# AN EXPANDED SIMULATION LANGUAGE FOR PARTITIONED SYSTEMS

by

William Randle Moore

---

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 7 2

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Wm. R. Moore

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

John V. Wait

JOHN V. WAIT
Professor of Electrical Engineering

Sept. 22, 1971
Date

## ACKNOWLEDGMENTS

The project described in this thesis is part of a continuing study of on-line digital simulation directed by Professor Granino A. Korn. The writer is indebted to Professor Korn and to Professor John V. Wait, who suggested the topic, for their guidance and for providing an ideal atmosphere in which to work. The writer is grateful to the National Science Foundation for support of this study under NSF Grant GK 15224. Thanks are also due to Dr. R. H. Mattson, Head, Electrical Engineering Department, and Dean Fahey, of the College of Engineering for their contribution of University facilities.

TABLE OF CONTENTS

## LIST OF ILLUSTRATIONS

LIST OF TABLES

## ABSTRACT

DARE1X is an expanded version of the DARE I simulation language developed under project DARE (Differential Analyzer REplacement) at the Computer Science Research Laboratory, Department of Electrical Engineering, The University of Arizona. Like previous DARE languages, DARE1X provides on-line continuous-system simulation on a medium-sized general-purpose digital computer. In addition, DARE1X provides a convenient test environment for the development of new numerical integration methods and simulation techniques. In particular, special techniques for fixed linear system simulation have been implemented which will also permit future development of techniques for simulation of systems that can be partitioned into a fixed linear segment and a non-linear and/or time-varying segment. Sampled-data system simulation is also possible. Several minor "bugs" in the original DARE I have been corrected, operation from a fast-access disk has been implemented, and program storage on DECtape has been added.

DARE1X is written for a Digital Equipment Corporation PDP-9 with Extended Arithmetic Element, Automatic Priority Interrupt, 16K of core memory, an RS-Ø9 disk, one DECtape transport, and a specially designed simulation console.

# CHAPTER 1

## INTRODUCTION

The digital simulation language described here is an extension of a system originally developed as DARE I by J. R. Goltz (1970). DARE I was the first in a series of interactive digital simulation languages in project DARE (Differential Analyzer REplacement). These languages have all been implemented on a Digital Equipment Corporation PDP-9 computer with Extended Arithmetic Element, Automatic Priority Interrupt, 16K of core memory, one or two DECtape transports and/or an RS-Ø9 disk, and a specially designed simulation console.

DARE I is a complete operating system for the PDP-9 including a Monitor or Executive program, a flexible text editing program, a language translator, a FORTRAN IV compiler, a linking loader, and a data output routine. The entire system requires five overlays of the 16K memory, all of which are transparent to the user. The actual simulation is executed in software implemented double-precision floating point arithmetic which yields better than eight decimal digits of significance.

The DARE I language allows direct input, on a CRT typewriter, of first-order ordinary differential

equations (state equations) describing a system. The input language is FORTRAN-like in its syntax but free-format in style. Provision is made for entering a simulation control logic program, FORTRAN IV subroutines and external functions, and non-linear function specification by means of tables. A large library of standard FORTRAN functions is also available. After entry by the user, all of his programs are translated into FORTRAN subroutines which are automatically compiled and loaded with system routines and a user selected numerical integration rule to form the actual simulation program. Provision is made for data input at run-time via a CRT typewriter, and display of one or two simulation variables during problem execution. After the simulation is complete, the user may view all of the simulation output variables on the CRT display console in various combinations or obtain a variety of hard copy outputs.

DARE1X is an extension of DARE I incorporating corrections for minor "bugs" in the original system. More important are the new features that have been added: (1) provision has been made for user definition of his own numerical integration method or special simulation algorithm and his own run-time system control program; (2) a more efficient method of simulating linear systems has been included, currently making possible simulation of fixed linear systems and making possible the future

development of simulation techniques for systems that can be partitioned into a linear segment and a non-linear and/or time-varying segment; and (3) a new magnetic tape program storage feature has been made available. DARE1X is upward compatible with DARE I in all of its important features. It provides, in addition to the interactive problem solution oriented goal of DARE I, a convenient test environment for new numerical integration methods and simulation techniques.

# CHAPTER 2

# SYSTEM OPERATION

In the following discussion it must be assumed that the reader is familiar with the user operation of the DARE I system as explained in Goltz (1970) and in Wait (1970).

## 2.1 Derivative Block 2

DARE1X has two Derivative Blocks for system definition instead of one as in DARE I. Derivative Block 1 is identical to the Derivative Block of DARE I. Derivative Block 2 has been added to extend the capabilities of DARE1X to simulation of fixed linear and partitioned systems. Its use requires an integration routine that is capable of calling different subroutines to integrate the separate state equations in either or both Derivative Blocks, or else one that is capable of implementing some special purpose simulation algorithm that may make use of both Derivative Blocks.

Derivative Block 2 is accessed by typing OPEN D2 or O D2 when in the Editor COMMAND MODE.

Derivative Block 2 can be used in two different modes, linear or non-linear. The mode which is chosen is determined by the problem application and the choice of

an integration method capable of exercising that mode. In either mode, the subroutine resulting from entries in Derivative Block 2 is named DIFEQ2. As in DARE I, the Derivative Block 1 subroutine is named DIFFEQ.

In the non-linear mode, Derivative Block 2 serves to simply extend the problem definition from that in Derivative Block 1. In this mode Derivative Block 2 may contain state equations and any other DARE I statements that are allowed in Derivative Block 1. The total number of state equations in both Blocks must be twenty or less. This mode lends extra flexibility to the design of simulation algorithms. For example, a hypothetical integration routine might be written which would integrate the state equations in Derivative Block 1 with a fixed time step (DT), and would integrate the state equations of Derivative Block 2 with an error-controlled variable time step.

The second Derivative Block may also be used for simulation of fixed linear systems with specially developed linear simulation techniques (Wait, Trevor and Puls 1969; Wait 1969a; Wiatrowski 1971) and an added overlay of PDP-9 memory for matrix calculations. To make use of this linear system feature, the user must declare Derivative Block 2 to be linear by entering on the first line of that block the word LINEAR followed, on the same line, by the names chosen for the linear state variables. The state names must be separated by commas. If the list requires more than one

line, the symbol $ is used to begin any continuation lines. The rules regarding variable names in DARE I apply. The order of the fixed linear system must be ten or less and the number of its inputs must be three or less.

The remainder of Derivative Block 2, when operating in the linear mode, may be used only to specify the input equations for the fixed linear system. There may be no other DARE I statements, and any variable name appearing on the left of an equal sign will be taken to be an input variable. A maximum of three input equations is allowed. These may reference any variables defined in Derivative Block 1, but may not, of course, contain any reference to the linear system state variables of Derivative Block 2. Any equation in Derivative Block 1 may refer to any of the states or input variables of Derivative Block 2. Table 1 outlines the rules regarding the use of Derivative Block 2 in the linear mode.

The actual fixed linear system specification, subject only to the order of the system and the number of its inputs as specified in Derivative Block 2, is done after problem compilation by entering coefficients of the standard A and B matrices in an added overlay which is made available to the user just prior to the Run-Time overlay; viz., the linear system model is described by the matrix equation:

Table 1. Linear Mode Content of Derivative Block 2

| | |
|---|---|
| A. | Derivative Block 2 must be declared linear by entering LINEAR on the first line followed by the linear state names separated by commas. |
| B. | The only other entries allowed are comments and the linear system input equations. The latter may refer to any other defined variables, other than the linear states, or FORTRAN functions. |
| C. | A maximum of 10 states may be specified. |
| D. | A maximum of 3 inputs may be specified. |

$$\dot{\underset{\sim}{X}}(t) = \underset{\sim\sim}{A}\underset{\sim}{X}(t) + \underset{\sim\sim}{B}\underset{\sim}{U}(t)$$

where $\underset{\sim}{X}$ is the state vector and $\underset{\sim}{U}$ is the input vector. The output equation, usually specified in terms of a $\underset{\sim}{C}$ matrix, is not used, since more general output functions of the states may be written as equations in Derivative Block 1.

Any of the DARE I special statements, such as DISPLAY, must appear only in Derivative Block 1 when Derivative Block 2 is LINEAR. If it is desired that one of the inputs to the fixed linear system be an output variable which will be stored for later plotting or listing, that variable will have to be defined in Derivative Block 1 and passed through a dummy input name to Derivative Block 2. (See Section 3.2 for an example of this.)

The user is left with the responsibility of making sure that the integration routine used with a LINEAR Derivative Block 2 is capable of this type of simulation. Currently two system routines are available for this purpose. They are chosen by selecting Method 8 or Method 9 with the Method switch on the DARE console. These methods are not capable of integrating state equations in Derivative Block 1; they simply implement efficient fixed linear simulation algorithms.

Method 8 is the zero-order hold-invariant method in Wait (1969a). The algorithm is

$$\underset{\sim}{X}(n+1) = e^{\underset{\sim}{A}T}X(n) + \underset{\sim}{M}_o\underset{\sim}{B}\underset{\sim}{U}(n)$$

where $\underset{\sim}{X}$, $\underset{\sim}{A}$, $\underset{\sim}{B}$, and $\underset{\sim}{U}$ are as stated previously, T is the time step (DT), and $\underset{\sim}{M}_o$ is given by

$$\underset{\sim}{M}_o = \underset{\sim}{A}^{-1}[e^{\underset{\sim}{A}T} - \underset{\sim}{I}].$$

This algorithm is exact (except for round-off) for fixed linear systems with step or piece-wise constant inputs.

Method 9 is the linear interpolation or polygonal hold-invariant method in Wait (1969a). The algorithm is

$$\underset{\sim}{X}(n+1) = e^{\underset{\sim}{A}T}\underset{\sim}{X}(n) + [\underset{\sim}{M}_o - \underset{\sim}{M}_1]\underset{\sim}{B}\underset{\sim}{U}(n) + \underset{\sim}{M}_1\underset{\sim}{B}\underset{\sim}{U}(n+1)$$

where $\underset{\sim}{M}_1$ is given by

$$\underset{\sim}{M}_1 = \frac{(\underset{\sim}{A}^{-1})^2}{T}[e^{\underset{\sim}{A}T} - \underset{\sim}{A}T - \underset{\sim}{I}].$$

This method gives exact results for fixed linear systems having step, ramp, or piece-wise linear inputs.

The Linear Overlay, described in more detail later, calculates the M matrices above and passes their values along to the Run-Time system. Only the matrices required for the particular algorithm are passed along and as much pre-processing as possible is done in the Linear Overlay, i.e., the whole matrix $[\underset{\sim}{M}_o - \underset{\sim}{M}_1]\underset{\sim}{B}$ is passed along instead of $\underset{\sim}{M}_o$, $\underset{\sim}{M}_1$, and $\underset{\sim}{B}$ separately.

Partitioned system simulation algorithms have not yet been implemented, but are under development (Wait 1969b).

## 2.2 User Defined Integration Method

One of the principal reasons for the development of DARE I was to provide a test environment for the development of new numerical integration methods and simulation techniques. Debugging such new methods was awkward within the DARE I structure.

DARE1X provides a more convenient approach to this problem. The user may enter his own FORTRAN IV integration routine in the Method Block and debug it on-line in the DARE1X environment, taking full advantage of the text editing facilities and extended FORTRAN diagnostics available. Also by judicious use of the FORTRAN Blocks, execution diagnostics may be produced by user programmed

debugging routines. (Appendix C provides a listing of the device unit numbers for this purpose.)

Actually, the term "integration method" is too restrictive. As demonstrated by the second example of Chapter 3, the Method Block can implement a variety of simulation techniques, such as difference equations, which have nothing to do with integration.

Once a user-developed simulation or integration method has proven its utility, it may be permanently entered in the DARE1X library (see Appendix D).

The Method Block is accessed in the Editor COMMAND MODE by typing OPEN M or O M. Before attempting to program a simulation method, the user should be quite familiar with the run-time operation of DARE1X. Unlike other DARE1X problem blocks, the Method Block is not processed by the Translator, and the user is totally responsible for its content. A thorough study of the second example in Chapter 3 is recommended before attempting use of the Method Block. Table 2 contains the COMMON statements that may appear in the Method Block. Only those COMMON blocks actually used in the method are required to be present. The subroutine must be named INTRGX. The standard RETURN statement must appear somewhere to return control from the routine, the independent variable T must be incremented upon each entry, and the physical end of the routine must be marked with an END statement.

Table 2. COMMON Blocks Required in the Method Block

| Block Name | Variable Names | Use |
|---|---|---|
| /STADER/ | GN(20) | Vector of state derivative values |
| /STATEV/ | X(20) | Vector of state variable values from Derivative Block 1. Also includes states in Derivative Block 2 if it is in non-linear mode. |
| /SYSVAR/ | T | The independent variable |
| | DT | The integration method time step (usually only for Derivative Block 1) |
| | TMAX | The maximum value of T |
| | LDOUT* | Data storage flag |
| | DTMAX | Maximum DT in variable step integration methods |
| | DTMIN | Minimum DT |
| | EMAX | Maximum error bound in variable step integration method |
| | EMIN | Minimum error bound |
| | LINIT* | Initialization flag |
| | NORDER** | Order of the system being simulated (includes Derivative Blocks 1 and 2) |
| | SWA,SWB,SWC | Free parameters available for user definition |
| | RUNNO | Run counter for multi-run simulations |
| | NPT** | Number of data output points |
| | TNEXT | Time of the next data output |
| /DB2VAR/ | DT2 | Time step used for Derivative Block 2 |
| | NORDR1** | Order of the system in Derivative Block 1 |
| | NORDR2** | Order of the system in Derivative Block 2 |
| | IORDER** | Number of inputs to fixed linear system in Derivative Block 2 |
| /LSTATV/ | Y(10) | Vector of state variables from Derivative Block 2 when it is LINEAR |

Table 2.--Continued

| | | |
|---|---|---|
| /LINPUV/ | QINP(3) | Vector of fixed linear input variables from Derivative Block 2 when it is LINEAR |
| /MATRIX/ | EAT(10,10) | State transition matrix for linear simulation techniques |
| | MØB(10,3) | Matrices used in linear simulation techniques |
| | M1B(10,3) | |
| | DIFB(10,3) | Contains MØB-M1B unless Method Block used, in which case it contains the B matrix |
| | A(10,10) | The A matrix for linear systems |
| /LINALG/ | MODE** | Mode number for variable mode linear simulation algorithms |

*LOGICAL variables.

**INTEGER variables.

As in DARE I, all variables beginning with L are automatically assumed to be LOGICAL; those beginning with I, J, K, M, and N are assumed to be INTEGER; and all others are assumed to be DOUBLE PRECISION. It is not necessary to declare a variable's type except to change one of the default types. It is not possible to declare a variable to be REAL.

The subroutines accessed by the simulation method are DIFFEQ and DIFEQ2 (note only one F), for the programs resulting from entries in Derivative Block 1 and Derivative Block 2 respectively. DIFFEQ contains the equations to calculate the values of the derivatives of the state variables of Derivative Block 1. DIFEQ2 also contains derivative equations when it is in non-linear mode. When Derivative Block 2 is LINEAR, it contains only the fixed linear system input equations, if there are any. Since variables in either block can depend on variables in the other block, consideration must be given to those variables calculated first.

When there is a user-defined simulation or integration method, the Method switch in the upper left hand corner of the DARE simulation console is ignored and the method number is set internally by the Translator to $16_{10}$. This causes all of the system parameters associated with the integration methods to be placed in the symbol tables for definition at Run-Time. This also causes all of the

matrices of the /MATRIX/ COMMON block to be calculated and passed to the Run-Time system if Derivative Block 2 is LINEAR. In this case the B matrix is passed in the array DIFB which normally contains $[\underset{\sim}{M}_o - \underset{\sim}{M}_1]\underset{\sim}{B}$.

It should be noted that for correct data storage of variables other than state variables, the simulation method should terminate its processing at each time step by making a call to DIFFEQ so that these variables are updated to correspond to the newly calculated values of the state variables. The values of the derivative expressions which are also obtained at this time will still be available at the next time stop so that DIFFEQ need not be called at the beginning of the routine. The initial call to DIFFEQ at the beginning of the run is accomplished by the Run subroutine. The same procedure is required for DIFEQ2 if it is operating in non-linear mode.

In DARE I it was thought that confusion could develop between user-defined symbols and subroutines and system symbols and subroutines. To avoid this, several system symbols included the letter Z and the Z key was disabled (through software modification) so that a user could not confuse the system. This is not as serious a problem as originally thought, but in order to provide flexibility in variable names and file names for DECtape storage while still minimizing the possibility for confusion, the letter Z can be obtained by typing the % key.

The addition of the linear simulation algorithms necessitated adding several matrix manipulating routines to the system library. These are available for use in the Method Block, the Run Block, and the FORTRAN Blocks. Listings are provided in Appendix E.

## 2.3 User Defined Run Subroutine

In DARE1X as in DARE I, SUBROUTINE RUN is the run-time control program that sets up problem execution, initiates data storage, performs bookkeeping, and calls the integration routine. All the DARE systems provide a standard RUN subroutine that is automatically loaded from the system library (.LIBR) at run-time. However, to provide the experienced user complete flexibility in designing simulation techniques, DARE1X also offers the option of entering a user-defined RUN subroutine in the Run Block, which is accessed by the command OPEN (or O) R.

If SUBROUTINE RUN is entered by the user, it is compiled and loaded in place of the standard system RUN subroutine. As in the Method Block, the user is totally responsible for the content of the Run Block. It is not examined by the Translator. It must begin with SUBROUTINE RUN, must contain all of the COMMON statements of Table 3 as well as those of Table 2, and end with RETURN and END statements.

Table 3. Additional COMMON Blocks Required in the Run Block

| Block Name | Variable Names | Use |
|---|---|---|
| /UNDVAR/ | PARAM(20) | Vector of undefined input parameter values |
| /DEFVAR/ | DEFV(20) | Vector of non-state variable output values |
| /ABORT/ | NAB* | Flag which, when .TRUE., indicates TMAX, DT, and NPT are inconsistent for precise data output intervals |

*INTEGER variable.

Before attempting to define a RUN subroutine, one must have a complete understanding of DARE1X run-time execution. A study of the listing of the standard DARE1X Run subroutine in Appendix B is absolutely necessary, and its form should be followed closely.

## 2.4 Run-Time Interaction

Whenever Derivative Block 2 is declared linear, an additional overlay of PDP-9 core memory is loaded by the Run-Time System. This Linear Overlay provides the mechanism for defining the linear system to be simulated. The standard A and B matrices must be specified to this overlay, which in turn calculates other matrices that are

used in the linear system simulation algorithms (see Section 2.1).

The following functions are performed by the Linear Overlay:

1. Coefficients of the A and B matrices are accepted when entered by the user or read-in from paper tape.
2. Coefficients of the A and B matrices that have been specified may be displayed, typed on the Teletype, or punched on paper tape.
3. The state variable and input variable names that the user specified in Derivative Block 2 may be displayed in column vector form showing the order which determines the arrangement of the A and B matrix coefficients.
4. The linear simulation algorithm mode can be chosen by the user.
5. DT2, the linear integration algorithm time step, may be entered.
6. The matrices required by the linear simulation algorithms, EAT, MØB, M1B, and DIFB, are calculated and stored on the disk for retrieval by the Run-Time System.

All of the above functions are controlled by the user commands listed in Table 4. When a coefficient is

Table 4. Linear Overlay Commands

| Command | Function |
|---|---|
| Amn or Bmn | Display the $m^{th}$ row of the A or B matrix |
| Amn = number or Bmn = number | Set coefficient Amn or Bmn equal to "number" and then display the $m^{th}$ row of the matrix |
| MOD or M | Display the simulation mode options and DT2 |
| MOD = number | Set MOD equal to "number" and display mode options and DT2 |
| DT2 or D | Same as MOD |
| DT2 = number | Set DT2 equal to "number" and display mode options and DT2. Note: This same value must be maintained in the Run-Time System |
| PUNCH or P | Punch the A and B matrices on paper tape |
| READ or R | Read the A and B matrices from paper tape |
| TYPE or T | Type the non-zero elements of the A and B matrices, DT2 and MOD |
| NAMES or N | Display the names of the linear state variables and the linear input variables in column vector order |
| CLEAR | Set the elements of A and B, as well as DT2 and MOD, to zero. May not be abbreviated. |
| CONTINUE or C | Proceed with the calculation of the linear system matrices and exit to the Run-Time System for problem execution |
| LINEAR | Issued in the Run-Time system to return to the Linear Overlay. May not be abbreviated. |

specified, it is read in a D1Ø.Ø format which is a free field format reading a maximum of ten characters in I, F, E, or D format and converting them to the DARE1X DOUBLE PRECISION internal format. No extraneous spaces are allowed in the specification commands left of the equal sign. It should be noted that matrix element subscripts are expressed modulo 10 and are not separated by commas or enclosed in parentheses; e.g., $A_{10,2}$ is expressed as AØ2. Any coefficient not explicitly specified by the user is taken to be zero.

The coefficients of the A and B matrices, as well as DT2 and MOD, are preserved between overlays on a disk file, and they need not be respecified except to change them. If, however, the order of the linear system of Derivative Block 2 is changed, or the number of the linear system inputs is changed, A, B, DT2, and MOD are set to zero and must be re-entered.

It is anticipated that user-defined linear simulation algorithms may be programmed to operate in several modes. Provision is made to specify this mode by giving a value to the variable MOD (see Appendix A). Currently none of the system simulation methods utilize this parameter, so it can be ignored.

It is important to note that the value of DT2 specified in the Linear Overlay must not be changed in the Run-Time system. If this is done, the linear simulation

algorithms will fail. On the other hand, if the user issues the CONTINUE command before specifying DT2, the system will gently remind him that he must do so.

Another important detail is that the A matrix must have some non-zero entry. If it does not, the Linear Overlay routine which calculates $e^{\underset{\sim}{A}T}$ will go into an infinite loop. However, it is unlikely that a null A matrix would ever be specified except by accident.

If the user wishes to return to the Linear Overlay from the Run-Time system he may do so by typing LINEAR (cannot be abbreviated) when the CRT screen reads "READY FOR INPUT DATA."

The user familiar with DARE I will note that at Run-Time a new parameter, PTS, is displayed at data input time. This parameter can be used to set the number of output points for data storage and display purposes by setting it equal to some positive integer value less than 257. However, because of the system requirements for producing data output at precise time intervals, the user may not specify values for both DT and PTS. Either DT or PTS must be left zero. The system will then determine convenient values for the zero parameters. If PTS is not specified, its value is calculated as

$$PTS = \frac{TMAX}{K_c \cdot DT} + 1$$

where $K_c$ is the smallest positive integer such that PTS is less than 257. If DT is not specified it is calculated as

$$DT = \frac{TMAX}{K_c \cdot (PTS-1)}$$

where here $K_c$ is the smallest integer such that DT is less than Ø.ØØ1TMAX. If both PTS and DT are left unspecified DT will be set equal to Ø.001TMAX and PTS will be calculated by the first formula above.

The above manipulations make the POINTS statement in Derivative Block 1 unnecessary. This statement is still accepted by the Translator, but is ignored.

All of the above manipulations make the data communication or data output interval (COMINT) equal to some integral multiple of DT.

## 2.5 DECtape Input/Output

The DARE I system provided for permanent program retention by allowing the user to store programs and read them into memory by way of paper tape. This method is cumbersome and frequently prone to machine error, especially with long programs. DARE1X has retained this capability for upward compatibility and for use with small programs, but it has added the facility of program storage on DECtape, Digital Equipment Corporation's directory-oriented magnetic tape.

Three commands are provided in the Editor phase of DARE1X for this feature: READ (or R), WRITE (or W), and DELETE. The user issues these commands by typing them (or the first letter with a READ or WRITE) followed by an option, when required, in parentheses and a user chosen file name of five characters or less.

When a WRITE command is typed with no option, all of the non-empty DARE1X problem blocks will be copied to the DECtape mounted on Unit 1 (WRITE ENABLED) and each will be given a file name formed by appending a key character to the end of the user supplied file name. The key character indicates the DARE1X problem block from which the file came and gives each block a unique name. (See Table 5 for the list of key characters.)

A subsequent READ issued without an option will read into the system all of the files whose file names begin with the file name specified in the command. These new files will be sorted on the key characters and placed in the appropriate DARE1X blocks, replacing any other text that was previously in those blocks.

The option (A) has the same effect as no option, i.e., it READs, WRITEs, or DELETEs all of the blocks or files it refers to.

If any option other than (A) is given, it indicates that only a single block is to be affected. The option itself consists of the first letter of the block name and,

Table 5. Key Characters for DECtape Problem Storage Files

| Block Name | Key Character |
|---|---|
| FORTRAN Block 1 | 1 |
| FORTRAN Block 2 | 2 |
| FORTRAN Block 3 | 3 |
| FORTRAN Block 4 | 4 |
| FORTRAN Block 5 | 5 |
| FORTRAN Block 6 | 6 |
| FORTRAN Block 7 | 7 |
| FORTRAN Block 8 | 8 |
| FORTRAN Block 9 | 9 |
| Derivative Block 1 | : |
| Derivative Block 2 | F |
| Logic Block | ; |
| Output Block | < |
| Table Block 1 | = |
| Table Block 2 | > |
| Table Block 3 | ? |
| Table Block 4 | @ |
| Table Block 5 | A |
| Table Block 6 | B |
| Table Block 7 | C |
| Table Block 8 | D |
| Table Block 9 | E |
| Run Block | G |
| Method Block | H |

if appropriate, the block number (required for the Derivative Blocks, FORTRAN Blocks, and Table Blocks). Only one option is allowed for each command issued. For example, READ (D2) SIM, or equivalently, R(D2)SIM, will read the Derivative Block 2 file, from DECtape 1, which was stored there by either a WRITE (D2) SIM or a WRITE SIM command (the first WRITE stores only Derivative Block 2 under file name SIM, while the second stores all currently non-empty blocks under file name SIM).

The DELETE command should be used with care. It cannot be abbreviated by its first letter. DELETE effectively removes from the DECtape the file (or files) that it references by erasing the directory entry for that file. This command follows the same option rules as READ and WRITE.

If the user issues a WRITE command and the system finds a file on the DECtape that already has the name the user specified, it will inform him and ask if that already existent file is to be replaced by the WRITE command. Answering NO causes a return to the Editor COMMAND MODE. A YES answer will replace the file. The user may frequently find this occurring when he is in program development and is constantly updating his stored programs. But occasionally it is anticipated that one user may inadvertently specify a file name previously used by another user. If this appears to be the case, the

original file should not be replaced and the user should specify a new file name.

The above warning is not issued, of course, with a DELETE command--another reason for care in its use.

If any of the DECtape I/O commands are issued and a DECtape has not been mounted on Unit 1 with WRITE ENABLED, an error message will be displayed on the screen and the system will wait for the user to properly mount the tape. It will proceed with the execution of the command after any CRT keyboard key is struck.

In storing problem blocks on DECtape it is not possible to store the Run-Time data. To preserve these data a paper tape must be punched, using the PUNCH DATA TAPE button, or comments should be entered in Derivative Block 1 to indicate what data may be appropriate.

It is suggested that whenever using the laboratory supplied DARE1X USER TAPE for program storage (as opposed to a user's private tape), the user should form his file names from some part of his own name and place, in each of the DARE1X blocks, a comment that indicates his name and how long that file is to be retained. Periodically laboratory personnel will examine the tape and remove inactive programs.

# CHAPTER 3

## SAMPLE PROBLEMS

Since the simulation language described here is an extension of DARE I, the examples presented will serve to demonstrate only the prominent added features of DARE1X. The full range of problem applications in DARE I is also available in DARE1X.

### 3.1 A Simple Linear System Simulation

The first example illustrates the ease with which fixed linear systems can be simulated using Derivative Block 2 in linear mode. The fourth-order system described by the transfer function

$$G(S) = \frac{10^3}{S^4 + 111.4S^3 + 1155S^2 + 1500S + 10^3}$$

is implemented using the problem specifications of Figures 1 and 2. This system has widely separated eigenvalues, making it a "stiff" system. A fourth-order Runge-Kutta integration rule would require a step size less than Ø.028 just to maintain stability (Wait et al. 1969) while simulating the system with Method 8, the zero-order hold algorithm, provides excellent results (Figures 3 and 4) for

```
* DERIVATIVE BLOCK 1:

* EXAMPLE PROBLEM 1 - THE DISPLAY
* STATEMENT AND ANY OTHER SPECIAL DARE
* STATEMENTS MUST APPEAR IN DER. BLK. 1
* WHEN DER. BLK. 2 IS IN LINEAR MODE
*
      DISPLAY XOUT,Q,T


* DERIVATIVE BLOCK 2:

* EXAMPLE PROBLEM 1 - FOURTH ORDER
* LINEAR SYSTEM SIMULATION
*
* NOTE: STATE AND INPUT NAMES ARE
* USER CHOSEN
*
      LINEAR XOUT,Q,R,S
      FINP1=STEP+RAMP*T


* DATA:

DT = 1.0E-02
TMAX = 1.0E+01
PTS =
DT2 = 1.0000000D-01
XOUT =
Q =
R =
S =
STEP = 1
RAMP =
```

Figure 1. Problem Listing for Example 1

DARE I PROBLEM LISTING - LINEAR OVERLAY

```
A12 =   1.0000000D+00
A23 =   1.0000000D+00
A34 =   1.0000000D+00
A41 =  -1.0000000D+03
A42 =  -1.5000000D+03
A43 =  -1.1550000D+03
A44 =  -1.1140000D+02
B41 =   1.0000000D+03

DT2 =   1.0000000D-01     MOD =  1
```

Figure 2. Linear Overlay Data for Example 1

DARE I OUTPUT LISTING

| TIME | XOUT | Q |
|---|---|---|
| 0.0000000 | 0.0000000 | 0.0000000 |
| .50000000 | 6.6131335E-02 | .28869055 |
| 1.0000000 | .25761341 | .44383685 |
| 1.5000000 | .48412649 | .44348074 |
| 2.0000000 | .68823480 | .36510184 |
| 2.5000000 | .84515997 | .26157067 |
| 3.0000000 | .95084573 | .16373321 |
| 3.5000000 | 1.0123017 | 8.5992390E-02 |
| 4.0000000 | 1.0408284 | 3.1998359E-02 |
| 4.5000000 | 1.0478773 | -6.1324634E-04 |
| 5.0000000 | 1.0429553 | -1.6793978E-02 |
| 5.5000000 | 1.0329221 | -2.1914476E-02 |
| 6.0000000 | 1.0221052 | -2.0607646E-02 |
| 6.5000000 | 1.0128043 | -1.6320936E-02 |
| 7.0000000 | 1.0059008 | -1.1301757E-02 |
| 7.5000000 | 1.0014120 | -6.8018095E-03 |
| 8.0000000 | .99892099 | -3.3547341E-03 |
| 8.5000000 | .99786704 | -1.0414089E-03 |
| 9.0000000 | .99771674 | 2.9769584E-04 |
| 9.5000000 | .99804430 | 9.1405715E-04 |
| 10.000000 | .99855278 | 1.0607322E-03 |
| 10.100000 | .99865855 | 1.0531316E-03 |

Figure 3. Output Data Listing for Example 1

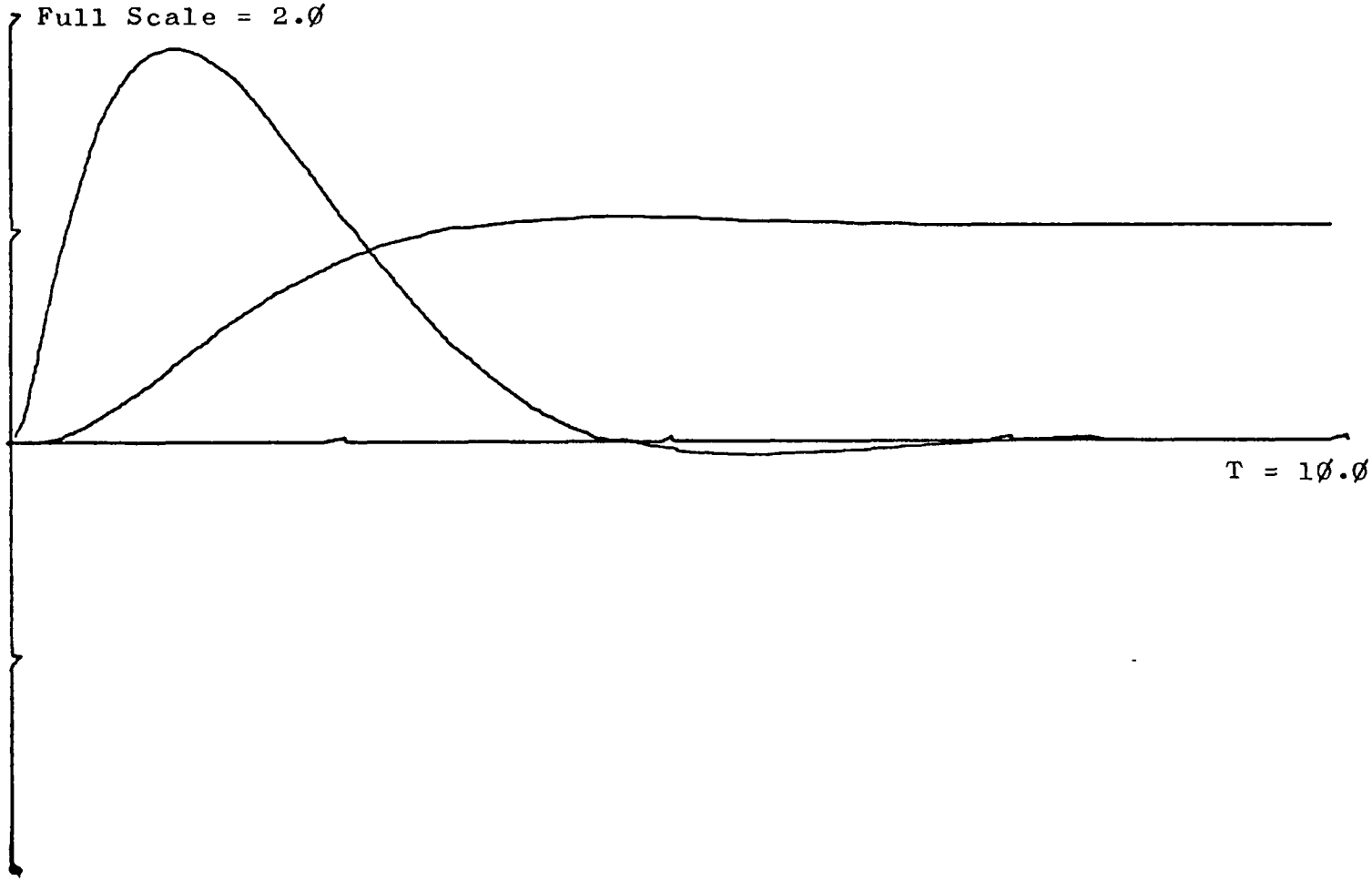


Figure 4. Plotting Board Output for Example 1, XOUT and Q vs. TIME

a unit step input with a time step of 0.1, and this value can easily be more than doubled without loss of accuracy.

In Figure 1 it should be noted that the DISPLAY statement, requesting a Run-Time display of the system output and its second derivative, must be placed in Derivative Block 1. In Derivative Block 2, the order in which the state names are listed determines the order in which those states will appear in the state vector.

The A and B matrices that are input to the Linear Overlay are shown in Figure 2 as they are printed by the TYPE command. Note that when there is no choice of MODE in the fixed linear simulation algorithm, as is the case with Method 8 used here, the TYPE command shows MOD = 1.

### 3.2 A Digital Control System

The second example is more complex, showing the range of flexibility available from DARE1X. The problem is to implement the "dead-beat" digital controller of Figure 5 for a continuous-time system (Gupta and Hasdorff 1970). This demonstrates the possibilities of partitioned system simulation as well as the use of the Method Block for special purpose user-defined simulation algorithms.

Derivative Block 1 specifies the continuous-time system and Derivative Block 2 is used as a dummy linear system to specify the order of the digital controller and

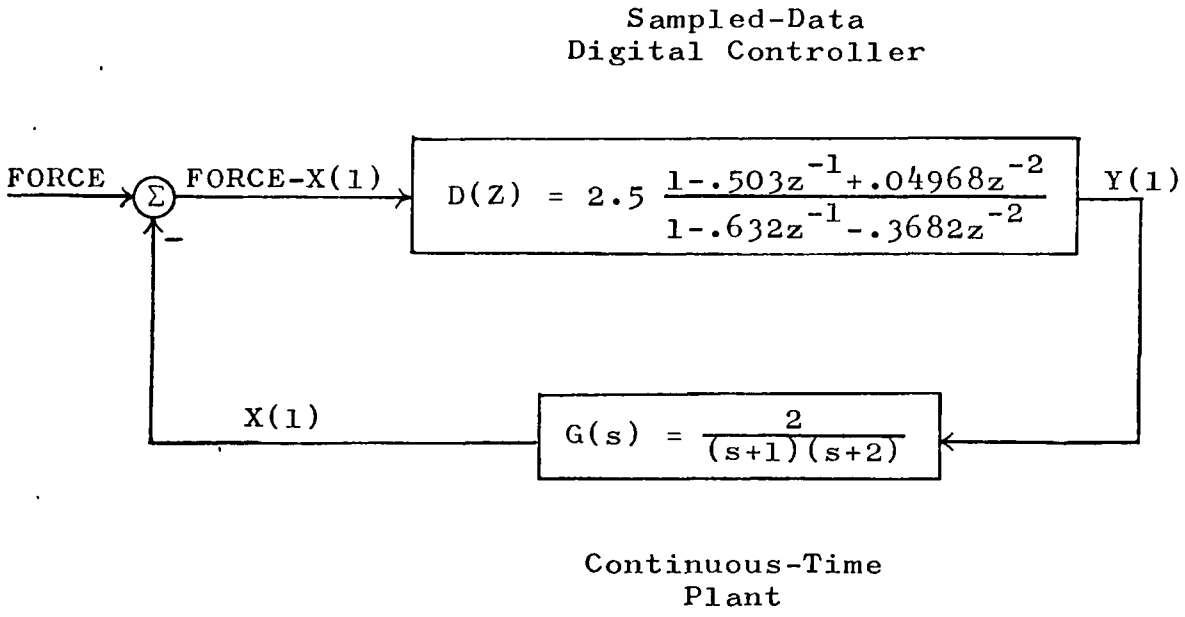


Figure 5. Block Diagram for "Dead-Beat" Control System of Example 2

to link it with the linear system through the input variable CTRL.

The Method Block implements integration of the Derivative Block 1 state equations using a well known second-order Runge-Kutta method. The digital controller is implemented with difference equations. The canonical state variable model of Figure 6 is used (Huelsman 1970). In z-transform notation, the transfer function of the general digital system is

$$D(z) = K_d \frac{a_0 z + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_m z^{-m}}{1 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_k z^{-k}}$$

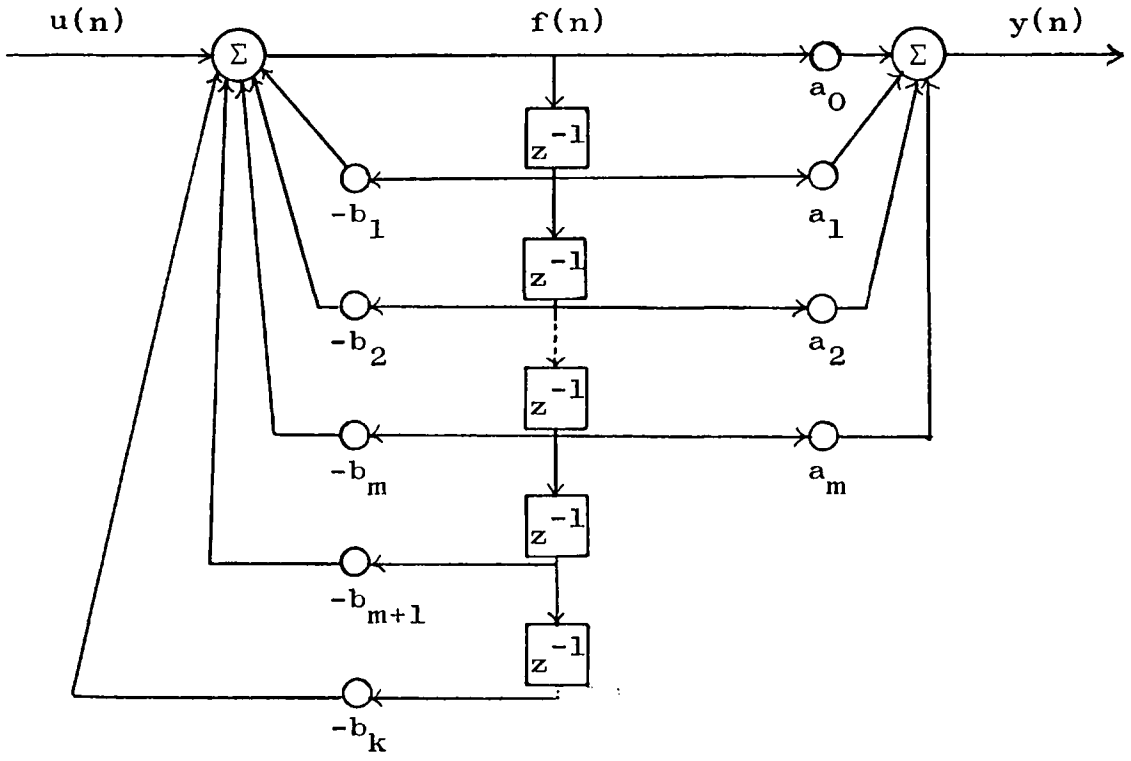


Figure 6. Canonical State Variable Model of a Digital System

where $0 \leq m \leq 3$ and $0 \leq k \leq 3$ is required by the simulation method. The a's are input as the elements of the first row of the A matrix and the b's are the first row of the B matrix; that is, $a_0$ = A11, $a_1$ = A12, $b_0$ = B11, etc. $K_d$ must be entered in Derivative Block 1 as a multiplier of Y1, the output of the controller.

Careful study of the fully commented Method Block of Figure 7 is recommended before attempting the use of the Method Block in defining one's own simulation algorithm. Listings of other standard integration methods are available from laboratory personnel (Goltz 1969).

Figure 8 is the Linear Overlay output of the A and B coefficients that define the controller. Figures 9 and 10 are the hard-copy outputs of the system response to a unit step input. Note the "dead-beat" characteristic of the response; i.e., the output is very nearly equal to the magnitude of the step input at the sample times (one second) even though the overall response is quite oscillatory.

```
* DERIVATIVE BLOCK 1:

* EXAMPLE PROBLEM 2 - 2ND-ORDER
* CONTINUOUS TIME PLANT FOR DIGITAL
* CONTROL
*          G(S)=2/(S+1)(S+2)
* CTRL IS THE INPUT TO THE CONTROLLER
*
X1' = 2.* X2 - 2.* X1
X2' = -1.* X2 + 2.5* Y1
CTRL= FORCE - X1
*
DISPLAY X1,Y1,T


* DERIVATIVE BLOCK 2:

* EXAMPLE PROBLEM 2 - DUMMY LINEAR
* SYSTEM TO IMPLEMENT A 2ND-ORDER
* DIGITAL CONTROLLER
*
LINEAR Y1,Y2,Y3
Q1 = CTRL
Q2 = 0.
Q3 = 0.


* METHOD SUBROUTINE:

 SUBROUTINE INTRGX
*
* DARE1X EXAMPLE - USER DEFINED SIMULA-
* TION METHOD.
* IMPLEMENTS SIMULATION OF A CONTINUOUS
* TIME PLANT IN DER. BLK. 1 USING A
* 2ND-ORDER RUNGE-KUTTA INTEGRATION RULE
* AND A DISCRETE TIME DIGITAL CONTROLLER
* IN DER. BLK. 2 USING THE CANONICAL
* STATE VARIABLE MODEL.
*
* NOTE THAT ONLY VARIABLES BEGINNING
* WITH THE LETTERS I-N NEED TO BE
* DECLARED DOUBLE PRECISION
*
 DOUBLEPRECISION MØB(10,3),M1B(10,3)
 DIMENSION USAV(10),GTEM(10),STEM(10)
*
```

Figure 7. Problem Listing for Example 2

```
*  ONLY THE COMMON BLKS. ACTUALLY USED
*  NEED BE INCLUDED HERE
*
 COMMON/MATRIX/EAT(10,10),M0B,MIB,
$B(10,3),A(10,10)
 COMMON/LINPUV/QINP(3)
 COMMON/STATEV/X(10)
 COMMON/LSTATV/Y(10)
 COMMON/STADER/G(10)
 COMMON/DB2VAR/DT2,NORDR1,NORDR2,IORDER
 COMMON/SYSVAR/T,DT,TMAX,LD,DX,DN,EX,EN,
$LINIT,ND,SA,SB,SC,RNO,NPT,TXT
*
*  CHECK FOR LINIT=.TRUE. FIRST TIME THRU
*
       IF(LINIT) GO TO 900
*
*  CHECK FOR SAMPLE TIME FOR THE CONTROL
*
1      IF(T.GE.TSAMPL) GO TO 300
*
*  INTEGRATE THE STATE EQUATIONS USING
*  2ND-ORDER RUNGE-KUTTA
*
5      DO 100 I=1,NORDR1
       STEM(I)=X(I)
       GTEM(I)=G(I)
100    X(I)=X(I)+DT*G(I)
*
*  TIME IS INCREMENTED FOR THE ENTIRE
*  ALOGRITHM HERE.
*
       TEM=K
       T=TEM*DT
       K=K+1
       CALL DIFFEQ
       DO 200 I=1,NORDR1
200    X(I)=STEM(I)+S*(GTEM(I)+G(I))
*
*  CALL DIFFEQ AGAIN TO UPDATE ANY
*  DEFINED VARIABLES THAT MAY BE IN
*  DER. BLK. 1
*
       CALL DIFFEQ
       RETURN
*
*  COME HERE WHEN IT'S TIME TO SAMPLE
*  THE OUTPUT OF THE PLANT AND FORM A
*  NEW CONTROL.
*  FIRST SHIFT THE CANONICAL STATES
```

Figure 7.--Continued Problem Listing for Example 2

```
*
300     IEND=NORDR2-1
        IST=NORDR2+1
        DO 305 I=1,IEND
        IT=IST-I
        ITS=IT-1
305     USAV(IT)=USAV(ITS)
        USAV(1)=0.
*
* GET THE CONTROLLER INPUT
*
        CALL DIFEQ2
*
* CALCULATE THE NEW CONTROL AND THE
* NEW STATES
*
        CALL MCVMUL(B,USAV,STEM,1,IORDER)
        USAV(1)=QINP(1)-STEM(1)
        CALL MCVMUL(A,USAV,Y,1,NORDR2)
*
* UPDATE THE SAMPLE TIME
*
        TEM=K2
        TSAMPL=DT2*TEM*.9999999
        K2=K2+1
*
* UPDATE THE VARIABLES IN DER. BLK. 1
*
        CALL DIFFEQ
        GO TO 5
*
* COME HERE TO DO THE INITIALIZATION
*       NOTE: "Z" IS OBTAINED FROM THE
*             "PER CENT"  KEY
*
900     K=K2=1
        TSAMPL=T
        DO 905 I=1,10
905     USAV(I)=0.
        S=0.5*DT
        LINIT=.FALSE.
        GO TO 1
        END


* OUTPUT BLOCK:

D (S,T) X1,Y1,T
L (1.) X1,Y1,T
L (3.9,4.1) X1,T
```

Figure 7.--Continued Problem Listing for Example 2

```
* DATA:

DT = 1.0E-02
TMAX = 5.0E+00
PTS =
DTMAX =
DTMIN =
EMAX =
EMIN =
SWA =
SWB =
SWC =
DT2 = 1.0000000D+00
X1 =
X2 =
Y1 =
Y2 =
Y3 =
FORCE = 1
```

Figure 7.--Continued Problem Listing for Example 2

DARE I PROBLEM LISTING - LINEAR OVERLAY

```
A11 =  1.0000000D+00
A12 = -5.0300000D-01
A13 =  4.9680000D-02
B11 =  1.0000000D+00
B12 = -6.3200000D-01
B13 = -3.6800000D-01

DT2 =  1.0000000D+00     MOD =  0
```

Figure 8. Linear Overlay Data for Example 2

| TIME | X1 | Y1 |
|---|---|---|
| 0.0000000 | 0.0000000 | 0.0000000 |
| .26000000 | .13107891 | 1.0000000 |
| .50000000 | .38708208 | 1.0000000 |
| .75000000 | .69601952 | 1.0000000 |
| 1.0000000 | .99895591 | 1.0000000 |
| 1.2600000 | 1.1688503 | .13004409 |
| 1.5000000 | 1.1720697 | .13004409 |
| 1.7500000 | 1.1011098 | .13004409 |
| 2.0000000 | 1.0000548 | .13004409 |
| 2.2600000 | .93780694 | .49928793 |
| 2.5000000 | .93673433 | .49928793 |
| 2.7500000 | .96288683 | .49928793 |
| 3.0000000 | 1.0000732 | .49928793 |
| 3.2600000 | 1.0229590 | .36341241 |
| 3.5000000 | 1.0233279 | .36341241 |
| 3.7500000 | 1.0136763 | .36341241 |
| 4.0000000 | .99996635 | .36341241 |
| 4.2600000 | .99153024 | .41348235 |
| 4.5000000 | .99139385 | .41348235 |
| 4.7500000 | .99495105 | .41348235 |
| 5.0000000 | 1.0000042 | .41348235 |
| 5.0200000 | 1.0004129 | .39503183 |

Figure 9. Output Data Listing for Example 2

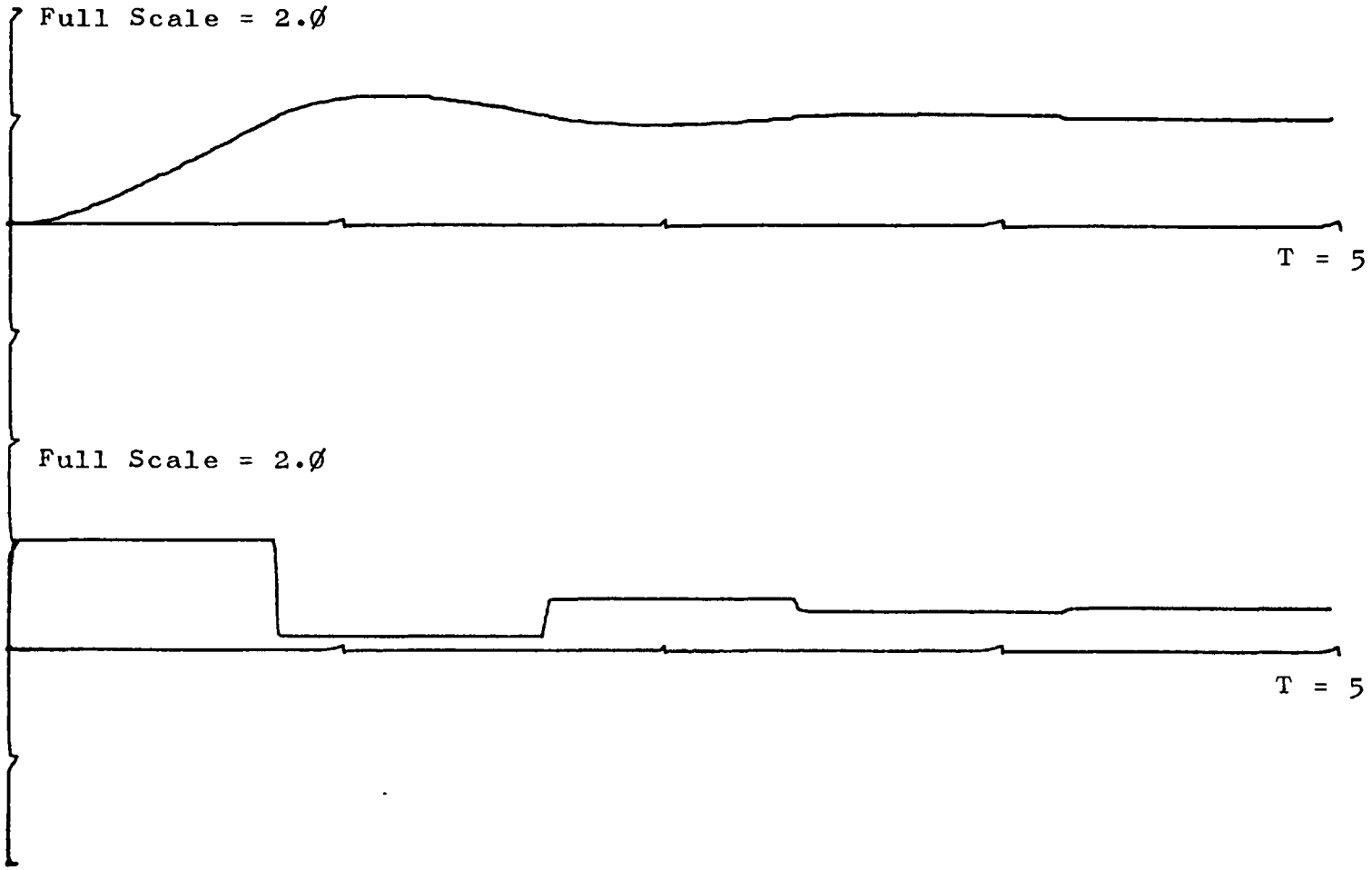


Figure 10. Plotting Board Output for Example 2, X1 and Y1 vs. Time

# CHAPTER 4

## SYSTEM PROGRAM INFORMATION

DARE1X retains the basic overall structure of DARE I, with the addition of the Linear Overlay which is optionally loaded by the Run-Time System.

All of the overlays except the FORTRAN IV compiler were modified for DARE1X to provide the processing necessary for implementation of its expanded facilities. The major changes are summarized below. For a complete understanding of these program modifications, refer to the fully documented program listings available from laboratory personnel (Moore 1971).

1. Monitor (D1XMON)
    a. System registers 22 and 23 are used for block-file bit maps and 25 is used as a Linear Overlay switch.
    b. .DAT+1 is made available to the Run-Time System.
    c. .SIM+22 and .SIM+23 were added to store the order of Derivative Blocks 1 and 2 respectively.
    d. .SYSPR was augmented to include Linear Overlay loading information.

e. The IOPS4 error was modified to indicate a probable DECtape-not-ready condition.

f. CRA. modified to output the character Z when the % key is struck.

g. The non-resident section was modified to zero system registers 22 and 23 unless AC switch Ø is up. This provides the facility, as in DARE I, of saving the problem blocks, in case of system failure, by putting AC Ø in the up position. This is effective only if the problem has once been compiled.

2. Editor/Translator (D1XEDT)

a. New blocks Derivative Block 2, Run Block, and Method Block were added.

b. DECtape program storage added (see Table 6 for block file names).

c. Instead of maintaining the entire text of all the problem blocks in core simultaneously, the block-file was created. Each block is made into a separate file and stored on the disk when it is not being displayed by the Editor. In this way each block has approximately 5000 words of text buffer area available to it. (See Table 6 for the block-file file names.)

Table 6. Disk Block-File Names

| Block Name | Block-File Name |
|---|---|
| FORTRAN Block 1 | EDTD1A |
| FORTRAN Block 2 | EDTD1B |
| FORTRAN Block 3 | EDTD1C |
| FORTRAN Block 4 | EDTD1D |
| FORTRAN Block 5 | EDTD1E |
| FORTRAN Block 6 | EDTD1F |
| FORTRAN Block 7 | EDTD1G |
| FORTRAN Block 8 | EDTD1H |
| FORTRAN Block 9 | EDTD1I |
| Derivative Block 1 | EDTD1J |
| Derivative Block 2 | EDTD1V |
| Logic Block | EDTD1K |
| Output Block | EDTD1L |
| Table Block 1 | EDTD1M |
| Table Block 2 | EDTD1N |
| Table Block 3 | EDTD1O |
| Table Block 4 | EDTD1P |
| Table Block 5 | EDTD1Q |
| Table Block 6 | EDTD1R |
| Table Block 7 | EDTD1S |
| Table Block 8 | EDTD1T |
| Table Block 9 | EDTD1U |
| Run Block | EDTD1W |
| Method Block | EDTD1X |

d. Appropriate modifications were made to all block access functions to allow them to access the disk block-files.

e. The block-file bit maps of system registers 22 and 23 are used to indicate which block-files are currently active in the Editor. This was necessitated by the lack of the .FSTAT function in the disk handler routine used.

f. The command structure was augmented to permit a and b.

g. Appropriate flags and processing routines were modified and/or added to handle Derivative Block 2 and the Method Block. (Note that the Run Block is handled just like the FORTRAN Blocks and may, indeed, be used as one. However, the Method Block causes additional processing to take place.)

h. Information concerning the Linear Mode of Derivative block 2 is written on the disk file LINEDT SRC which is read by the Linear Overlay.

i. Minor bugs were corrected that existed in the DARE I Translator.

3. Loader (D1XLOD)

a. A check is made for the existence of a user-defined integration method to defeat the

default loading of one of the system integration routines.

b. The method subroutine file name (METNAM) was changed so that it ends in an alphabetic character instead of a number.

4. Run-Time System (D1XRTS)

a. A check is made to see if Derivative Block 2 is LINEAR. If it is, the Run-Time System dumps itself onto the disk (this makes it unnecessary to repeat the loader process) and exits to the Linear Overlay.

b. After returning from the Linear Overlay (if there is one) the Run-Time System reads the file produced by the Linear Overlay to obtain the various matrices necessary for the linear simulation algorithms.

c. Tests are made on TMAX, DT, and NPT to determine if they are consistent for precise data output intervals.

d. Access is provided to the LINEAR state variable initial conditions.

e. The little used feature of data listing in the Run-Time System was removed, as well as the unsuccessful attempt to change the integration method by depressing the RESET button. (This operation requires the modification of symbol

tables, necessitating re-translation, compiling, and loading.)

f. The run-time display routines were modified to provide access to the new linear state variables from Derivative Block 2.

g. The LINEAR command was implemented to return to the Linear Overlay from the Run-Time system.

5. Data Output Overlay (D1XOUT)

a. Access is provided to the Output Block directly through its block-file.

It should also be noted, as shown in Table 1, that the variables NPT and TNEXT have been added to the system COMMON block /SYSVAR/. These parameters are used to provide precise output intervals for data storage.

The added Linear Overlay (D1XMAT) was necessitated by the large amount of core memory required to calculate and store the matrices required for the linear simulation algorithms. Calculation of these matrices in a separate overlay frees more memory in the Run-Time System where it is needed for actual implementation of the simulation algorithms.

The order of the A and B matrices, the order of the non-linear system in Derivative Block 1 (for partitioned system simulation), the integration method number, and the user-provided names of the linear states and inputs are

provided to the Linear Overlay by the Translator-produced disk file LINEDT SRC. The matrices required by the linear simulation algorithms, DT2, MOD, and the order of A and B are passed to the Run-Time System in a disk file named LINRTS SRC.

In order to make the expanded Editor/Translator fit on the system tape, it was necessary to move the Editor/Translator and Loader overlays to different locations. This change is indicated in the system tape directory and in the .SYSPR table in the Monitor.

# CHAPTER 5

## SUGGESTIONS FOR FUTURE DEVELOPMENT

It has been found that there are two major obstacles to modification of the DARE I system: (1) system debugging is difficult since DARE I does not execute under any standard operating system; and (2) the 16K memory capacity of the PDP-9 is barely adequate for the Run-Time overlay. It is felt that before any further expansion is undertaken, a study should be made of the possibility of rewriting DARE1X under the standard operating system to execute as a CHAIN program.

Much of the DARE1X Monitor functions as a large symbol table and program flag storage area. This was necessitated by the need to pass common information between overlays, and the DECtape used with the original DARE I system was too slow for storage of this information. Now that a fast access disk is available, this information could be stored there and retrieved with the .TRAN disk handler function without noticeable delay and with considerable gain in core availability in the overlays where this information, or some part of it, is not needed. Elimination of this Monitor function would be a big step in allowing the DARE1X system to function under a standard

operating system, making available all of the standard debugging techniques.

# APPENDIX A

## MODIFYING THE LINEAR OVERLAY

The Linear Overlay of DARE1X is written modularly to make it easy to modify. The functions of the various routines are explained below.

D1XMAT is the "main" program. It is a Macro assembly language program that sets up the .DAT slots for the overlay and initializes the subroutine that services interrupts generated by the DARE console buttons. It also contains the subroutine EXIT which initiates loading of the Run-Time Overlay when the Linear Overlay processing is complete.

CAC. is another assembly language program which, in conjunction with the FORTRAN subroutine WRITE, performs the encoding of the variable DT2 and places its value in the monitor .SYMBL table so that it can be passed along to the Run-Time Overlay.

OVERL6 is called by D1XMAT and becomes the main FORTRAN control program. Its function is to simply pass control from one phase of the Overlay to the next.

READDK is the first phase of the Linear Overlay. It reads from disk file LINEDT SRC the information created by the Translator concerning the LINEAR Derivative Block 2.

This information is the order of the fixed linear system, the number of its inputs, the order of the system in Derivative Block 1 (Ø if there are no state equations in Derivative Block 1), the Method number (16 if there is a user defined method), and names of the states and inputs in Derivative Block 2.

READSC is the heart of the Linear Overlay. It reads user commands from the CRT keyboard, processes some of these commands directly (CLEAR, NAMES, PUNCH, READ, TYPE, and matrix element specification commands), and passes control to other routines for the rest. When it is initially called by OVERL6, READSC also checks to see if previous processing by the Linear Overlay has created the file LINLIN SRC containing values of the A and B matrices, DT2 and MOD. If this file is present on the disk, the order of the A and B matrices in it are compared with those specified by the Translator for the current problem. If they are the same, A, B, DT2, and MOD are read from the disk and stored in COMMON. If they are not the same, all variables are set to zero.

OPTION is called by READSC to process the MODE or DT2 commands. Here is where some changes will most likely be made in the future. By checking the Method number, this routine can determine whether any variable modes of operation are available for a particular simulation algorithm that is invoked by declaring Derivative Block 2 to be

LINEAR. Currently, checks are made only for Method numbers 8, 9, and 16. Methods 8 and 9 do not make use of the variable mode option so a branch is done in OPTION to a WRITE statement that tells the user that no choice is available. Internally MOD is set to 1 just so it will have a non-zero value. For Method 16, the user-defined method, any value can be given to MOD and used by the simulation algorithm. Here OPTION branches to tell the user that the mode is free. When a new LINEAR method is installed in the system, OPTION will have to be modified so that the new method number is checked and a branch is provided to a WRITE statement that tells the user what his choices are. A zero value for MOD may not be used except with a user defined method in the Method Block.

MATCAL is the routine called in response to the CONTINUE command. It calculates the matrices required by the fixed linear-simulation algorithms (Wiatrowski 1971). Here also future changes may be required. Checks must be made at appropriate points for the Method number. In order to save time and disk space, only those matrices actually needed for a particular algorithm should be calculated and stored on the disk. Here care must be taken to assure data format consistency so that the Run-Time system will be able to correctly read the Linear Overlay data. The matrices EAT ($e^{\underset{\sim}{AT}}$), MØB, M1B, DIFB, and A are all provided for, and all will be read, in that order, by

the Run-Time system. As soon as a matrix is calculated, it should be written on the disk by calling the routine WRITDK. If a matrix is not used in a particular algorithm, WRITDK must be called to write a zero data entry for that matrix (see the next paragraph). By writing out the matrices as soon as they are calculated, temporary array storage can be freed for use in other calculations, thus saving memory.

As indicated above, some entry must be made in the disk file LINRTS SRC for each of the five linear simulation matrices which will be read at Run-Time by the subroutine REDLIN. WRITDK scans each array that is passed to it and writes only the elements (and their subscripts) which have non-zero value onto the disk file. The end of each array is marked by a zero data entry, and the end of the file is marked by entering zero subscripts. WRITDK is called with a control parameter, ISW, that specifies exactly what is to be done. ISW must be 1 when EAT is passed to WRITDK. This causes the file LINRTS SRC to be entered for output (.ENTER) and then EAT is written in the file. ISW equal to 2 is used to write the file LINLIN SRC (this should not concern the user). When ISW is 3, the matrix passed to WRITDK is scanned and its non-zero elements are written on the disk. When the array is exhausted, a zero entry is made which is checked by REDLIN in the Run-Time Overlay to mark the end of an array. Setting ISW to 4 causes only a zero data entry to be written. This is done when one of

the five required arrays is not to be used. When REDLIN encounters this entry it knows to skip the next array. If ISW is 5, WRITDK writes out the matrix passed to it, marks the file end by entering Ø subscripts for a zero data entry, and then closes LINRTS SRC (.CLOSE).

For a thorough understanding of the disk writing operations, the reader should refer to the DARE1X listings of the Linear Overlay (Moore 1971).

If it becomes necessary to calculate and pass along matrices other than those provided, modification of WRITDK, MATCAL, and the Run-Time routine REDLIN will have to be made. It may, however, be possible to use some of the arrays to pass information other than that which their mnemonic names indicate. For example, DIFB was originally intended to contain MØB-M1B, but when the Method Block is used, DIFB is used to pass just the B matrix.

The routines MCAL, MATMUL, SCAMUL, MATCLR, and ADDID are matrix manipulation routines used by MATCAL and should not need future modification.

The Linear Overlay can be tested in any of the standard CSRL disk operating systems. Two actions are necessary to implement testing. First D1XMAT must be assembled using the P option at assembly time to give some value to the parameter TEST. The following command sequence accomplishes this.

```
MONITOR
$ A TTA -10↵
$ MACRO↵
MACRO
> P, B ← D1XMAT↵
TEST = 1↵
↑D EOT
↑P ↑P
```

The underlined portion indicates user commands; the symbol ↵ indicates a carriage return; and the symbol ↑ means hold the CTRL key down while typing the following character.

The second thing that must be done before testing is to make the proper device assignments before loading. A single Monitor command does this.

```
MONITOR
$ A DKD1 1,1Ø↵
```

Although .DAT slot 10 is not used by the disk but is a dummy .DAT slot used by CAC., it must be assigned to DKD1 to prevent loading of extra handlers.

Loading is then accomplished in the same fashion as indicated below for a non-test load.

Note that the file LINEDT RTS must be artificially established before the Linear Overlay can be tested. Its form can be deduced by examination of READDK.

To place a modified version of the Linear Overlay on the DARE1X system tape, first make sure D1XMAT has been assembled without the P option. Then proceed with the following command sequence.

```
MONITOR
$ A DKD1 1,2,3,4,5,1Ø↵
$ LOAD
LOADER
> D1XMAT, CAC., WRITE, OVERL6, READDK, READSC↵
> WRITDK, OPTION, MATCAL, MCAL, MATMUL↵
> SCAMUL, ADDID, MATCLR (ALT MODE)
```

At this point the LOADER will type out the load map, terminating with ↑S. Now the paper tape program D1WRIT must be read-in at location 100 by depressing the I-O RESET key then the READ IN key on the PDP-9 console. The use of D1WRIT is covered in Appendix D of Goltz (1970). The BLOCK NO. required by D1WRIT for the Linear Overlay is $220_8$.

If the SIZE of the core load ($37637_8$ - last address from loader map) is greater than $32226_8$, the Overlay will not fit on the system tape. It is not anticipated that this will occur for any reasonable modifications. If it does happen, and no programming tricks can be used to reduce the memory requirements, then the system tape must be restructured. This is a somewhat tedious process that

will not be covered here and should not be attempted except by an experienced PDP-9 systems programmer.

After the new Linear Overlay is on the system tape, the .SYSPR table in the Monitor must be modified to reflect the new size of the overlay. Using the system program PATCH, locations 246, 247, and 250 in the Monitor must be changed. After assigning the proper DECtape unit on which the system tape is mounted to .DAT-14 and calling PATCH, the following commands are issued.

```
PATCH
> KM9 ↵
> L 245 ↵
> 00245/000220>  ↵
  00246/751144> xxxxxx ↵
  00247/0011002> yyyyyy ↵
  00250/037505> zzzzzz (ALT MODE)
> EXIT ↵
```

Location 245 should contain $220_8$ which is the starting block for the Linear Overlay on the system tape. This should not be changed. Location 246 should be changed to the two's complement of the Overlay SIZE (xxxxxx), 247 should be one less than the last address on the load map (yyyyyy), and 250 should be the address at which D1XMAT is loaded. Location 250 need not be changed unless D1XMAT is altered.

At this time the modifications should be complete; however, in the interests of up-to-date documentation, the source version of the Monitor should be changed to indicate the new entries in the .SYSPR table. After a short user trial period to insure that all modifications are working properly, all back-up tapes should be updated.

# APPENDIX B

## RUN SUBROUTINE LISTING

The listing which follows shows the exact form of the DARE1X standard Run Subroutine with explanatory comments. This is the form that is acceptable by the regular CSRL operating system FORTRAN compiler.

For the Run Block, change all comment lines so that they begin with "$" instead of "C."

```
      SUBROUTINE RUN
C
C RUN-TIME SIMULATION CONTROL PROGRAM
C FOR DAREIX SIMULATION SYSTEM
C
C DECLARE ALL LOGICAL AND DOUBLE
C PRECISION VARIABLES
C
      LOGICAL LDOUT,LINIT,DTDAT
      DOUBLE PRECISION X(20),Y(10),
     $DEFV(20),PARAM(20),T,DT,TMAX,
     $DTMAX,DTMIN,EMAX,EMIN,SWA,SWB,SWC,
     $RUNNO,TNEXT,DT2,DTSAV,DT2SAV,
     $XSAV(20),YSAV(10),COMINT,TMAXP,
     $AIK,PTS,TOL
C
      COMMON /STATEV/X
      COMMON /LSTATV/Y
      COMMON /DEFVAR/DEFV
      COMMON /UNDVAR/PARAM
      COMMON /SYSVAR/T,DT,TMAX,LDOUT,
     $DTMAX,DTMIN,EMAX,EMIN,LINIT,
     $NORDER,SWA,SWB,SWC,RUNNO,NPT,
     $TNEXT
      COMMON /DB2VAR/DT2,NORDR1,NORDR2,
     $IORDER
      COMMON /ZLOC/II
C
C /ZSTORZ/ SHOULD HOLD ALL LOCAL
C VARIABLES USED ONLY IN RUN
C
      COMMON /ZSTORZ/DTSAV,DT2SAV,XSAV,
     $YSAV,COMINT,TMAXP,AIK,PTS,TOL,IK,
     $TLITE,STPINT,DTDAT
C
C SET UP EARLY EXIT POINT; USED BY
C SYSTEM PROGRAMS
C
      ASSIGN 6000 TO II
C
C SAVE IC'S AND DT'S
C
      DO 10 I=1,NORDR1
10    XSAV(I)=X(I)
      DO 20 I=1,NORDR2
20    YSAV(I)=Y(I)
      DTSAV=DT
      DT2SAV=DT2
```

```
C
C SEE IF DT2 IS USED. IF SO BASE CALCU-
C LATION OF COMINT ON DT2, NOT DT.
C
      CALL WHATDT(DTDAT)
      IF(DTDAT) DT=DT2
C
C NOW CHECK FOR Ø PARAMETERS AND ASSIGN
C DEFAULT VALUES. NOTE TMAX CANNOT BE Ø;
C BUT ONE OR BOTH OF DT & PTS MUST BE Ø.
C
      IF(DT.EQ.Ø.DØ) GO TO 1ØØØ
C
C HERE IF PTS=Ø OR IF DT2 IS USED
C
499   IK=Ø
5ØØ   IK=IK+1
      PTS=TMAX/DT
      NPT=PTS
      NPT=1+NPT/IK
      IF(NPT.GT.256) GO TO 5ØØ
      GO TO 2ØØØ
C
C HERE IF DT=Ø
C
1ØØØ  IF(NPT.EQ.Ø) GO TO 15ØØ
      PTS=NPT
      T=Ø.ØØ1DØ*TMAX
      IK=Ø
11ØØ  IK=IK+1
      AIK=IK
      DT=TMAX/(AIK*(PTS-1.DØ))
      IF(DT.GT.T) GO TO 11ØØ
      GO TO 2ØØØ
C
C HERE IF DT=PTS=Ø
C
15ØØ  DT=Ø.ØØ1DØ*TMAX
      GO TO 499
C
C SET UP SYSTEM ROUTINES. NOTE THAT
C SETDZS MUST BE CALLED WITH A DUMMY
C ARGUMENT IN DARE1X.
C (FIRST RESET DT IF DT2 WAS USED ABOVE)
C
2ØØØ  IF(DTDAT) DT=DTSAV
      IF(LDOUT) CALL SETSTR
      CALL SETDZS(IK)
```

```
      CALL SETSTP
C
C SET COMMUNICATION INTERVALS
C
      PTS=NPT-1
      COMINT=TMAX/PTS
      IF(DTMAX.GT.COMINT) DTMAX=COMINT
      STPINT=TMAX/11.D0
      TOL=-0.0001D0*COMINT
      TMAXP=TMAX+TOL
C
C INITIALIZE
C
      LINIT=.TRUE.
      RUNNO=RUNNO+1.D0
      T=0.D0
      TNEXT=COMINT
      TLITE=STPINT*.999999
      IK=1
C
C CALCULATE AND STORE THE INITIAL POINT
C
      CALL DIFFEQ
      CALL DIFEQ2
      IF(LDOUT) CALL STORE
      CALL RUNDIS
C
C CARRY OUT THE SIMULATION!
C
4000  CALL INTRGX
      IF(T-TLITE) 4001,4002,4002
4002  CALL ZSTRIP
      TLITE=TLITE+STPINT
C
C TEST FOR DATA STORAGE TIME
C
4001  IF((T-TNEXT).LT.TOL) GO TO 4000
C
C DO STORAGE AND UPDATE DISPLAY
C
5000  IF(LDOUT) CALL STORE
      CALL RUNDIS
C
C TEST "TERMINATE" STATEMENT IF THERE IS
C ONE
C
      CALL TERMIN
```

```
C SET NEW DATA COMMUNICATION TIME
C
      IK=IK+1
      AIK=IK
      TNEXT=AIK*COMINT
C
C TEST FOR END OF RUN
C
      IF(T-TMAXP) 4000,4000,6000
C
C RUN COMPLETE; STORE FINAL POINT AND
C RE-ESTABLISH IC'S AND DT'S
C
6000  IF(LDOUT) CALL FINSTR
      DO 7000 I=1,NORDR1
7000  X(I)=XSAV(I)
      DO 8000 I=1,NORDR2
8000  Y(I)=YSAV(I)
      DT=DTSAV
      DT2=DT2SAV
      RETURN
      END
```

## APPENDIX C

## RUN-TIME DEVICE UNIT NUMBERS

To do input/output to the DARE peripheral devices in the FORTRAN language, the following device numbers should be used. These devices can be accessed at Run-Time through one of the FORTRAN Blocks, the Method Block, or the Run Block.

| | |
|---|---|
| CRT Screen--Output | 3 |
| CRT Keyboard--Input | 2 |
| Teletypewriter--Output only | 4, 5 |

## APPENDIX D

## ADDING NEW INTEGRATION METHODS

Before adding a new integration or simulation method it should be thoroughly tested in the DARE1X environment using the Method Block. Once the method is properly functioning it should be saved in its final form on a DECtape by using the WRITE command in the Editor. Then further processing is necessary under one of the standard operating systems.

First, the method file name must be changed. This can be accomplished while making other changes in the CRT Editor. All that is required is to type the new name after the CLOSE command. The new name must be the characters METHD followed by a letter of the alphabet corresponding to the Method number by which this method is to be known. This is different from DARE I. Method 1 must have file name METHDA, while Method 10 must have file name METHDJ.

Next, all DOUBLE PRECISION and LOGICAL variables must be explicitly declared. This is not necessary when testing in the Method Block, but since the method must now be compiled by the standard FORTRAN IV compiler, these declarations are required.

After compiling, the binary object code should be resident on DK1. In order to properly place the routine on the system tape, it must first be written into the system while the system is resident on DKØ. This insures that it will be formatted properly for the disk refresh function. The process is as follows. With the new method on DK1, place the DARE1X system on DKØ using the paper tape program RFSAV. Next, load a DECtape based operating system, assign DKA to DAT slots 6 and 7, and use PIP to transfer the method from DK1 to DKØ. Now use the RFSAV program to transfer the updated system back to the disk refresh DECtape.

The new method will now be physically resident on the system tape, but before it can be accessed the proper entry must be made in the .METH table in the Monitor. Here there is one word allotted to each method which informs the system whether the method is defined and what system parameters it uses. The bit functions are outlined below. When a particular parameter is to be used, its bit should be a 1; otherwise it should be Ø.

Bit Ø: always 1

Bit 1: DTMAX

Bit 2: DTMIN

Bit 3: EMAX

Bit 4: EMIN

Bit 5: SWA

Bit 6: SWB

Bit 7: SWC

Bit 8: DT2

All other bits must be zero.

To make the proper entry in the .METH table for a new method, the system program PATCH is used. Mount the DARE1X system tape on Unit 1, WRITE ENABLED, assign DTA1 to .DAT slot -14, and call patch. Then type the following:

> KM9 ↵

> L 147 ↵

Patch will then type out an address which is the beginning address of the .METH table. To this address add the number of the new method minus 1 (octal radix) yielding nnnnn and type

> L nnnnn ↵ .

PATCH will then type the current contents of the .METH entry (should be all zeros) after which the user should type the new entry that indicates what system parameters the method will use, followed by ALT MODE. This completes the installation of the new method; however, the new .METH entry should also be inserted in the source version of the Monitor program.

If a new method which makes use of the Linear Overlay is installed, other modifications may be required to the Linear Overlay (see Appendix A).

# APPENDIX E

## MATRIX MANIPULATION LIBRARY ROUTINES

The listings on the following pages are of FORTRAN subroutines that have been placed in the DARE1X library (.LIBR) for matrix manipulation. They may be called from the FORTRAN Blocks, the Method Block, the Run Block, the Logic Block, and from a PROCED section of Derivative Block 1.

Note that the argument lists are not all of parallel construction and that some use singly-subscripted arrays and some use doubly-subscripted arrays.

```
      SUBROUTINE MATMUL(P,Q,R,IR1,IRC,
     #IC2)
C
C MULTIPLY AN IR1*IRC MATRIX ON LEFT
C BY AN IRC*IC2 MATRIX ON RIGHT WHERE
C FIRST INDEX IS NUMBER OF ROWS
C
      DOUBLE PRECISION P,Q,R
      DIMENSION P(10,10),Q(10,10),
     1 R(10,10)
      DO 10 I=1,IR1
      DO 10 K=1,IC2
      P(I,K)=0.D0
      DO 10 J=1,IRC
10    P(I,K)=P(I,K)+Q(I,J)*R(J,K)
      RETURN
      END
```

```
      SUBROUTINE SCAMUL(B, C, D, IR, JC)
C
C SUBROUTINE TO MULTIPLY AN IR*JC
C ARRAY BY A SCALAR
C
      DOUBLE PRECISION B, C, D
      DIMENSION B(10,10), C(10,10)
      DO 10 I=1, IR
      DO 10 J=1, JC
10    B(I, J)=D*C(I, J)
      RETURN
      END
```

```
      SUBROUTINE MATCLR(S, IR, JC)
C
C SUBROUTINE TO ZERO OUT AN IR*JC ARRAY
C
      DOUBLE PRECISION S
      DIMENSION S(10,10)
      DO 10 I=1, IR
      DO 10 J=1, JC
10    S(I, J)=0.D0
      RETURN
      END
```

```
      SUBROUTINE ADDID(R,S,IR,JC)
C
C SUBROUTINE TO ADD THE IDENTITY MATRIX
C TO AN IR*JC ARRAY
C
      DOUBLE PRECISION R,S
      DIMENSION R(10,10),S(10,10)
      DO 10 I=1,IR
      DO 10 J=1,JC
      R(I,J)=S(I,J)
10    IF(I.EQ.J) R(I,I)=S(I,I)+1.0D0
      RETURN
      END
```

```
      SUBROUTINE MCVMUL(A,B,C,IR,IC)
C
C SUBROUTINE TO MULTIPLY AN IR*IC
C MATRIX ON LEFT BY AN IC*1 VECTOR ON
C RIGHT
C       NOTE: ARGUMENT LIST DOES NOT
C       PARALLEL THAT OF THE OTHER
C       DAREIX MATRIX ROUTINES
C
      DOUBLE PRECISION A(10,10),B(10),
     #C(10)
      DO 1 I=1,IR
      C(I)=0.D0
      DO 1 J=1,IC
1     C(I)=C(I)+A(I,J)*B(J)
      RETURN
      END
```

## LIST OF REFERENCES


Goltz, John R.

1969 DARE I Program Listings, Computer Science Research Laboratory, Electrical Engineering Dept., Univ. of Ariz.

1970 DARE-I, An On-Line Digital Simulation System, Ph.D. Dissertation, Department of Electrical Engineering, Univ. of Ariz.

Gupta, Someshwar C., and Lawrence Hasdorff

1970 Fundamentals of Automatic Control, John Wiley and Sons, Inc., Chapter 9.

Huelsman, Lawrence P.

1970 Active Filters: Lumped, Distributed, Integrated, Digital, and Parametric, McGraw-Hill, Chapter 5.

Moore, William R.

1971 DARE1X Program Listings, Computer Science Research Laboratory, Electrical Engineering Dept., Univ. of Ariz.

Wait, John V.

1969a A Note on Designing Algorithms to Simulate Fixed Linear Systems Using Polynomial Input Approximations, CSRL Memo No. 196R, Revised 1970, Computer Science Research Laboratory, Electrical Engineering Dept., Univ. of Ariz.

1969b Proposed Simulation Methods Combining Linear Difference Equations and Numerical Integration CSRL Memo No. 191, Computer Science Research Laboratory, Electrical Engineering Dept., Univ. of Ariz.

1970 How to Use the CSRL Digital Computer, Vol. III, DARE I User's Manual, CSRL Memo No. 205, Computer Science Research Laboratory, Electrical Engineering Dept., Univ. of Ariz.

Wait, John V., Alexander Trevor, and James H. Puls

1969 A Conversational-Mode FORTRAN Program for Time-Domain Analysis of Fixed Linear Systems, ACL Memo No. 161, Electrical Engineering Dept., Univ. of Ariz.

Wiatrowski, Claude A.

1971 A Simulation Method Combining Linear Difference Equations and Numerical Integration, Unpublished Report, Computer Science Research Laboratory, Electrical Engineering Dept., Univ. of Ariz.