The Mathematical Sciences Group

CR-86355

Report on

A Program for Interactive Computation In Linear Systems Theory

Prepared by

Thomas S. Englar

of

The Mathematical Sciences Group, Inc.
7100 Baltimore Avenue
College Park, Maryland

Final Report May 1970

Under Contract NAS 12-583

for

Office of Control Theory and Application
NASA/Electronics Research Center
Cambridge, Massachusetts

FOREWORD

This report describes a computer program for use in linear system design and analysis.

The purpose of the report is twofold. First, it provides a complete instruction manual for use of the program. Second, it describes the structure of the program and gives sufficient detail about implementation that a reasonably skilled FORTRAN programmer should be able to modify or add instructions to the program, thus adapting the structure to his particular requirements.

The program itself, and this report, have been prepared under Contract NAS-12-583 for NASA's Office of Control Theory and Application (OCTA) at the Electronics Research Center in Cambridge, Massachusetts.

I would like to thank all the members of OCTA for their assistance and enthusiasm in discussing the possible applications of work produced under this contract. In particular the technical officer, Dr. William Wolovich, has been a constant help and encouragement in the development, and an enthusiastic user of the program.

TABLE OF CONTENTS

INTRODUCTION

This report describes a computer program for use in the analysis and design of linear dynamical systems. The program is written in FORTRAN IV for use on remote terminals which access a time-shared, conversational computer system.

The intent has been to have a structure in which arbitrary subroutines could be used, their output monitored, parameter changes made, and the process repeated until a satisfactory conclusion has been reached. To mechanize these aims we have set up an interpretive program which requests commands from the user and operates upon data stored in the machine.

In accordance with present-day thinking in systems theory, both time domain and frequency domain analysis methods should be available. Therefore the first stage of programming constructed a basic set of subroutines for manipulating constant matrices. The second stage would have performed the same task for matrices whose elements are ratios of polynomials. Only a beginning has been made on this task. In addition some vector space operations, using matrices to represent spaces, have been made available. These mechanize the operations required in the Wonham-Morse Geometric Approach to the decoupling problem [1].

The first chapter is an instruction manual on how to use the program.

The second chapter outlines some possible applications, discusses limitations on size and accuracy, and suggests some improvements.

The third chapter is devoted to information which a programmer would need to modify the program. It explains program structure and data storage techniques as well as giving some of the changes which would probably be required in changing machines.

To learn to use the program quickly and efficiently, I suggest that the user read Chapter 1 quickly, and then provide himself with a copy of Appendix I and begin operations at a terminal.

The program has not been extensively used as yet in applications, but one significant problem has been solved. All computations to decouple a hovering helicopter for purposes of manual control were performed by the program. This work is described in [2].

Some comparisons should be made between this program and ASP, the Automatic Synthesis Program [3]. The significant difference is the conversational mode of operation used here as opposed to the batch mode of ASP. Even when batch turn-around is immediate, which is rare, it is difficult to pick up in the middle of a run to change a parameter. It is also inconvenient to change the course of the program on the basis of intermediate results. Briefly the man-machine interaction allows decisions to be made as required on the basis of visible results; this removes the necessity of planning ahead for all possible contingencies.

As far as content is concerned, the prospectus for this program more than covers the capabilities of ASP, which only handled matrix operations and quadratic control and filtering problems.

The present program could easily include these, as well as vector space operations, and matrix operations over the field of rational forms in a single variable.

CHAPTER 1

MANUAL FOR PROGRAM USE

The program is designed for use on a conversational system accessed through a remote terminal. To a certain extent the program is self-instructional; for this reason, and also because it is pedagogically sound, the technique of this chapter will be to simulate the action of the computer if its instructional capabilities were fully utilized.

To this end we will assume that the parts of computer operation which are dependent upon the installation have been performed; that is, an executable version of the program has been loaded and execution has started. Remarks by the computer and by the user will be in upper case, as they actually appear on the terminal. User replies will be distinguished by underlining. All other text appearing in this report is additional comment and explanation.

* * * * * *

1.1 Initiation and Read-In

DO YOU WANT HELP IN USING THE PROGRAM?
ANSWER YES OR NO FOLLOWED BY A RETURN

A yes answer,

YES

produces the following printout. (All input by the user should be followed by a return punch.)

THIS PROGRAM ENABLES YOU TO ACCESS VARIOUS OPERATIONAL PROGRAMS IN AN ARBITRARY FASHION, RETAINING THE COMPUTATIONAL OUTPUT IN THE MACHINE. TO ACCESS AN OPERATION, TYPE ITS DICTIONARY ENTRY NUMBER WHEN THE COMPUTER REQUESTS IT.

***

THE DICTIONARY ENTRIES ARE: (see Appendix I for complete listing)

***

1. READ: ENABLES THE USER TO ENTER A MATRIX FROM THE TERMINAL
2. WRITE: ENABLES THE USER TO LOOK AT HIS RESULTS.
3. ADD: ADDS TWO MATRICES.
4. GINVERSE: COMPUTES THE GENERALIZED INVERSE OF A MATRIX.
5. SUBTRACT: SUBTRACTS TWO MATRICES.
6. MULTIPLY: MULTIPLIES TWO MATRICES.
7. DELETE: DELETES A MATRIX FROM STORAGE.
8. TRANSPOSE: FORMS THE TRANSPOSED MATRIX.
9. COPY: COPIES A MATRIX INTO ANOTHER STORAGE AREA, GIVING IT ANOTHER NAME.
10. TRACE: COMPUTES THE TRACE OF A MATRIX.
11. ALTER: PRINTS OUT THE M,N ELEMENT OF A MATRIX AND ENABLES THE USER TO ALTER IT.
12. ROOTS: FINDS THE ROOTS OF A POLYNOMIAL.
13. ORBIT: COMPUTES THE CONTROLLABILITY MATRIX.
14. MEMORY: LISTS MATRICES PRESENTLY STORED.
15. LUENBERGER: FINDS, FOR (A,B) THE TRANSFORM TO LUEN FORM 1
16. JUXTC: FORMS MATRIX (A,B) FROM A AND B.
17. MEXP: FORMS MATRIX EXPONENTIAL.

18. REDUCE: GIVEN AN N×M MATRIX OF RANK R , PRODUCES AN N×R MATRIX WITH ORTHONORMAL COLUMNS. SUCH A MATRIX WILL BE CALLED A SPACE.

19. PERP: GIVEN A SPACE, FINDS THE ORTHOGONAL COMPLEMENT.

20. INTER: GIVEN TWO SPACES, FINDS THEIR INTERSECTION.

21. PMULT: GIVEN TWO POLYNOMIALS, COMPUTES THEIR PRODUCT.

22. RMDET: GIVEN A MATRIX WHOSE ELEMENTS ARE RATICS OF POLYNOMIALS, FINDS ITS DETERMINANT.

23. RMADD: GIVEN TWO MATRICES WHOSE ELEMENTS ARE RATIOS OF POLYNOMIALS, FINDS THEIR SUM.

24. RMSUBT: GIVEN TWO MATRICES WHOSE ELEMENTS ARE RATIOS OF POLYNOMIALS, FINDS THEIR DIFFERENCE.

****

QUANTITIES IN THE COMPUTER ARE ACCESSED BY NAME - A NAME BEING COMPOSED OF SIX CHARACTERS. NAMES MUST ALWAYS BE TYPED AS THEY WERE ORIGINALLY TYPED - BLANKS ARE IMPORTANT. IF YOU FOLLOW THE DICTIONARY ENTRY NUMBER WITH A ', ', A '1', AND A 'RETURN', YOU WILL RECEIVE HELP WITH THAT ENTRY ON THAT EXECUTION. FOLLOW THE DICTIONARY ENTRY NUMBER WITH A 'RETURN' ALONE TO DELETE THIS EXPLANATORY PRINTING. SINCE THERE ARE CURRENTLY NO MATRICES IN THE COMPUTER, YOUR FIRST INSTRUCTION WILL BE #1, READ.

Hereafter a matrix whose elements are ratios of polynomials will be referred to as a rational matrix or a matrix of rational fractions.

A no answer to the initial question would have skipped the preceding information and immediately printed the request which now follows.

TYPE DICTIONARY ENTRY NUMBER

Please notice that your reply to this should be

1

or

1,1

either of which should, of course, be followed by a return punch.

The reason why the first instruction must be used to read in a matrix needs to be explained. When an instruction is activated by typing its dictionary entry number, it does not, in general, ask for data to be typed in. It asks for the name of a storage area which contains the required data. The order of events is that a matrix is entered via the Read instruction, #1, and then this matrix is retained for use in a variety of ways. Since at this point we have just started execution of the program, it follows that there is nothing in storage upon which to operate - hence we ask to read in some data.

We will assume that

1,1

is typed, giving the following printout:

THIS INSTRUCTION ENABLES YOU TO PUT DATA INTO THE MACHINE IN THE FORM OF A RECTANGULAR ARRAY - A MATRIX. THE COMPUTER FIRST ASKS FOR THE NAME OF THE ARRAY. THIS MAY BE AT MOST SIX CHARACTERS - PLEASE TYPE

THE NAME, FOLLOWED BY A 'RETURN', AFTER THE COMPUTER ASKS 'GIVE MATRIX NAME'. NEXT THE COMPUTER WILL ASK WHAT TYPE OF MATRIX WILL ENTER AND HOW MANY ROWS AND COLUMNS IT HAS. MATRIX TYPES ARE SET UP FOR YOUR CONVENIENCE AND ARE:

0: ARBITRARY. EVERY ELEMENT MUST BE TYPED, ONE ROW AT A TIME.

1: SCALAR MATRIX. A MATRIX WHICH IS ZERO EVERYWHERE EXCEPT FOR THE ELEMENTS WHOSE ROW AND COLUMN INDICES ARE EQUAL. THESE ELEMENTS ALL HAVE THE SAME VALUE, THIS VALUE OF THE SCALAR WILL BE REQUESTED IF YOU SPECIFY TYPE 1. THESE MATRICES NEED NOT BE SQUARE.

2: SYMMETRIC. ONLY THE TOP TRIANGLE NEED BE TYPED. WHEN THE COMPUTER REQUESTS THAT YOU TYPE THE NEXT ROW, BEGIN WITH THE DIAGONAL ELEMENT. THESE MATRICES NEED NOT BE SQUARE, BUT MUST HAVE AT LEAST AS MANY ROWS AS COLUMNS.

3: COMPANION FORM. THESE MATRICES MUST HAVE AT LEAST AS MANY ROWS AS COLUMNS. THE 'TOP' SQUARE MATRIX WILL BE A COMPANION MATRIX, THE 'BOTTOM' A ZERO MATRIX. ONLY THE LAST ROW OF THE COMPANION MATRIX NEED BE TYPED.

4: RATIONAL MATRIX. MATRICES WHOSE ELEMENTS ARE RATIOS OF POLYNOMIALS. A ZERO POLYNOMIAL IS CONSIDERED TO HAVE DEGREE -1. ALL POLYNOMIALS ARE ENTERED IN ASCENDING DEGREE. THE HIGHEST DEGREE TERM MUST BE NONZERO.

* * * *

WHEN THE COMPUTER ASKS FOR MATRIX TYPE, #ROWS, #COLS; PLEASE TYPE THE MATRIX TYPE, ',', THE NUMBER OF ROWS, ',', AND THE NUMBER OF COLUMNS, 'RETURN'. IN ALL NUMERICAL DATA INPUT, AT MOST FOUR ELEMENTS, SEPARATED BY COMMAS, WILL BE ACCEPTED ON EACH TELETYPE LINE. IF MORE THAN FOUR ELEMENTS MUST BE TYPED, THE ADDITIONAL ELEMENTS MAY BE TYPED ON

SUBSEQUENT LINES. DECIMAL POINTS MUST BE USED. THE READ STATEMENT CANNOT BE USED TO REDEFINE A MATRIX IN STORAGE. USE DELETE (7.) AND THEN READ OR USE A NEW NAME. POLYNOMIALS SHOULD BE ENTERED AS VECTORS OF COEFFICIENTS IN ORDER OF ASCENDING DEGREE. THE COEFFICIENT OF HIGHEST DEGREE MUST BE NONZERO.

The machine now types

GIVE MATRIX NAME .

This message would have been typed immediately, without the intervening printout if the reply to the preceding request had been

1

rather than the

1,1

which we assumed.

In any case, we now type in any (not more than six) characters which will then be used as a name to call the matrix. (The maximum number of characters - here given as six - is dependent upon the machine and the program; six is a conservative figure. For more information see the programmer information in Chapter 3.)

For simplicity, suppose we call the matrix A . We must type

A .

From now on the matrix must be accessed by typing A in the same way; blanks are important. If b represents a blank, then

A

and

bA

are two different names. Terminal blanks are irrelevant however,

Ab,

A,

and

Abbbbb

are all the same name.

The computer now requests,

GIVE MATRIX TYPE, #ROWS, #COLS.

Matrix type refers to the numbers 0-4 above. This should be entered without a decimal point and followed by a comma.

The number of rows should be entered without a decimal point and followed by a comma.

The number of columns should be entered without a decimal point and followed by a return.

To enter a row 4-vector, then, one should type:

0,1,4 .

The computer would then request:

TYPE ROW 1 OF A .

The four coefficients of A should then by typed, with decimal points, separated by commas; for instance:

1., 1.0E-4, 1.0E10, 0.003 .

All matrix elements must be entered with decimal points and with at most four elements per type line. Thus if A were a row 3-vector one would have typed the coefficients as

1., 1.0E-4, 1.0E10 ,

whereas if A were a row 6-vector, one would have typed four on one row and two on the next:

1., 1.0E-4, 1.0E10, 0.003

2., 3. .

Polynomials of degree n should be entered as 1 by (n+1) vectors, the first coefficient entered being the constant term. The coefficient of highe t degree must be nonzero.

Some more detail is required on the matrix types. Type 0 is self-explanatory, and when this type is requested the computer will

successively ask the user to

TYPE ROW 1 OF A

TYPE ROW 2 OF A , etc.

When type 1 is requested, the computer types out

SCALAR = .

The user then types the value of the diagonal elements, e.g.,

3.14159 .

Notice that this option does not require that the matrix be square; it will define any dimension matrix, and the elements whose row index and column index are the same will have the value of the input scalar. This option is useful for entering the identity matrix or any dimension of zero matrix.

When type 2 is requested the computer will print

TYPE ROWS BEGINNING AT DIAGONAL .
TYPE ROW 1 OF A .

Action then proceeds as with the type zero option. Remember that only elements from the diagonal to the right need by typed. As an additional convenience type 2 matrices may have more rows than columns, the additional rows being filled with zeroes. For instance, to enter the matrix

$$A = \begin{bmatrix} 3 & 4 & 2 \\ 4 & 6 & 1 \\ 2 & 1 & 5 \\ 0 & 0 & 0 \end{bmatrix},$$

the conversation would go as follows

TYPE DICTIONARY ENTRY NUMBER

1

GIVE MATRIX NAME

A

MATRIX TYPE, #ROWS, #COLS

2, 4, 3

TYPE ROWS BEGINNING AT DIAGONAL

TYPE ROW 1 OF A

3., 4., 2.

TYPE ROW 2 OF A

6., 1.

TYPE ROW 3 OF A

5.

TYPE DICTIONARY ENTRY NUMBER

At this point, A would be in storage, complete.

When type 3 is requested, the computer will print

TYPE LAST ROW OF A

When these elements are typed, a companion form matrix will have been stored. As an additional convenience, type 3 matrices may have more than columns, the additional rows being filled with zeroes. For instance, to enter the matrix

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -5 & -6 & -3 \\ 0 & 0 & 0 \end{bmatrix},$$

the conversation would go as follows

TYPE DICTIONARY ENTRY NUMBER

1

GIVE MATRIX NAME

B

MATRIX TYPE, #ROWS, #COLS

3, 4, 3

TYPE LAST ROW OF B

-5., -6.0, -3.

TYPE DICTIONARY ENTRY NUMBER

At this point B is in storage.

To illustrate type 4, consider the scalar matrix

$$S = \frac{1 + 2s + 2s^2}{3s + 4s^2 + s^3 + s^4} \quad .$$

The conversation would go as follows.

TYPE DICTIONARY ENTRY NUMBER

1

GIVE MATRIX NAME

S

MATRIX TYPE, #ROWS, #COLS

4, 1, 1

DEG NUM 1 1

2

TYPE NUM COEFFS

1., 2., 2.

DEG DEN 1 1

4

TYPE DENOM COEFFS

0., 3., 4., 1.

1.

TYPE DICTIONARY ENTRY NUMBER

If the matrix has more than one element, input type 4 will continue down the first column in this fashion, requesting element 2,1 next, then down the second, etc. Notice that while all other input has been by row, this option requires input by column. The reasons for this are explained in Chapter 3.

These are the five input options available at present.

1.2 Output

Instruction #2 enables the user to write out matrices in storage. As with input, matrices with scalar elements are printed by rows, matrices of rational fractions are printed by columns. To obtain output, type 2 when the entry number is requested. Because of the simplicity of this instruction, no information printout is given, that is,

2, 1

and

2

produce the same computer response.

1.3 Constant Matrix Arithmetic

Instruction #3 adds two matrices, instruction #5 subtracts two matrices, instruction #6 multiplies two matrices. In each case a name must be supplied for the output matrix. #3 and #5 may be replacement operations, that is, the result may be stored in place of one of the arguments. #6 may not be used in this manner.

In these and other instructions, the output matrices can be defined implicitly; they need not have been previously defined by the READ instruction.

To form

$$C = A + B ,$$

$$C = A - B ,$$

or

$$C = AB ,$$

the conversations would go as follows:

TYPE DICTIONARY ENTRY NUMBER

3	or	5	or	6
ADD		FROM		MULTIPLY
A		A		A
TO		SUBTRACT		AND
B		B		B

INTO	INTO	INTO
C	C	C

TYPE DICTIONARY ENTRY NUMBER

At this point the operations have been performed and C is available for printout or further computation.

1.4 Generalized Inverse

Instruction #4 computes the generalized inverse of the input matrix. In addition it tells which columns of the matrix are linearly independent, prints the rank, and the volume of the parallelipiped determined by the linearly independent columns.

Briefly, the generalized inverse of an $m \times n$ matrix A is the matrix $A^{\dagger} = A^{-1}$. If A is of maximal rank then either $(A'A)^{-1}$ or $(AA')^{-1}$ is defined and $A^{\dagger} = (A'A)^{-1}A'$ or $A^{\dagger} = A'(AA')^{-1}$.

The volume mentioned above is the absolute value of the determinant for nonsingular matrices. For example, letting (A) be the volume, when

$$A = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad (A) = \sqrt{2}$$

the one dimensional volume or length.

When

$$A = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}, \quad v(A) = \sqrt{2} ,$$

the length of the linearly independent vector.

When

$$A = \begin{bmatrix} -1 & 1 \\ -1 & 2 \end{bmatrix}, \quad v(A) = 1 ,$$

the absolute value of the determinant.

At the completion of this instruction, the computer will print

TOL = 1.0E-6 , WANT TO CHANGE TOL?

If the user answers yes, the computer will accept a new value for TOL . TOL is used as a zero tolerance and for most single precision uses should be about 10^{-6} . A number is considered zero is the ratio of that number to the largest number in computations leading to it is less than TOL . TOL is used to test linear independence of vectors and the degree of polynomials (by testing highest degree of coefficients). Decreasing TOL will tend to make matrices have higher rank, spaces have higher dimension, polynomials have higher degree, rational fractions less reducible, and, if carried too far, errors have greater magnitudes.

1.5 Administrative Instructions

Instruction #7 deletes a matrix from storage, thus freeing the name to be used again. Remember that Read, #1, cannot be used to redefine a matrix.

Instruction #9 copies a matrix into another area under another name, thus giving two copies of the matrix.

Instruction #11 allows an individual matrix element to be accessed and changed. This is very convenient when correcting input errors or when changing parameters.

Instruction #14 prints out a catalog of all matrices in storage, giving their name and dimensions.

1.6 Trace and Transpose

Instruction #8 forms the transpose of a matrix. This cannot be done as a replacement.

Instruction #10 computes the trace of a matrix and prints it out.

1.7 Polynomial Root Solver

Instruction #12 calculates the roots of a polynomial. The polynomial must be stored as a vector of dimension one greater than its degree, in order of increasing degree. The highest degree

coefficient must be nonzero. The roots of an nth degree polynomial will be stored in an n x 2 array with real points in the first column, imaginary parts in the second column.

1.8 Orbit

Given an n x r matrix B and an n x n matrix A, this instruction forms the matrix

$$Q = [B, AB, \ldots, A^{n-1}B] .$$

The conversation is as follows:

ORBIT OF

B

UNDER

A

INTO

Q

1.9 Canonical Form

Instruction #15 computes a Luenberger Canonical Form [D. G. Luenberger, Canonical Forms for Linear Multivariable Systems, IEEE Transactions or Automatic Control, June 1967] which attempts to

make each of the block companion matrices have the same size. In the single-input case, the canonical form becomes the usual companion form with coefficients on the last row.

The program first asks for a generating matrix

$$Q = [B, AB, \ldots, A^{n-1}B]$$

then for the system matrix A, and then for a name under which to store the transformation matrix S such that SAS^{-1} is in canonical form.

1.10 Matrix Juxtaposition

Given an $n \times r$ matrix A and an $n \times m$ matrix B, instruction #16 forms the $n \times (r+m)$ matrix $[A,B]$.

1.11 Matrix Exponential

Given a square matrix A and a time T, this instruction forms the exponential and integral exponential matrices

$$e^{TA} \quad \text{and} \quad \int_0^T e^{\tau A}\, d\tau \quad .$$

Taylor series expansions are used to compute these matrix functions. For this reason, there is an upper limit to the value of T which can be used to compute the function accurately directly from the series. The program computes this limit and if the input

T is larger, halves T until it is less than the limit, computes the functions for this smaller T and from them computes the functions at the required T value. There is also an option by which the program will use the upper limit T . This is useful for investigating infinite-time or steady-state conditions. A complete description of the analysis involved in this instruction may be found in Chapter II, Reference 3.

The conversation goes as follows:

EXPONENTIAL OF

A

INTO

ETA

INTEGRAL EXPONENTAL INTO

IETA

SHOULD I USE LARGEST TIME STEP?

YES or NO

0.5

TYPE DICTIONARY ENTRY NUMBER

1.12 Matrix Reduction

Instruction #18 takes an $n \times m$ matrix A of rank r and computes an $n \times r$ matrix V with orthonormal columns such that range (A) = range (V) .

1.13 Vector Space Operations

Given a space, that is, an $n \times r$ matrix V with orthonormal columns, instruction #19 computes the orthogonal complement of range (V) . That is, an $n \times (n-r)$ matrix P with orthonormal columns such that

$$P' V = 0 .$$

Given spaces V_1 and V_2 , instruction #20 computes a space $V_3 = V_1 \cap V_2$.

Space here always refers to a matrix which has orthonormal columns, that is,one which has been operated on by #18.

1.14 Polynomial Multiplication

Instruction #21 multiplies two polynomials which are stored as described in 1.7 , instruction #12, polynomial root solver.

1.15 Determinant of Rational Matrix

Instruction #22 computes the determinant of a matrix of rational fractions. The determinant is stored as a rational fraction.

1.16 Rational Arithmetic

Instructions #23, #24, #29, and #30 add, subtract, form the conjugate transpose $(A'(-s))$, and multiply matrices whose elements are ratios of polynomials in a single variable.

1.17 Polynomial Arithmetic

Instructions #25, #26, #27, #28, and #31 perform polynomial division, highest common divisor, subtraction, addition, and least common multiple. Polynomials of degree n are assumed to be stored as $1 \times (n + 1)$ matrices.

1.18 Spectral Factorization

Given a rational matrix A such that $A'(-s) = A(s)$, the program determines a matrix $H(s)$ such that $H'(-s)H(s) = A(s)$. This is done by obtaining $\Delta(s)$ and $\overline{A}(s)$, the polynomial and the polynomial matrix such that $A(s) = \Delta^{-1}\overline{A}$. We then compute $\overline{H}(s)$ such that $\overline{H}'(-s)\overline{H}(s) = \overline{A}(s)$ and $h(s)$ such that $h(-s)h(s) = \Delta(s)$.

* * * * * *

This is the dictionary available at present.

CHAPTER 2

APPLICATIONS, LIMITATIONS, AND EXTENSIONS

This program provides a framework into which any set of operational subroutines can be put and then easily called for use.

With linear systems analysis being the field of interest, we have incorporated all the operations of matrix and vector algebra, some operations with vector spaces, the matrix exponential, and some operations with matrices over the field of rational fractions. To complete the work properly, one would like to have available instructions which could solve the decoupling problem or the quadratic optimization problem by means of one operation. At present these can be done by the user, but only by iteration of instructions.

There are really very few limitations on the program. At present we are limited to 50 matrices in memory (limitations imposed by dimensions of MTNMS and ISTPLC) but this does not seem to be a serious constraint. We are limited in input-output to matrices with 20 columns and polynomials of degree 19 (limitations imposed by the dimension of Y). We are also limited in the total storage of numbers by the dimension of X . These easily changed restrictions are the only size constraints.

At present the Fortran Source Program requires about 50000 characters. On the PDP-10 at ERC this is a reasonably small program and we have used the system with no problems. On the Honeywell 416/516

system we have had to divide the program into two links. The first link contains the first 16 instructions and the second link has the rest. These distinct programs call each other from the disk as required by the user. This has worked very satisfactorily.

There are some implicit limitations caused by the fact that the program is written in single precision. The subroutines will proceed formally regardless of matrix dimensions. However some matrix operations - in particular inversion - are sensitive to numerical problems which depend upon dimension. Depending upon the number of digits carried by the given computer in a single precision floating point word, the critical dimension varies; however, with 27-bit significance, there are poorly conditioned systems of order eight which will give inaccurate results.

To assess the influence of numerical errors it is advisable, whenever possible, for the user to do his computation twice, changing his methods or at least modifying the order in which operations are done. This will test the stability of the results and give a measure of their accuracy.

An example of this is the problem of finding eigenvalues. At present there is no eigenvalue routine, but eigenvalues can be found easily if a cyclic vector is known. Since the procedure then involves a matrix inversion, it is advisable to perform the analysis with two different cyclic vectors and compare the resulting eigenvalues. The conversational mode of operation is well suited to this and such a procedure is a sound engineering practice under all circumstances.

Of course it would be possible to change to a double precision mode. This appears simple because in the main program there are very few floating point variables (X,Y,Z1,TOL) . However to do this the subroutines would have to be examined very carefully.

Everyone can suggest additional instructions for inclusion, of course, and since these depend upon personal interest, none will be suggested here. There are, however, a number of possible non-operational changes which would increase the value of the program.

There should be a capability of reading and saving matrices on disk files. This will enable the program to accept input from other programs without typing. In addition it will provide a margin of safety. In case the user is thrown off the computer, his intermediate output will still be available on disk.

The conventions by which matrices are stored, either having constant elements or rational fraction elements, are sound; however, there is at present not enough flexibility in going from one mode to another. At the very least, one should be able to make a constant matrix into an RF matrix, a vector (polynomial) into a companion matrix or a rational fraction, and vice versa. Along the same lines it is unnecessary to have an RF add (#23) and a constant add (#3). Because of the code it is possible to include these in the same instruction. Mixed mode capability does not seem to be an immediate requirement, however the capability of mode change is. For instance we store polynomials simply as vectors but the sum of two vectors is very different from the sum of two polynomials, and this must be accounted for in the formalism.

The aforementioned are fairly simple changes to implement. Beyond them is a large, fairly vague area involved with making the program easier to use. There are a few concrete suggestions about how this might be done.

It seems essential that conversation with the computer be simplified and that iterative operations be allowed. Iterative operations will require that the list structure allow a new matrix to be inserted in an old area regardless of size. This in retrospect is probably desirable anyway. Iterative computation can then be implemented by special delimiters to isolate a block of instructions which is saved and repeated.

To simplify the conversation, we should as a first step, punch the dictionary entry, a delimiter, and the first argument on one line. Thus we might type

```
DELETE: A
```

or even

```
DELETE: A : B .
```

This would not be too difficult to mechanize and since it would cut out at least one computer response, would speed up activity considerably.

CHAPTER 3

PROGRAM STRUCTURE

The entire program has been written if Fortran IV and is currently in use on both a PDP-10 timesharing system at ERC and on a Honeywell 416/516 system through a commercial timesharing system in Silver Spring, Maryland.

Because the Honeywell system is in some ways more representative of timesharing operations, we will use excerpts from it for illustration. This chapter will start at the beginning of the program and proceed with execution, stopping for asides to explain the structure as questions arise. Listings are not included in this report because they are too long; copies of the program may be obtained from COSMIC, the NASA program depository at the University of Georgia.

The program starts:

* * * * *

```
1     DIMENSION X(500)
2     DIMENSION ISTPLC(50)
3     DIMENSION MATNAM(3),MATNM1(3),MATNM2(3),MTNMS(150)
4     COMMON/COMM1/MTNMS,NUMMAT,ISTPLC,NEWSTR,NUMRS,NUMCLS,X
5     COMMON/COMM2/IYES,IHELP,IPU,NDICEN,TOL
6     COMMON/COMM3/NAVSTR
7     IPU=9
8     TOL=1.0E-6
```

```
 9     NUMMAT=0
10     NEWSTR=1
11     CALL INFO(2,3)
12     READ(IPU,24)IHELP
13     IYES='YE'
14 24  FORMAT(3A2)
15     IF(IHELP.EQ.IYES)CALL INFO(4,37)
16  5  WRITE(IPU,6)
17  6  FORMAT(30H TYPE DICTIONARY ENTRY NUMBER)
```

* * * *

The purpose of these instruction is to initialize certain locations and to get the user into operation. IPU is set to the Fortran unit number for the teletype; this changes at different installations. TOL is set to 10^{-6} (see Chapter 1, 1.4). NUMMAT , the number of matrices in storage, is zeroed. NEWSTR , the location where the next matrix will be stored, is set to one.

3.1 Matrix Storage

All matrix data is stored in the singly-dimensioned array X . NEWSTR tells the component of X in which the next matrix to be defined will start.

Constant matrices are stored in X columnwise, preceded by their dimensions. That is if a new, m x n matrix A is defined either by the Read instruction, #1, or implicitly by an operation,

then we have

$$X(NEWSTR) = m$$

$$X(NEWSTR+1) = n$$

$$X(NEWSTR+2) = a_{11}$$

$$\vdots$$

$$X(NEWSTR+m) = a_{1m-1}$$

$$\vdots$$

$$X(NEWSTR+mn+1) = a_{mn}$$

Thus far we have had no problems in carrying fixed point numbers in floating point, however we sometimes take precautions when converting back to fixed point, e.g.

$$L = X(I) + 0.1 \ .$$

A matrix whose elements are rational fractions is handled in much the same way, except that the first element contains a code. For instance if the new matrix were the rational fraction

$$\frac{a_0 + a_1 s}{b_0 + b_1 s + b_2 s^2} \ ,$$

then it would be stored as

X(NEWSTR) = 10.5	Storage +0.5
X(NEWSTR+1) = 1	Number of rows
X(NEWSTR+2) = 1	Number of columns
X(NEWSTR+3) = 2	Numerator degree + 1
X(NEWSTR+4) = a_0	
X(NEWSTR+5) = a_1	
X(NEWSTR+6) = 3	Denominator degree + 1
X(NEWSTR+7) = b_0	
X(NEWSTR+8) = b_1	
X(NEWSTR+9) = b_2	

Here the 10.5 gives the storage required, 10 locations, and the code 0.5 saying that the matrix is a rational matrix. It is essential to remember that the first location of a rational matrix must be set up after the matrix itself has been stored.

That part of X past NEWSTR , that is, beyond the storage being used, is used as working area by the program and subroutines. Therefore the capabilities of the program in terms of the size of matrices handled can be increased merely by increasing the dimension of the array X .

At present the number of matrices handled is limited to 50 by the dimensions of ISTPLC(50) and MTNMS(150) .

Each matrix name is allowed three words of storage. The list of all names is kept in the array MTNMS . To find a matrix,

comparison is made with MTNMS . When a match is found with the kth name, then the matrix is stored starting at X(ISTPLC(k)) .

* * * * * *

The call to INFO prints out lines two and three of the information file asking if the user needs help. The reply is read and compared with IYES . Finally the program asks for the dictionary entry number to be read. Depending upon the number (1-24) control is transferred to the proper area. At the end of an instruction control returns to external formula number 5.

The discussion above concerning how matrices are stored tells practically all the specialized knowledge required for additional programming. However it is helpful to understand the actual mechanism of executing an instruction. For this purpose we will give the code for Add, instruction #3, and explain its purposes.

3.2 Instruction Execution

* * * *

```
1C  NDICEN IS 3, INSTRUCTION IS ADD
2 30 Z1=1
3    WRITE(IPU,83)
4 83 FORMAT(4H ADD)
5    READ(IPU,24)MATNAM
6    WRITE(IPU,84)
7 84 FORMAT(3H TO)
```

```
 8  9 READ(IPU,24)NATNM1
 9    WRITE(IPU,61)
10 61 FORMAT(5H INTO)
11    READ(IPU,24)MATNM2
12    CALL IMANUM(MATNAM,1,I)
13    IF(I.LT.0)GOTO 5
14    CALL IMANUM(MATNM1,1,J)
15    IF(J.LT.0)GOTO 5
16    L=ISTPLC(I)
17    NUMRS=X(L)
18    NUMCLS=X(L+1)
19    M=ISTPLC(J)
20    I1=X(M)
21    IF(NUMRS.EQ.I1)GOTO 34
22 31 FORMAT(1H,3A2,fH AND,3A2,21H ARE NOT COMFORMABLE.)
23 35 WRITE(IPU,31)MATNAM,MATNM1
24    GOTO 5
25 34 I1=X(M+1)
26    IF(NUMCLS.NE.I1)GOTO 35
27    CALL IMANUM(MATNM2,0,K)
28    IF(K.LT.0)GOTO 5
29    N=ISTPLC(K)
30    CALL ADD1(X(L),1.,X(M),Z1,X(N))
31    GOTO 5
```

* * * *

This code forms

$$C = A + Z1*B$$

where Z1 is a scalar,
so Z1 is set to one (2).

The computer prints the word ADD (3) and accepts the name of the first argument (5). The computer prints the word TO (6) and accepts the name of the second argument (8). The computer prints the word INTO (9) and accepts the name under which A + B is to be stored (11).

A call to IMANUM(MATNAM,1,I) will set I equal to the ordinal number in storage of the matrix whose name is contained in MATNAM (12). If that name is not found, I is set to -1, and execution of the instruction is terminated by a transfer to 5 (13). In (14) and (15) the same operations are performed on the second argument.

These are constant matrices, so we know that the number of rows of A is contained in X(L) (17) where L = ISTPLC(I) (16) (these remarks follow from the description in 3.1 of matrix storage). Also the number of columns is contained in X(L+1) . The number of rows in the second argument is obtained and checked, as is the number of columns. If the matrices do not have the same dimensions, execution is deleted with a transfer to 5 .

A call to IMANUM(MATNM2,0,K) will set K equal to the ordinal number of MATNM2 . The zero indicates that MATNM2 need not have been previously defined but can be implicitly defined here. The operation of IMANUM is described more fully in Appendix II.

Finally the call to ADD1 forms the sum

$$C = A + Z1*B$$

and stores the result beginning at X(N) .

With more or less complexity, all other instructions perform their appropriate operations. An attempt has been made to confine the main program to the administrative details and handle the mechanics in subroutines. For instance in the instruction listed above, #3, the actual addition is done by ADD1 , a subroutine which expects to find the matrix dimensions stored in the first two locations.

3.3 General Commentary

The method we have used for storing matrices - by column, in a singly-dimensioned array - is not new and has certain advantages even in normal Fortran programming if storage is at a premium and various dimensions will be used in a main program - which must have fixed dimensions. However our motivation for doing it was solely to have a single storage area to which matrices could be

added indefinitely. It still seems that this is the only way in which this end could be achieved.

This method entails some inconvenience in accessing elements, requiring index computation that the FORTRAN compiler could handle automatically in a doubly-indexed array. However it serves its purpose and at the same time is completely compatible with normal Fortran matrix routines, provided variable dimensioning is used to communicate to the subroutine that the storage is packed. That is, the matrix and the array in which it is stored have the same dimensions. For instance the generalized inverse routine which we use was written using doubly dimensioned arrays, but required no changes to incorporate.

At present, matrices may be stored in previously defined areas only if the storage requirements are the same for the new and old matrices. They may not be stored if the new matrix requires less storage or more storage than the old matrix. The mechanism by which this is done is contained in the writeup for IMANUM in Appendix II. There are certain disadvatages to this procedure if the program is extended to iterative operations. At present, however, it is not a serious restriction.

Before calling IMANUM when setting up a new matrix of rational fractions, it is essential that NUMRS be set to one and NUMCLS set to (the total storage required by the matrix minus two).

This is a device to enable IMANUM to define the proper amount of storage.

In most subroutines dealing with rational fractions, the output array must be extendable for working area. For instance in the subroutine which reduces to lowest terms,

RFREDE(A,TOL) ,

it is assumed that the locations in A following the rational fraction may be used freely for working space. This procedure was followed because of our method of matrix storage.

CHAPTER 4

REFERENCES

1. W. M. Wonham and A. S. Morse, Decoupling and Pole Assignment in Linear Multivariable Systems: A Geometric Approach, NASA ERC PM-66, October, 1968; SIAM J. Control, v. 8, no. 1, Feb. 1970, pp. 1-18.

2. W. A. Wolovich and R. S. Shirley, A Frequency Domain Approach to Handling Quality Design, to be presented at 1970 Joint Automatic Control Conference.

3. R. E. Kalman and T. S. Englar, A User's Manual for ASP-C, NASA CR-475, June, 1966.

GLOSSARY

Fixed Meaning

IHELP — Used to receive user replies to yes/no questions and to receive the code telling whether informational material should be printed.

IPU — Fortran unit number for teletype.

ISTPLC — A 50-dimensional array. The k^{th} entry gives the location in X of the initial word of the k^{th} matrix.

ITYPE — Matrix input type.

IYES — Holds the word YES to check user replies.

MATNAM, MATNM1, MATNM2 — Three dimensional arrays used to hold matrix names.

MTNMS — A 150-dimensional array in which the matrix names are stored, 3 words per name.

NAVSTR — Tells where the next location of free, available storage is in X. Between instructions, NAVSTR=NEWSTR. This variable is only used in complicated instructions where a large number of working areas must be defined.

NDICEN — Used to receive the dictionary entry number.

NEWSTR Location in X where next matrix will be stored.

NUMMAT Number of matrices in storage.

NUMRS,

NUMCLS Usually used to hold the dimension of a matrix. Used in IMANUM to check space requirements.

TOL Relative zero tolerance, initialized to 10^{-6}.

X A 2000-dimensional array. All matrices are stored here. Most scratch work is done in X.

Y A 20-dimensional variable used for intermediate storage (buffering) of input and output.

Z1 Free floating point variable for internal use.

APPENDIX I

REFERENCE DATA

A. The Following Instructions are Available

1. Enables the User to Enter a Matrix from the Terminal.
2. Enables the User to Look at his Results.
3. Adds Two Matrices.
4. Computes the Generalized Inverse of a Matrix.
5. Subtracts Two Matrices.
6. Multiplies Two Matrices.
7. Deletes a Matrix from Storage.
8. Forms the Transposed Matrix.
9. Copies a Matrix into Another Storage Area, Giving It Another Name.
10. Computes the Trace of a Matrix.
11. Prints Out the m,n element of a Matrix and Enables User to Alter It.
12. Finds the Roots of a Polynomial.
13. Computes the Controllability Matrix.
14. Lists Matrices Presently Stored.
15. Finds, for (A,B) the Transform to Luen Form 1
16. Forms Matrix (A,B) from A and B.
17. Forms Matrix Exponential.

18. Given an n×m Matrix of Rank r , Produces an n×r Matrix with Orthonormal Columns. Such a Matrix Will be Called a Space.

19. Given a Space, Finds the Orthogonal Complement.

20. Given Two Spaces, Finds Their Intersection.

21. Given Two Polynomials, Computes Their Product.

22. Given A Matrix Whose Elements are Ratios of Polynomials, Finds Its Determinant.

23. Given Two Matrices Whose Elements are Rational Fractions, Finds Their Sum.

24. Given Two Matrices Whose Elements are Rational Fractions, Finds Their Difference.

25. Given Two Polynomials, Finds Their Quotient and Remainder.

26. Given Two Polynomials, Finds Their Highest Common Divisor.

27. Given Two Polynomials, Finds Their Difference.

28. Given Two Polynomials, Finds Their Sum.

29. Forms the Conjugate Transpose A'(-s) of a Rational Matrix A(s) .

30. Given Two Rational Matrices, Finds Their Product.

31. Given Two Polynomials, Finds Their Least Common Multiple.

32. Given a Hermitian Rational Matrix, Performs a Spectral Factorization.

B. General Facts

1. Read (#1) cannot redefine a matrix.

2. Matrices can be defined implicitly as results of operations.

APPENDIX II

SUBROUTINES

1. NEWMAT (MATNAM)

Stores the contents of MATNAM at the end of the list of names in MTNMS. Increases NUMMAT by one, updates NEWSTR and NAVSTR , and stores in ISTPLC the location of MATNAM.

In other words it handles the bookkeeping required for placing a new matrix in the memory.

2. IMANUM (MATNAM,J,I)

If J is 1, the name in MATNAM must have been previously defined; that is, the matrix is a required argument. If J is not equal to 1 then the matrix is output and may be put either in an old storage area or a new storage area.

I gives the ordinal number of MATNAM in the storage list. If J equals 1, but MATNAM is not found, a message will be printed and the execution deleted. If J equals zero a new storage area will be set up.

If J equals zero and MATNAM is already in memory, storage requirements are checked for consistency.

3. GINV2(A,U,AFLAG,ATEMP,MR,NR,NC,TOL)

This program computes the generalized inverse $A^{\dagger}$ of A and stores $A^{\dagger}$ in the area A . The subroutine itself prints out the

rank, which columns are linearly independent, and the volume of the parallelpiped formed by the linearly independent columns.

The program uses a modification of the Gram-Schmidt orthogonalization procedure.

MR is the first dimension number of the area in which A is stored. NR is the number of rows in A , NC is the number of columns in A . Because of the way in which this program stores matrices, we always have MR=NR .

TOL is a zero tolerance. The length of a column vector is computed after and before orthogonalization. If the ratio of these numbers is less than TOL , then that vector is considered to be linearly dependent, i.e. zero after orthogonalization.

4. DOT(X,Y,NR,FAC)

This subroutine is used by GINV2 . It takes the inner product of the NR-vectors X and Y and stores the result in FAC .

5. MULT(X,M,N,M1,NUMRS,I1,NUMCLS)

This subroutine forms the product of two matrices.

The first matrix, A , is NUMRS by I1 and a_{11} is stored at X(M) . The second matrix, B , is I1 by NUMCLS and b_{11} is stored at X(N) . The product AB is stored starting at X(M1) .

6. MULLER(COE,N1,ROOTR,ROOTI)

This subroutine finds the roots of the polynomial of degree N1 whose N1+1 coefficients are stored, in order of increasing

degree, in COE . The N1 roots are stored, real parts in ROOTR , imaginary parts in ROOTI .

7. INFO(I,J)

Prints out the I^{th} through Jth lines inclusive of an information file. Printing of line-terminating blanks is suppressed.

8. TRANP(A,B)

This is the first of several subroutines which are specifically designed for the matrix storage procedure used in this program. The subroutine forms the transpose of A . The dimensions of A are in A(1) and A(2) , with the elements following columnwise. The dimensions and elements of the transpose are stored in B .

No attempt is made to check that B has sufficient storage area.

9. SETSC(A,M,N,Z1)

Stores M in A(1) , stores N in A(2). In A(3) to A(2+M*N) are stored the elements of an M×N matrix A such that

$$A_{ij} = Z1*\delta_{ij} .$$

10. ADD1(A,X,B,Y,C)

X and Y are scalars. The matrix C is constructed:

$$C = X*A+Y*B .$$

Dimensions are not checked, but we set C(1) = A(1) , C(2) = A(2) .

11. MULT1(A,B,C)

C = A*B .

C(1) = A(1) , C(2) = B(2) , dimensions are not checked.

12. COPY (A,B)

A(1) through A(2+A(1)*A(2)) are copied into B(1) through B(2+A(1)*A(2)) .

13. JUXTC(A,B,C)

A and B are A(1) × A(2) and B(1) × B(2) matrices. C is formed as the A(1) × (A(2)+B(2)) matrix

[A,B] .

14. MEXP(A,T,AE,AIE,B,D,E,AN)

This subroutine computes the exponential matrix

AE = exp(T*A)

and the integral exponential

$$AIE = \int_0^T \exp(Z*A)d\tau \quad .$$

AN is the norm of the matrix A (see subroutine 15 below).

As usual, the dimension n of A is obtained from A(1) and this number is stored in AE(1), AE(2), AIE(1), and AIE(2).

B, D, and E are respectively n^2, n, and n dimensional working areas.

The exponentials are computed by the Taylor Series. In order to do this with good accuracy, certain precautions must be taken. The entire procedure is analyzed in NASA CR-475.

15. MNORM(A,X)

Computes the norm X of the matrix A.

$$\|X\| = \min\left\{\max_i \sum_j |a_{ij}| \ , \ \max_j \sum_i |a_{ij}|\right\} \ .$$

16. PERP(A,B)

Given a matrix A with orthonormal columns, the matrix B is defined having maximal rank, with orthonormal columns, such that

$$B' A = 0 \ .$$

Thinking of A as defining a space, B defines the orthogonal complement of A.

The output of this subroutine can be a matrix which is $n \times 0$; this represents the space spanned by the zero vector.

17. REDUCE(A,B,NUMRS,NUMCLS,TOL,M)

This subroutine takes the NUMRS by NUMCLS matrix A and determines its rank M. It then forms the NUMRS by M matrix B

having the same column space as M but having orthonormal columns.

18. PDIV(Q,P,A,R,TOL)

Divides polynomial Q by polynomial P , constructing the quotient A and the remainder R .

$$Q = A*P + R .$$

The highest degree coefficient r_h of R must satisfy

$$r_h > TOL*Z1$$

where

$$Z1 = \max \{\text{absolute value, coefficients of } Q\} .$$

The degree of R is reduced until this condition is satisfied.

The degree of a polynomial is contained in the first element of its storage area. As usual, the degree of R is strictly less than the degree of P .

19. PGAUSA(A,C,B,IR,NUM,TOL)

A is an IR-vector of rational fractions. Storage convention requires that rational fractions be stored in conformity with the convention for polynomial storage; that is, polynomials are stored in order of increasing degree, highest degree terms are nonzero, and polynomials are preceded in storage by their degree plus one. Therefore the vector will be stored

NUM 1,1 DEG

1,1 SUM COEFF

DENOM 1,1 DEG

1,1 DENOM COEFF

NUM 2,1 DEG

2,1 NUM COEFF

etc.

B is an (IR+1) by IR matrix of rational fractions stored columnwise as described above.

TOL is, as usual, the zero tolerance required by the polynomial addition routine.

This subroutine performs gaussian elimination to eliminate the first column of B . It is assumed that the missing first element of the vector A is one. The IR by IR matrix which is the result of this elimination is stored according to the same convention, beginning at C(1) .

NUM is the first unused location in C .

20. PADDB(B,X,C,Y,D,TOL)

X and Y are scalars, B and C are polynomials stored according to our convention. This subroutine forms D = X*B + C*Y .

TOL is the zero tolerance, used when deg B = deg C .

21. PMULT(B,A,C)

This subroutine computes the product

$$C = B*A$$

of polynomials. All polynomials are stored in our conventional way.

22. MULTIN(A,B,C)

A and B are rational fractions stored in our conventional way. This subroutine forms

$$C = A^{-1}*B .$$

23. PCOPY(A,B)

Copies the polynomial A into the polynomial B .

24. RATADC(A,X,B,Y,C,TOL)

This subroutine forms the sum

$$C = X*A + Y*B$$

where X and Y are scalars and A and B are rational fractions.

25. HCDD(A,B,C,TOL)

This subroutine finds the highest common divisor C of the polynomials A and B .

26. RFREDE(A,TOL)

Destructively reduces the rational fraction A to lowest terms.

27. CTRFMH(A,B)

Puts the conjugate transpose of the rational matrix A into B . A(1) contains the number of storage locations for A plus one half. Standard storage for RF matrix.

28. CCRF(A,B,M)

Copies the conjugate of the rational fraction A into B . A(1) contains degree of numerator A plus one. Standard storage for rational fraction.

M tells number of items stored. M = deg Num + deg Denom + 4 .

29. DMRFG(A,I,J)

J is the number of storage locations which must be skipped to find the I^{th} rational fraction in A after the one beginning at A(1) .

30. RFCPO(A,B)

Copies the rational fraction A into B .

31. RFMMI(A,B,C,TOL)

Forms the product AB of the rational matrices A and B , placing the result in C .

32. RFMJ(A,B,C)

Forms the product AB of the rational fractions A and B, placing the result in C.

33. PLCMQ(A,B,C,Y,X,TOL)

Computes the least common multiple C of the polynomials A and B. Y and X are scratch areas.

34. RMCDR(A,Y,X,TOL)

Finds a common denominator Y for the rational matrix A. X is a scratch area.

35. RMTPMS(A,Y,X,M,TOL)

This converts the rational matrix A to a polynomial matrix X. Y is input, the common denominator for A. They $Y^{-1}X$ (if reduced) would be A. All denominators of A need not by Y. M - 1 is the highest degree appearing in X.

36. PMCOPT(A,B,C)

Puts the polynomial matrix A into the array B = C. A is stored as follows

A(1) = # elts in A + 0.5

A(2) = # rows

A(3) = # cols

A(4) = 1 + deg $p_{11}(s)$

$A(5) = p_{110}$

$A(6) = p_{111}$

etc.

B is stored in a $3 \times 3 \times 21$ array for use in the TUEL program described below.

C must be the same as B in the calling sequence internally one is considered a vector and the other a $3 \times 3 \times 21$ array - purely for convenience.

37. SPECDU(ALPHA,GAMMA,DELTA,BETA,WORK,IRAY,IR,NP,IEND,EPS,LGOUT, TEMP,YTEMP)

This is the subroutine obtained from W. G. Tuel's executive program for spectral factorization (Computer Algorithm for Spectral Factorization of Rational Matrices, W. G. Tuel, Jr., IBM Journal, March 1968, pp. 163-170).

38. S2Z(ALPHA,BETA,MD,NP,IR,JH,JTEM,JWORK,IRAY,NPOW)

Performs the transformation from s-plane to z-plane (discrete time).

39. RICC(Q,Y,MD,IR,IEND,EPS,Y)

Solves a discrete Riccati equation.

40. WWT(X,WT,MD,M,IR,IFLAG,WK,TEMP)

Solves for the W matrix which factors the Riccati solution.

41. Z2S(A,B,MD,MIR,JH,JTEM,JWORK,NPOW)

Returns to s-plane from z-plane.

42. PDINV(A,AI,M,N,DET)

Matrix inversion routine.

43. CZECH(BETA,ALPHA,MD,M,IR,FLFX1,FLFX2,PROD)

Checks answer by attempting to compute the input matrix from its factor.

END