 until n do read coe
                    end
              end;
              read del;
              if del≠0 then
              begin for i:=1 step 1 until 2*del+1 do read coe
              end;
              if key(1) then print punch(3),££l2?NXDATA?,
              print ££l2r10??;
              restart
        end;
        procedure mxprod(A,B,C);
        array A,B,C;
        begin integer aa,ab,ac,ra2,rb2,j,jstop,l,lstop,m,mstart,sa; real sum;
              aa:=address(A), sa:=size(A)+aa-1;
              ab:=address(B); ac:=address(C),
              ra2:=range(A,2), rb2:=range(B,2);
              if aa=ab or aa=ac or range(C,2) ≠ ra2
              or range(C,1) ≠ rb2 or range(A,1) ≠ range(B,1)
              then
              begin write(£mxprod error?);
                    outerror
              end;
              for aa:=aa step ra2 until sa do
              begin jstop:=aa+ra2-1, mstart:=ac-1,
                    for j:=aa step 1 until jstop do
                    begin m:=mstart:=mstart+1;
                          lstop:=ab+rb2-1; sum:=0;
                          for l:=ab step 1 until lstop do
                          begin sum:=sum+location(l)*location(m),
                                      elliott(3,0,ra2,0,2,4,m);
                          end;
                          location(j):=sum
                    end;
                    ab:=ab+rb2
              end
        end mxprod;
        extra:=iA:=0,
        instring(A,iA);
        print ££l4?Response of Linear Systems Programme I£l2??,
        iA:=0;
        outstring(A,iA);
        if key(2) then
        begin iA:=0;
              print punch(2),££r10l4??,outstring(A,iA),£Tape 2?
        end;
        if key(1) then
        begin iA:=0;
              print punch(3),££l??,outstring(A,iA)
        end;
        read h,
        if h=0 then
        begin if key(1) then print punch(3),££l2?NXDATA?,
              print ££l2r10??,
              restart
        end,
        read count,cg,y[3],y[4],y[5],y[6];
        begin array f4[1:y[4]],f51,f52[1:y[5]],cc[1:8];
              if ty(4) then
              begin for i:=1 step 1 until y[4] do read f4[i]
              end,
              if ty(5) then
              begin for i:=1 step 1 until y[5] do read f51[i];
                    for i:=1 step 1 until y[5] do read f52[i]
              end;
              for i:=1 step 1 until 8 do  read cc[i];
              read p[9],p[10],p[11];
              q[9]:=8+3*p[9]; r[9]:=p[9],
              q[10]:= [9]+2*p[10]; r[10]:=r[9]+p[10];
              q[11]:=q[10]+2*p[11], r[11]:=r[10]+p[11];
              q[12]:= [11]+y[6];
              begin array e,c[1:1,1:q[12]],d[1:q[12],1:q[12]],b[1:r[11],1:2];
                    procedure setzero;
                    begin integer i,j,
                          for i:=1 step 1 until q[12] do
                          begin for j:=1 step 1 until q[12] do d[i,j]:=0
                          end
                    end setzero;
                    procedure testc;
                    begin integer i;
                          for i:=1 step 1 until 8 do
                          begin if abs(c[1,i])<5₁₀-6 then p[i]:=0 else p[i]:=1
                          end
                    end testc,
                    procedure convert,
                    begin integer i;
                          for i:=1 step 1 until q[12] do c[1,i]:=e[1,i]
                    end convert;
                    procedure normalise,
                    begin integer i,j; real nn,
                          nn:=0;
                          for i:=1 step 1 until q[12] do nn:=nn+abs(c[1,i]),
                          nn:=nn/q[12];
                          if key(2) then print punch(2),££l4??;
                          for i:=1 step 1 until q[12] do
                          begin c[1,i]:=c[1,i]/nn;
                                if key(2) then print punch(2),c[1,i]
                          end,
                          cg:=cg*nn,
                          if key(2) then print punch(2),££l2??,cg
                    end normalise;
                    for i:=1 step 1 until 8 do c[1,i]:=cc[i];
                    if q[11]>8 then
                    begin for i:=9 step 1 until q[11] do read c[1,i];
                          for i:=1 step 1 until r[11] do read b[i,1],b[i,2]
                    end;
                    if q[12]>q[11] then
                    begin for i:=q[11]+1 step 1 until q[12] do c[1,i]:=0
                    end,
                    normalise;
                    testc;

comment s Response chapter;

if ty(3) then
begin real a,w; integer h,i,j;
      for h:=1 step 1 until y[3] do
      begin setzero, count:=count+1;
            if tp(1) then
            begin write(£u4(t) into filter number?);
                  outerror
            end;
            for i:=2 step 1 until 6 do
            begin if tp(i) then d[i,i-1]:=1
            end;
            if tp(7) then d[7,6]:=2,
            if tp(8) then d[8,7]:=3,
            if tp(9) then
            begin for i:=1 step 1 until p[9] do
                  begin j:=6+3*i; a:=b[i,1];
                        d[j,j]:=d[j+1,j+1]:=d[j+2,j+2]:=-a;
                        d[j+1,j]:=d[j,4]:=1, d[j+2,j+1]:=2
                  end
            end;
            if tp(10) then
            begin for i:=1 step 1 until p[10] do
                  begin w:=b[r[9]+i,2]; j:=q[9]+2*i-1,
                        d[j+1,4]:=1; d[j+1,j]:=-w; d[j,j+1]:=w
                  end
            end,
            if tp(11) then
            begin for i:=1 step 1 until p[11] do
                  begin a:=b[r[10]+i,1], w:=b[r[10]+i,2];
                        j:=q[10]+2*i-1;
                        d[j,j]:=d[j+1,j+1]:=-a,
                        d[j+1,4]:=1; d[j+1,j]:=-w; d[j,j+1]:=w
                  end
            end;
            mxprod(e,c,d);
            convert,
            normalise;
            testc
      end
end;

comment (s+k) Response chapter;

if ty(4) then
begin real k,a,w; integer h,i,j;
      for h:=1 step 1 until y[4] do
      begin k:=f4[h]; count:=count+1,
            setzero;
            if tp(1) then
            begin write(£u4(t) into filter number?),
                  outerror
            end,
            for i:=2 step 1 until 6 do
            begin if tp(i) then
                  begin d[i,i-1]:=1, d[i,i]:=k
                  end
            end,
            if tp(7) then
            begin d[7,6]:=2, d[7,7]:=k
            end,
            if tp(8) then
            begin d[8,7]:=3, d[8,8]:=k
            end;
            if tp(9) then
            begin for i:=1 step 1 until p[9] do
                  begin a:=b[i,1]; j:=6+3*i;
                        d[j,j]:=d[j+1,j+1]:=d[j+2,j+2]:=k-a;
                        d[j,4]:=1; d[j+1,j]:=1; d[j+2,j+1]:=2
                  end
            end;
            if tp(10) then
            begin for i:=1 step 1 until p[10] do
                  begin w:=b[r[9]+i,2]; j:=q[9]+2*i-1;
                        d[j,j]:=d[j+1,j+1]:=k;
                        d[j+1,4]:=1, d[j,j+1]:=w; d[j+1,j]:=-w
                  end
            end;
            if tp(11) then
            begin for i:=1 step 1 until p[11] do
                  begin a:=b[r[10]+i,1], w:=b[r[10]+i,2];
                        j:=q[10]+2*i-1;
                        d[j,j]:=d[j+1,j+1]:=k-a,
                        d[j+1,4]:=1; d[j,j+1]:=w, d[j+1,j]:=-w
                  end
            end,
            mxprod(e,c,d);
            convert;
            normalise;
            testc
      end
end;
```

```
comment (s↑2+2ns+m↑2) Response chapter;

if ty(5) then
begin real x3,n,m,a,w, integer h,i,j;
      for h:=1 step 1 until y[5] do
      begin n:=f51[h]; m:=f52[h];
            setzero;
            x3:=m↑2-n↑2, count:=count+1;
            if tp(1) then
            begin write(£u4(t) into filter number?);
                  outerror
            end,
            if tp(2) then
            begin write(£u3(t) into filter number?);
                  outerror
            end,
            for i:=3 step 1 until 6 do
            begin if tp(i) then
                  begin d[i,i-2]:=1; d[i,i-1]:=2*n, d[i,i]:=m↑2
                  end
            end;
            if tp(7) then
            begin d[7,5]:=2; d[7,6]:=4*n, d[7,7]:=m↑2
            end;
            if tp(8) then
            begin d[8,6]:=6; d[8,7]:=6*n; d[8,8]:=m↑2
            end;
            if tp(9) then
            begin for i:=1 step 1 until p[9] do
                  begin a:=b[i,1], j:=6+3*i;
                        d[j,3]:=d[j+1,4]:=1, d[j,4]:=2*n-a,
                        d[j,j]:=d[j+1,j+1]:=d[j+2,j+2]:=(a-n)↑2+x3;
                        d[j+1,j]:=2*(n-a); d[j+2,j]:=2;
                        d[j+2,j+1]:=4*(n-a)
                  end
            end;
            if tp(10) then
            begin for i:=1 step 1 until p[10] do
                  begin w:=b[r[9]+i,2]; j:=q[9]+2*i-1;
                        d[j,j]:=d[j+1,j+1]:=m↑2-w↑2,
                        d[j,j+1]:=2*w*n, d[j+1,j]:=-2*w*n,
                        d[j,4]:=w, d[j+1,3]:=1; d[j+1,4]:=2*n
                  end
            end,
            if tp(11) then
            begin for i:=1 step 1 until p[11] do
                  begin a:=b[r[10]+i,1], w:=b[r[10]+i,2],
                        j:=q[10]+2*i-1,
                        d[j,j]:=d[j+1,j+1]:=(a-n)↑2+x3-w↑2,
                        d[j+1,j]:=2*w*(a-n); d[j,j+1]:=-d[j+1,j],
                        d[j,4]:=w; d[j+1,3]:=1, d[j+1,4]:=2*n-a
                  end
            end;
            mxprod(e,c,d),
            convert;
            normalise;
            testc
      end
end,

comment 1/s Response chapter;

if ty(6) then
begin real a,w,d1, integer h,i,j,
      for h:=1 step 1 until y[6] do
      begin setzero; count:=count+1,
            for i:=1 step 1 until 5 do
            begin if tp(i) then d[i,i+1]:=1
            end;
            if tp(6) then d[6,7]:=1/2;
            if tp(7) then d[7,8]:=1/3,
            if tp(8) then d[8,9]:=1/4,
            if extra≠0 then
            begin for i:=1 step 1 until extra do
                  begin j:=8+i, d[j,j+1]:-1/(4+i)
                  end
            end;
            if tp(9) then
            begin for i:=1 step 1 until p[9] do
                  begin a:=b[i,1]; j:=6+3*i+extra;
                        d[j,j+1]:=d[j+1,j+2]:=d[j+2,j+3]:=-1/a,
                        d[j,5]:=1/a; d[j+1,j+1]:=-1/a↑2;
                        d[j+1,5]:=1 a↑2, d[j+2,j+2]:=-2/a↑2,
                        d[j+2,j+1]:=-2 a↑3, d[j+2,5]:=2/a↑3
                  end
            end;
            if tp(10) then
            begin for i:=1 step 1 until p[10] do
                  begin w:=b[r[9]+i,2]; j:=q[9]+2*i-1,
                        d[j,j+2]:--1/w, d[j+1,j+1]:=d[j,5]:=1/w
                  end
            end;
            if tp(11) then
            begin for i:=1 step 1 until p[11] do
                  begin a:=b[r[10]+i,1], w:=b[r[10]+i,2],
                        j:=q[10]+2*i-1, d1:=a↑2+w↑2,
                        d[j,j+1]:=d[j+1,j+2]:=-a/d1, d[j+1,5]:=a/d1,
                        d[j,5]:=d[j+1,j+1]:=w/d1; d[j,j+2]:=-w/d1
                  end
            end,
            mxprod(e,c,d);
            convert,
            normalise;
            testc,
            if p[8]=0 and extra=0 and (p[9]+p[10]+p[11])≠0 then
            begin for i:=1 step 1 until 3*p[9]+2*(p[10]+p[11]) do
                  begin j:=8+i,
                        c[1,j]:=c[1,j+1], c[1,j+1]:=0
                  end
            end,
            if (extra=0 and p[8]≠0) or extra≠0 then
            begin extra:-extra+1;
                  for i:=9 step 1 until 11 do q[i]:=q[i]+1
            end
      end
end,
for i:=1 step 1 until q[12] do c[1,i]:=cg*c[1,i];

comment Output function standard print out,

print ££12s4?Output Function£1??;
for i:=1 step 1 until 5 do
begin if tp(i) then
      print c[1,i],sameline,££s2?u?,leadzero(£?),5-i,£(t)?,
      if i=4 then print ££1??
end,
for i:=6 stop 1 until 8 do
begin if tp(i) then
      print c[1,i],sameline,££s2?t↑?,leadzero(£?),i-5
end,
if extra≠0 then
begin for i:=9 step 1 until extra+8 do
      begin if abs(c[1,i])>₁₀-6 then
            print c[1,i],sameline,££s2?t↑?,leadzero(£?),i-5
      end
end,
if tp(9) then
begin for i:=1 step 1 until p[9] do
      begin j:=6+3*i+extra;
            print ££12?exp(-at)*(u(t),t,t↑2)£s2?a=?,sameline,b[i,1];
            print c[1,j],c[1,j+1],c[1,j+2]
      end
end,
if tp(10) then
begin for i:=1 step 1 until p[10] do
      begin j:=q[9]+2*i-1,
            print ££12?sinwt,coswt£s2?w=?,sameline,b[r[9]+i,2],
            print c[1,j],c[1,j+1]
      end
end;
if tp(11) then
begin for i:=1 step 1 until p[11] do
      begin j:=q[10]+2*i-1,
            print ££12?exp(-at)*(sinwt,coswt)£s2?a=?,
                  sameline,b[r[10]+i,1],££s2?w=?,b[r[10]+i,2];
            print c[1,j],c[1,j+1]
      end
end;

comment Output to Programme II (Mean Square Programme),

read SAF;
if SAF≠0 then
begin if key(1) then print punch(3),££1?SAF DATA?;
      iA:=0,
      print punch(2),££1q??,outstring(A,iA),££u1??,
      print punch(2),p[9],p[11];
      if tp(9) then
      begin for i:=1 step 1 until p[9] do
            begin j:=6+3*i+extra,
                  print punch(2),c[1,j],c[1,j+1],c[1,j+2]
            end
      end,
      if tp(11) then
      begin for i:=1 step 1 until p[11] do
            begin j:=q[10]+2*i-1;
                  print punch(2),c[1,j],c[1,j+1]
            end
      end;
      if tp(9) then
      begin for i:=1 step 1 until p[9] do print punch(2),b[i,1]
      end;
      if tp(11) then
      begin for i:=1 step 1 until p[11] do
            print punch(2),b[r[10]+i,1],b[r[10]+i,2]
      end;
      read cg,i,j;
      print punch(2),cg,i,j;
      h:=4*(i+j),
      if h≠0 then
      begin for i:=1 step 1 until h do
            begin read cg;
                  print punch(2),cg
            end
      end;
      print punch(2),££1r10??
end;

comment Tabulation of output function;

read del;
if del≠0 then
begin real to,t,tf,h1,fn,a,w; integer i,j,h,m2,R4,B2,
      read to; R4:=0;
      for h:=1 step 1 until del do
      begin switch ssss :=L6,L7;
            read h1,tf,
            if h=1 then print ££13?Tabulated Output Function?;
            print ££12s3?time£s10?function£s6?interval=?,sameline,h1,££1??;
            if R4=0 then
            begin R4:=1; t:=to; fn:=0;
                  goto L6
            end;
      L7:   B2:=entier(abs((tf-to)/h1)+5₁₀-5);
            for m2:=1 step 1 until B2 do
            begin t:=to+h1*m2;
                  fn:=0;
            L6:        if tp(5) then fn:=c[1,5];
                  if tp(6) then fn:=fn+c[1,6]*t;
                  if tp(7) then fn:=fn+c[1,7]*t↑2;
                  if tp(8) then fn:=fn+c[1,8]*t↑3;
                  if extra≠0 then
                  begin for i:=1 step 1 until extra do
                        fn:=fn+c[1,8+i]*t↑(3+i)
                  end;
                  if tp(9) then
                  begin for i:=1 step 1 until p[9] do
                        begin a:=b[i,1], j:=6+3*i+extra,
                        fn:=fn+exp(-a*t)*(c[1,j]+c[1,j+1]*t+c[1,j+2]*t↑2)
                        end
            end;
                  if tp(10) then
                  begin for i:=1 step 1 until p[10] do
                        begin w:=b[r[9]+i,2]; j:= [9]+2*i-1;
                              fn:=fn+c[1,j]*sin(w*t)+c[1,j+1]*cos(w*t)
                        end
                  end,
                  if tp(11) then
                  begin for i:=1 step 1 until p[11] do
                        begin a:=b[r[10]+i,1], w:=b[r[10]+i,2];
                              j:= [10]+2*i-1,
                  fn:=fn+exp(-a*t)*(c[1,j]*sin(w*t)+c[1,j+1]*cos(w*t))
                        end
                  end,
                  print t,sameline,££s6??,fn;
                  if R4=1 then
                  begin R4:=2; goto L7
                  end
            end;
            to:=tf
      end
end,

print ££12r10??;
if key(1) then print punch(3),££12?NXDATA?;
restart
            end
      end
end,
```

App.A(c) Programme 1, Tape 2(cont'd)

Appendix B

ERROR INDICATIONS IN PROGRAMME I

(a) Transfer function numerator errors (see section 4.2.1)

Error indication	Cause of error indication	Method of correction
u4(t) into filter number x	A $u_4(t)$ function has been passed through: (a) s, generating $u_5(t)$ function, (b) s + k, generating $u_5(t)$ function, (c) $s^2 + 2ns + m^2$, generating $u_6(t)$ function. These are unacceptable to the next filter of the chain.	Break transfer function up into two stages, and run as shown in the example in section 4.2.1.
u3(t) into filter number x	A $u_3(t)$ function has been passed through an $s^2 + 2ns + m^2$ filter, thus generating a $u_5(t)$ function. This function is unacceptable to the next filter of the chain.	

(b) Transfer function denominator errors (see section 4.2.2)

Error indication	Cause of error indication	Method of correction
Failure case I filter number x	A t^2e^{-kt} function has been passed through: (a) $1/(s + k)$, generating t^3e^{-kt}, (b) $1/(s + k)^2$, generating t^4e^{-kt}, or, (c) $1/(s + k)^3$, generating t^5e^{-kt}.	Re-run without filter, and calculate the response of that filter by hand.
Failure case II filter number x	A te^{-kt} function has been passed through (a) $1/(s + k)^2$, generating t^3e^{-kt}, or, (b) $1/(s + k)^3$, generating t^4e^{-kt}.	x refers to the first filter of the type (see section 4.2.2.)
Failure case III filter number x	An e^{-kt} function has been passed through a $1/(s + k)^3$ filter, generating t^3e^{-kt}.	
Failure case IV filter number x	In a $1/(s^2 + 2ns + m^2)$ filter, then either: (a) $n^2 = m^2$, (b) $n^2 > m^2$, or, (c) $m = 0$.	Re-factorise the $(s^2 + 2ns + m^2)$ factor and repeat the calculation.
Failure case V filter number x	(a) An $e^{-at}(\sin \omega t, \cos \omega t)$ has been passed through a $1/(s^2 + 2ns + m^2)$ filter where $a = n$, and $\omega^2 = m^2 - n^2$, or (b) A $1/(s^2 + 2ns + m^2)^r$ has been used in the transfer function.	Re-run without filter, and calculate the response of that filter by hand. In (b), x refers to the second filter of the type.

Appendix C

PROGRAMME II - MEAN SQUARE PROGRAMME

(a) List of variables

Section deriving system autocorrelation function (S.A.F.)

A[1:m, 1:m]	real	matrix A of Ref.3, part I
a[1:1, 1:m]	real	coefficients of input vector
B[1:m, 1:m]	real	matrix B of Ref.3, part I
b[1:1, 1:m]	real	used to store matrix product aA
c[1·1, 1:m]	real	used to store matrix product aAB (i.e. coefficients of S.A.F.)
f1[1:p1]	real	holds a's of exp-poly functions in I.V. and S.A.F.
f2[1·1, 1:p2]	real	holds a's of exp-trig functions in I.V. and S.A.F.
f2[2:2, 1.p2]	real	holds w's of exp-trig functions in I.V. and S.A.F.
k,K,x,n,n1,w,w1,d	real	store coefficients when setting up A and B matrices (also used in setting up E matrix)
m	integer	number of coefficients in input vector (I.V.)
p1	integer	number of exp-poly functions in I.V.
p2	integer	number of exp-trig functions in I.V.
Z[0:10]	integer	stores data title
zz	integer	constant used in storing data title

Mean square evaluation section.

bb[1:1, 1:mm]	real	used to store matrix product cE
cc[1:mm, 1:1]	real	holds coefficients of input autocorrelation function (I.A.F.)
coe	real	coefficient of $u_1(\tau)$ in I.A.F.
E[1:mm, 1:mm]	real	matrix F of Ref.3, part I
ff1[1:pp1]	real	holds a's of exp-poly functions in I.A.F.
ff2[1:1, 1:pp2]	real	holds a's of exp-trig functions in I.A.F.
ff2[2:2, 1:pp2]	real	holds w's of exp-trig functions in I.A.F.
mm	integer	number of coefficients in I.A.F.
pp	integer	value 1 if $u_1(\tau)$ present in I.A.F., otherwise zero
pp1	integer	number of exp-poly functions in I.A.F.

pp2	integer	number of exp-trig functions in I.A.F.
s[1:1, 1:1]	real	used to store matrix product cEcc (i.e. mean square value)
v	real	used to store coefficients while setting up E matrix

```
Response of Linear Systems  Programme II   E.Huntley  L.J.Hazlewood;

begin integer p1,p2,pp,pp1,pp2,mm,m,zz,h,i,j,r; integer array Z[0:10];
      boolean procedure key(n);
      value n; integer n;
      begin elliot(7,0,0,0,2,3,n);
            key:=n≠0
      end;
      procedure mxprod(A,B,C),
      array A,B,C;
      begin integer aa,ab,ac,ra2,rb2,j,jstop,l,lstop,m,mstart,sa; real sum;
            aa:=address(A); sa:=size(A)+aa-1;
            ab:=address(B), ac:=address(C);
            ra2:=range(A,2); rb2:=range(B,2`;
            for aa:=aa step ra2 until sa do
            begin jstop:=aa+ra2-1; mstart:=ac-1;
                  for j:=aa step 1 until jstop do
                  begin m:=mstart:=mstart+1;
                        lstop:=ab+rb2-1; sum:=0;
                        for l:=ab step 1 until lstop do
                        begin sum:=sum+location(l)*location(m);
                              elliott(3,0,ra2,0,2,4,m);
                        end;
                        location(j):=sum
                  end;
                  ab:=ab+rb2
            end
      end mxprod,
      zz:=0;
      instring(Z,zz);
      print ££14?Response of Linear Systems  Programme II£12??;
      zz:=0;
      outstring(Z,zz);
      if key(1) then
      begin zz:=0;
            print punch(3),££1??,outstring(Z,zz)
      end;
      read p1,p2,
      m:=3*p1+2*p2;

      comment System Autocorrelation Function Section;
      begin real k,K,x,n,n1,w,w1,d,v,coe;
            array A,B[1:m,1:m],a,b,c[1:1,1:m],f1[1:p1],f2[1:2,1:p2];
            for i:=1 step 1 until m do read a[1,i];
            if p1≠0 then
            begin for i:=1 step 1 until p1 do read f1[i]
            end;
            if p2≠0 then
            begin for i:=1 step 1 until p2 do read f2[1,i],f2[2,i]
            end;

            comment formation of matrix B;
            for i:=1 step 1 until m do
            begin for j:=1 step 1 until m do B[i,j]:=0
            end;
            if p1≠0 then
            begin for i:=1 step 1 until p1 do
                  begin j:=3*i-2;
                        B[j,j]:=a[1,j];
                        B[j,j+1]:=B[j+1,j]:=a[1,j+1],
                        B[j,j+2]:=B[j+2,j]:=a[1,j+2];
                        B[j+1,j+1]:=2*a[1,j+2]
                  end
            end;
            if p2≠0 then
            begin for i:=1 step 1 until p2 do
                  begin j:=3*p1+2*i-1;
                        B[j,j+1]:=B[j+1,j]:=a[1,j];
                        B[j+1,j+1]:=a[1,j+1];
                        B[j,j]:=-a[1,j+1]
                  end
            end;

            comment formation of matrix A;
            if p1≠0 then
            begin for h:=1 step 1 until p1 do
                  begin k:=f1[h]; i:=3*h-2;
                        for r:=1 step 1 until h do
                        begin K:=f1[r]; d:=K+k; j:=3*r-2;
                              A[i,j]:=1/d;
                              A[i+1,j]:=1/d↑2;
                              A[i+2,j]:=A[i+1,j+1]:=2/d↑3;
                              A[i+2,j+1]:=6/d↑4;
                              A[i+2,j+2]:=24/d↑5;
                              if h≠r then
                              begin A[i,j+1]:=1/d↑2;
                                    A[i,j+2]:=2/d↑3;
                                    A[i+1,j+2]:=6/d↑4
                              end
                        end
                  end
            end;
            if p2≠0 then
            begin for h:=1 step 1 until p2 do
                  begin n:=f2[1,h]; w:=f2[2,h]; i:=3*p1+2*h-1;
                        if p1≠0 then
                        begin for r:=1 step 1 until p1 do
                              begin k:=f1[r]; x:=k+n; d:=x↑2+w↑2; j:=3*r-2;
                                    A[i,j]:=w/d;
                                    A[i,j+1]:=2*w*x/d↑2;
                                    A[i,j+2]:=2*w*(3*x↑2-w↑2)/d↑3;
                                    A[i+1,j]:=x/d;
                                    A[i+1,j+1]:=(x↑2-w↑2)/d↑2;
                                    A[i+1,j+2]:=2*x*(x↑2-3*w↑2)/d↑3
                              end
                        end;
                        for r:=1 step 1 until h do
                        begin j:=3*p1+2*r-1;
                              if r=h then
                              begin x:=w↑2+n↑2; d:=4*x;
                                    A[i,j]:=w↑2/d/n;
                                    A[i+1,j]:=w/d;
                                    A[i+1,j+1]:=(x+n↑2)/d/n
                              end
                              else
                              begin n1:=f2[1,r]; w1:=f2[2,r];
                                    x:=(n+n1)↑2+(w+w1)↑2; d:=(n+n1)↑2+(w-w1)↑2;
                                    A[i,j]:=((n+n1)/d-(n+n1)/x)*.5;
                                    A[i+1,j]:=((w+w1)/x+(w-w1)/d)*.5;
                                    A[i,j+1]:=((w+w1)/x-(w-w1)/d)*.5;
                                    A[i+1,j+1]:=((n+n1)/d+(n+n1)/x)*.5
                              end
                        end
                  end
            end;
            for i:=1 step 1 until m-1 do
            begin for j:=1 step 1 until m-i do A[i,i+j]:=A[i+j,i]
            end;
            mxprod(b,a,A);
            mxprod(c,b,B);

            comment System Autocorrelation Function Standard Print Out;
            print ££1s4?System Autocorrelation Function£1??,
            if p1≠0 then
            begin for i:=1 step 1 until p1 do
                  begin j:=3*i-2;
                        for h:=1 step 1 until 3 do
                        begin if abs(c[1,j+h-1])>⏨-8 then
                              print ££1??,c[1,j+h-1],sameline,££s2?exp(-k|T|)*|T|↑?,
                                    leadzero(£?),h-1,££s4?k=?,freepoint(8),f1[i]
                        end
                  end
            end;
            if p2≠0 then
            begin for i:=1 step 1 until p2 do
                  begin j:=3*p1+2*i-1;
                        if abs(c[1,j])>⏨-8 then
                        print ££1??,c[1,j],sameline,££s2?exp(-n|T|)sin(w|T|)£s4?n=?,
                              f2[1,i],££s4?w=?,f2[2,i];
                        if abs(c[1,j+1])>⏨-8 then
                        print ££1??,c[1,j+1],sameline,££s2?exp(-n|T|)cos(w|T|)£s4?n=?,
                              f2[1,i],££s4?w=?,f2[2,i]
                  end
            end;
            print ££12??;
            comment End of System Autocorrelation Function Section;

            read coe,pp1,pp2;
            if abs(coe)>⏨-6 then pp:=1 else pp:=0;
            mm:=pp+3*pp1+2*pp2;

            comment Mean Square Section;
            begin array E[1:m,1:mm],bb[1:1,1:mm],cc[1:mm,1:1],
                        s[1:1,1:1],ff1[1:pp1],ff2[1:2,1:pp2];
                  if pp≠0 then cc[1,1]:=coe,
                  if mm≠pp then
                  begin for i:=pp+1 step 1 until mm do read cc[i,1];
                  end;
                  if pp1≠0 then
                  begin for i:=1 step 1 until pp1 do read ff1[i]
                  end;
                  if pp2≠0 then
                  begin for i:=1 step 1 until pp2 do read ff2[1,i],ff2[2,i]
                  end;

                  comment formation of matrix E;
                  if p1≠0 then
                  begin for h:=1 step 1 until p1 do
                        begin k:=f1[h]; i:=3*h-2;
                              if pp≠0 then
                              begin E[i,1]:=1;
                                    E[i+1,1]:=E[i+2,1]:=0
                              end;
                              if pp1≠0 then
                              begin for r:=1 step 1 until pp1 do
                                    begin K:=ff1[r]; j:=pp+3*r-2; d:=K+k;
                                          E[i,j]:=2/d;
                                          E[i,j+1]:=E[i+1,j]:=2/d↑2;
                                          E[i,j+2]:=E[i+1,j+1]:=E[i+2,j]:=4/d↑3;
                                          E[i+1,j+2]:=E[i+2,j+1]:=12/d↑4,
                                          E[i+2,j+2]:=48/d↑5
                                    end
                              end;
                              if pp2≠0 then
                              begin for r:=1 step 1 until pp2 do
                                    begin K:=ff2[1,r]; v:=ff2[2,r];
                                          j:=pp+3*pp1+2*r-1, d:=(k+K)↑2+v↑2,
                                          E[i,j]:=2*v/d;
                                          E[i,j+1]:=2*(k+K)/d;
                                          E[i+1,j]:=4*v*(k+K)/d↑2;
                                          E[i+1,j+1]:=2*((k+K)↑2-v↑2)/d↑2,
                                          E[i+2,j]:=4*v*(3*(k+K)↑2-v↑2)/d↑3,
                                          E[i+2,j+1]:=4*(k+K)*((k+K)↑2-3*v↑2)/d↑3
                                    end
                              end
                        end
                  end;
                  if p2≠0 then
                  begin for h:=1 step 1 until p2 do
                        begin n:=f2[1,h]; w:=f2[2,h], i:=3*p1+2*h-1,
                              if pp≠0 then
                              begin E[i,1]:=0;
                                    E[i+1,1]:=1
                              end;
                              if pp1≠0 then
                              begin for r:=1 step 1 until pp1 do
                                    begin K:=ff1[r]; j:=pp+3*r-2; d:=(K+n)↑2+w↑2,
                                          E[i,j]:=2*w/d;
                                          E[i+1,j]:=2*(n+K)/d;
                                          E[i,j+1]:=4*w*(n+K)/d↑2;
                                          E[i+1,j+1]:=2*((n+K)↑2-w↑2)/d↑2;
                                          E[i,j+2]:=4*w*(3*(n+K)↑2-w↑2)/d↑3;
                                          E[i+1,j+2]:=4*(n+K)*((n+K)↑2-3*w↑2)/d↑3
                                    end
                              end;
                              if pp2≠0 then
                              begin for r:=1 step 1 until pp2 do
                                    begin K:=ff2[1,r], v:=ff2[2,r]; j:=pp+3*pp1+2*r-1;
                                          d:=(n+K)↑2+(w+v)↑2; x:=(n+K)↑2+(w-v)↑2;
                                          E[i,j]:=(n+K)/x-(n+K)/d,
                                          E[i+1,j+1]:=(n+K)/x+(n+K)/d;
                                          E[i+1,j]:=(w+v)/d-(w-v)/x,
                                          E[i,j+1]:=(w+v)/d+(w-v)/x
                                    end
                              end
                        end
                  end;
                  mxprod(bb,c,E);
                  mxprod(s,bb,cc);
                  print sameline,£Mean Square=?,s[1,1],££12r10??;
                  comment End of Mean Square Section;

                  if key(1) then print punch(3),££1?NXDATA?;
                  restart
            end
      end
end;
```

Appendix D

PROGRAMME III - AUTOCORRELATION FUNCTION PROGRAMME

(a) List of variables and procedures

a,v,k,m,n	real	variables in I.F. and T.F.
b1[1:r3]	real	holds a's of exp-poly and exp-trig input functions
b2[1:r3]	real	holds v's of exp-trig input functions
cg	real	constant gain of T.F., later used for normalising factor
cnt	integer	number of factors dealt with
coe	real	holds coefficient of $u_1(\tau)$ on input
D[1:q3, 1:q3]	real	response matrix
del	integer	number of large tabulation intervals
e[1:1, 1:q3]	real	output coefficient vector
exq	integer	number of locations required in the coefficient vectors to accommodate the coefficients of the generated functions
exr	integer	number of locations required in the b1 and b2 arrays to accommodate the a's and v's of the generated functions
g[1:1, 1:q3]	real	input coefficient vector
ny[0]	integer	not significant, always zero
ny[1] ny[2] etc.	integer	number of factors of each $1/(s + k)^n$ type (i.e. value of the index n), excluding n = 1
p	integer	value 1 if $u_3(\tau)$ present in input, otherwise zero
pp	integer	value 1 if $u_1(\tau)$ present in input, otherwise zero
p1	integer	number of exp-poly functions in input
p2	integer	number of exp-trig functions in input
q1	integer	position of last coefficient of exp-poly function in input vector
q2	integer	position of last coefficient of exp-trig functions in input vector
q3	integer	length of final coefficient vector

rr1, rr2, RR1, RR2	integer	used in $1/(s + k)$ chapter when input function is passed through repeated factors
r1	integer	position of the last a, v values in exp-poly functions in the b1 and b2 arrays
r2	integer	position of the last a, v values in exp-trig functions in the b1 and b2 arrays
r3	integer	length of final b1 and b2 arrays
t1[1:y[1]]	real	holds k's of $1/(s + k)$ factors
t21[1:y[2]]	real	holds n's of $1/(s^2 + 2ns + m^2)$ factors
t22[1:y[2]]	real	holds m's of $1/(s^2 + 2ns + m^2)$ factors
t4[1:y[4]]	real	holds k's of $(s + k)$ factors
t51[1:y[5]]	real	holds n's of $(s^2 + 2ns + m^2)$ factors
t52[1:y[5]]	real	holds m's of $(s^2 + 2ns + m^2)$ factors
y[1]	integer	number of $1/(s + k)$ factors
y[2]	integer	number of $1/(s^2 + 2ns + m^2)$ factors
y[3]	integer	number of s factors
y[4]	integer	number of $(s + k)$ factors
y[5]	integer	number of $(s^2 + 2ns + m^2)$ factors
yy1	integer	total number of factors in $1/(s + k)^n$ terms
yy2	integer	total number of terms of the type $1/(s + k)^n$
zz	integer	constant used in storing data title
Z[0:10]	integer	stores data title

Variables in $1/(s + k)$ chapter

ak	integer	position of exp-poly input function with parameter k for a single $1/(s + k)$ factor (otherwise zero)
akn	integer	position of exp-poly input function with parameter k for a repeated $1/(s + k)$ factor (otherwise zero)
d	real	auxiliary variable
m1	integer	used in computation of $1/(s + k)^n$ terms, determines the current factor
x	integer	position of first coefficient of generated exp-poly function

Variables in $1/(s^2 + 2ns + m^2)$ chapter

c	real	$(a^2 + m^2)^2 - 4a^2 n^2$
d	real	$a^2 + m^2 - 2n^2$
e1,e2,e3,e4 f1,f2,f3,f4 g1,g2,g3,g4	real	auxiliary variables
x	integer	position of first coefficient of generated exp-trig function
x1	real	$(m^2 - n^2)^{\frac{1}{2}}$
x2	real	$m^2 - n^2$

Variables in $(s^2 + 2ns + m^2)$ chapter

c,d	real	auxiliary variables
x2	real	$m^2 - n^2$

Variables in tabulation block

B1	integer	used in tabulation of first value of function
B2	integer	number of time increments in one large time interval
fn	real	value of function at time t
h1	real	time increment
m2	integer	count variable
to	real	initial time value in one large time interval
tf	real	final time value in one large time interval
t	real	current value of time

List of procedures

key(n)	takes logical value TRUE or FALSE if the key on the computer console of value n is switched on
readr (n,B)	reads in values of the real array B[i] for i taking values from 1 to n
outerror (string)	prints the string on the output device, and on the teleprinter if key(1) is on. String is a set of characters between the £? string quotes. Reads in tabulation data (if any). If key(1) on prints NXDATA on the teleprinter and then restarts the programme. Uses real variable co to read in the real numbers
set(n)	Used in the section of the programme which determines the $1/(s + k)^n$ terms by testing values of k

mxprod (A,B,C)	performs matrix operation A:= B × C
setzero	sets all elements of R.M. to zero
testg	tests coefficients of $u_3(\tau)$ and $u_1(\tau)$ and sets the corresponding p and pp to zero or 1 as required
convert	converts elements of the e array to corresponding elements in the g array
normalise	normalises the input coefficient vector; ave used to accumulate coefficients, and find the mean; normalising factor accumulated in cg

```
Response of Linear Systems  Programme III   E.Huntley  L.J.Hazlewood;

begin integer h,i,j,zz,p1,p2,q1,q2,q3,r1,r2,r3,exq,exr,yy1,yy2,rr1,rr2,RR1,RR2,p,pp,cnt,del;
        integer array y[1:5],ny[0:20],Z[0:10]; real cg,a,v,k,m,n,coe;
        boolean procedure key(n);
        value n; integer n,
        begin elliot(7,0,0,0,2,3,n);
              key:=n≠0
        end;
        procedure readr(n,B);
        value n; integer n; array B,
        begin integer i,
              for i:=1 step 1 until n do read B[i]
        end;
        procedure outerror(string);
        string string;
        begin integer i; real co;
              print ££l??,string,££s?filter number£s??,sameline,leadzero(£?),cnt;
              if key(1) then print punch(3),££l??,string,££s?filter number£s??,
              sameline,leadzero(£?),cnt;
              read del;
              if del≠0 then
              begin for i:=1 step 1 until 2*del+1 do read co
              end,
              if key(1) then print punch(3),££l2?NXDATA?,
              print ££l2r10??;
              restart
        end;
        procedure set(n);
        value n; integer n;
        begin j:=j+1, ny[j]:=n
        end;
        procedure mxprod(A,B,C);
        array A,B,C;
        begin integer aa,ab,ac,ra2,rb2,j,jstop,l,lstop,m,mstart,sa; real sum;
              aa:=address(A); sa:=size(A)+aa-1;
              ab:=address(B); ac:=address(C);
              ra2:=range(A,2); rb2:=range(B,2);
              for aa:=aa step ra2 until sa do
              begin jstop:=aa+ra2-1; mstart:=ac-1;
                    for j:=aa step 1 until jstop do
                    begin m:=mstart:=mstart+1,
                          lstop:=ab+rb2-1; sum:=0;
                          for l:=ab step 1 until lstop do
                          begin sum:=sum+location(l)*location(m);
                                elliott(3,0,ra2,0,2,4,m);
                          end;
                          location(j):=sum
                    end;
                    ab:=ab+rb2
              end
        end mxprod;
        cnt:=zz:=0;
        instring(Z,zz),
        print ££l4?Response of Linear Systems Programme III£l2??;
        zz:=0;
        outstring(Z,zz);
        if key(1) then
        begin zz:=0;
              print punch(3 ,££l??,outstring(Z,zz)
        end;
        read cg,y[1],y[2],y[3],y[4],y[5], cg:=cg↑2;
        zz:=0,
        if key(2) then print punch(2),££r10l4??,outstring(Z,zz);

begin array t1[1:y[1]],t21,t22[1:y[2]],t4[1:y[4]],t51,t52[1:y[5]],
      if y[1]≠0 then readr(y[1],t1);
      if y[2]≠0 then
      begin readr(y[2],t21); readr(y[2],t22)
      end;
      if y[4]≠0 then readr(y[4],t4),
      if y[5]≠0 then
      begin readr(y[5],t51); readr(y[5],t52)
      end;
      ny[0]:=j:=yy2:=yy1:=0;
      if y[1]≠0 then
      begin if y[1]>1 then
            begin if y[1]=2 and abs(t1[1]-t1[2])<⏨-6 then set(2);
                  if y[1]>2 then
                  begin for i:=3 step 1 until y[1] do
                        begin if abs(t1[i]-t1[i-1])<⏨-6 and
                              abs(t1[i-1]-t1[i-2])<⏨-6 then set(3) else
                              if abs(t1[i-1]-t1[i-2])<⏨-6 and ny[j]≠3 then
                              set(2) else
                              if abs(t1[i]-t1[i-1])<⏨-6 and i=y[1] then set(2)
                        end
                  end
            end;
            yy2:=j;
            if yy2=0 then RR1:=y[1] else
            begin rr1:=1; RR1:=ny[1];
                  for i:=1 step 1 until yy2 do yy1:=yy1+ny[i]
            end
      end;
      exq:=3*(y[1]-yy1+yy2)+2*y[2],
      exr:=y[1]-yy1+yy2+y[2];
      read coe,p1,p2,
      q1:=2+3*p1; q2:=q1+2*p2;  3:=q2+exq;
      r1:=p1; r2:=r1+p2, r3:=r2+exr;
      begin array e,g[1:1,1:q3],D[1:q3,1:q3],b1,b2[1:r3],
            procedure setzero;
            begin integer i,j;
                  for i:=1 step 1 until q3 do
                  begin for j:=1 step 1 until q3 do D[i,j]:=0
                  end
            end setzero;
            procedure convert;
            begin integer i;
                  for i:=1 step 1 until q3 do g[1,i]:=e[1,i]
            end convert,
            procedure testg;
            begin if abs(g[1,1])<5⏨-6 then p:=0 else p:=1,
                  if abs(g[1,2])<5⏨-6 then pp:=0 else pp:=1
            end testg;
            procedure normalise;
            begin real ave; integer i;
                  ave:=0;
                  for i:=1 step 1 until q3 do ave:=ave+abs(g[1,i]),
                  ave:=ave/q3,
                  if key(2) then print punch(2),££l4??;
                  for i:=1 step 1 until q3 do
                  begin g[1,i]:=g[1,i]/ave;
                        if key(2) then print punch(2),g[1,i]
                  end;
                  cg:=cg*ave,
                  if key (2) then print punch (2),££l2??,cg
            end normalise,
            g[1,1]:=0, g[1,2]:=coe;
            if q2>2 then
            begin for i:=3 step 1 until q2 do read g[1,i]
            end;
            if exq≠0 then
            begin for i:=r2+1 step 1 until q3 do g[1,i]:=0
            end;
            if r2≠0 then
            begin for i:=1 step 1 until r2 do read b1[i],b2[i]
            end,
            if exr≠0 then
            begin for i:=r2+1 step 1 until r3 do b1[i]:=b2[i]:=0
            end,
            normalise;
            testg,

comment 1 (s+k) Response chapter;

if y[1]≠0 then
begin real d, integer h,m1,x,ak,akn; switch s:=L2,L3,
      rr2:=ak:=akn:=0, RR2:=y[1]-yy1+1,
      for h:=1 step 1 until y[1] do
begin k:=t1[h], cnt:=cnt+1;
            if h>RR2 then
            begin m1:=m1+1;
                  if m1=1 or m1=2 then
                  begin D[x,x]:=.5/k↑2, D[x,x+1]:=.5/k
                  end,
                  if m1=2 then
                  begin D[x+1,x]:=.25/k↑3;
                        D[x+1,x+1]:=.25/k↑2, D[x+1,x+2]:=.25/k
                  end
            end
            else
            begin m1:=0; setzero; x:=q2+1
            end;
            if p≠0 then
            begin D[1,2]:=-1, D[1,x]:=.5*k
            end,
            if pp≠0 then D[2,x]:=.5/k;
            if h>RR2 then goto L2;
            if p1≠0 then
            begin for i:=1 step 1 until p1 do
                  begin a:=b1[i]; j:=3*i;
                        if abs(k-a)<⏨-6 then
                        begin if abs(g[1,j+2])>⏨-6 then
                              outerror(£failure case I?);
                              if h>y[1]-yy1 then
                              begin if abs(g[1,j+1])>⏨-6 then
                              outerror(£failure case II?);
                              if ny[rr1]=3 then
                              outerror(£failure case III?),
                              akn:=j
                              end
                              else ak:=j;
                              D[j,j]:=.5/k↑2, D[j,j+1]:=.5/k;
                              D[j+1,j]:=.25/k↑3;
                              D[j+1,j+1]:=.25/k↑2; D[j+1,j+2]:=.25/k
                        end
                        else
                        begin d:=k↑2-a↑2,
                              D[j,j]:=D[j+1,j+1]:=D[j+2,j+2]:=1/d;
                              D[j,x]:=-D[j,j]*a/k;
                              D[j+1,j]:=-2*a/d↑2,
                              D[j+2,j+1]:=2*D[j+1,j];
                              D[j+1,x]:=(k↑2+a↑2)/(k*d↑2),
                              D[j+2,j]:=2*(k↑2+3*a↑2)/d↑3,
                              D[j+2,x]:=-2*a*(3*k↑2+a↑2)/(k*d↑3)
                        end
                  end
            end;
            if p2≠0 then
            begin for i:=1 step 1 until p2 do
                  begin a:=b1[r1+i], v:=b2[r1+i]; j:=q1+2*i-1;
                        d:=(k↑2+v↑2+a↑2)↑2-4*k↑2*a↑2,
                        D[j,j]:=D[j+1,j+1]:=(k↑2+v↑2-a↑2)/d,
                        D[j,j+1]:=-2*a*v/d, D[j+1,j]:=-D[j,j+1];
                        D[j,x]:=v*(k↑2+v↑2+a↑2)/(k*d);
                        D[j+1,x]:=-a*(k↑2-v↑2-a↑2)/(k*d)
                  end
            end;
L2:         mxprod(e,g,D);
            convert;
            normalise,
            testg;
            begin array z[1:3], switch ss:=L4;
                  if h≤y[1]-yy1 then
                  begin if ak≠0 then
                        begin g[1,ak]:=g[1,ak]+g[1,x],
                              g[1,x]:=0, ak:=0; goto L3
                        end
                        else
                        begin z[1]:=g[1,x], z[2]:=z[3]:=0,
                              g[1,x]:=0;  goto L4
                        end
                  end,
```

App.D(b) Programme 3

```
                if yy2≠0 and y[1]-yy1+RR1=h then
                begin if h≠y[1] then
                    begin rr1:=rr1+1, RR1:=RR1+ny[rr1],
                          rr2:=rr2+1; RR2:=RR2+ny[rr2]
                    end;
                    if akn≠0 then
                    begin g[1,akn]:=g[1,akn]+g[1,x];
                          g[1,akn+1]:=g[1,akn+1]+g[1,x+1],
                          g[1,x]:=g[1,x+1]:=0,
                          akn:=0; goto L3
                    end;
                    z[1]:=g[1,x], z[2]:=g[1,x+1], z[3]:=g[1,x+2],

                    g[1,x]:=g[1,x+1]:=g[1,x+2]:=0,
                    goto L4
                end;
                goto L3;
        L4:         if p2≠0 then
                begin for i:=1 step 1 until 2*p2 do
                    begin j:=x+3-i; g[1,j]:=g[1,j-3]
                    end;
                    for i:=1 step 1 until p2 do
                    begin j:=r2+2-i;
                          b1[j]:=b1[j-1]; b2[j]:=b2[j-1]
                    end
                end,
                for i:=1 step 1 until 3 do
                begin j:=3*p1+2+i, g[1,j]:=z[i]
                end;
                p1:=p1+1,   1:=q1+3, r1:=r1+1;
                q2:= 2+3, r2:=r2+1,
                b1[p1]:=k,  b2[p1]:=0
            end;
L3:     end
end;

comment 1/(s↑2+2ns+m↑2) Response chapter;

if y[2]≠0 then
begin real x2,x1,c,d,e1,e2,e3,e4,f1,f2,f3,f4,g1,g2,g3,g4,
      integer h,x;
      for h:=1 step 1 until y[2] do
      begin n:=t21[h]; m:=t22[h]; cnt:=cnt+1; x2:=m↑2-n↑2,
            if x2<0 or abs(x2)<⏨-6 or abs(m)<⏨-6 or abs(n)<⏨-6 then
            outerror(£failure case IV?);
            x1:=sqrt(x2);
            setzero,
            x:= 2+1,
            if p≠0 then
            begin D[1,x]:=.25/x1, D[1,x+1]:=-.25/n
            end,
            if pp≠0 then
            begin D[2,x]:=.25/(x1*m↑2); D[2,x+1]:=.25/(n*m↑2)
            end,
            if p1≠0 then
            begin for i:=1 step 1 until p1 do
                  begin a:=b1[i], j:=3*i; c:=(a↑2+m↑2)↑2-4*a↑2*n↑2;
                        D[j,j]:=D[j+1,j+1]:=D[j+2,j+2]:=1/c;
                        d:=a↑2+m↑2-2*n↑2;
                        D[j+1,j]:=4*a*d/c↑2, D[j+2,j+1]:=2*D[j+1,j],
       D[j+2,j]:=4*(d*(5*a↑4+4*a↑2*m↑2-8*a↑2*n↑2-m↑4)-8*a↑2*n↑2*x2)/c↑3,
                        D[j,x]:=a*(a↑2+3*m↑2-4*n↑2)/(2*x1*c*m↑2);
                        D[j,x+1]:=a*(d-2*n↑2)/(2*n*c*m↑2),
                D[j+1,x]:=(c*(d-2*n↑2)+4*m↑2*(a↑2-m↑2)*d)/(2*x1*c↑2*m↑2),
     D[j+1,x+1]:=((a↑4-m↑4)*(d-2*n↑2)-4*a↑2*n↑2*(a↑2+3*m↑2-4*n↑2))/(2*n*c↑2*m↑2);
D[j+2,x]:=a*(d*(d↑2-12*n↑2*x2)*(a↑2+3*m↑2 +x2*(6*d↑2-8*n↑2*x2)*(a↑2-3*m↑2))/(x1*c↑3*m↑2),
D[j+2,x+1]:=a*(d*(d↑2-12*n↑2*x2)*(a↑2-3*m↑2)-n↑2*(6*d↑2-8*n↑2*x2)*(a↑2+3*m↑2))/(n*c↑3*m↑2)
                  end
            end;
            if p2≠0 then
            begin for i:=1 step 1 until p2 do
                  begin a:=b1[r1+i]; v:=b2[r1+i], j:=q1+2*i-1,
                        if abs(n↑2-a↑2)<⏨-6 and abs(x2-v↑2)<⏨-6 then
                        outerror(£failure case V?);
                        g4:=8*n*x1*m↑2;
                     g1:=g4*((n+a)↑2+(x1+v)↑2), g2:=g4*((n-a)↑2+(x1+v)↑2),
                     g3:=g4*((n+a)↑2+(x1-v)↑2), g4:=g4*((n-a)↑2+(x1-v)↑2),
            f1:=(n*(x1+v)+x1*(n+a))/g1; f2:=(n*(x1+v)+x1*(n-a))/g2;
            f3:=(n*(x1-v)+x1*(n+a))/g3; f4:=(n*(x1-v)+x1*(n-a))/g4,
            e1:=(n*(n+a)-x1*(x1+v))/g1, e2:=(-n*(n-a)+x1*(x1+v))/g2,
            e3:=(-n*(n+a)+x1*(x1-v))/g3, e4:-(n*(n-a)-x1*(x1-v))/g4,
                        D[j,j]:=D[j+1,j+1]:=f1+f2+f3+f4;
                        D[j+1,j]:=e1+e2+e3+e4, D[j,j+1]:=-D[j+1,j];
                        D[j,x]:=f1+f2-f3-f4; D[j,x+1]:=-e1+e2-e3+e4;
                        D[j+1,x]:=e1+e2-e3-e4, D[j+1,x+1]:=f1-f2+f3-f4
                  end
            end;
            mxprod(e,g,D),
            convert;
            normalise;
            testg;
            p2:=p2+1,  2:=q2+2; r2:=r2+1;
            b1[r2]:=n, b2[r2]:=x1
      end
end;

comment s Response chapter,

if y[3]≠0 then
begin integer h;
      for h:=1 step 1 until y[3] do
      begin cnt:=cnt+1, setzero,
            if p≠0 then outerror(£u3(T) into?);
            if pp≠0 then D[2,1]:=-1;
            if p1≠0 then
            begin for i:=1 step 1 until p1 do
                  begin a:=b1[i], j:=3*i;
                        D[j,j]:=D[j+1,j+1]:=D[j+2,j+2]:=-a↑2,
                        D[j,2]:=D[j+1,j]:=2*a,
                        D[j+2,j+1]:=4*a, D[j+1,2]:=D[j+2,j]:=-2
                  end
            end,
            if p2≠0 then
            begin for i:=1 step 1 until p2 do
                  begin a:=b1[r1+i], v:=b2[r1+i]; j:=q1+2*i-1,
                        D[j,j]:=D[j+1,j+1]:=v↑2-a↑2,
                        D[j,j+1]:=2*a*v, D[j+1,j]:=-D[j,j+1],
                        D[j,2]:=-2*v, D[j+1,2]:=2*a
                  end
            end;
            mxprod(e,g,D);
            convert;
            normalise,
            testg
      end
end;

comment (s+k) Response chapter;

if y[4]≠0 then
begin integer h;
      for h:=1 step 1 until y[4] do
      begin cnt:=cnt+1, k:=t4[h], setzero;
            if p≠0 then outerror(£u3(T) into?),
            if pp≠0 then
            begin D[2,1]:=-1; D[2,2]:=k↑2
            end;
            if p1≠0 then
            begin for i:=1 step 1 until p1 do
                  begin a:=b1[i]; j:=3*i,
                        D[j,j]:=D[j+1,j+1]:=D[j+2,j+2]:=k↑2-a↑2;
                        D[j,2]:=D[j+1,j]:=2*a, D[j+2,j+1]:=4*a,
                        D[j+1,2]:=D[j+2,j]:=-2
                  end
            end,
            if p2≠0 then
            begin for i:=1 step 1 until p2 do
                  begin a:=b1[r1+i]; v:=b2[r1+i], j:= 1+2*i-1,
                        D[j,j]:=D[j+1,j+1]:=k↑2+v↑2-a↑2,
                        D[j,j+1]:=2*a*v, D[j+1,j]:=-D[j,j+1],
                        D[j,2]:=-2*v; D[j+1,2]:=2*a
                  end
            end;
            mxprod(e,g,D),
            convert;
            normalise,
            testg
      end
end;

comment (s↑2+2ns+m↑2) Response chapter;

if y[5]≠0 then
begin real c,d,x2, integer h,
      for h:=1 step 1 until y[5] do
      begin cnt:=cnt+1; n:=t51[h], m:=t52[h], x2:=m↑2-n↑2,
            setzero;
            if p≠0 then outerror(£u3(T) into?),
            if pp≠0 then outerror(£u1(T) into?);
            if p1≠0 then
            begin for i:=1 step 1 until p1 do
                  begin a:=b1[i]; j:=3*i; c:=(a↑2+m↑2)↑2-4*a↑2*n↑2,
                        D[j,j]:=D[j+1,j+1]:=D[j+2,j+2]:=c,
                        D[j+1,j]:=-4*a*(a↑2+m↑2-2*n↑2);
                        D[j+2,j+1]:=2*D[j+1,j]; D[j,1]:=-2*a;
                        D[j+2,j]:=4*(3*a↑2+m↑2-2*n↑2); D[j+1,1]:=2,
                        D[j,2]:=-2*a*(a↑2+2*m↑2-4*n↑2), D[j+2,2]:=-12*a,
                        D[j+1,2]:=2*(3*a↑2+2*m↑2-4*n↑2)
                  end
            end,
            if p2≠0 then
            begin for i:=1 step 1 until p2 do
                  begin a:=b1[r1+i]; v:=b2[r1+i]; j:=q1+2*i-1;
                        d:=(n↑2-x2)-(a↑2-v↑2), c:=4*(n↑2*x2-a↑2*v↑2);
                        D[j,j]:=D[j+1,j+1]:=d↑2+c,
                        D[j,j+1]:=4*a*v*d, D[j+1,j]:=-D[j,j+1];
                        D[j,1]:=2*v, D[j+1,1]:=-2*a,
                        D[j,2]:=2*v*(2*(m↑2-2*n↑2)-(v↑2-3*a↑2));
                        D[j+1,2]:=-2*a*(2*(m↑2-2*n↑2)-(3*v↑2-a↑2))
                  end
            end;
            mxprod(e,g,D);
            convert;
            normalise;
            testg
      end
end;
for i:=1 step 1 until q3 do g[1,i]:=g[1,i]*cg;
cg:=0;

comment Output Autocorrelation Function Standard Print Out,

print ££l s4?Output Autocorrelation Function?;
if p≠0 then print g[1,1],sameline,££s2?u3(T)?;
if pp≠0 then print g[1,2],sameline,££s2?u1(T)?;
if p1≠0 then
begin for i:=1 step 1 until p1 do
      begin a:=b1[i]; j:=3*i; cg:=cg+g[1,j];
            print ££l2?exp(-a|T|)*(1,|T|,|T|↑2)£s5?a=?,sameline,a,££l??;
            print g[1,j],g[1,j+1],g[1,j+2]
      end
end;
if p2≠0 then
begin for i:=1 step 1 until p2 do
      begin a:=b1[r1+i]; v:=b2[r1+i]; j:=q1+2*i-1, cg:=cg+g[1,j+1];
            print ££l2?exp(-a|T|)*(sin(v|T|),cos(v|T|))£s2?a=?,
            sameline,a,££s2?v=?,sameline,v,££l??;
            print g[1,j],g[1,j+1]
      end
end;
print ££l2?Mean S uare =?,sameline,cg,

comment Tabulation Section;

read del;
if del≠0 then
begin real to,t,tf,h1,fn; integer m2,B1,B2;
      read to; B1:=0;
      print ££l2?Tabulated Output Autocorrelation Function?;
      for h:=1 step 1 until del do
      begin switch sss:=L5,L6;
            read h1,tf;
            print ££l2s2?time shift£s6?function£s6?interval=?,
                  sameline,h1,££l??;
            if B1=0 then
            begin B1:=1; t:=to; fn:=0; goto L5
            end;
      L6:   B2:=entier(abs((tf-to)/h1)+5⏨-5);
            for m2:=1 step 1 until B2 do
            begin t:=to+h1*m2; fn:=0;
            L5:       if p1≠0 then
                  begin for i:=1 step 1 until p1 do
                        begin a:=b1[i], j:=3*i,
                  fn:=fn+exp(-a*t)*(g[1,j]+g[1,j+1]*t+g[1,j+2]*t↑2)
                        end
                  end;
                  if p2≠0 then
                  begin for i:=1 step 1 until p2 do
                        begin a:=b1[r1+i]; v:=b2[r1+i]; j:=q1+2*i-1;
                  fn:=fn+exp(-a*t)*(g[1,j]*sin(v*t)+g[1,j+1]*cos(v*t))
                        end
                  end,
                  print ££l s2??,sameline,t,££s5??,fn;
                  if B1=1 then
                  begin B1:=2; goto L6
                  end
            end;
            to:=tf
      end
end;
print ££l2r10??;
if key(1) then print punch(3),££l2?NXDATA?,
restart
      end
   end
end;
```

Appendix E

ERROR INDICATIONS IN PROGRAMME III

(a) Transfer function numerator errors (see section 8.2)

Error indication	Cause of error indication	Method of correction
$u3(\tau)$ into filter number x	A $u_3(\tau)$ function has been passed through: (a) s, generating $u_5(\tau)$ function, (b) s + k, generating $u_5(\tau)$ function, (c) $s^2 + 2ns + m^2$, generating $u_7(\tau)$ function. These are unacceptable to the next filter of the chain.	Break transfer function up into two stages, and run as shown in the example for Programme I in section 4.2.1.
$u1(\tau)$ into filter number x	A $u_1(\tau)$ function has been passed through an $s^2 + 2ns + m^2$ filter, thus generating a $u_5(\tau)$ function. This function is unacceptable to the next filter of the chain.	

(b) Transfer function denominator errors (see section 8.2)

Error indication	Cause of error indication	Method of correction
Failure case I filter number x	A $\lvert\tau\rvert^2 e^{-k\lvert\tau\rvert}$ function has been passed through: (a) $1/(s + k)$, generating $\lvert\tau\rvert^3 e^{-k\lvert\tau\rvert}$, (b) $1/(s + k)^2$, generating $\lvert\tau\rvert^4 e^{-k\lvert\tau\rvert}$, or, (c) $1/(s + k)^3$, generating $\lvert\tau\rvert^5 e^{-k\lvert\tau\rvert}$.	Re-run without filter, and calculate the response of that filter by hand. x refers to the first filter of the type
Failure case II filter number x	A $\lvert\tau\rvert e^{-k\lvert\tau\rvert}$ function has been passed through: (a) $1/(s + k)^2$, generating $\lvert\tau\rvert^3 e^{-k\lvert\tau\rvert}$, or (b) $1/(s + k)^3$, generating $\lvert\tau\rvert^4 e^{-k\lvert\tau\rvert}$.	
Failure case III filter number x	An $e^{-k\lvert\tau\rvert}$ function has been passed through a $1/(s + k)^3$ filter generating $\lvert\tau\rvert^3 e^{-k\lvert\tau\rvert}$.	
Failure case IV filter number x	In a $1/(s^2 + 2ns + m^2)$ filter, then either: (a) $n^2 = m^2$, (b) $n^2 > m^2$, (c) $m = 0$, or, (d) $n = 0$.	If (a) or (b), refactorise the $(s^2 + 2ns + m^2)$ factor and repeat the calculation. If (c) or (d), transfer function is inadmissible.

REFERENCES

No.	Author	Title, etc
1	J. F. Koenig	A method for obtaining the time response of any linear system. I.E.E.E. Trans on Automatic control, AC 9, No.4, October 1964
2	E. Huntley	A matrix formulation for the time response of time-invariant linear systems to discrete inputs. R.A.E. Technical Report 66038 (A.R.C. 28188) (1966) Int. J. Control, 1966, Vol.4, No.1, 49-72
3	E. Huntley	Matrix methods for the analytic determination of the output autocorrelation functions of linear systems for stationary random inputs. R.A.E. Technical Report 66387, (1966) Part I System autocorrelation function method. (A.R.C. 29278) Part II Serial method (A.R.C. 29279) Int. J. Control Vol. 10, No.1,1-12 and 13-27
4	E. Huntley	The response analysis of linear multivariable systems by an extension of the serial/matrix technique. Int. J. Control, 1967, Vol.5, No.5, 437-450
5	K. Steiglitz	Rational transform approximation via the Laguerre Spectrum. Jo. of Franklin Institute, Nov. 1965, 387-394

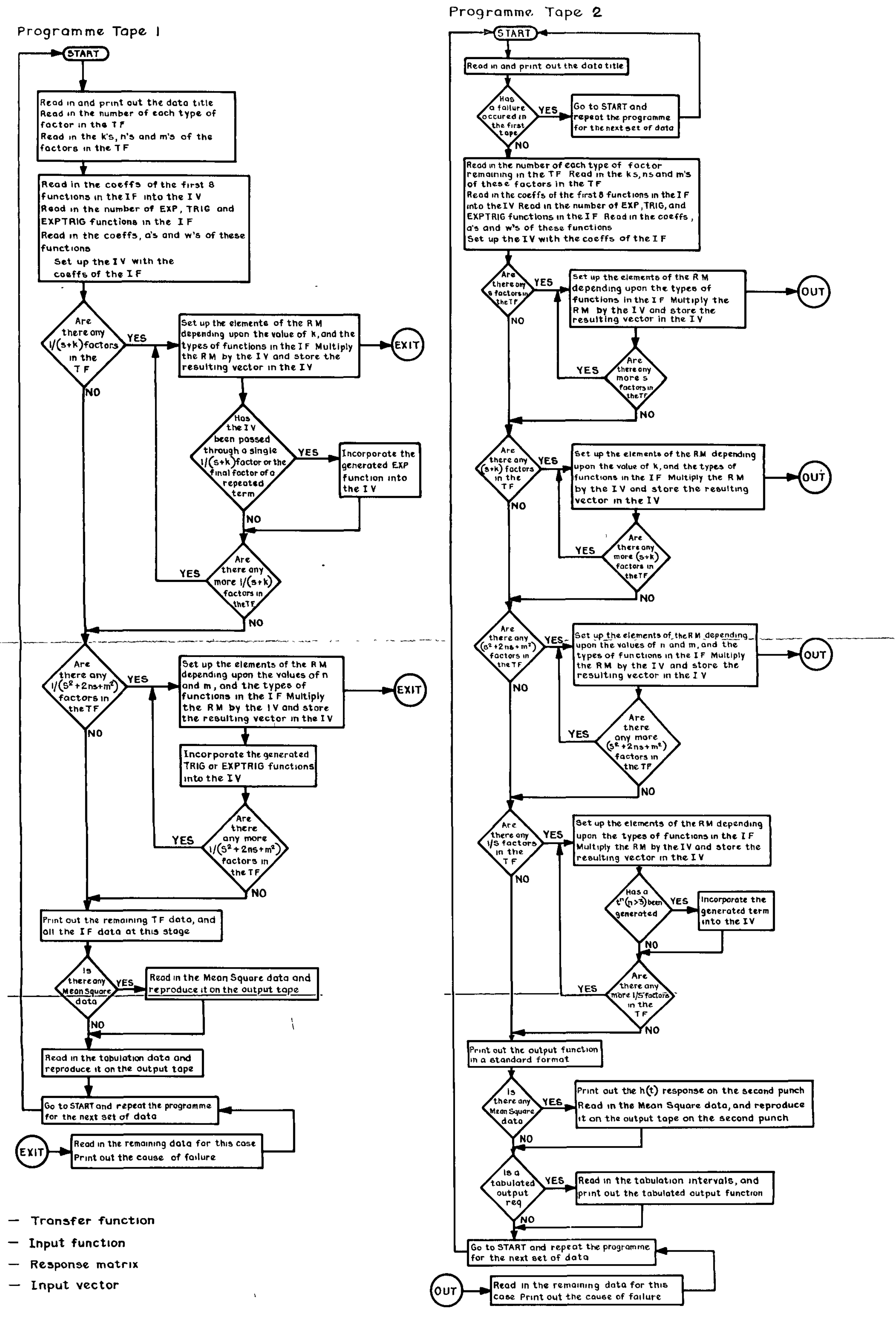


Fig.1 Flow diagram for Programme I: Tapes 1 and 2

IF — Transfer function
IF — Input function
RM — Response matrix
IV — Input vector

RESPONSE OF LINEAR SYSTEMS PROGRAMME I

TRANSFER FUNCTION

TITLE	£ ?

A TITLE OF UP TO 30 CHARACTERS MUST BE INSERTED BETWEEN £ AND ? CHARACTERS, IN THE BLOCK ABOVE

ALL THE VALUES OF THE INTEGER ARRAY $y[1 \; 6]$ MUST BE FILLED IN A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FACTOR IS NOT PRESENT IN THE TRANSFER FUNCTION

THE k's, n's AND m's NEED NOT BE FILLED IN IF THEIR CORRESPONDING y IS ZERO

UP TO SIX FACTORS OF ONE TYPE ARE ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO A BLOCK TO ALLOW FOR MORE THAN SIX FACTORS IF NECESSARY

FACTORS OF THE TYPE $1/(s+k)^n (n \leq 3)$ MUST BE DECLARED AS $1/(s+k) \quad (s+k)$, ALL THE FACTORS BEING ALLOWED FOR IN $y[1]$, AND "n" k's BEING WRITTEN IN THE "$1/(s+k)$ FACTOR" BLOCK k's OF NON-REPEATED FACTORS MUST PRECEDE THOSE OF ANY REPEATED FACTORS

CONSTANT GAIN	

NUMBER OF $1/(s+k)$ FACTORS INCLUDING REPEATED FACTORS		$y[1]$
NUMBER OF $1/(s^2+2ns+m^2)$ FACTORS, FOR $n<m$		$y[2]$
NUMBER OF s FACTORS		$y[3]$
NUMBER OF $(s+k)$ FACTORS		$y[4]$
NUMBER OF $(s^2+2ns+m^2)$ FACTORS		$y[5]$
NUMBER OF $1/s$ FACTORS		$y[6]$

VALUES OF k's IN $1/(s+k)$ FACTORS
$k_1, k_2, \quad, k_{y[1]}$

VALUES OF n's AND m's IN $1/(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \quad, n_{y[2]}$	$m_1, m_2, \quad, m_{y[2]}$

VALUES OF k's IN $(s+k)$ FACTORS
$k_1, k_2, \quad, k_{y[4]}$

VALUES OF n's AND m's IN $(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \quad, n_{y[5]}$	$m_1, m_2, \quad, m_{y[5]}$

A

INPUT FUNCTION

A

TYPE OF FUNCTION	COEFFICIENT
$u_4(t)$	
$u_3(t)$	
$u_2(t)$	
$u_1(t)$	
$u_0(t)$	
t	
t^2	
t^3	

$e^{-at}(\alpha u_0(t)+\beta t+\gamma t^2)$	
NUMBER OF FUNCTIONS OF THIS TYPE	$p[9]$

$\alpha \sin(\omega t)+\beta \cos(\omega t)$
$p[10]$

$e^{at}(\alpha \sin(\omega t)+\beta \cos(\omega t))$
$p[11]$

B

B

α_1		α_1		α_1	
β_1		β_1		β_1	
γ_1					
α_2		α_2		α_2	
β_2		β_2		β_2	
γ_2					
α_3		α_3		α_3	
β_3		β_3		β_3	
γ_3					

a_1		a_1		a_1	
ω_1		ω_1		ω_1	
a_2		a_2		a_2	
ω_2		ω_2		ω_2	
a_3		a_3		a_3	
ω_3		ω_3		ω_3	

C

C

INSERT A ZERO HERE	O

TABULATION

NUMBER OF LARGE TIME INTERVALS - del	

t_0	
h_0	
t_1	
h_1	
t_2	
h_2	
t_3	
h_3	
t_4	

THE INTEGER ARRAY $p[9, \; ,11]$ DETERMINES HOW MANY OF EACH TYPE OF FUNCTION ARE PRESENT IN THE INPUT A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FUNCTION IS NOT PRESENT THE COLUMNS UNDER EACH TYPE OF FUNCTION NEED NOT BE FILLED IN IF THEIR CORRESPONDING p' IS ZERO ONLY p OF THE $\alpha, \beta, (\gamma)$ AND a, ω BLOCKS NEED BE FILLED IN

SPACES FOR ONLY THREE OF EACH TYPE OF FUNCTION HAVE BEEN ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO THE APPROPRIATE BLOCK TO ALLOW FOR MORE THAN THREE OF A GIVEN TYPE IF NECESSARY IN THE $p[9]$ BLOCK, FOR EACH 'a' ITS CORRESPONDING 'ω' MUST BE WRITTEN AS ZERO IN THE $p[10]$ BLOCK, FOR EACH 'ω', ITS CORRESPONDING 'a' MUST BE WRITTEN AS ZERO

IF NO TABULATION OF THE OUTPUT FUNCTION IS REQUIRED THEN THE INTEGER 'del' MUST BE WRITTEN AS ZERO OTHERWISE 'del' IS THE NUMBER OF LARGE TIME INTERVALS, ie THE SUFFIX OF THE FINAL VALUE OF t

– – – ► – INDICATES THE ORDER IN WHICH THE DATA MUST BE PUNCHED

EACH NUMBER MUST BE TERMINATED BY A NEW LINE

FIG. 2 BLANK DATA SHEET FOR PROGRAMME I

RESPONSE OF LINEAR SYSTEMS PROGRAMME I

TRANSFER FUNCTION

SEE SECTION 4

CONSTANT GAIN	0 634102

NUMBER OF $1/(s+k)$ FACTORS INCLUDING REPEATED FACTORS	3	y [1]
NUMBER OF $1/(s^2+2ns+m^2)$ FACTORS, FOR $n<m$	1	y [2]
NUMBER OF s FACTORS	1	y [3]
NUMBER OF $(s+k)$ FACTORS	0	y [4]
NUMBER OF $(s^2+2ns+m^2)$ FACTORS	2	y [5]
NUMBER OF $1/s$ FACTORS	0	y [6]

VALUES OF k's IN $1/(s+k)$ FACTORS
$k_1, k_2, \ldots, k_{y[1]}$
−0·31401
5 549
5 549

VALUES OF n's AND m's IN $1/(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \ldots, n_{y[2]}$	$m_1, m_2, \ldots, m_{y[2]}$
3 7372	6·21034

VALUES OF k's IN $(s+k)$ FACTORS
$k_1, k_2, \ldots, k_{y[4]}$

VALUES OF n's AND m's IN $(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \ldots, n_{y[5]}$	$m_1, m_2, \ldots, m_{y[5]}$
8 8949	10 3265
−4·7884	5·26

A

TITLE	£ ILLUSTRATIVE EXAMPLE ?

A TITLE OF UP TO 30 CHARACTERS MUST BE INSERTED BETWEEN £ AND ? CHARACTERS, IN THE BLOCK ABOVE

ALL THE VALUES OF THE INTEGER ARRAY y[1, 6] MUST BE FILLED IN A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FACTOR IS NOT PRESENT IN THE TRANSFER FUNCTION

THE k's, n's AND m's NEED NOT BE FILLED IN IF THEIR CORRESPONDING y IS ZERO

UP TO SIX FACTORS OF ONE TYPE ARE ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO A BLOCK TO ALLOW FOR MORE THAN SIX FACTORS IF NECESSARY

FACTORS OF THE TYPE $1/(s+k)^n$ $(n \leq 3)$ MUST BE DECLARED AS $1/(s+k)$ $(s+k)$ ALL THE FACTORS BEING ALLOWED FOR IN y[1], AND 'n' k's BEING WRITTEN IN THE '$1/(s+k)$ FACTOR' BLOCK k's OF NON-REPEATED FACTORS MUST PRECEDE THOSE OF ANY REPEATED FACTORS

INPUT FUNCTION

SEE SECTION 4

A

TYPE OF FUNCTION	COEFFICIENT
$u_4(t)$	0
$u_3(t)$	0
$u_2(t)$	0
$u_1(t)$	0
$u_0(t)$	3 507
t	0
t^2	6·25
t^3	0

$e^{-at}(\alpha u_0(t)+\beta t+\gamma t^2)$	
NUMBER OF FUNCTIONS OF THIS TYPE	2
	p [9]

B

α_1	−2 9131
β_1	0
γ_1	0
α_2	0
β_2	0·5928
γ_2	0 31601
α_3	
β_3	
γ_3	

a_1	2 79923
ω_1	0
a_2	1 304
ω_2	0
a_3	
ω_3	

$\alpha \sin(\omega t)+\beta \cos(\omega t)$
0
p [10]

α_1	
β_1	
α_2	
β_2	
α_3	
β_3	

a_1	
ω_1	
a_2	
ω_2	
a_3	
ω_3	

$e^{at}(\alpha \sin(\omega t)+\beta \cos(\omega t))$
1
p [11]

B

α_1	0
β_1	5·3857
α_2	
β_2	
α_3	
β_3	

a_1	1 9486
ω_1	7·2619
a_2	
ω_2	
a_3	
ω_3	

C

C

INSERT A ZERO HERE	0

TABULATION

NUMBER OF LARGE TIME INTERVALS — del	3

t_0	3
h_0	0 1
t_1	5
h_1	1
t_2	6
h_2	0 05
t_3	10·5
h_3	
t_4	

THE INTEGER ARRAY p[9, 11] DETERMINES HOW MANY OF EACH TYPE OF FUNCTION ARE PRESENT IN THE INPUT A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FUNCTION IS NOT PRESENT THE COLUMNS UNDER EACH TYPE OF FUNCTION NEED NOT BE FILLED IN IF THEIR CORRESPONDING "p" IS ZERO ONLY "p" OF THE α, β, (γ) AND a, ω BLOCKS NEED BE FILLED IN

SPACES FOR ONLY THREE OF EACH TYPE OF FUNCTION HAVE BEEN ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO THE APPROPRIATE BLOCK TO ALLOW FOR MORE THAN THREE OF A GIVEN TYPE IF NECESSARY IN THE p[9] BLOCK, FOR EACH "a" ITS CORRESPONDING "ω" MUST BE WRITTEN AS ZERO IN THE p[10] BLOCK, FOR EACH "ω", ITS CORRESPONDING "a" MUST BE WRITTEN AS ZERO

IF NO TABULATION OF THE OUTPUT FUNCTION IS REQUIRED THEN THE INTEGER "del" MUST BE WRITTEN AS ZERO OTHERWISE "del" IS THE NUMBER OF LARGE TIME INTERVALS, i.e. THE SUFFIX OF THE FINAL VALUE OF t

– – – ► — INDICATES THE ORDER IN WHICH THE DATA MUST BE PUNCHED

EACH NUMBER MUST BE TERMINATED BY A NEW LINE

FIG. 3 EXAMPLE OF COMPLETED DATA SHEET

RESPONSE OF LINEAR SYSTEMS PROGRAMME I

TRANSFER FUNCTION

SEE RESULTS FROM DEUCE POLYNOMIAL FACTORISATION PROGRAMME (FIG 5)

CONSTANT GAIN	·02191

NUMBER OF $1/(s+k)$ FACTORS INCLUDING REPEATED FACTORS	2	y [1]
NUMBER OF $1/(s^2+2ns+m^2)$ FACTORS, FOR $n<m$	3	y [2]
NUMBER OF s FACTORS	0	y [3]
NUMBER OF $(s+k)$ FACTORS	1	y [4]
NUMBER OF $(s^2+2ns+m^2)$ FACTORS	3	y [5]
NUMBER OF $1/s$ FACTORS	0	y [6]

VALUES OF k's IN $1/(s+k)$ FACTORS
$k_1, k_2, \ldots, k_{y[1]}$
·57937185
5·4301403

VALUES OF n's AND m's IN $1/(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \ldots, n_{y[2]}$	$m_1, m_2, \ldots, m_{y[2]}$
50668008	·83243297
·26989885	1·4162948
·63016501	2·2942107

VALUES OF k's IN $(s+k)$ FACTORS
$k_1, k_2, \ldots, k_{y[4]}$
13 251822

VALUES OF n's AND ms IN $(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \ldots, n_{y[5]}$	$m_1, m_2, \ldots, m_{y[6]}$
−1·9804204	2·0518301
−1·993108	2·1119914
−1·4371839	2·0556478

A

TITLE	£ IMPULSE RESPONSE OF A TRANSFER FUNCTION BY STEIGLITZ ?

A TITLE OF UP TO 30 CHARACTERS MUST BE INSERTED BETWEEN £ AND ? CHARACTERS, IN THE BLOCK ABOVE

ALL THE VALUES OF THE INTEGER ARRAY y[1, 6] MUST BE FILLED IN A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FACTOR IS NOT PRESENT IN THE TRANSFER FUNCTION

THE k's, n's AND m's NEED NOT BE FILLED IN IF THEIR CORRESPONDING y IS ZERO

UP TO SIX FACTORS OF ONE TYPE ARE ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO A BLOCK TO ALLOW FOR MORE THAN SIX FACTORS IF NECESSARY

FACTORS OF THE TYPE $1/(s+k)^n (n \leq 3)$ MUST BE DECLARED AS $1/(s+k)\ (s+k)$, ALL THE FACTORS BEING ALLOWED FOR IN y[1], AND "n" k's BEING WRITTEN IN THE "$1/(s+k)$ FACTOR" BLOCK k s OF NON-REPEATED FACTORS MUST PRECEDE THOSE OF ANY REPEATED FACTORS

INPUT FUNCTION

UNIT IMPULSE , $u_1(t)$

A

TYPE OF FUNCTION	COEFFICIENT
$u_4(t)$	0
$u_3(t)$	0
$u_2(t)$	0
$u_1(t)$	1
$u_0(t)$	0
t	0
t^2	0
t^3	0

$\epsilon^{-at}(\alpha u_0(t)+\beta t+\gamma t^2)$	$\alpha \sin(\omega t)+\beta \cos(\omega t)$	$\epsilon^{-at}(\alpha \sin(\omega t)+\beta \cos(\omega t))$
NUMBER OF FUNCTIONS OF THIS TYPE: 0 p [9]	0 p [10]	0 p [11]

B

B α_1, β_1, γ_1	α_1, β_1	α_1, β_1
α_2, β_2, γ_2	α_2, β_2	α_2, β_2
α_3, β_3, γ_3	α_3, β_3	α_3, β_3
a_1, ω_1, a_2, ω_2, a_3, ω_3	a_1, ω_1, a_2, ω_2, a_3, ω_3	a_1, ω_1, a_2, ω_2, a_3, ω_3

C

C

INSERT A ZERO HERE	0

TABULATION

NUMBER OF LARGE TIME INTERVALS - del	3

t_0	0
h_0	·1
t_1	4
h_1	05
t_2	9
h_2	·1
t_3	10
h_3	
t_4	

THE INTEGER ARRAY p[9, ,11] DETERMINES HOW MANY OF EACH TYPE OF FUNCTION ARE PRESENT IN THE INPUT A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FUNCTION IS NOT PRESENT THE COLUMNS UNDER EACH TYPE OF FUNCTION NEED NOT BE FILLED IN IF THEIR CORRESPONDING "p" IS ZERO ONLY "p" OF THE $\alpha, \beta, (\gamma)$ AND a, ω BLOCKS NEED BE FILLED IN

SPACES FOR ONLY THREE OF EACH TYPE OF FUNCTION HAVE BEEN ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO THE APPROPRIATE BLOCK TO ALLOW FOR MORE THAN THREE OF A GIVEN TYPE IF NECESSARY IN THE p[9] BLOCK, FOR EACH "a" ITS CORRESPONDING "ω" MUST BE WRITTEN AS ZERO IN THE p[10] BLOCK, FOR EACH "ω", ITS CORRESPONDING "a" MUST BE WRITTEN AS ZERO

IF NO TABULATION OF THE OUTPUT FUNCTION IS REQUIRED THEN THE INTEGER "del" MUST BE WRITTEN AS ZERO OTHERWISE "del" IS THE NUMBER OF LARGE TIME INTERVALS, ie THE SUFFIX OF THE FINAL VALUE OF t

——▶— INDICATES THE ORDER IN WHICH THE DATA MUST BE PUNCHED

EACH NUMBER MUST BE TERMINATED BY A NEW LINE

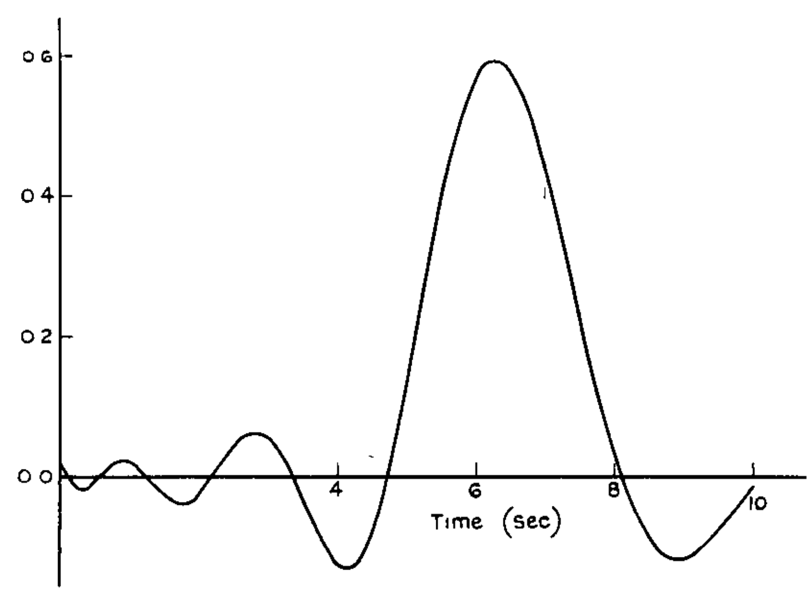
FIG.4 COMPLETED DATA SHEET FOR WORKED EXAMPLE

Results from the DEUCE polynomial factorization programme

```
Numerator
                 -13 2518217
                  1 98042042 ± i 0 536601929
                  1 99310798 ± i 0 698590264
                  1 43718389 ± i 1 46975862

Denominator      -0 579371846
                 -5 43014026
                 -0 506680077 ± i 0 660091341
                 -0 269898855 ± i 1 39034011
                 -0 630165014 ± i 2 20596800
```

Results from Programme I

```
Response of Linear Systems Programme I

Impulse response of a transfer function by Steiglitz

  Output function

exp(-at)*(u(t),t,t↑2)  a=  57937185
 3 9767338
  00000000
  00000000

exp(-at)*(u(t),t,t↑2)  a= 5 4301403
- 27189447
  00000000
  00000000

exp(-at)*(sinwt,coswt)  a=  50668008  w=  66009134
-5 0906952
-1 8236981

exp(-at)*(sinwt,coswt)  a=  26989885  w= 1 3903401
  33789070
-1 5287643

exp(-at)*(sinwt,coswt)  a=  63016501  w= 2 2059680
  92167295
- 33046917

  Tabulated output function

     time          function       interval=  10000000

    00000000        02195952
    10000000        00439558
    20000000      - 01066167
    30000000      - 01696600
       |               |
       |               |
       |               |
   9 9000000      - 02310279
   10 000000      - 01004826
```



Plotted response

Fig 5 Results for worked example (section 5)

Input function

$$3\ 21\, U_3(t) + 4{\cdot}21\, U_2(t) + 5{\cdot}21\, U_1(t) + 6{\cdot}21\, U_0(t)$$
$$+ 2t + 3t^2 + 4t^3$$
$$+ e^{-\cdot 5t}\,(6\ 25 + 3{\cdot}25\,t + 0{\cdot}25\,t^2)$$
$$+ 5\ 2 \sin(1{\cdot}5t) + {\cdot}2\cos(1{\cdot}5t)$$
$$+ e^{-2{\cdot}5t}\,(6\ 33\sin(3{\cdot}9t) + 6{\cdot}22\cos(3\ 9t))$$

Transfer function

$$\frac{(s+3{\cdot}2)(s^2+3{\cdot}2s+18{\cdot}49)\,s}{(s+3\ 2)(s^2+3{\cdot}2s+18{\cdot}49)\,s}$$

Output function

Response of linear systems programme I

Test Ch 123456 S01)

Output function

3·2100000 u3(t)
4·2100001 u2(t)
5·2099996 u1(t)

6 2100001 u0(t)
1·9999998 t↑1
2·9999999 t↑2
3·9999998 t↑3

exp(−at)*(u(t), t, t↑2) a= ·50000000
6·2499999
3·2500000
·24999996

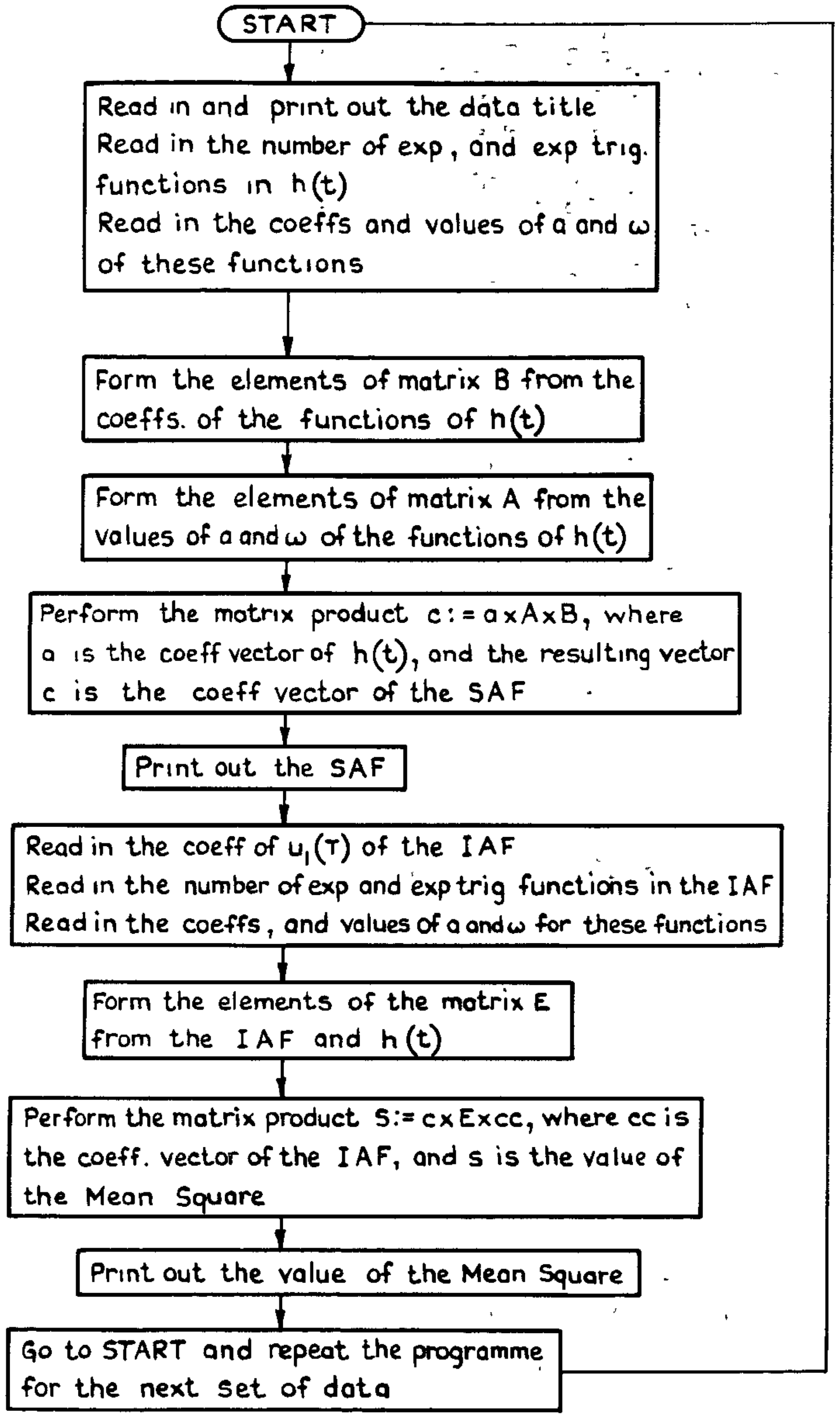
exp(−at)*(u(t), t, t↑2) a= 3 2000000
·00000000
00000000
·00000000

sinwt, coswt W= 1·5000000
5·2000002
·19999996

exp(−at)*(sinwt, coswt) a= 2·5000000 W=3·9000000
6·3300000
6·2200001

exp(−at)*(sinwt, coswt) a= 1·6000000 W= 3·9912404
·00000000
00000000

Fig. 6 Example for test of accuracy



(SAF = System Autocorrelation Function
IAF = Input Autocorrelation Function)

Fig 7 Flow diagram for programme II

RESPONSE OF LINEAR SYSTEMS. PROGRAMME II.

MEAN SQUARE PROGRAMME
(USED IN CONJUNCTION WITH PROGRAMME I)

TRANSFER FUNCTION

TITLE | & ?

A TITLE OF UP TO 30 CHARACTERS MUST BE INSERTED BETWEEN & AND ? CHARACTERS, IN THE BLOCK ABOVE.

ALL THE VALUES OF THE INTEGER ARRAY y[1, ,6] MUST BE FILLED IN. A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FACTOR IS NOT PRESENT IN THE TRANSFER FUNCTION

THE K's, n's AND m's NEED NOT BE FILLED IN IF THEIR CORRESPONDING "y" IS ZERO

UP TO SIX FACTORS OF ONE TYPE ARE ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO A BLOCK TO ALLOW FOR MORE THAN SIX FACTORS IF NECESSARY.

FACTORS OF THE TYPE $1/(s+k)^n$ $(n \leq 3)$, MUST BE DECLARED AS $1/(s+k)$ $(s+k)$. ALL THE FACTORS BEING ALLOWED FOR IN y[1], AND "n" K's BEING WRITTEN IN THE "$1/(s+k)$ FACTOR" BLOCK k's OF NON-REPEATED FACTORS MUST PRECEDE THOSE OF ANY REPEATED FACTORS

CONSTANT GAIN

NUMBER OF $1/(s+k)$ FACTORS, INCLUDING REPEATED FACTORS		y[1]
NUMBER OF $1/(s^2+2ns+m^2)$ FACTORS FOR $n<m$		y[2]
NUMBER OF s FACTORS		y[3]
NUMBER OF $(s+k)$ FACTORS		y[4]
NUMBER OF $(s^2+2ns+m^2)$ FACTORS		y[5]
NUMBER OF 1/s FACTORS		y[6]

VALUES OF k's IN $1/(s+k)$ FACTORS
$k_1, k_2, \ldots, k_{y[1]}$

VALUES OF n's AND m's IN $1/(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \ldots, n_{y[2]}$	$m_1, m_2, \ldots, m_{y[2]}$

VALUES OF k's IN $(s+k)$ FACTORS
$k_1, k_2, \ldots, k_{y[4]}$

VALUES OF n's AND m's IN $(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \ldots, n_{y[5]}$	$m_1, m_2, \ldots, m_{y[5]}$

A

INSERT THE FOLLOWING DATA. (THIS IS TO PRODUCE THE SYSTEM AUTOCORRELATION FUNCTION)

A

0
0
0
1
0
0
0
0
0
0
0
1

THE SYSTEM AUTOCORRELATION FUNCTION IS PRODUCED BY THE PROGRAMME, AND DEPENDS ONLY UPON THE TRANSFER FUNCTION DATA

IF ONLY THE SYSTEM AUTOCORRELATION FUNCTION IS REQUIRED THEN AN ARBITRARY INPUT AUTOCORRELATION FUNCTION SHOULD BE ENTERED BELOW.

INPUT AUTOCORRELATION FUNCTION

COEFFICIENT OF $u_1(\tau)$

B

$e^{-a\tau}(\alpha u_0(\tau)+\beta\tau+\gamma\tau^2)$ — NUMBER OF FUNCTIONS OF THIS TYPE — pp1

$e^{-a\tau}(\alpha \sin(\omega\tau)+\beta\cos(\omega\tau))$ — pp2

B

α_1
β_1
γ_1
α_2
β_2
γ_2
α_3
β_3
γ_3
a_1
a_2
a_3

α_1
β_1
α_2
β_2
α_3
β_3
a_1
ω_1
a_2
ω_2
a_3
ω_3

THE COEFFICIENT OF $u_1(\tau)$ MUST BE FILLED IN. A ZERO MUST BE WRITTEN IF THIS FUNCTION IS NOT PRESENT IN THE INPUT AUTOCORRELATION FUNCTION.

THE INTEGERS pp1 AND pp2 DETERMINE HOW MANY OF EACH TYPE OF FUNCTION ARE PRESENT IN THE INPUT AUTOCORRELATION FUNCTION. A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FUNCTION IS NOT PRESENT.

THE COLUMNS UNDER EACH TYPE OF FUNCTION NEED NOT BE FILLED IN IF THEIR CORRESPONDING "pp" IS ZERO. ONLY "pp" OF THE $\alpha, \beta, (\gamma)$ AND a, ω BLOCKS NEED BE FILLED IN.

INSERT A ZERO HERE | 0

SPACES FOR ONLY THREE OF EACH TYPE OF FUNCTION HAVE BEEN ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO THE APPROPRIATE BLOCK TO ALLOW FOR MORE THAN THREE OF A GIVEN TYPE IF NECESSARY.

- - - ▶ - - - INDICATES THE ORDER IN WHICH THE DATA MUST BE PUNCHED.

EACH NUMBER MUST BE TERMINATED BY A NEW LINE

A

FIG. 8 BLANK DATA SHEET FOR PROGRAMME II

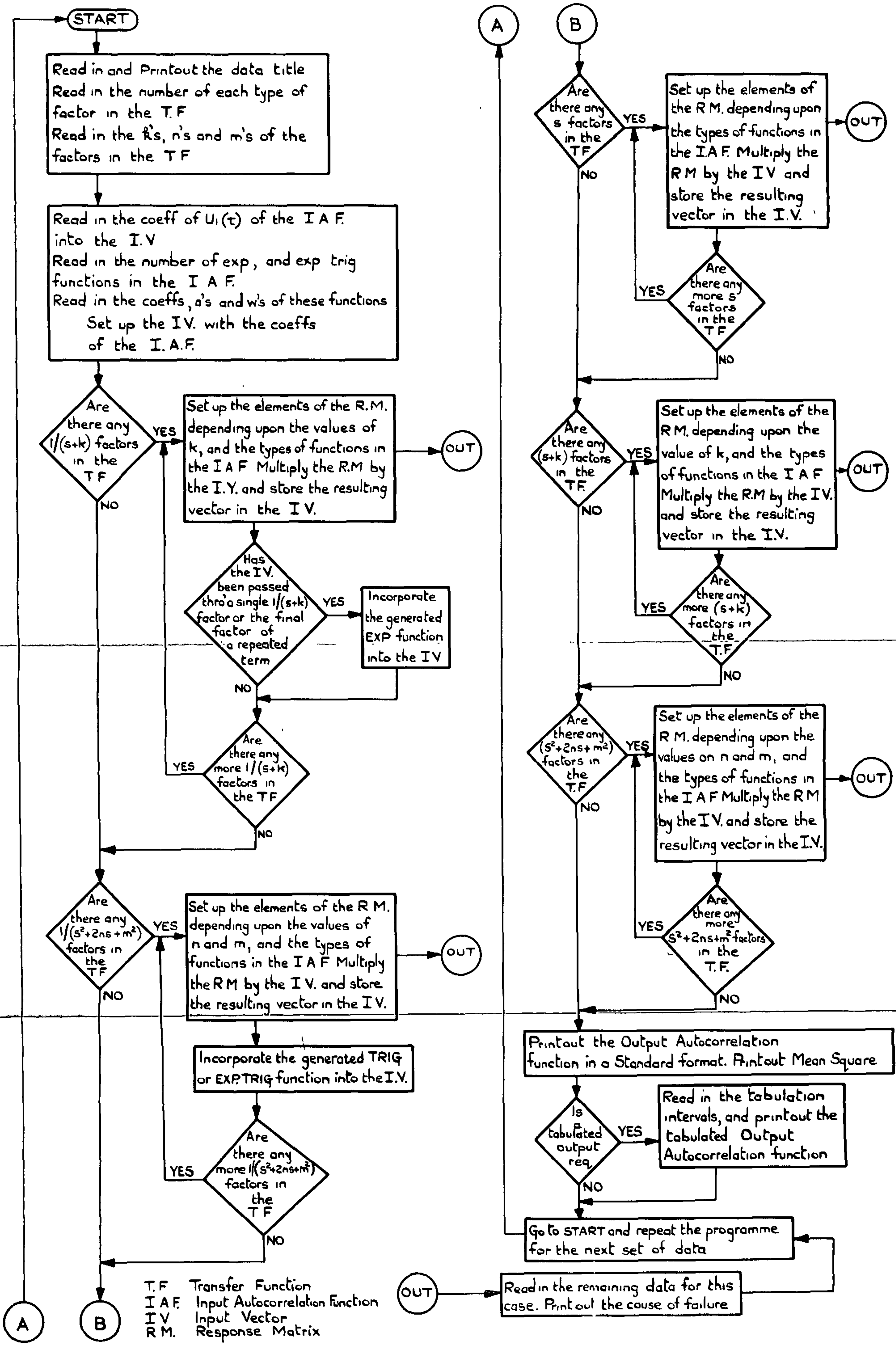


Fig.9 Flow diagram for Programme III

RESPONSE OF LINEAR SYSTEMS. PROGRAMME III

TRANSFER FUNCTION

TITLE £ ?

A TITLE OF UP TO 30 CHARACTERS MUST BE INSERTED BETWEEN £ AND ? CHARACTERS, IN THE BLOCK ABOVE.

ALL THE VALUES OF THE INTEGER ARRAY y [1, 5] MUST BE FILLED IN. A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FACTOR IS NOT PRESENT IN THE TRANSFER FUNCTION.

THE k's, n's AND m's NEED NOT BE FILLED IN IF THEIR CORRESPONDING "y" IS ZERO.

UP TO SIX FACTORS OF ONE TYPE ARE ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO A BLOCK TO ALLOW FOR MORE THAN SIX FACTORS IF NECESSARY.

FACTORS OF THE TYPE $1/(s+k)^n$ ($n \leq 3$) MUST BE DECLARED AS $1/(s+k)$ $(s+k)$, ALL THE FACTORS BEING ALLOWED FOR IN y [1], AND "n" k's BEING WRITTEN IN THE "$1/(s+k)$ FACTOR" BLOCK. k's OF NON-REPEATED FACTORS MUST PRECEDE THOSE OF ANY REPEATED FACTORS.

CONSTANT GAIN		
NUMBER OF $1/(s+k)$ FACTORS INCLUDING REPEATED FACTORS		y [1]
NUMBER OF $1/(s^2+2ns+m^2)$ FACTORS, FOR $n < m$		y [2]
~~NUMBER OF s FACTORS~~		y [3]
NUMBER OF $(s+k)$ FACTORS		y [4]
NUMBER OF $(s^2+2ns+m^2)$ FACTORS		y [5]

VALUES OF k's IN $1/(s+k)$ FACTORS
$k_1, k_2, \ldots k_{y[1]}$

VALUES OF n's AND m's IN $1/(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \ldots n_{y[2]}$	$m_1, m_2, \ldots m_{y[2]}$

VALUE OF k's IN $(s+k)$ FACTORS
$k_1, k_2, \ldots k_{y[4]}$

VALUES OF n's AND m's IN $(s^2+2ns+m^2)$ FACTORS	
$n_1, n_2, \ldots n_{y[5]}$	$m_1, m_2, \ldots m_{y[5]}$

A

INPUT AUTOCORRELATION FUNCTION

A

COEFFICIENT OF $u_1(\tau)$	

$e^{-a\lvert\tau\rvert}(\alpha+\beta\lvert\tau\rvert+\gamma\lvert\tau\rvert^2)$ NUMBER OF FUNCTIONS OF THIS TYPE	p1
α_1	
β_1	
γ_1	
α_2	
β_2	
γ_2	
α_3	
β_3	
γ_3	
a_1	
ω_1	
a_2	
ω_2	
a_3	
ω_3	

B

$e^{-a\lvert\tau\rvert}(\alpha\sin(\omega\lvert\tau\rvert)+\beta\cos(\omega\lvert\tau\rvert))$	p2
α_1	
β_1	
α_2	
β_2	
α_3	
β_3	
a_1	
ω_1	
a_2	
ω_2	
a_3	
ω_3	

B

THE COEFFICIENT OF $u_1(\tau)$ MUST ALWAYS BE FILLED IN. A ZERO MUST BE WRITTEN IF THIS TYPE OF FUNCTION IS NOT PRESENT IN THE INPUT AUTOCORRELATION FUNCTION.

TABULATION

NUMBER OF LARGE TIME INTERVALS – del	
τ_0	
h_0	
τ_1	
h_1	
τ_2	
h_2	
τ_3	
h_3	
τ_4	

THE INTEGERS p1 AND p2 DETERMINE HOW MANY OF EACH TYPE OF FUNCTION ARE PRESENT IN THE INPUT. A ZERO MUST BE WRITTEN WHEREVER A GIVEN TYPE OF FUNCTION IS NOT PRESENT. THE COLUMNS UNDER EACH TYPE OF FUNCTION NEED NOT BE FILLED IN IF THEIR CORRESPONDING "p" IS ZERO. ONLY "p" OF THE $\alpha, \beta, (\gamma)$ AND a, ω BLOCKS NEED BE FILLED IN. SPACES FOR ONLY THREE OF EACH TYPE OF FUNCTION HAVE BEEN ALLOWED FOR ON THIS SHEET, BUT THE USER MAY ADD EXTRA ROWS TO THE APPROPRIATE BLOCK TO ALLOW FOR MORE THAN THREE OF A GIVEN TYPE IF NECESSARY. IN THE p1 BLOCK, FOR EACH "a", ITS CORRESPONDING "ω" MUST BE WRITTEN AS ZERO.

IF NO TABULATION OF THE OUTPUT AUTOCORRELATION FUNCTION IS REQUIRED THEN THE INTEGER "del" MUST BE WRITTEN AS ZERO. OTHERWISE "del" IS THE NUMBER OF LARGE TIME INTERVALS, ie THE SUFFIX OF THE FINAL VALUE OF τ.

---→– INDICATES THE ORDER IN WHICH THE DATA MUST BE PUNCHED

EACH NUMBER MUST BE TERMINATED BY A NEW LINE

FIG 10 BLANK DATA SHEET FOR PROGRAMME III

Printed in England for Her Majesty's Stationery Office by the Royal Aircraft Establishment, Farnborough. Dd 501371 K 4.

DETACHABLE ABSTRACT CARD

A.R.C. C.P. No. 1124
July 1969
518.5 : 621.374.32 : 519.283 : 512.831 : 519.242 : 517.511

Hazlewood, L. J.
Huntley, E.

ALGOL PROGRAMMES FOR THE RESPONSE ANALYSIS OF LINEAR SYSTEMS WITH DETERMINISTIC OR RANDOM INPUTS

In previous publications the so-called serial/matrix technique has been developed for the response analysis of systems defined by time-invariant ordinary differential equations. One paper describes how an explicit formulation for the output function may be easily obtained when the input function is deterministic. A second gives the output autocorrelation function and output mean square value when the input is a stationary random process.

This paper gives computer programmes in ALGOL which implement these ideas. The programmes are described primarily from the point of view

(Over)

A.R.C. C.P. No. 1124
July 1969
518.5 : 621.374.32 : 519.283 : 512.831 : 519.242 : 517.511
Hazlewood, L. J.
Huntley, E.

ALGOL PROGRAMMES FOR THE RESPONSE ANALYSIS OF LINEAR SYSTEMS WITH DETERMINISTIC OR RANDOM INPUTS

In previous publications the so-called serial/matrix technique has been developed for the response analysis of systems defined by time-invariant ordinary differential equations. One paper describes how an explicit formulation for the output function may be easily obtained when the input function is deterministic. A second gives the output autocorrelation function and output mean square value when the input is a stationary random process.

This paper gives computer programmes in ALGOL which implement these ideas. The programmes are described primarily from the point of view

(Over)

A.R.C. C.P. No. 1124
July 1969
518.5 : 621.374.32 : 519.283 : 512.831 : 519.242 : 517.511
Hazlewood, L. J.
Huntley, E.

ALGOL PROGRAMMES FOR THE RESPONSE ANALYSIS OF LINEAR SYSTEMS WITH DETERMINISTIC OR RANDOM INPUTS

In previous publications the so-called serial/matrix technique has been developed for the response analysis of systems defined by time-invariant ordinary differential equations. One paper describes how an explicit formulation for the output function may be easily obtained when the input function is deterministic. A second gives the output autocorrelation function and output mean square value when the input is a stationary random process.

This paper gives computer programmes in ALGOL which implement these ideas. The programmes are described primarily from the point of view

(Over)

of the user with illustrative examples to demonstrate the use of prepared data sheets but sufficient information is included to enable users to develop the programmes further if required.

of the user with illustrative examples to demonstrate the use of prepared data sheets but sufficient information is included to enable users to develop the programmes further if required.

of the user with illustrative examples to demonstrate the use of prepared data sheets but sufficient information is included to enable users to develop the programmes further if required.

C.P. No. 1124

Published by
HER MAJESTY'S STATIONERY OFFICE

To be purchased from
49 High Holborn, London w c.1
13a Castle Street, Edinburgh EH 2 3AR
109 St Mary Street, Cardiff CF1 1JW
Brazennose Street, Manchester 2
50 Fairfax Street, Bristol BS1 3DE
258 Broad Street, Birmingham 1
7 Linenhall Street, Belfast BT2 8AY
or through any bookseller

C.P. No. 1124

SBN 11 470312 4