AD-A115 511 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 1/3
DESIGN OF ADVANCED DIGITAL FLIGHT CONTROL SYSTEMS VIA COMMAND G--ETC(U)
DEC 81 R M FLOYD
UNCLASSIFIED AFIT/GE/EE-81-20-VOL-2 NL

1 OF 3
AD
A115511

AD A115511

DDC

1

DTIC
ELECTE
JUN 14 1982
S
E
D

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved for public release and sale; its distribution is unlimited.

82 06 14 197

AFIT/GE/EE-81-20

(1)

DESIGN OF ADVANCED DIGITAL FLIGHT CONTROL SYSTEMS VIA COMMAND GENERATOR TRACKER (CGT) SYNTHESIS METHODS

THESIS VOLUME II

AFIT/GE/EE-81-20- Vol-2   Richard M. Floyd
Capt USAF

DTIC
SELECTE
JUN 14 1982
S E

Approved for public release; distribution unlimited

AFIT/GE/EE/81-20

DESIGN OF ADVANCED DIGITAL FLIGHT CONTROL SYSTEMS VIA COMMAND GENERATOR TRACKER (CGT) SYNTHESIS METHODS

THESIS VOLUME II

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Accession For

NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

by

Richard M. Floyd, B.S.

Capt USAF

Graduate Electrical Engineering

DTIC COPY INSPECTED 2

December 1981

Approved for public release; distribution unlimited

Contents

Page

List of Figures

Appendix A

CGTPIF Programmer's Guide

A.1 Introduction

CGTPIF is a controller design program which executes interactively. Three design paths are offered: (1) design of a Proportional-plus-Integral (PI) regulator via linear-quadratic (LQ) methodology; (2) design of a Command Generator Tracker, either open-loop (CGT) or closed-loop (CGT/PI); and (3) design of a Kalman filter (KF). These three designs are components of a final controller implemented as a Command Generator Tracker, with an inner-loop proportional-plus-integral regulator, and a Kalman filter for state estimation (CGT/PI/KF). For each design path there is a corresponding set of routines to evaluate the quality of the design achieved.

The program is written in FORTRAN IV and consists of about 2500 lines of source code. In addition, numerous routines are employed from a library of matrix routines described in Reference 24. Since the resulting program is large both in code and in memory utilization for array storage, direct and complete loading of the program exceeds memory limits for interactive execution on the ASD CYBER computer system. A technique referred to as "segmentation" is employed to provide selective loading of routines during execution so that memory usage remains within the limits for interactive execution.

This guide first discusses various general aspects of the program relevant to a programmer wishing to understand the code and wishing to obtain an executable object file. Later sections discuss the specific execution paths and the computations performed by each routine.

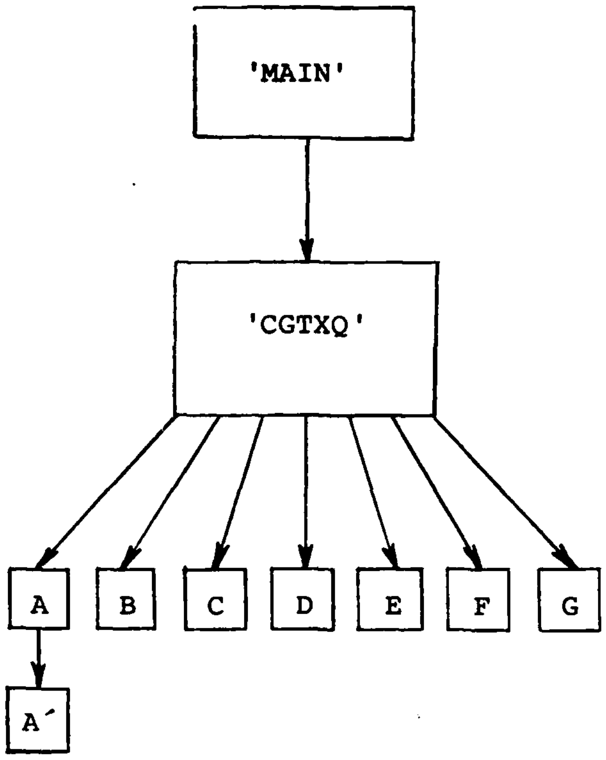
A.2 Program Structure

All of CGTPIF's execution logic and computations are embodied in a large set of routines which require no modification in order to apply the program to various different specific design problems. These invariant routines are referred to here collectively as CGTPIF SUBS. Among these routines, a single subroutine, CGTXQ, serves as the overall executive for program execution.

Additional routines comprising CGTPIF are the main program routine (MAIN) and various optional user-provided routines. MAIN simply defines temporary file names, allocates total array storage, then calls CGTXQ. CGTXQ then determines all execution options and calls the appropriate routines of CGTPIF SUBS. The optional routines are called from within the CGTPIF SUBS routines (at the user's discretion), and if not specifically needed for the design of interest may be omitted (i.e., the user need not provide "dummy" subroutines). IF CGTPIF is directed by the user to call such optional routines but which the user has not provided, dummy routines within CGTPIF SUBS are called. These dummy routines allow the call to be completed and signal CGTPIF that functional routines do not exist in the object file. Thus execution of the program is not affected if the user directs execution of optional routines that he has not implemented.

The available executable object file for CGTPIF provides specific array allocations and can handle systems with states and other vector variables dimensioned in the range of 10-20, approximately (the specific dimensionalities are given in a later section). In many cases the available CGTPIF will be directly applicable to a variety of different problems without modification. However, if the memory allocation is to be changed and/or if any of the optional routines are to be implemented, then these will require compilation. The CGTPIF SUBS routines would require no modification under these circumstances.

The general structure of CGTPIF is shown in Figure A-1. The blocks emanating from CGTXQ comprise the primary computational components of the program. At any given instant during program execution, the routines actually loaded in memory are MAIN, subroutine CGTXQ, and the subroutines associated with a single computational block called by CGTXQ. In addition, certain routines utilized by several different computational blocks are loaded in conjunction with CGTXQ.



A :	Establish Dynamics Model
A´:	Optional User Routines
B :	Controller Setup Computations
C :	Design PI Regulator
D :	Design CGT or CGT/PI Controller
E :	Evaluate Controller
F :	Design Kalman Filter
G :	Evaluate Filter

Fig. A-1. CGTPIF General Structure

A.3 Segmentation (Ref 13)

As mentioned above, only certain routines of CGTPIF are actually in memory at any time during execution. This selective loading is achieved using a CYBER loader option termed "Segmentation."

Segmentation is achieved using Job Control Language and segmentation directives. Source code requires no modification, and is simply compiled in the usual manner but without immediate execution. The object files for all source code and all library routines are then manipulated according to the segmentation directives to create a segmented object file. This object file may then be executed like any other executable file.

As the segmented file executes, segments of object code are loaded and unloaded to achieve the memory-resident program structure defined by the segmentation directives. All loader operations are performed automatically. For the user, execution proceeds as though the entire program were resident at all times.

For CGTPIF, the Job Control commands and the segmentation directives needed to achieve a segmented executable object file are invariant. A listing of the job commands is given in Appendix E. More detail on segmentation may be found in Reference 13.

A.4 Use of Library Routines

Routines described in Reference 24 are maintained in a program library in object form. CGTPIF employs many of the 'LIBRARY' routines in performing necessary computations.

The LIBRARY routines execute very efficiently. Array subscripting is single-indexed to reduce the overhead execution time incurred simply in computing array element addresses. However, as a byproduct of this single-indexing technique and the array storage mechanism of FORTRAN, the row dimension allocation within which arrays are stored must be the same for all arrays used by a library subroutine call (in some cases in which matrix transposes are involved, a column dimensioning constraint is also imposed). The routines included in the CGTPIF SUBS package accommodate these requirements on the effective row/column array allocations in each case that LIBRARY routines are employed.

Three named common blocks of CGTPIF are included to effect communication with the LIBRARY routines: /MAIN1/, /MAIN2/, and /INOU/. These provide two temporary arrays, two parameters related to the row dimension used for array storage, and three parameters defining files to be used for input/output (I/O). Later sections of this guide discuss these and all other common blocks in detail.

A.5 Array Storage

A significant characteristic of CGTPIF is its applicability to problems having a large variety of dimensionalities with system orders as great as 10 to 20. This is achieved by efficient techniques for array storage, adaptive addressing of individual arrays, and careful coding to avoid generating unnecessary temporary storage areas. The resulting code is not typical of the coding frequently encountered in matrix routines but is not in itself especially difficult to understand.

The basic principle in the array storage technique is simple. A small number of one-dimensional arrays are allocated corresponding to specific computational elements of the program. Within each allocated vector, individual arrays are stacked linearly according to the standard FORTRAN convention (storage by columns). Each array occupies only as many storage locations as required to contain all its elements, and for each a starting address in the appropriate linear stack is computed. Any array then can be located through its starting address in the larger vector. Thus within a given total allocation for each computational element, individual arrays of many different specific dimensions can be stored. Each array used can be considered "full" (the "allocated" dimensions and actual dimensions are identical).

The usual method for achieving variable array dimensioning involves specific fixed dimensioning of many individual arrays. Although this corresponds more with the ordinary conception of arrays and makes the code simple to write, there are disadvantages. Often the overall problem size which can be handled is smaller since all allocations assume the maximum value of every specified dimension is simultaneously attained. Also, problems having different sets of dimensions inconsistent with the fixed dimensions may cause individual array allocations to be exceeded while other arrays have enough excess storage locations to accommodate the need. But that storage is not free to be portioned out among the arrays suffering the short-fall and the problem cannot be accommodated.

In CGTPIF, many of the matrix computations work with arrays which are "in place" in the large vector storage areas. In cases in which augmented matrices are formed, arrays may be moved from permanent storage to form a partition of a new matrix. Also, in using the LIBRARY routines it is sometimes necessary to move arrays from their full storage mode to other temporary storage of larger row dimension. Finally, other arrays are sometimes moved from partitions of larger arrays to permanent full storage.

Figure A-2 illustrates several aspects of array storage using a model of a partitioned matrix $\underline{M}$. While it is represented in the figure as two-dimensional, storage is actually one-dimensional and CGTPIF works with single value addresses within $\underline{M}$. Note that $\underline{M}$ is in full storage

$$\underline{M} = \left[\begin{array}{c|c} \underline{A}_1 & \underline{A}_2 \\ \hline \underline{A}_3 & \underline{A}_4 \end{array}\right]$$

$\underline{A}_1$:	n-by-m
$\underline{A}_2$:	n-by-p
$\underline{A}_3$:	r-by-m
$\underline{A}_4$:	r-by-p
$\underline{M}$:	(n+r)-by-(m+p)

Fig. A-2. Partitioned Matrix $\underline{M}$

mode and $\underline{A}_1$ is stored in the manner typical of variable dimensioning (matrix $\underline{A}_1$ of dimension n-by-m stored in array $\underline{M}$ allocated (n+r)-by-(m+p) locations).

Suppose that array $\underline{M}$ is itself stored within a larger vector $\underline{V}$ and that the first element of $\underline{M}$ is at location LM in $\underline{V}$. Columns of $\underline{M}$ are stored in consecutive locations in $\underline{V}$. Figure A-3 shows $\underline{M}$'s first column within $\underline{V}$. Note the following addresses are equivalent:

$$\begin{aligned}
&\mathrm{ADDR}(\underline{V}(LM))=\mathrm{ADDR}(\underline{M}(1))=\mathrm{ADDR}(\underline{A}_1(1,1))\\
&\mathrm{ADDR}(\underline{V}(LM+n-1))=\mathrm{ADDR}(\underline{M}(n))=\mathrm{ADDR}(\underline{A}_1(n,1))\\
&\mathrm{ADDR}(\underline{V}(LM+n))=\mathrm{ADDR}(\underline{M}(n+1))=\mathrm{ADDR}(\underline{A}_3(1,1))\\
&\mathrm{ADDR}(\underline{V}(LM+n+r-1))=\mathrm{ADDR}(\underline{M}(n+r))=\mathrm{ADDR}(\underline{A}_3(r,1))\\
&\mathrm{ADDR}(\underline{V}(LM+n+r))=\mathrm{ADDR}(\underline{M}(n+r+1))=\mathrm{ADDR}(\underline{A}_1(1,2)) \qquad \text{(A-1)}
\end{aligned}$$

where 'ADDR' is an address function giving the absolute memory storage location.

Similarly, the addresses of all elements of matrices $\underline{A}_1$, $\underline{A}_2$, $\underline{A}_3$, and $\underline{A}_4$ have equivalents which specify the corresponding address within $\underline{V}$ and $\underline{M}$. In the moving of arrays mentioned previously and in computations involving arrays it is necessary that such equivalences among addresses be readily determined. CGTPIF includes several routines specifically dealing with such array manipulations.

Programmers often encounter difficulties in working with arrays that are not fixed in size. The array storage techniques employed in CGTPIF are readily understandable if careful thought is given to the actual arrangement of

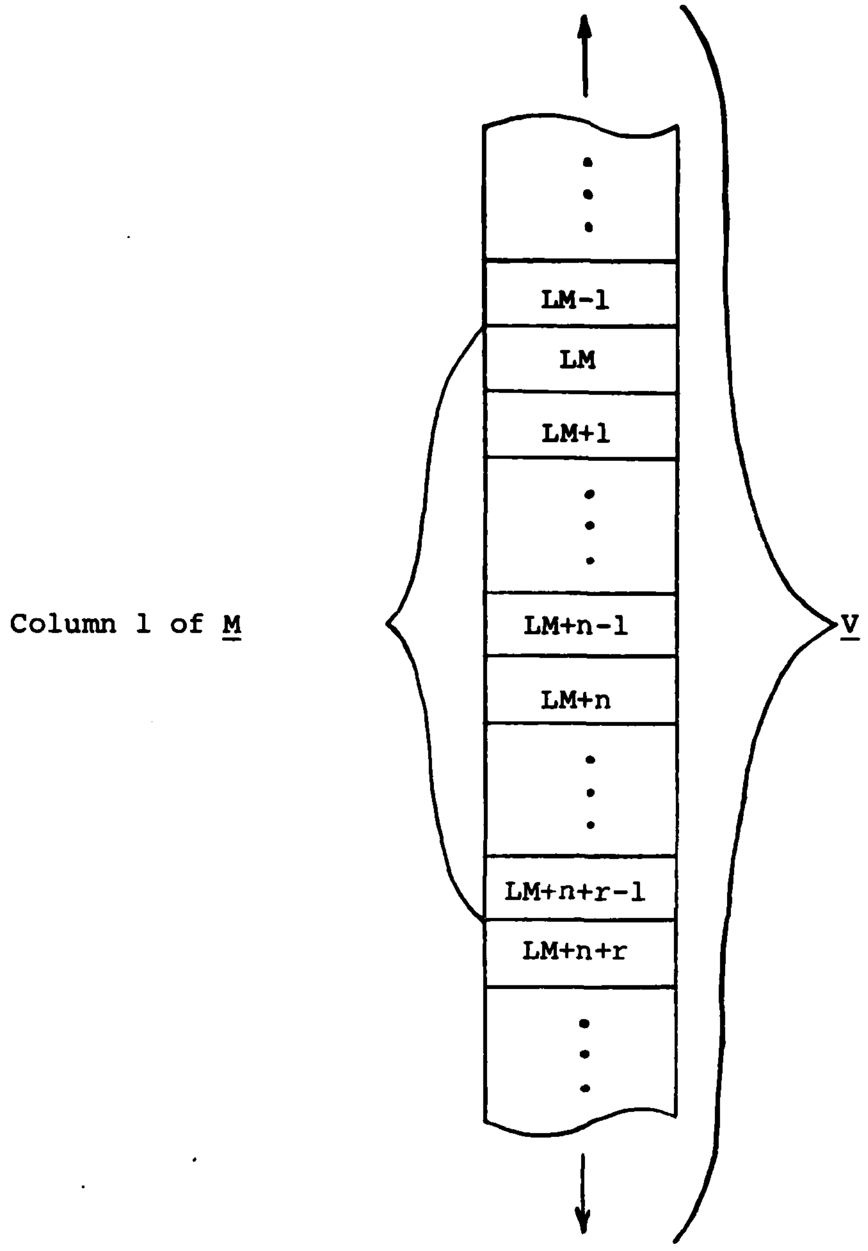


Fig. A-3. Column 1 of $\underline{M}$ within $\underline{V}$

arrays within program memory. If one has not previously considered such aspects of array storage in FORTRAN programs, it may be useful to determine various address equivalences among $\underline{A}_1$, $\underline{A}_2$, $\underline{A}_3$, $\underline{A}_4$, $\underline{M}$, and $\underline{V}$ of Figures A-2 and A-3.

A.6 Common Blocks

CGTPIF uses named Common blocks exclusively. A total of twenty-five Commons are used. Some provide communication with the LIBRARY routines, others communicate general program information, others provide temporary array storage, and others are associated with specific computational elements. The last-mentioned Commons will be discussed in groupings according to the computational element to which each group pertains. The elements of each Common are given here but will be described by type only (integer, real, scalar, vector). Information about array dimensioning is given in the discussion of the 'MAIN' routine. Specific definition of the elements of each Common are given in descriptions of the routines of CGTPIF SUBS.

A.6.1 LIBRARY Commons. Three Common blocks communicate with thè LIBRARY routines:

```
COMMON/MAIN1/NDIM,NDIM1,COM1
COMMON/MAIN2/COM2
COMMON/INOU/KIN,KOUT,KPUNCH
```

NDIM, NDIM1, KIN, KOUT, and KPUNCH are integer scalars. COM1 and COM2 are real arrays providing temporary storage, and are used occasionally for this purpose by CGTPIF also.

Further details are given in the discussion of the MAIN program.

A.6.2 General Commons. Two Commons communicate general information:

```
COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,
LFLKF,LTEVAL,LABORT

COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
```

All the variables are scalar and all but TSAMP are integer; TSAMP is a real. Further detail about the elements of each Common are included in the discussions of routines CGTXQ and MAIN for /DESIGN/ and /FILES/, respectively.

A.6.3 Temporary Storage Commons. Three Commons provide arrays for temporary storage:

```
COMMON/SYSMTX/NVSM,SM

COMMON/ZMTX1/NVZM,ZM1

COMMON/ZMTX2/ZM2
```

NVSM and NVZM are integer scalars. SM, ZM1, and ZM2 are real arrays. The dimensioning of the arrays is discussed in the description of MAIN.

A.6.4. Computational Element Commons. Sets of Common blocks are associated with computational elements A, B, C, D, and F of Figure A-1. More detail about the elements of each Common is given in the later sections describing the routines of each corresponding computational element.

A.6.4.1 Set A: Establish Dynamics Model. Three different dynamics models may be employed--for each, three Common blocks are used.

Design Model:

```
COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,
NWPNWD,NNPR

COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,
LC,LDY,LEY,LHP,LR

COMMON/DSNMTX/NVDM,NODY,NOEY,DM
```

Truth Model:

```
COMMON/NDIMT/NNT,NRT,NMT,NWT

COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT

COMMON/TRUMTX/NVTM,TM
```

Command Model:

```
COMMON/NDIMC/NNC,NRC,NPC

COMMON/LOCC/LPHC,LBDC,LCC,LDC

COMMON/CMDMTX/NVCM,NEWCM,NODC,CM
```

DM, TM, and CM are real arrays. All other variables are integer scalars. The various models are discussed in the next section of this guide.

A.6.4.2 Set B: Controller Setup. A pair of Commons is associated with the setup computations for the controller:

```
COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL

COMMON/CONTROL/NVCTL,CTL
```

CTL is a real array. All other variables are integer scalars.

A.6.4.3 Set C: Design PI Regulator.

A pair of Commons is used for the PI design:

```
COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ

COMMON/CREGPI/NVRPI,RPI
```

RPI is a real array. All other variables are integer scalars.

A.6.4.4 Set D: Design CGT or CGT/PI.

The design of the CGT or CGT/PI controller uses a pair of Commons:

```
COMMON/LCGT/LA11,LA13,LA21,LA23,LA12,LA22,LKXA11,
LKXA12,LKXA13

COMMON/CCGT/NVCGT,CGT
```

CGT is a real array. All other variables are integer scalars.

A.6.4.5 Set F: Design Kalman Filter.

The design of the Kalman filter uses a pair of Commons:

```
COMMON/LKF/LEADSN,LFLTRK,LFCOV

COMMON/CKF/NVFLT,FLT
```

FLT is a real array. All other variables are integer scalars.

A.7 Dynamics Models

CGTPIF employs three time-invariant dynamics models for computations: a "design" model, a "truth" model, and a "command" model. Each model is defined initially as a continuous-time system, then is discretized by CGTPIF. Any of the models may be established by user-provided subroutines, if desired. This section defines each model;

a later section discussing computational element A describes the manner in which the models may be entered into CGTPIF.

A.7.1 Design Model. The design model consists of a system state differential equation, a disturbance state differential equation, an output equation, and a measurement equation, as follow:

$$\dot{\underset{\sim}{x}}(t) = \underline{A}\underset{\sim}{x}(t) + \underline{B}\underline{u}(t) + \underline{E}_x\underset{\sim}{n}_d(t) + \underline{G}\underset{\sim}{w}(t) \qquad \text{(A-2a)}$$

$$\dot{\underset{\sim}{n}}_d(t) = \underline{A}_n\underset{\sim}{n}_d(t) + \underline{G}_n\underset{\sim}{w}_d(t) \qquad \text{(A-2b)}$$

$$\underset{\sim}{y}(t) = \underline{C}\underset{\sim}{x}(t) + \underline{D}_y\underline{u}(t) + \underline{E}_y\underset{\sim}{n}_d(t) \qquad \text{(A-2c)}$$

$$\underset{\sim}{z}(t_i) = \underline{H}\underset{\sim}{x}(t_i) + \underline{H}_n\underset{\sim}{n}_d(t_i) + \underset{\sim}{v}(t_i) \qquad \text{(A-2d)}$$

The under-tilde denotes the variable as being modeled as a random process. $\underset{\sim}{x}$ and $\underset{\sim}{n}_d$ are the Gaussian system and disturbance state vectors respectively; $\underset{\sim}{w}$ and $\underset{\sim}{w}_d$ are independent stationary zero-mean white Gaussian noises with covariances

$$E\{\underset{\sim}{w}(t)\underset{\sim}{w}^T(t+\tau)\} = \underline{Q}\delta(\tau) \qquad \text{(A-3a)}$$

$$E\{\underset{\sim}{w}_d(t)\underset{\sim}{w}_d^T(t+\tau)\} = \underline{Q}_n\delta(\tau) \qquad \text{(A-3b)}$$

The vectors $\underset{\sim}{y}$ and $\underset{\sim}{z}$ are the output and measurement vectors, respectively. $\underset{\sim}{v}$ is stationary zero-mean white Gaussian discrete-time noise independent of both $\underset{\sim}{w}$ and $\underset{\sim}{w}_d$ and of covariance

$$E\{\underset{\sim}{v}(t_i)\underset{\sim}{v}^T(t_j)\} = \underline{R}\delta_{ij} \qquad \text{(A-4)}$$

The dimensionalities for the design model are,

n = number of system states

r = number of system inputs

p = number of system outputs

m = number of system measurements

d = number of disturbance states

w = number of independent system noises

w_D = number of independent disturbance noises (A-5)

CGTPIF requires that the number of system inputs and outputs be equal: $p=r$. Also, the number of disturbance states cannot be greater than the number of system states: $d \leq n$ (due to setup for solution of the CGT equations of Section A.11.5, in which the maximum row dimension is assumed to be n).

The dimensions of the matrices defining the design model are given in row, column specification as

$\underline{A}$:	n-by-n
$\underline{B}$:	n-by-r
$\underline{E}_x$:	n-by-d
$\underline{G}$:	n-by-w
$\underline{Q}$:	w-by-w
$\underline{C}$:	p-by-n
$\underline{D}_y$:	p-by-r
$\underline{E}_y$:	p-by-d
$\underline{H}$:	m-by-n
$\underline{H}_n$:	m-by-d
$\underline{R}$:	m-by-m
$\underline{A}_n$:	d-by-d

$$\underline{G}_n : \text{d-by-}w_D$$

$$\underline{Q}_n : w_D\text{-by-}w_D \tag{A-6}$$

The design model is a dynamic model of the system for which the controller is to be designed. The Kalman filter will estimate the states of the design model and these will be employed by the controller for feedforward and feedback control.

A.7.2 Truth Model. The truth model consists of a state differential equation, a measurement equation, and two equations relating the system and disturbance states of the design model to the truth model states, as follow:

$$\dot{\underset{\sim}{x}}_t(t) = \underline{A}_t \underset{\sim}{x}_t(t) + \underline{B}_t \underline{u}_t(t) + \underline{G}_t \underset{\sim}{w}_t(t) \tag{A-7a}$$

$$\underset{\sim}{z}_t(t_i) = \underline{H}_t \underset{\sim}{x}_t(t_i) + \underset{\sim}{v}_t(t_i) \tag{A-7b}$$

$$\underset{\sim}{x}'(t) = \underline{T}_{DT} \underset{\sim}{x}_t(t) \tag{A-7c}$$

$$\underset{\sim}{n}'_d(t) = \underline{T}_{NT} \underset{\sim}{x}_t(t) \tag{A-7d}$$

with $\underset{\sim}{x}_t$ the truth model state and modeled as a Gaussian random process. $\underset{\sim}{x}'$ and $\underset{\sim}{n}'_d$ correspond to states of the design model (equation (A-2)). $\underset{\sim}{w}_t$ and $\underset{\sim}{v}_t$ are independent stationary zero-mean white Gaussian continuous and discrete-time noises with covariances

$$E\{\underline{w}_t(t)\underline{w}_t^T(t+\tau)\} = \underline{Q}_t \delta(\tau) \tag{A-8a}$$

$$E\{\underline{v}_t(t_i)\underline{v}_t^T(t_j)\} = \underline{R}_t \delta_{ij} \tag{A-8b}$$

The dimensionalities for the truth model are,

n_T = number of system states

r_T = number of system inputs

m_T = number of system measurements

w_T = number of independent noises (A-9)

CGTPIF requires that the numbers of measurements and of inputs be equal for both truth and design models: $m_T = m$ and $r_T = r$.

The dimensions of the matrices defining the truth model are given in row, column specification as

$\underline{A}_t$: n_T-by-n_T

$\underline{B}_t$: n_T-by-r_T

$\underline{G}_t$: n_T-by-w_T

$\underline{Q}_t$: w_T-by-w_T

$\underline{H}_t$: m_T-by-n_T

$\underline{R}_t$: m_T-by-m_T

$\underline{T}_{DT}$: n-by-n_T

$\underline{T}_{NT}$: d-by-n_T (A-10)

The truth model represents the same dynamic system as the design model, but generally may be of greater dimension and complexity. It is intended to provide as complete and accurate a description as possible of the system dynamics, consistent with the design objectives.

A.7.3 <u>Command</u> <u>Model</u>. The command model is defined by a state differential equation and an output equation:

$$\underline{\dot{x}}_m(t) = \underline{A}_m\underline{x}_m(t) + \underline{B}_m\underline{u}_m(t) \quad \text{(A-11a)}$$

$$\underline{y}_m(t) = \underline{C}_m\underline{x}_m(t) + \underline{D}_m\underline{u}_m(t) \quad \text{(A-11b)}$$

The dimensionalities of the command model are,

n_M = number of model states

r_M = number of model inputs

p_M = number of model outputs (A-12)

CGTPIF requires that the numbers of outputs of the command and design models be equal: $p_M = p$. Also, the number of command model states cannot be greater than the number of system states of the design model: $n_M \leq n$ (due to setup for solution of the CGT equation of Section A.11.5 in which the maximum row dimension is assumed to be n).

The dimensions of the matrices defining the command model are given in row, column specification as

$\underline{A}_m$: n_M-by-n_M

$\underline{B}_m$: n_M-by-r_M

$\underline{C}_m$: p_M-by-n_M

$\underline{D}_m$: p_M-by-r_M (A-13)

The command model represents the dynamics that the controlled system is intended to follow. Typically it is of relatively low dimension since the desired dynamics are usually characterized by first- or second-order descriptions.

A.8 File Usage

In addition to the input/output (I/O) communication directly with the user terminal, CGTPIF uses four files for I/O. The 'DATA' file is used for input, files 'SAVE' and 'LIST' are used for output, and file 'PLOT' is used for input and output. Because of the close relationship between the SAVE and DATA files, they are discussed first.

A.8.1 SAVE File. During program execution the user may direct CGTPIF to write any of the system models to the SAVE file. If the PI design path has been executed, then the existing sets of PI gains are automatically written to SAVE just prior to program termination. An integer code number written along with each output to the file identifies each set of data: the design, command, and truth models are codes 1, 2, and 3, respectively; the PI gains are code 4. A code of -1 is written to indicate that no more data is on the file.

A.8.2 DATA File. A previously created SAVE file may be given the local file name DATA. During program execution, CGTPIF can be directed to read system models and PI gains from DATA as needed. If the data sought by CGTPIF is not on the DATA file, a message is written to the terminal and execution proceeds on an alternative path, as appropriate.

A.8.3 LIST File. During program execution results of computations are output to the LIST file under format direction. After program execution is stopped, LIST may be routed to a line-printer for listing.

A.8.4 PLOT File. The PLOT file is used by CGTPIF during controller and filter evaluations. Variables derived from time-response simulations are written to PLOT at each time sample. When the time-response run is complete, selected variables are read from PLOT to generate line-printer plots of the results.

A.9 Description of Routine 'MAIN'

MAIN specifies files to be used and their FORTRAN unit designations (e.g., 'INPUT' is unit 5); it allocates all array dimensions for the Common blocks and calls subroutine CGTXQ. A listing of MAIN is in Appendix D.

The appropriate unit designations for files SAVE, DATA, PLOT, LIST, and of the user terminal are set in the variables KSAVE, KDATA, KPLOT, KLIST, and KTERM, respectively of the /FILES/ Common. The variable KIN of /INOU/ is set to the unit designator for the INPUT file.

Array allocation requires two steps: arrays are allocated by specifying a vector length for each array in its common declaration; the length of each array is then set in an appropriate integer variable which communicates array allocations to CGTPIF SUBS through the Commons.

Denoting the integer vector lengths allocated for the various individual Common arrays as n_1, n_2, n_3, ..., n_{10}, arrays are allocated as follows:

COMMON/MAIN1/NDIM,NDIM1,COM1(n_1)
COMMON/MAIN2/COM2(n_1)
COMMON/SYSMTX/NVSM,SM(n_2)
COMMON/ZMTX1/NVZM,ZM1(n_3)
COMMON/ZMTX2/ZM2(n_3)
COMMON/DSNMTX/NVDM,NODY,NOEY,DM(n_4)
COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(n_5)
COMMON/TRUMTX/NVTM,TM(n_6)
COMMON/CONTROL/NVCTL,CTL(n_7)
COMMON/CREGPI/NVRPI,RPI(n_8)
COMMON/CCGT/NVCGT,CGT(n_9)
COMMON/CKF/NVFLT,FLT(n_{10})

Note that the arrays of /MAIN1/ and /MAIN2/ have the same allocations (n_1); /ZMTX1/ and /ZMTX2/ also have the same allocations (n_3).

The corresponding statements setting the integer variables to the array allocations are

NDIM = n_1
NVSM = n_2
NVZM = n_3
NVDM = n_4
NVCM = n_5
NVTM = n_6

NVCTL = n_7

NVRPI = n_8

NVCGT = n_9

NVFLT = n_{10}

The allocations needed for each array are expressed as functions of the system dimensions of equations (A-5), (A-10), and (A-12). In the equations to follow, 'MAX' is a function which takes the largest value from among its arguments. Also, the following names will be used to represent certain sums of dimensions (these names correspond to mnemonics employed within CGTPIF, e.g., "npld" mnemonically represents "n plus d" and "na" represents "n augmented").

$$\text{npld} = n+d \qquad \text{(A-14a)}$$

$$\text{nnpr} = n+r \qquad \text{(A-14b)}$$

$$\text{nwpnwd} = w+w_D \qquad \text{(A-14c)}$$

$$\text{na} = n+d+n_t \qquad \text{(A-14d)}$$

The array allocations needed are,

$$\text{/MAIN1/, /MAIN2/: } n_1 \geq \text{MAX}\{[\text{MAX}(\text{npld},\text{nnpr})]^2, n_t^2\} \qquad \text{(A-15a)}$$

$$\text{/SYSMTX/: } n_2 \geq \text{MAX}\{606,$$

$$\begin{bmatrix} n(\text{npld}+r+p+m+w) + p(r+d) + m(m+d) \\ +d(d+w_D) + w^2 + w_D^2 \end{bmatrix},$$

$$[(n_M+p_M)\ (n_M+r_M)],$$

$[n_t(n_t+r_t+m_t+w_t+npld) + m_t^2 + w_t^2]$,

$[n(3n+2MAX(d,n_M)) + p(n_M)]$,

$[nnpr(3nnpr+r)]$,

$na^2\}$ (A-15b)

/ZMTX1/, /ZMTX2/: $n_3 \geq MAX[n_1, na^2]$ (A-15c)

/DSNMTX/: $n_4 \geq npld(2npld+n+p+m+nwpnwd) + r(n+p) + m^2 + d^2 + w^2 + w_D^2$ (A-15d)

/CMDMTX/: $n_5 \geq n_M(n_M+r_M+p_M) + r_M(p_M)$ (A-15e)

/TRUMTX/: $n_6 \geq n_t(2n_t+npld+m+r) + m^2$ (A-15f)

/CONTROL/: $n_7 \geq nnpr(2nnpr+p)$ (A-15g)

/CREGPI/: $n_8 \geq r(4r+n) + nnpr^2$ (A-15h)

/CCGT/: $n_9 \geq (n+2p)\ (n_m+r_m+d)$ (A-15i)

/CKF/: $n_{10} \geq npld[(2npld+m)+1]$ (A-15j)

Routines of CGTPIF SUBS which use these arrays employ these equations to verify sufficient allocation has been provided. If not, a message is written which specifies the array in question and the necessary allocation; execution then is aborted.

The MAIN listed in Appendix D can accommodate problems of dimensions given as follow:

$n \leq 15$

$npld \leq 15$

$r \leq 5$

$p \leq 5$

$m \leq 15$

$w \leq 15$

$nwpnwd \leq 15$

$n_M \leq 10$

$r_M \leq 5$

$p_M \leq 5$

$n_t \leq 20$

$r_t \leq 5$

$m_t \leq 15$

$w_t \leq 20$ (A-16)

In these expressions, the substitutions of equations (A-14a) and (A-14c) have been used to impose constraints on the total number of design model system and disturbance states. These allocations are sufficient for problems all of whose dimensions are equal to the numbers given in equation (A-16). Moreover, other combinations of dimensions, some greater than and some less than these specific dimensions, will also be accommodated. For the set of dimensions appropriate to one's design problem, the equations of equation set (A-15) may be used to determine if existing allocations are adequate; or the problem may be attempted and CGTPIF will signal any inadequacies in available allocations (if any exist). The specific values of allocations (n_1 through n_{10}) given by the MAIN or Appendix D are,

$$\begin{aligned}
n_1 &= 400 \\
n_2 &= 2125 \\
n_3 &= 1225 \\
n_4 &= 1750 \\
n_5 &= 225 \\
n_6 &= 1725 \\
n_7 &= 900 \\
n_8 &= 575 \\
n_9 &= 400 \\
n_{10} &= 690
\end{aligned} \qquad \text{(A-17)}$$

Since there are two arrays of length n_1 and also two of length n_3, this represents a total of $(11640)_{10}$ words of memory for array storage. As implemented in segmented form, the memory utilized during execution is the sum of the memory required by the largest of the load segment sets and the memory required by the loader itself (about $(10000)_8$ words). The arrays allocated by MAIN are always in memory. The largest set of segments loaded at any time includes the segment which utilizes optional user-provided routines (described in the next section). Thus the total array allocations and the memory required to implement any optional routines effectively determine the execution load size attained by CGTPIF. For the CYBER system and interactive execution, the total memory which is available for array storage and optional routines is about $(13000)_{10}$ words.

A.10 Optional Routines: Define Dynamics Models

The user may choose to enter any of the three dynamics models by using subroutines. Each model definition requires at least two specific subroutines. These subroutines may then call any additional routines to accomplish the necessary computations--routines of CGTPIF SUBS, LIBRARY, or any user-provided subroutines may be used. In the listing of Appendix D, DSND, DSNM, TRTHD, TRTHM, ACDATA, GUSTS, and TBLUP1 are all optional routines used to establish design and truth models of the longitudinal dynamics of an aircraft subject to atmospheric turbulence.

For each model defined by subroutines, one subroutine must establish the dimensions of the model, and another must set the values for all matrices of that model. Each routine must have the appropriate name and argument list specified below. All model arrays appearing in the argument lists must be allocated in full manner: the array dimensions specified by "Dimension" statements within the routines must be exactly those implied by the routine specifying model dimensionalities and the array sizing given by equations (A-6), (A-10), and (A-13). For example, if the number of design model states (n) is established as 10, then according to equation (A-6) the system matrix must be explicitly dimensioned A(10,10) in the subroutine which sets array values for the design model. All of these arrays are initialized to zero before the array setting routines are called, so it is necessary only to set non-zero array elements within the subroutines. Any arrays of dimension one in both row and column are actually scalars and need not be included in a Dimension statement. Any arrays with row or column dimension of zero are in fact nonexistent arrays and must not be included in Dimension statements, although they still must be included in the subroutine's argument list (since calls to these routines from within CGTPIF assume full argument lists).

A.10.1 Design Model. The two routines required for the design model are 'DSND' and 'DSNM'. The first specifies dimensions of the model while the second sets the array elements for that model.

DSND has a single argument:

```
SUBROUTINE DSND(ND)
```

with ND an integer vector of length seven. In DSND the elements of ND are set to the dimensions given by equation (A-5) and in the order shown. Thus, for example, element 1 is set to the value n, element 2 is set to the value r, and so on.

DSNM has 14 arguments:

```
SUBROUTINE DSNM(A,B,EX,G,Q,C,DY,EY,H,HN,R,AN,
   GN,QN)
```

Each argument is an array defined in equation (A-2), (A-3), or (A-4). Note the order in which the arrays appear in the argument list is the same as the order of the arrays listed in equation (A-6) and the dimensions given in that equation must be specified in DSNM. Thus, for example, if DSND sets n=5 then the matrix $\underline{A}$ must be dimensioned A(5,5) in DSNM.

A.10.2 Truth Model. The two routines required for the truth model are 'TRTHD' and 'TRTHM'. The first specifies dimensions of the model while the second sets the array elements for that model

TRTHD has a single argument:

```
SUBROUTINE TRTHD(ND)
```

with ND an integer vector of length four. In TRTHD the elements of ND are set to the dimensions given by equation (A-9) and in the order shown.

TRTHM has 8 arguments:

```
SUBROUTINE TRTHM(AT,BT,GT,QT,HT,RT,TDT,TNT)
```

Each argument is an array defined in equation (A-7) or (A-8). Note the order in which the arrays appear in the argument list is the same as the order of the arrays listed in equation (A-10) and the dimensions given in that equation must be specified in TRTHM.

A.10.3 Command Model. The two routines required for the command model are 'CMDD' and 'CMDM'. The first specifies dimensions of the model while the second sets the array elements for that model.

CMDD has a single argument:

SUBROUTINE CMDD(ND)

with ND an integer vector of length three. In CMDD the elements of ND are set to the dimensions given by equation (A-12) and in the order shown.

CMDM has 4 arguments:

SUBROUTINE CMDM(AM,BM,CM,DM)

Each argument is an array defined in equation (A-11). Note the order in which the arrays appear in the argument list is the same as the order of the arrays listed in equation (A-13) and the dimensions given in that equation must be specified in CMDM.

A.11 CGTPIF SUBS

In contrast to the routines described in Sections A.9 and A.10, the routines of CGTPIF SUBS require no modification to apply to specific design problems. In discussing CGTPIF SUBS some detail as to the operation of specific routines is given. For users who may elect to attempt modification of routines, a detailed examination of the source code is essential.

The executive routine for CGTPIF SUBS-CGTXQ- is discussed first. Each major computational element and constituent routines are then discussed in turn.

A.11.1 CGTXQ. The overall execution logic of CGTPIF is determined by routine 'CGTXQ'. Specific execution of the designs is achieved by calls to other individual routines, which are in turn executives to routines comprising the various computational elements shown in Figure A-1.

Figure A-4 gives a flowchart of CGTXQ which emphasizes the major program decisions. Blocks representing calls to the particular computational elements give (1) the name of the routine which is executive to that element, and (2) the letter code used in Figure A-1 to represent that element.

All "flag" variables used by CGTXQ and CGTPIF SUBS are integers. A value of zero implies that the condition flagged is not true. While a non-zero value generally implies the condition is true, positive and negative values sometimes distinguish between different attributes of that condition. Flags which pertain to general program logic are included in the /DESIGN/ Common; flags which relate strictly to specific computational elements are passed as arguments in calls to the respective executive routines.

The elements of Common /DESIGN/ are defined as

"NVCOM":	The smaller of the array allocations of /MAIN1/ and /ZMTX1/. Throughout much of CGTPIF SUBS, the same array sizes are needed for COM1, COM2, ZM1, and ZM2. NVCOM is tested to determine if sufficient allocation is available for the temporary arrays.
"TSAMP":	The controller sample period (in seconds).

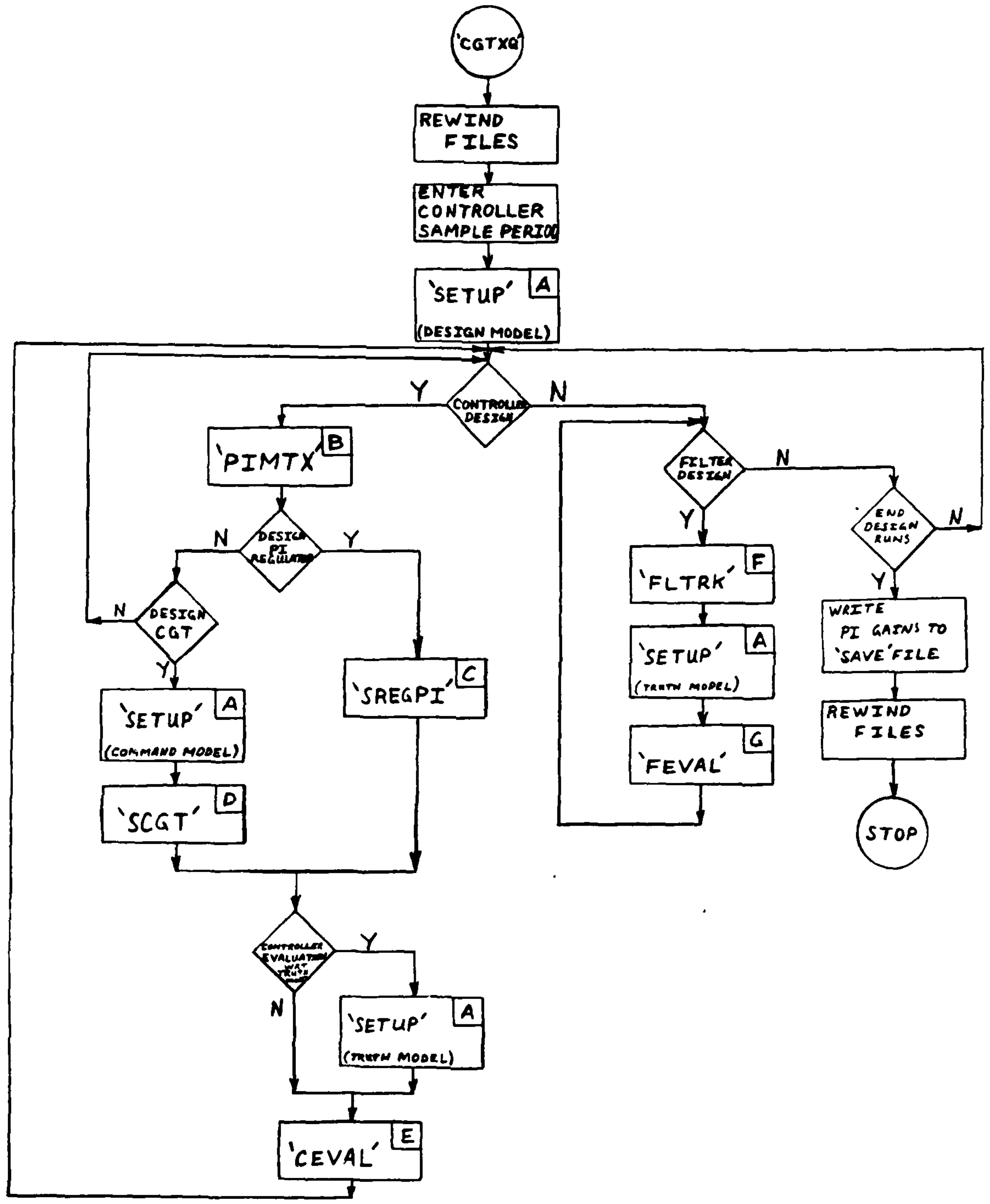


Fig. A-4. CGTXQ Flowchart

"LFLRPI":	Flag variable indicating availability of PI gains in program storage. A value of 0 means the gains are not available. Values of -1 and +1 mean the gains are available and have been obtained either from the DATA file or by computation in the current program execution, respectively.
"LFLCGT":	Flag variable indicating if CGT design/evaluation is in execution. A value of 1 means a CGT design has been determined, while values of 0 and -1 mean the converse. Moreover, a value of -1 signifies that an open-loop CGT design is infeasible (PI gains not available and design system unstable).
"LFLKF":	Flag variable indicating filter design is in execution.
"LTEVAL":	Flag variable indicating controller evaluation is with respect to truth model.
"LABORT":	Flag variable indicating execution abort status. If LABORT is positive then execution will abort due to insufficient array allocation, and the specific value is the allocation needed. If LABORT is negative, the abort is due to dimensional incompatibility as mentioned in Section A.7 for each model. If the incompatibility affects the design model the program aborts execution; for the other models only the specific execution path is aborted.

"IPI", "ICGT", "ITRU", and "IFLTR" are additional flags related to specific computations. IPI and IFLTR test the successful execution of the computations of routines 'PIMTX' and 'FLTRK', respectively. The other two flags have values according to:

"ICGT":	Flag tests if command model is established. A non-zero value indicates the command model is established, and if negative that it has also been written to the SAVE file. If the command model is not established, ICGT is zero.
"ITRU":	Flag tests if truth model is established. Specific values have the same significance as for ICGT, but with respect to the truth model.

CGTXQ includes other decision tests not shown in Figure A-4. These are not discussed since they involve obvious tests on the flags defined above and the code is simple.

A.11.2 <u>SETUP</u>. Routine 'SETUP' serves as an intermediary in establishing the various dynamic models used by CGTPIF. It calls one of three other routines according to the value of its input argument "ITYPE". The design, command, or truth models are established for ITYPE = 1, 2, or 3, respectively.

The routines 'SDSN', 'SCMD', and 'STRTH' actually establish each of the models. Each uses the routine 'RSYS' to enter the continuous-time model representation. The model's arrays are stored initially in vector SM of /SYSMTX/ Common. Each model is discretized (using the sample period T = TSAMP) and the new arrays defining the discrete-time models are stored in permanent vectors DM, CM, or TM of the /DSNMTX/, /CMDMTX/, or /TRUMTX/ Commons, respectively.

The continuous-time models are entered with subroutine RSYS. The models may be entered directly from the user terminal, from the DATA file, or using optional user-provided routines described in Section A.10.

The computations performed under SDSN, SCMD, and STRTH are discussed below. The routines which perform each computation are indicated following the equations.

A.11.2.1 <u>SDSN</u>. SDSN calls RSYS to read in the dimensions of the design model and the arrays defining it. The dimensions are stored in the variables of /NDIMD/. The first seven variables are the dimensions of equation (A-5) in order and the final three are the sums of dimensions of equations (A-14a), (A-14c), and (A-14b), respectively. A call to 'DSCRTD' then gives the discretized model.

An augmented system description is formed with the system and disturbance states:

$$\underset{\sim}{\underline{x}}_a = \begin{bmatrix} \underset{\sim}{\underline{x}}(t) \\ \underset{\sim}{\underline{n}}_d(t) \end{bmatrix} \quad , \quad \underset{\sim}{\underline{w}}_a(t) = \begin{bmatrix} \underset{\sim}{\underline{w}}(t) \\ \underset{\sim}{\underline{w}}_d(t) \end{bmatrix} \qquad \text{(A-18)}$$

and partitioned matrices describing the dynamics of the augmented system are formed:

$$\underline{A}_a = \left[\begin{array}{c|c} \underline{A} & \underline{E}_x \\ \hline \underline{O} & \underline{A}_n \end{array}\right] \tag{A-19a}$$

$$\underline{B}_a = \left[\begin{array}{c} \underline{B} \\ \hline \underline{O} \end{array}\right] \tag{A-19b}$$

$$\underline{G}_a = \left[\begin{array}{c|c} \underline{G} & \underline{O} \\ \hline \underline{O} & \underline{G}_n \end{array}\right] \tag{A-19c}$$

$$\underline{Q}_a = \left[\begin{array}{c|c} \underline{Q} & \underline{O} \\ \hline \underline{O} & \underline{Q}_n \end{array}\right] \tag{A-19d}$$

with component matrices defined in equations (A-2) and (A-3). Matrices $\underline{A}_a$ and $\underline{G}_a$ are stored permanently in vector "DM" for reuse in Kalman filter design.

The corresponding discrete-time augmented state transition model is,

$$\underset{\sim}{\underline{x}}_a(t_{i+1}) = \underline{\Phi}_a \underset{\sim}{x}_a(t_i) + \underline{B}_{a_d} \underline{u}(t_i) + \underset{\sim}{w}_{a_d}(t_i) \tag{A-20}$$

where, assuming $\underline{u}$ is constant over a sample period,

$$\underline{\Phi}_a = e^{\underline{A}_a T} \tag{A-21a}$$

$$\underline{B}_{a_d} = \int_0^T \underline{\Phi}_a(T-\tau)\underline{B}_a d\tau \qquad \text{(A-21b)}$$

where $\underline{\Phi}_a(T-\tau)=e^{\underline{A}_a(T-\tau)}$ and the strength of $\underset{\sim}{\underline{w}}_{a_d}$ is given by

$$\underline{Q}_{a_d} = \int_0^T \underline{\Phi}_a(T-\tau)\underline{G}_a\underline{Q}_a\underline{G}_a^T\underline{\Phi}_a^T(T-\tau)\,d\tau \qquad \text{(A-21c)}$$

Matrix $\underline{\Phi}_a$ is stored permanently in vector "FLT" of /CKF/ and $\underline{Q}_{a_d}$ is stored permanently in vector DM.

$\underline{\Phi}_a$ and $\underline{B}_{a_d}$ may be partitioned to the component dimensions to yield

$$\underline{\Phi}_a = \left[\begin{array}{c|c} \underline{\Phi} & \underline{E}_{x_d} \\ \hline \underline{O} & \underline{\Phi}_n \end{array}\right] \qquad \text{(A-22a)}$$

$$\underline{B}_{a_d} = \left[\begin{array}{c} \underline{B}_d \\ \hline \underline{O} \end{array}\right] \qquad \text{(A-22b)}$$

Matrices $\underline{\Phi}$, $\underline{E}_{x_d}$, $\underline{\Phi}_n$, and $\underline{B}_d$ are stored permanently in vector DM. The deterministic discrete-time design model then is,

$$\underline{x}(t_{i+1}) = \underline{\Phi x}(t_i) + \underline{B}_d\underline{u}(t_i) + \underline{E}_{x_d}\underline{n}_d(t_i) \qquad \text{(A-23a)}$$

$$\underline{n}_d(t_{i+1}) = \underline{\Phi}_n\underline{n}_d(t_i) \qquad \text{(A-23b)}$$

$$\underline{y}(t_i) = \underline{Cx}(t_i) + \underline{D}_y\underline{u}(t_i)+\underline{E}_y\underline{n}_d(t_i) \qquad \text{(A-23c)}$$

Matrices $\underline{C}$, $\underline{D}_y$, and $\underline{E}_y$ are as originally defined and are retained in vector DM. Equations (A-23a-c) are used to propagate the time response for the design model in the controller evaluation routines.

An augmented measurement matrix is formed and stored in vector DM:

$$\underline{H}_a = [\underline{H} \mid \underline{H}_n] \qquad \text{(A-24)}$$

where $\underline{H}$ and $\underline{H}_n$ are as in equation (A-2d).

The noise strengths $\underline{Q}$, $\underline{Q}_n$, and $\underline{R}$ of equations (A-3) and (A-4) are also stored in vector DM so that they are available for modification in the Kalman filter design path.

To avoid unnecessary computations in later code, if matrix $\underline{E}_y$ does not exist or if $\underline{D}_y$ or $\underline{E}_y$ are zero matrices then the variables "NODY" or "NOEY" (/DSNMTX/) are set to 1 as appropriate. In other circumstances these variables are zero and computations involving these arrays are carried out.

Equations (A-18), (A-19), (A-21), (A-22), and (A-24) are computed under the direction of DSCRTD. Routine 'QDSCRT' (called by DSCRTD) forms the partitioned matrix $\underline{Q}_a$ and computes the matrix $\underline{Q}_{a_d}$ using the LIBRARY routine 'INTEG'. DSCRTD computes $\underline{\Phi}_a$ and $\underline{B}_{a_d}$ using a call to the LIBRARY routine 'DSCRT'.

A.11.2.2 SCMD. SCMD begins by testing if PI regulator gains are available. If they are, a CGT/PI design will be pursued by routine 'SCGT'. If not already available, the user may choose that the PI gains be read from the DATA file. If the gains are not available and the system is stable then an open-loop CGT design will be pursued by SCGT; otherwise, if not stable, no CGT design is allowed and SCMD is exited. The logic is represented in the flowchart of Figure A-5. The remainder of SCMD is indicated by the block "establish command model" and is described below.

The command model may be established repeatedly during program execution. It is entered with a call to routine RSYS. The dimensions of the model are stored in the variables of /NDIMC/ and in the order shown in equation (A-12). A call to 'DSCRTC' then gives the discretized model:

$$\underline{x}_m(t_{i+1}) = \underline{\Phi}_m \underline{x}_m(t_i) + \underline{B}_{m_d} \underline{u}_m(t_i) \qquad \text{(A-25a)}$$

$$\underline{y}_m(t_i) = \underline{C}_m \underline{x}_m(t_i) + \underline{D}_m \underline{u}_m(t_i) \qquad \text{(A-25b)}$$

where, for $\underline{u}_m$ constant over a sample period

$$\underline{\Phi}_m = e^{\underline{A}_m T} \qquad \text{(A-26a)}$$

$$\underline{B}_{m_d} = \int_0^T \underline{\Phi}_m(T-\tau)\underline{B}_m d\tau \qquad \text{(A-26b)}$$

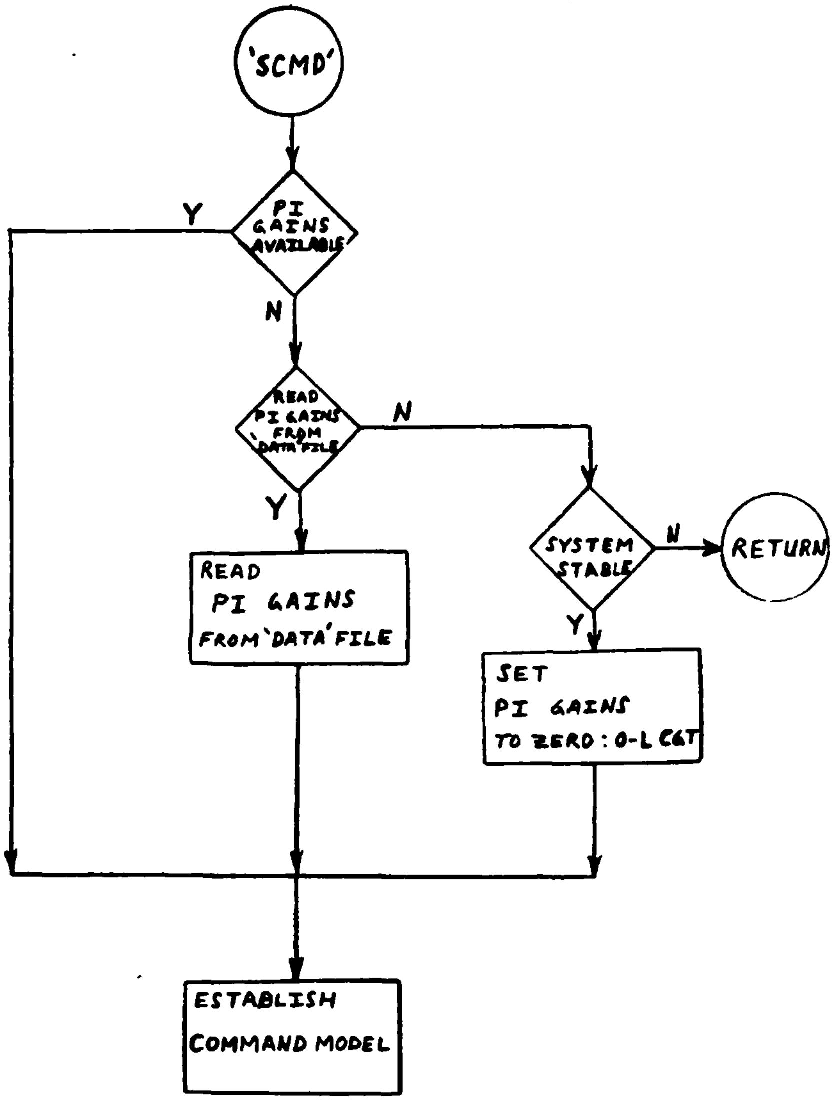

'SCMD'
PI GAINS AVAILABLE
Y
N
READ PI GAINS FROM 'DATA' FILE
N
Y
READ PI GAINS FROM 'DATA' FILE
SYSTEM STABLE
N
RETURN
Y
SET PI GAINS TO ZERO: O-L C&T
ESTABLISH COMMAND MODEL

Fig. A-5. SCMD Entry Logic

and $\underline{A}_m$, $\underline{B}_m$, $\underline{C}_m$, and $\underline{D}_m$ are as defined in equation (A-11).

Matrices $\underline{\Phi}_m$ and $\underline{B}_{m_d}$ are computed with a call to routine DSCRT of the LIBRARY. $\underline{\Phi}_m$, $\underline{B}_{m_d}$, $\underline{C}_m$, and $\underline{D}_m$ are stored in vector "CM" of /CMDMTX/. Equations (A-25a,b) are used in propagating the model states and outputs in the controller evaluation routines.

In /CMDMTX/, the variable "NEWCM" signals that a new command model is being established (NEWCM non-zero). If the matrix $\underline{D}_m$ is zero, the variable "NODC" is set to a non-zero value.

A.11.2.3 STRTH. The continuous-time truth model is established with a call to routine RSYS. During program execution the truth model can be redefined as often as desired. The dimensions of the model are stored in the variables of /NDIMT/ in the order shown in equation (A-9). A call to 'DSCRTT' discretizes the truth model:

$$\underset{\sim}{x}_t(t_{i+1}) = \underline{\Phi}_t \underset{\sim}{x}_t(t_i) + \underline{B}_{t_d} \underline{u}_t(t_i) + \underset{\sim}{w}_{t_d}(t_i) \tag{A-27}$$

where, for $\underline{u}_t$ constant over a sample period,

$$\underline{\Phi}_t = e^{\underline{A}_t T} \tag{A-28a}$$

$$\underline{B}_{t_d} = \int_0^T \underline{\Phi}_t(T-\tau)\underline{B}_t d\tau \tag{A-28b}$$

and the strength of the noise $\underline{w}_{t_d}$ is,

$$\underline{Q}_{t_d} = \int_0^T \underline{\Phi}_t(T-\tau)\underline{G}_t\underline{Q}_t\underline{G}_t^T\underline{\Phi}_t^T(T-\tau)\,d\tau \tag{A-28c}$$

Matrices $\underline{\Phi}_t$, $\underline{B}_{t_d}$, $\underline{Q}_{t_d}$ as well as $\underline{H}_t$, $\underline{R}_t$, $\underline{T}_{DT}$, and $\underline{T}_{NT}$ are stored in vector "TM" of /TRUMTX/.

DSCRTT computes $\underline{\Phi}_t$ and $\underline{B}_{t_d}$ using routine DSCRT of LIBRARY. $\underline{Q}_{t_d}$ is computed using routine INTEG.

A.11.3 PIMTX. Computations that are necessary to the controller designs but independent of design iteration for a fixed design model are computed under the direction of PIMTX. The input argument IPI is set to 1 following successful computation; subsequent entries into PIMTX then test IPI and return immediately without recomputation of the information.

PIMTX forms an augmented matrix and then forms its inverse. The resulting matrix is termed the $\underline{\Pi}$ matrix. Partitions of the $\underline{\Pi}$ matrix into sub-arrays of the original component dimensions are then stored individually in the vector "CTL" of /CONTROL/.

$$\underline{\Pi} = \left[\begin{array}{c|c} (\underline{\Phi}-\underline{I}) & \underline{B}_d \\ \hline \underline{C} & \underline{D}_y \end{array}\right]^{-1} \tag{A-29a}$$

$$\underline{\Pi} = \left[\begin{array}{c|c} \underline{\pi}_{11} & \underline{\pi}_{12} \\ \hline \underline{\pi}_{21} & \underline{\pi}_{22} \end{array}\right] \tag{A-29b}$$

Matrices $\underline{\Phi}$, $\underline{B}_d$, $\underline{C}$, and $\underline{D}_y$ are as defined in equation (A-23). The $\underline{\Pi}$ matrix is used in computations for both the PI and CGT controllers.

PIMTX then calls routine 'CDIF', which sets up two augmented matrices for the control-difference PI regulator:

$$\underline{\Phi}_\delta = \left[\begin{array}{c|c} \underline{\Phi} & \underline{B}_d \\ \hline \underline{O} & \underline{I} \end{array}\right] \qquad \text{(A-30a)}$$

$$\underline{B}_\delta = \left[\begin{array}{c} \underline{O} \\ \hline \underline{I} \end{array}\right] \qquad \text{(A-30b)}$$

$\underline{\Phi}_\delta$ and $\underline{B}_\delta$ are stored in vector CTL of /CONTROL/.

A.11.4 <u>SREGPI</u>. Computations involved in the design of the PI regulator are directed by routine SREGPI. Routine 'WXUS' is called first to determine the quadratic weighting matrices of the discrete-time optimal cost function from the continuous-time input quadratic weights. Quadratic weighting matrices (assumed diagonal) are entered directly by the user from the terminal for costs assigned to output and input deviations and to input rates-- $\underline{Y}$, $\underline{U}_y$, and $\underline{U}_c$ respectively. These matrices are stored in vector "RPI" of /CREGPI/. An augmented perturbation state vector is defined to be

$$\underline{\bar{x}} = \begin{bmatrix} \delta\underline{x} \\ \delta\underline{u} \end{bmatrix} \qquad \text{(A-31)}$$

A weighting matrix on the state vector $\underline{\bar{x}}$ is formed as

$$\underline{X}_{c_{11}} = \underline{C}^T\underline{YC} \tag{A-32a}$$

$$\underline{X}_{c_{22}} = \underline{U}_y + \underline{D}_y^T\underline{YD}_y \tag{A-32b}$$

$$\underline{X}_{c_{12}} = \underline{C}^T\underline{YD}_y \tag{A-32c}$$

Routine 'FORMX' performs these computations and forms

$$\underline{X} = \begin{bmatrix} \underline{X}_{c_{11}} & \underline{X}_{c_{12}} \\ \underline{X}^T_{c_{12}} & \underline{X}_{c_{22}} \end{bmatrix} \tag{A-33}$$

The user is then given an opportunity to modify individual elements of $\underline{X}$ (symmetry is preserved automatically by WXUS), as for instance, to alter individual diagonal elements of $\underline{X}_{c_{11}}$. The associated continuous-time cost function is,

$$J = \tfrac{1}{2}\int_{t_0}^{t_{N+1}} \begin{bmatrix} \underline{\bar{x}}(t) \\ \underline{\bar{u}}(t) \end{bmatrix}^T \begin{bmatrix} \underline{X} & \underline{O} \\ \underline{O} & \underline{U}_c \end{bmatrix} \begin{bmatrix} \underline{\bar{x}}(t) \\ \underline{\bar{u}}(t) \end{bmatrix} dt \tag{A-34}$$

where $\underline{\bar{u}}$ is the control difference ("pseudo-rate")

$$\underline{\bar{u}} = \Delta\underline{u} \tag{A-35}$$

The corresponding discrete-time cost function is defined by

$$J = \sum_{i=0}^{N} \tfrac{1}{2}[\underline{\bar{x}}^T(t_i)\underline{X}_\delta\underline{\bar{x}}(t_i)+\underline{\bar{u}}^T(t_i)\underline{U}_\delta\underline{\bar{u}}(t_i)+2\underline{\bar{x}}^T(t_i)\underline{S}_\delta\underline{\bar{u}}(t_i)] \tag{A-36}$$

and the discrete costs are,

$$\underline{X}_\delta = \int_{t_i}^{t_i+T} \underline{\Phi}_\delta^T(\tau)\underline{X}\underline{\Phi}_\delta(\tau)d\tau \tag{A-37a}$$

$$\underline{U}_\delta = \int_{t_i}^{t_i+T} [\underline{B}_\delta^T(\tau)\underline{X}\underline{B}_\delta(\tau)+\underline{U}_c]d\tau \tag{A-37b}$$

$$\underline{S}_\delta = \int_{t_i}^{t_i+T} \underline{\Phi}^T(\tau)\underline{X}\underline{B}_\delta(\tau)d\tau \tag{A-37c}$$

in which

$$\underline{\Phi}_\delta(\tau) = \left[\begin{array}{c|c} \underline{\Phi}(\tau) & \underline{B}_d(\tau) \\ \hline \underline{O} & \underline{I} \end{array}\right] \tag{A-37d}$$

$$\underline{B}_d(\tau) = \int_0^\tau \underline{\Phi}(\sigma)\underline{B}d\sigma \tag{A-37e}$$

and $\underline{B}_\delta$ is as defined in equation (A-30b). The integrals of equations (A-37a,b,c) are approximated in a two-step computation. First, $\underline{\Phi}_\delta$ and $\underline{B}_\delta$ are treated as constants over the sample interval with value set to their respective averaged values at the beginning and end of the interval:

$$\underline{\bar{\Phi}}_\delta = \tfrac{1}{2}[\underline{I} + \underline{\Phi}_\delta] \qquad \text{(A-38a)}$$

and $$\underline{\bar{B}}_\delta = \tfrac{1}{2}[\underline{O} + \underline{B}_\delta] \qquad \text{(A-38b)}$$

in which $\underline{\Phi}_\delta$ and $\underline{B}_\delta$ are as defined in equations (A-30a,b). With these approximations, each of the integrands is constant over the integration time T, so the integrals are obtained as

$$\underline{\hat{X}}_\delta = T[\underline{\bar{\Phi}}^T \underline{X}\underline{\bar{\Phi}}_\delta] \qquad \text{(A-39a)}$$

$$\underline{\hat{U}}_\delta = T[\underline{\bar{B}}_\delta^T \underline{X}\underline{\bar{B}}_\delta + \underline{U}_c] \qquad \text{(A-39b)}$$

$$\underline{\hat{S}} = T[\underline{\bar{\Phi}}^T \underline{X}\underline{\bar{B}}_\delta] \qquad \text{(A-39c)}$$

This is a better approximate evaluation than simple Euler inegration provides. These three discrete-time costs are returned by WXUS as arguments "X","U", and "S", respectively.

The cost function of equation (A-36) includes the cross-weight $\underline{S}_\delta$ weighting products of states and inputs. Routine 'PXUP' is called to compute modified system and weighting matrices to allow the optimization to be framed in terms of state and input quadratic costs only (Ref 29): Define a modified system,

$$\underline{\bar{x}}(t_{i+1}) = \underline{\Phi}_\delta' \underline{\bar{x}}(t_i) + \underline{B}_\delta \underline{\bar{u}}'(t_i) \qquad \text{(A-40a)}$$

with

$$\underline{\Phi}_{\delta}' = \underline{\Phi}_{\delta} - \underline{B}_{\delta}\underline{U}_{\delta}^{-1}\underline{S}_{\delta}^{T} \qquad \text{(A-40b)}$$

$$\underline{\bar{u}}_{\delta}' = \underline{\bar{u}} + \underline{U}_{\delta}^{-1}\underline{S}_{\delta}^{T}\underline{\bar{x}} \qquad \text{(A-40c)}$$

for which the cost function becomes

$$J' = \sum_{i=0}^{N} \tfrac{1}{2}[\underline{\bar{x}}^{T}(t_i)\underline{X}_{\delta}'\underline{\bar{x}}(t_i) + \underline{\bar{u}}'^{T}(t_i)\underline{U}_{\delta}\underline{\bar{u}}'(t_i)] \qquad \text{(A-41a)}$$

and

$$\underline{X}_{\delta}' = \underline{X}_{\delta} - \underline{S}_{\delta}\underline{U}_{\delta}^{-1}\underline{S}_{\delta}^{T} \qquad \text{(A-41b)}$$

The cost function of equation (A-41a) is now in standard form for solution of the steady-state Riccati equation. PXUP returns matrices $\underline{\Phi}_{\delta}'$, $\underline{X}_{\delta}'$, and $\underline{U}_{\delta}^{-1}\underline{S}_{\delta}^{T}$ of equations (A-40b), (A-41b), and (A-40b), respectively. An additional matrix needed for the routine which computes the solution to the Riccati equation is also computed by PXUP:

$$\underline{U}_{\delta}' = \underline{B}_{\delta}\underline{U}_{\delta}^{-1}\underline{B}_{\delta}^{T} \qquad \text{(A-42)}$$

SREGPI next computes the steady-state solution to the discrete-time Riccati equation using routine 'DRIC' of LIBRARY. DRIC solves for $\underline{\bar{K}}_R$ in

$$\underline{\bar{K}}_R = \underline{\Phi}_{\delta}'^{T}\underline{\bar{K}}_R(\underline{I}+\underline{U}_{\delta}'\underline{\bar{K}}_R)^{-1}\underline{\Phi}_{\delta}' + \underline{X}_{\delta}' \qquad \text{(A-43)}$$

using an iterative procedure discussed in Reference 24. In addition to $\underline{\bar{K}}_R$, DRIC returns the closed-loop system matrix

$$\underline{\Phi}_{\delta CL} = (\underline{I}+\underline{U}'_{\delta}\underline{\bar{K}}_R)^{-1}\underline{\Phi}'_{\delta} \qquad \text{(A-44)}$$

which is stored in vector RPI.

Routine 'GCSTAR' then is called to compute the optimal feedback gain matrix for the original system in two steps:
The optimal feedback gains for the modified system of equation (A-40) are,

$$\underline{G}^{*\prime}_c = (\underline{U}_{\delta}+\underline{B}^T_{\delta}\underline{\bar{K}}_R\underline{B}_{\delta})^{-1}\underline{B}^T_{\delta}\underline{\bar{K}}_R\underline{\Phi}'_{\delta} \qquad \text{(A-45)}$$

and from these the optimal feedback gains for the original system are obtained:

$$\underline{G}^*_c = \underline{G}^{*\prime}_c + \underline{U}^{-1}_{\delta}\underline{S}^T_{\delta} \qquad \text{(A-46)}$$

which can be considered as partitioned into gains on the components of the state vector $\underline{\bar{x}}$ of equation (A-31). The optimal input then is,

$$\Delta\underline{u}^*(t_i) = -\left[\underline{G}^*_{c_1} \,\vdots\, \underline{G}^*_{c_2}\right]\begin{bmatrix}\delta\underline{x}(t_i)\\ \hdashline \delta\underline{u}(t_i)\end{bmatrix} \qquad \text{(A-47)}$$

SREGPI uses these partitions of $\underline{G}^*_c$ and partitions of the $\underline{\Pi}$ matrix of equation (A-29) to compute the gains $\underline{K}_x$ and $\underline{K}_z$ of the optimal PI regulator and stores them in vector RPI:

$$\underline{K}_x = \underline{G}^*_{c_1}\underline{\pi}_{11} + \underline{G}^*_{c_2}\underline{\pi}_{21} \quad \text{(A-48a)}$$

$$\underline{K}_z = \underline{G}^*_{c_1}\underline{\pi}_{12} + \underline{G}^*_{c_2}\underline{\pi}_{22} \quad \text{(A-48b)}$$

The PI regulator in incremental-form (Ref 32) utilizing these gains is implemented as,

$$\begin{aligned} \underline{u}(t_i) &= \underline{u}(t_{i-1}) - \underline{K}_x[\underline{x}(t_i)-\underline{x}(t_{i-1})] \\ &\quad - \underline{K}_z\left[[\underline{C} \quad \underline{D}_y]\begin{bmatrix}\underline{x}(t_{i-1}) \\ \underline{u}(t_{i-1})\end{bmatrix}\right] \end{aligned} \quad \text{(A-49)}$$

The controller evaluation routines which propagate the response of the PI regulated system to non-zero initial conditions use equation (A-49) to compute the control input. Note that this assumes that the outputs are to be driven to zero by the PI regulator. No provision is made for evaluation of the PI regulator in response to control inputs.

A.11.5 SCGT. Routine SCGT directs the computations involved in design of an open-loop CGT or closed-loop CGT/PI controller. The first set of computations are performed by routine 'CGTA'; the results depend only on the design and command models. Since the design model is invariant throughout program execution, CGTA is not called unless a new command model has been established (test value of variable NEWCM in /CMDMTX/).

The CGT theory formulates an "ideal" state and input trajectory to achieve exact matching with the command model outputs. These ideal trajectories are assumed to be expressable as linear functions of the command model's states and inputs and the disturbance states:

$$\begin{bmatrix} \underline{x}_I(t_i) \\ \hline \underline{u}_I(t_i) \end{bmatrix} = \left[\begin{array}{c|c|c} \underline{A}_{11} & \underline{A}_{12} & \underline{A}_{13} \\ \hline \underline{A}_{21} & \underline{A}_{22} & \underline{A}_{23} \end{array}\right] \begin{bmatrix} \underline{x}_m(t_i) \\ \hline \underline{u}_m(t_i) \\ \hline \underline{n}_d(t_i) \end{bmatrix} \qquad \text{(A-50)}$$

A set of equations are derived for the $\underline{A}_{11}$ through $\underline{A}_{23}$ partitions. They are,

$$\underline{A}_{11} = \underline{\pi}_{11}\underline{A}_{11}(\underline{\Phi}_m - \underline{I}) + \underline{\pi}_{12}\underline{C}_m \qquad \text{(A-51a)}$$

$$\underline{A}_{12} = \underline{\pi}_{11}\underline{A}_{11}\underline{B}_{m_d} + \underline{\pi}_{12}\underline{D}_m \qquad \text{(A-51b)}$$

$$\underline{A}_{13} = \underline{\pi}_{11}\underline{A}_{13}(\underline{\Phi}_n - \underline{I}) - \underline{\pi}_{11}\underline{E}_{x_d} - \underline{\pi}_{12}\underline{E}_y \qquad \text{(A-51c)}$$

$$\underline{A}_{21} = \underline{\pi}_{21}\underline{A}_{11}(\underline{\Phi}_m - \underline{I}) + \underline{\pi}_{22}\underline{C}_m \qquad \text{(A-51d)}$$

$$\underline{A}_{22} = \underline{\pi}_{21}\underline{A}_{11}\underline{B}_{m_d} + \underline{\pi}_{22}\underline{D}_m \qquad \text{(A-51e)}$$

$$\underline{A}_{23} = \underline{\pi}_{21}\underline{A}_{13}(\underline{\Phi}_n - \underline{I}) - \underline{\pi}_{21}\underline{E}_{x_d} - \underline{\pi}_{22}\underline{E}_y \qquad \text{(A-51f)}$$

Of these equations, those for $\underline{A}_{11}$ (equation A-51a) and $\underline{A}_{13}$ (equation A-51c) must be solved independently. The other equations then express the remaining $\underline{A}_{ij}$ matrices in terms of known matrices. The two equations to be solved are of the form

$$\underline{X} = \underline{AXB} + \underline{C} \tag{A-52}$$

for which an algorithm for solution is reported in Reference 4. This algorithm has been implemented in routines described in Reference 10. Certain conditions which must be met for a solution to exist are discussed in these references and in Reference 32, as well as in Section 3.3.3 of this thesis.

CGTA sets up equations (A-51a) and (A-51c) then calls routine 'AXBMXC' to solve for $\underline{A}_{11}$ and $\underline{A}_{13}$. AXBMXC solves each equation using routine 'SLVSHR'. Iterative refinement of the solution is pursued until the Euclidean norm of the error residual matrix is less than 10^{-6} (routine 'ENORM') or as many as three refining iterations. If the solution does not meet the error tolerance after three refinement steps, a message is printed and execution proceeds. The routines AXBMXC, SLVSHR, and ENORM are adaptations of routines described in Reference 10.

With $\underline{A}_{11}$ and $\underline{A}_{13}$ determined, CGTA proceeds to compute $\underline{A}_{12}$, $\underline{A}_{21}$, $\underline{A}_{22}$, and $\underline{A}_{23}$. All the $\underline{A}_{ij}$ matrices are stored in vector "CGT" of /CCGT/.

SCGT then calls routine 'CGTKX' to compute the gains employed by the CGT and CGT/PI controllers. For the open-loop CGT controller routine SCMD sets matrices $\underline{K}_x$

and $\underline{K}_z$ of equation (A-48) to zero. CGTKX computes gains on command model states and inputs and disturbance states, respectively, as

$$\underline{K}_{x_m} = \underline{K}_x\underline{A}_{11} + \underline{A}_{21} \tag{A-53a}$$

$$\underline{K}_{x_u} = \underline{K}_x\underline{A}_{12} + \underline{A}_{22} \tag{A-53b}$$

$$\underline{K}_{x_n} = \underline{K}_x\underline{A}_{13} + \underline{A}_{23} \tag{A-53c}$$

These three gains are stored in vector CGT.

The closed-loop CGT/PI control law is implemented in incremental form as

$$\begin{aligned}\underline{u}(t_i) &= \underline{u}(t_{i-1}) - \underline{K}_x[\underline{x}(t_i)-\underline{x}(t_{i-1})] \\ &+ \underline{K}_{x_m}[\underline{x}_m(t_i)-\underline{x}_m(t_{i-1})] \\ &+ \underline{K}_{x_u}[\underline{u}_m(t_i)-\underline{u}_m(t_{i-1})] \\ &+ \underline{K}_{x_n}[\underline{n}_d(t_i)-\underline{n}_d(t_{i-1})] \\ &+ \underline{K}_z\left\{[\underline{C}_m \quad \underline{D}_m]\begin{bmatrix}\underline{x}_m(t_{i-1})\\ \underline{u}_m(t_i)\end{bmatrix}\right. \\ &\left. - [\underline{C} \quad \underline{D}]\begin{bmatrix}\underline{x}(t_{i-1})\\ \underline{u}(t_{i-1})\end{bmatrix}\right\}\end{aligned} \tag{A-54}$$

The open-loop CGT is obtained by employing equation (A-54) with PI gains $\underline{K}_x$ and $\underline{K}_z$ both zero matrices, giving the effective result for the open-loop CGT control law as

$$\underline{u}(t_i) = \underline{u}(t_{i-1}) + \underline{A}_{21}[\underline{x}_m(t_i)-\underline{x}_m(t_{i-1})]$$
$$+ \underline{A}_{22}[\underline{u}_m(t_i)-\underline{u}_m(t_{i-1})]$$
$$+ \underline{A}_{23}[\underline{n}_d(t_i)-\underline{n}_d(t_{i-1})] \quad \text{(A-55)}$$

Equation (A-54) is used by the controller evaluation routines to compute control inputs for either CGT controller.

A.11.6 CEVAL. Routine 'CEVAL' is executive to a set of routines which perform evaluations of the PI, CGT, or CGT/PI controllers. If the PI regulator is being evaluated, the continuous-time domain mapped eigenvalues of the closed-loop matrix $\underline{\Phi}_{\delta CL}$ of equation (A-44) are computed and printed by the routine 'POLES'. The primary evaluation tool is the simulated time-response of the controlled system. For the PI regulator the response is generated for non-zero initial conditions and no commanded input. The system with either CGT controller is driven by step inputs on any one of the command model's inputs and by non-zero initial conditions on system and disturbance states, if desired. The system time response can be propagated using either the design model or the truth model state transition equations. Plots of the resulting time behavior of the states, inputs, and outputs of the system are printed at the user terminal and output to the LIST file.

Specific execution of the controller evaluation is affected by flags "LFLCGT" and "LTEVAL" of /DESIGN/. These signal the design as either of PI (LFLCGT=0) or CGT type, and indicate the evaluation is to be with respect to the design (LTEVAL=0) or truth models. The flowcharts of Figures A-6a,b,c show the basic decisions and execution paths pursued in the controller evaluation. As many as two plots of user-selected variables may be printed at the user terminal while plots of all relevant variables are also output to the LIST file. If the user wishes no plots printed at the terminal, the time-response simulation is not executed. Each plot can include as many as five variables plotted versus time.

Because the routines execute differently according to the specific conditions of the controller to be evaluated and the system model used for simulation, there are numerous tests and variant sections of code. Details finer than that shown in Figures A-6a,b,c are not discussed.

Response variables are stored at each time step in sets by type in the scratch vector SM of /SYSMTX/. A collection of several sets of variables is itself considered to be the set of all relevant variables for plotting at each sample time. Other sets of variables at one sample-time in the past are also stored in vector SM. The partitioning of SM occurs both in routine CEVAL and 'VOUTIC'.

Routine VOUTIC is used to establish initial conditions for the system and to define the desired plots. The

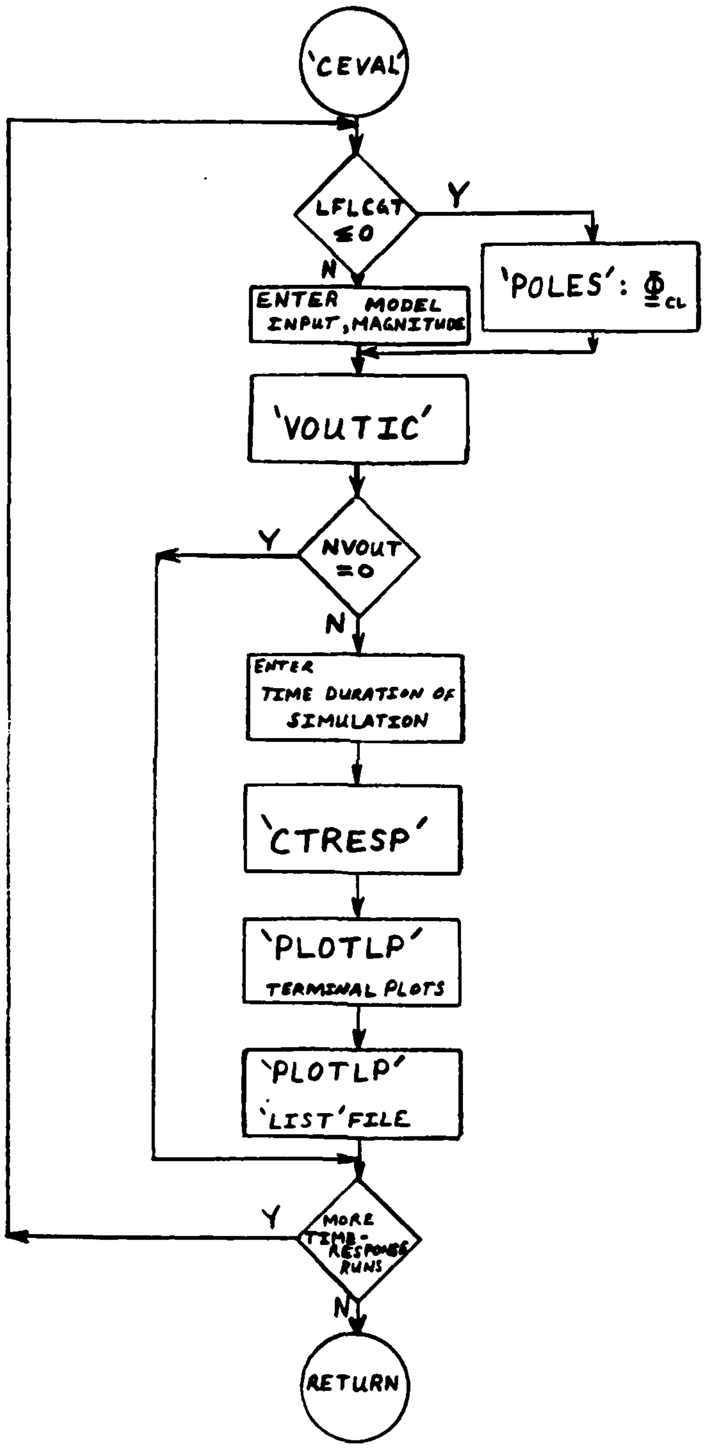


Fig. A-6a. CEVAL Flowchart

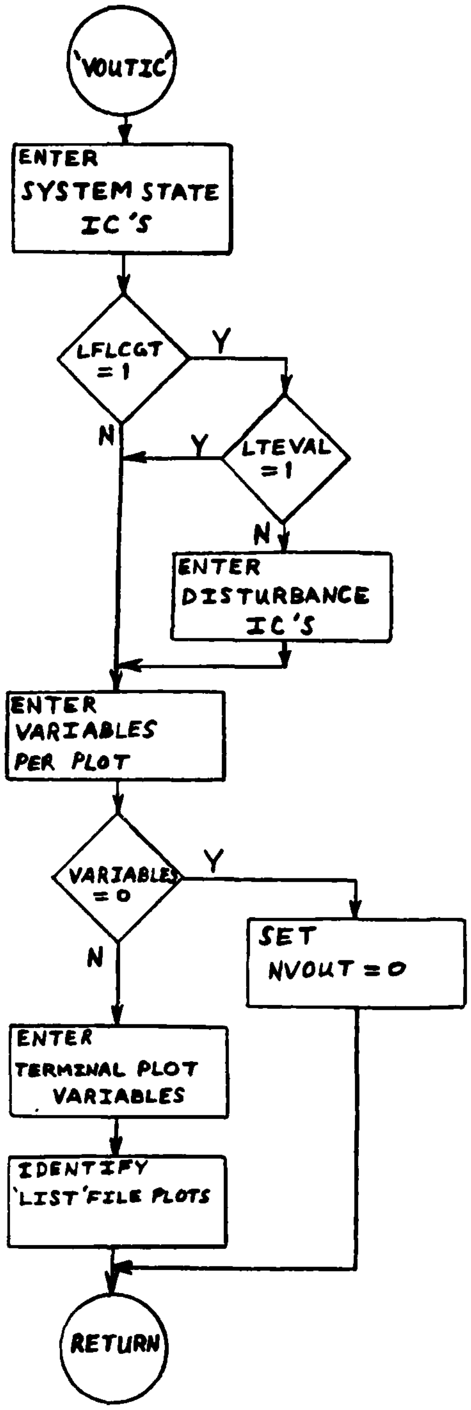


Fig. A-6b. VOUTIC Flowchart

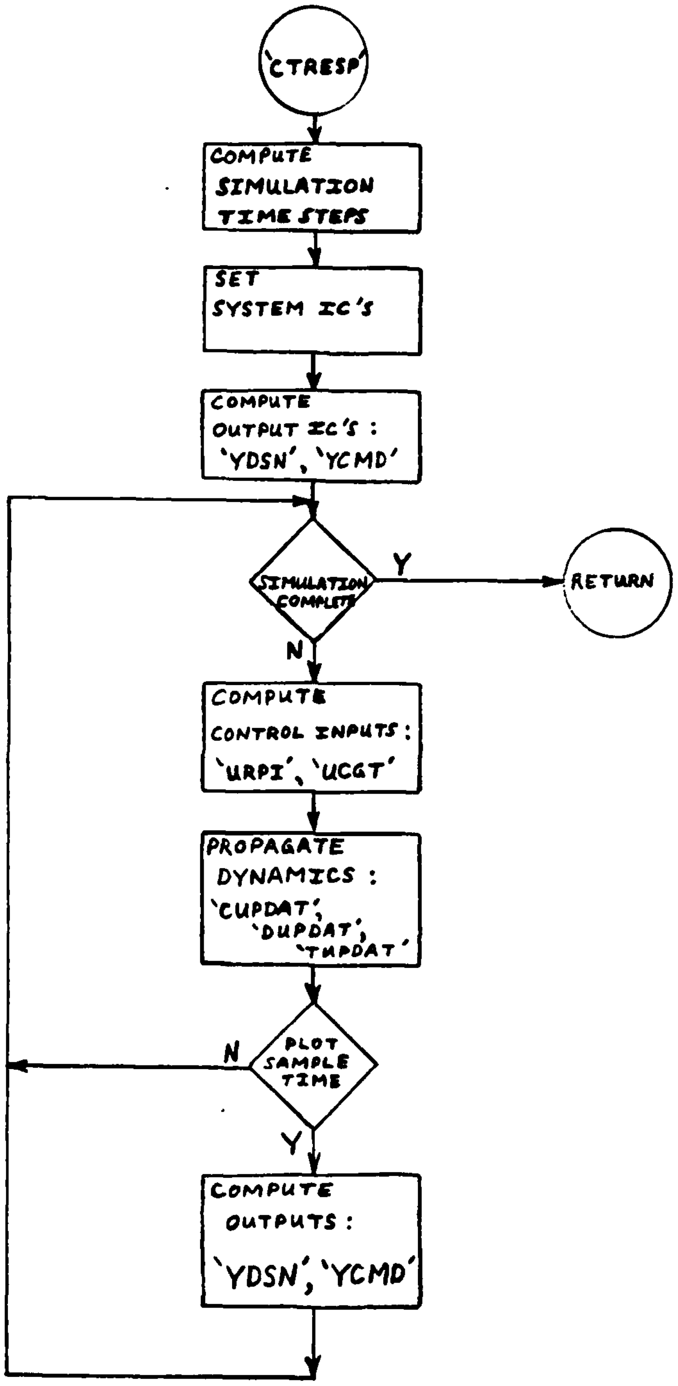

'CTRESP'
COMPUTE SIMULATION TIME STEPS
SET SYSTEM IC'S
COMPUTE OUTPUT IC'S : 'YDSN', 'YCMD'
SIMULATION COMPLETE
Y
RETURN
N
COMPUTE CONTROL INPUTS : 'URPI', 'UCGT'
PROPAGATE DYNAMICS : 'CUPDAT', 'DUPDAT', 'TMPDAT'
PLOT SAMPLE TIME
N
Y
COMPUTE OUTPUTS : 'YDSN', 'YCMD'

Fig. A-6c. CTRESP Flowchart

states of the model to be used for system time propagation are given initial values by the user. For the PI evaluation, the following variables may be plotted: system states, outputs, and inputs. For the CGT evaluation the disturbance states and the outputs of the command model may also be plotted. If a command model output is among the variables in a terminal plot, then all variables of the plot are plotted using a single scale range to facilitate evaluation of actual and commanded output responses. For all other plots each variable is scaled independently. Ordinarily the input argument "NVOUT" specifies the total number of relevant system variables available for plotting. VOUTIC sets NVOUT to zero if no plots are to be printed at the user terminal; CEVAL then does not perform a simulation.

Routine 'CTRESP' performs the time-response simulation. An input argument gives the total intended duration of the simulation ("TEND"). CTRESP executes the simulation as an integer loop with control inputs and model propagation computed during each pass. The value of TEND is adjusted so that the total time is an integer multiple of the controller sample period and of one hundred. Thus all plot samples coincide precisely with controller samples and the entire time interval is spanned by 100 evenly spaced samples. The loop is executed for the number of steps thus determined. A vector of response variables is written to the PLOT file at one hundred equally spaced time samples during the simulation.

In performing the simulation, eight primary routines are used. They are discussed briefly below:

'DUPDAT':	Propagates the states of the design model forward in time using equations (A-23a,b).
'CUPDAT':	Propagates the states of the command model forward in time using equation (A-25a).
'TUPDAT':	Propagates the states of the truth model forward in time using equation (A-27) without the noise input.
'XFDT':	Transforms the state vector of the truth model to the design model state and disturbance vectors using equations (A-7c,d).
'URPI':	Computes the control input due to the PI controller alone using equation (A-49).
'UCGT':	Computes the control input due to the CGT controller alone and adds it to the control given by URPI. The increment due to the CGT or CGT/PI alone is added as

$$\begin{aligned}\underline{u}(t_i) \leftarrow \underline{u}(t_i) &+ \underline{K}_{x_m}[\underline{x}_m(t_i)-\underline{x}_m(t_{i-1})] \\ &+ \underline{K}_{x_u}[\underline{u}_m(t_i)-\underline{u}_m(t_{i-1})] \\ &+ \underline{K}_{x_n}[\underline{n}_d(t_i)-\underline{n}_d(t_{i-1})] \\ &+ \underline{K}_z[\underline{C}_m \quad \underline{D}_m]\begin{bmatrix}\underline{x}_m(t_{i-1}) \\ \underline{u}_m(t_i)\end{bmatrix}\end{aligned} \qquad \text{(A-56)}$$

'YDSN':	Computes the outputs of the design model using equation (A-23c).
'YCMD':	Computes the outputs of the command model using equation (A-25b).

On return to CEVAL the PLOT file contains 101 sets of samples from the system time response simulation (sample at time=0. and one hundred additional samples at equal time intervals). Plots of selected variables to the user terminal include 51 sample points for each variable. If the time duration originally requested for the simulation spanned fewer than 50 controller sample periods, the terminal plots will have a duration equal to 50 times the controller sample period. Otherwise, alternate samples from among those on the PLOT file are plotted: the entire duration of the simulation is spanned but with time resolution half as fine as available from the PLOT file samples. Plots are then output to the LIST file. These plots include all sample points and all variables are plotted. Each plot includes the time-responses of five variables. Routine 'PLOTLP' computes and prints all plots to the terminal and the LIST file.

When plotting is complete, CEVAL provides the opportunity to perform additional simulations with the same controller. When no additional simulations are to be run, CEVAL is exited.

A.11.7 FLTRK. Routine 'FLTRK' effects design of a steady-state Kalman filter for the design model defined by equations (A-20) and (A-21). The measurement equation given by equations (A-2d) and (A-4) is rewritten in terms of the augmented state vector and augmented measurement matrix (equations (A-18) and (A-24)):

$$\underset{\sim}{z}(t_i) = \underline{H}_a \underset{\sim}{x}_a(t_i) + \underset{\sim}{v}(t_i) \qquad \text{(A-57)}$$

A call to routine 'KFLTR' of LIBRARY computes the steady-state covariance matrix and the Kalman filter gains. The covariance matrix is the solution $\overline{\underline{P}}_a$ satisfying

$$\overline{\underline{P}}_a = \underline{\Phi}_a\{\overline{\underline{P}}_a - \overline{\underline{P}}_a \underline{H}_a^T [\underline{H}_a \overline{\underline{P}}_a \underline{H}_a^T + \underline{R}]^{-1} \underline{H}_a \overline{\underline{P}}_a\} \underline{\Phi}_a^T + \underline{Q}_{a_d} \qquad \text{(A-58)}$$

and the Kalman filter gain matrix is,

$$\overline{\underline{K}}_F = \overline{\underline{P}}_a \underline{H}_a^T [\underline{H}_a \overline{\underline{P}}_a \underline{H}_a^T + \underline{R}]^{-1} \qquad \text{(A-59)}$$

The $\overline{\underline{P}}_a$ matrix employed is prior to update$(\underline{P}_a^-)$.

KFLTR uses an iterative technique described in Reference 24 to compute the matrix $\overline{\underline{P}}_a$. The filter gain matrix $\overline{\underline{K}}_F$ is stored in vector FLT of /CKF/. A vector of the standard deviations of the state estimates (square-roots of the diagonal elements of $\overline{\underline{P}}_a$) is also stored in FLT. An additional output of KFLTR is the measurement update matrix

$$\underline{M}_K = [\underline{I} - \overline{\underline{K}}_F \underline{H}_a] \qquad \text{(A-60)}$$

It is put into temporary storage in vector COM2 of /MAIN2/

for use in the filter evaluation routines.

The first execution of FLTRK in a given run of CGTPIF uses the matrices $\underline{Q}_{a_d}$ and $\underline{R}$ of equations (A-21c) and (A-4) as determined from initial entry of the design model. Subsequent executions of FLTRK begin by offering the user an opportunity to modify the system noise strength matrices $\underline{Q}$ and $\underline{Q}_n$ (no provision is made for direct entry of the augmented, discretized noise strength matrix $\underline{Q}_{a_d}$) and the measurement noise strength $\underline{R}$ of equations (A-3a), (A-3b), and (A-4), respectively. Routine 'QDSCRT' is called to form $\underline{Q}_a$ (equation (A-19d)) and compute the new discrete-time system noise covariance matrix $\underline{Q}_{a_d}$ as given by equation (A-21c). A new Kalman filter gain matrix is then computed as described above.

A.11.8 FEVAL. In 'FEVAL' the eigenvalues of the design model-Kalman filter system are computed with a call to POLES. The primary evaluation tool is a covariance analysis of the filter in which the filter's estimation error is evaluated in operating on measurements taken from the truth model. These "true" estimation error standard deviations are plotted along with the filter's computed error standard deviations.

The poles of the system with filter are the eigenvalues of

$$\underline{\Phi}_{KF} = \underline{M}_K \underline{\Phi}_a = [\underline{I} - \underline{\bar{K}}_F \underline{H}_a] \underline{\Phi}_a \qquad \text{(A-61)}$$

where $\underline{M}_K$, $\underline{\Phi}_a$, $\overline{\underline{K}}_F$, and $\underline{H}_a$ are given by equations (A-60), (A-21a), (A-59), and (A-24), respectively.

The covariance analysis entails propagation of an error covariance matrix through fifty filter (controller) sample periods. At each time sample a vector of true and filter computed estimation error standard deviations is written to the PLOT file. When the run is complete these are plotted pairwise (true/computed) for each state in a series of plots to the LIST file. The final RMS errors for each state are also printed at the user terminal.

Define an augmented state vector

$$\underset{\sim}{x}_c = \begin{bmatrix} \underset{\sim}{x}_t \\ \hline \hat{\underset{\sim}{x}} \end{bmatrix} \qquad \text{(A-62)}$$

with $\underset{\sim}{x}_t$ the truth model states and $\hat{\underset{\sim}{x}}$ the filter state estimates. Time propagation for the augmented state is given by

$$\underset{\sim}{x}_c(t_i^-) = \underline{\Phi}_c \underset{\sim}{x}_c(t_{i-1}^+) + \underset{\sim}{w}_{c_d}(t_{i-1}) \qquad \text{(A-63)}$$

where

$$\underline{\Phi}_c = \left[\begin{array}{c|c} \underline{\Phi}_t & \underline{0} \\ \hline \underline{0} & \underline{\Phi}_a \end{array}\right] \qquad \text{(A-64a)}$$

and $\underset{\sim}{w}_{c_d}$ is zero-mean white Gaussian discrete-time noise of discrete-time noise covariance

$$E\{\underset{\sim}{\underline{w}}_{c_d}(t_i)\underline{w}_{c_d}^T(t_j)\} = \underline{Q}_{c_d}\delta_{ij} \tag{A-64b}$$

with

$$\underline{Q}_{c_d} = \int_0^T \underline{\Phi}_c(T-\tau)\underline{G}_c\underline{Q}_t\underline{G}_c^T\underline{\Phi}_c^T(T-\tau)\,d\tau \tag{A-64c}$$

and

$$\underline{G}_c = \left[\begin{array}{c} \underline{G}_t \\ \hdashline \underline{O} \end{array}\right] \tag{A-64d}$$

In these equations $\underline{\Phi}_t$, $\underline{\Phi}_a$, $\underline{Q}_t$, and $\underline{G}_t$ are from equations (A-28a), (A-21a), (A-8a), and (A-7a), respectively.

Note that equation (A-64c) is actually

$$\underline{Q}_{c_d} = \left[\begin{array}{c:c} \underline{Q}_{t_d} & \underline{O} \\ \hdashline \underline{O} & \underline{O} \end{array}\right] \tag{A-65}$$

where $\underline{Q}_{t_d}$ is determined according to equation (A-28c). FEVAL uses $\underline{Q}_{t_d}$ directly rather than form the larger matrix $\underline{Q}_{c_d}$.

The measurement update equation is,

$$\underset{\sim}{\underline{x}}_c(t_i^+) = \underline{A}_c\underset{\sim}{\underline{x}}_c(t_i^-) + \underline{K}_c\underset{\sim}{\underline{v}}_t(t_i) \tag{A-66}$$

in which

$$\underline{A}_c = \left[\begin{array}{c:c} \underline{\underline{I}} & \underline{O} \\ \hdashline \underline{\overline{K}}_F\underline{H}_t & [\underline{I}-\underline{\overline{K}}_F\underline{H}_a] \end{array}\right] \tag{A-67a}$$

$$\underline{K}_c = \begin{bmatrix} \underline{O} \\ \hline \underline{\bar{K}}_F \end{bmatrix} \tag{A-67b}$$

Initial conditions for $\underset{\sim}{x}_c(t_0^+)$ are taken as the zero vector.

The covariance of the augmented state $\underset{\sim}{x}_c$ is assumed to be zero at the initial time $(\underline{P}_c(t_0))$ and is propagated forward by

$$\underline{P}_c(t_i^-) = \underline{\Phi}_c\underline{P}_c(t_{i-1}^+)\underline{\Phi}_c^T + \underline{Q}_{c_d} \tag{A-68a}$$

and

$$\underline{P}_c(t_i^+) = \underline{A}_c\underline{P}_c(t_i^-)\underline{A}_c^T + \underline{K}_c\underline{R}_t\underline{K}_c^T \tag{A-68b}$$

Note that in equation (A-68b)

$$\underline{K}_c\underline{R}_t\underline{K}_c^T = \left[\begin{array}{c|c} \underline{O} & \underline{O} \\ \hline \underline{O} & \underline{\bar{K}}_F\underline{R}_t\underline{\bar{K}}_F^T \end{array}\right] \tag{A-69}$$

FEVAL forms the lower right partition of equation (A-69) rather than forming the larger matrix $\underline{K}_c\underline{R}_t\underline{K}_c^T$.

The filter estimation error for the design model's system and disturbance states following measurement update is,

$$\underline{e}_c(t_i) = \underline{C}_c\underset{\sim}{x}_c(t_i^+) \tag{A-70}$$

with

$$\underline{C}_c = [-\underline{C}_t \mid \underline{I}] \tag{A-71a}$$

and

$$\underline{C}_t = [\underline{T}_{DT} \mid \underline{T}_{NT}] \tag{A-71b}$$

in which $\underline{T}_{DT}$ and $\underline{T}_{NT}$ are as defined by equations (A-7c) and (A-7d).

The estimation error covariance at each sample period is thus

$$\underline{P}_e(t_i) = \underline{C}_c \underline{P}_c(t_i^+) \underline{C}_c^T \tag{A-72}$$

The diagonal elements of $\underline{P}_e$ are the variances of the estimation error for each system and disturbance state. Taking the square-root of each to obtain the standard deviations, these and the standard deviations from the filter's computed covariance matrix are written to the PLOT file at each sample time.

Routine 'DACOV' forms matrix $\underline{C}_c$ of equation (A-71a) and then computes $\underline{P}_e(t_i)$ according to equation (A-72). The true and filter computed standard deviations of the estimation error are then determined and written to the PLOT file.

Routine 'ACOVUD' first forms matrix $\underline{\Phi}_c$ of equation (A-64a). The product $\underline{\Phi}_c \underline{P}_c(t_{i-1}^+) \underline{\Phi}_c^T$ is then obtained and $\underline{Q}_{t_d}$ is added to its upper left partition to give $\underline{P}_c(t_i^-)$ as in equation (A-68a). Next the matrix $\underline{A}_c$ of equation (A-67a) is formed. Then, after computing $\underline{A}_c \underline{P}_c(t_i^-) \underline{A}_c^T$

the product $\underline{\overline{K}}_F \underline{R}_t \underline{\overline{K}}_F^T$ is added to its lower right partition to obtain $\underline{P}_C(t_i^+)$ of equation (A-68b).

The adding of a given matrix to a partition of another as required for equations (A-68a) and (A-68b) is accomplished by routine 'FPADD'. Input arguments to FPADD specify the size of the partition to be dealt with ("NRY"-by-"NCY") and the starting address of that partition ("LADDR") in the large matrix.

FEVAL calls DACOV initially to determine the errors at time=0., then calls ACOVUD and DACOV repeatedly in a loop to obtain the errors at each time sample. When these samples are completed, plots of the results are output to the LIST file using calls to PLOTLP for each state. The RMS errors at the final time sample are printed at the terminal for each state.

A.11.9 <u>Utility Routines</u>. CGTPIF includes a number of routines which perform specific computations useful to several of the larger computational elements discussed in Sections A.11.2-A.11.8 above. Each routine will be discussed briefly. The function performed and the input/output arguments will be delineated. In a few cases variable "LABORT" (signifying abort of program execution) of /DESIGN/ or the model dimensions of /NDIMD/, /NDIMC/, or /NDIMT/ are modified; in all other cases only variables appearing as formal arguments are modified by the subroutines.

RSYS (A, L, ND, ITYPE, IWRT)

Routine 'RSYS' is used in entering any of the three dynamic models describing the design problem. It distinguishes among the models it is dealing with and provides prompts to the user appropriate to each. Its formal arguments are:

"A":	Output vector containing all arrays defining the dynamic model.
"L":	Output vector containing the starting addresses of each array within vector $\underline{A}$. The order of the array starting addresses is the same as in equations (A-6), (A-10), or (A-13).
"ND":	Output vector used internally by RSYS to store the dimensions of the model being entered. The order of the dimensions is the same as in equations (A-5), (A-9), or (A-12).
"ITYPE":	Input integer scalar signifying the model to be entered. Values of 1, 2, or 3 refer to design, command, or truth models, respectively.
"IWRT":	Input/output integer scalar indicating if specific model has been previously entered, and if so if it has been written to the SAVE file. For IWRT non-zero the model has been successfully entered; for IWRT negative the model has also been written to SAVE. IWRT is initialized to zero by the calling routine and set to

values by RSYS to control its functioning in subsequent calls dealing with the same model.

DSND (ND)

Routine 'DSND' is a dummy routine of the same name as an optional routine described in Section A.10. It is loaded if the user does not include the corresponding functional routine. It sets the first dimension of the model to zero, signaling RSYS that a "real" routine does not exist.

"ND":	Output vector intended to contain model dimensions. Its first element is set to zero.

CMDD (ND)

Same as for DSND.

TRTHD (ND)

Same as for DSND.

DSNM (A, B, EX, G, Q, C, DY, EY, H, HD, R, AN, GN, QN)

Routine'DSNM' is a dummy routine of the same name as an optional routine described in Section A.10. It merely "completes the load" in the event the user elects not to include the functional routine.

CMDM (AM, BM, CM, DM)

Same as for DSNM.

TRTHM (AT, BT, GT, QT, HT, RT, TDT, TNT)

Same as for DSNM.

DSNDM (ND, NAD)

Routine 'DSNDM' sets values into the design model dimension variables of /NDIMD/. It also stores the array dimensionalities of the model into a two-dimensional array for use by RSYS. Finally, it tests to determine if sufficient allocation has been provided in vector DM of /DSNMTX/; if not, LABORT of /DESIGN/ is set to flag allocation error.

"ND":	Input vector of model dimensions.
"NAD":	Output array of model matrix dimensions. Columns 1 and 2 are the [row,column] dimensions of each matrix in the order of the arguments of DSNM. For example, matrix "B" is argument 2 and is of dimension (n-by-m); thus DSNDM sets NAD(2,1)=n and NAD(2,2)=m.

CMDDM (ND, NAD)

Same as for DSNDM but for command model and common blocks /NDIMC/ and /CMDMTX/ are used.

TRTHDM (ND, NAD)

Same as for DSNDM but for truth model and common blocks /NDIMT/ and /TRUMTX/ are used.

ZMATIN (A, NR, NC, IZ)

Routine 'ZMATIN' is used to read in matrices entered by specifying the element address (row, column) and value. If an entry is attempted that is not in array bounds, a message is printed and the entry not accepted. A row entry of zero signals end of array entries.

"A":	Input/output array in full storage mode.
"NR":	Input integer scalar specifying row dimension of A.
"NC":	Input integer scalar specifying column dimension of A.
"IZ":	Input integer scalar affecting execution of ZMATIN. For IZ positive, matrix A is first zeroed. For IZ negative, A is constrained to be symmetric.

WFILED (NT, NP, ND, A)

Routine 'WFILED' is used to write data to the SAVE file. It executes four writes: (1) a pair of integer scalars specifying the data code and the number of data points; (2) an integer vector of length ten with the dimensions of the model in data; (3) a real vector containing data arrays; and (4) a pair of integer scalars, the first indicating end of data on SAVE file, the second a dummy. The end of data code (-1) is written on SAVE initially by CGTXQ; each execution of WFILED begins with a "backspace" on SAVE to allow the already existing end of data code to be overwritten. This ensures that the SAVE file data entries can be successfully read when used as a DATA file.

"NT":	Input integer scalar data code. Values of 1, 2, 3, or 4 correspond to design model, command model, truth model, or PI gains, respectively.

"NP": Input integer scalar specifying number of data elements in data vector.

"ND": Input integer vector of dimensions.

"A": Input real vector storing data to be saved.

READFS (A, ND, NT, IERR)

Routine 'READFS' reads data from the DATA file which was written by WFILED. It searches the DATA file for the code of the data set it is to read. If the data set is found, a call to 'FARRAY' reads the data. If not found, a message is written and an error flag set.

"A": Output real vector of array data.

"ND": Output integer vector of dimension data.

"NT": Input integer scalar specifying data set code (as for "NT" in WFILED).

"IERR": Output integer scalar error flag set non-zero if data set is not found on DATA file.

FARRAY (A, ND, NP)

Routine 'FARRAY' reads data sets from the DATA file.

"A": Same as "A" in READFS.

"ND": Same as "ND" in READFS.

"NP": Input integer scalar specifying number of data elements in data vector <u>A</u>.

TFRMTX (X1, X2, NR, NC, ITX)

Routine 'TFRMTX' transfers matrices between storage locations in cases when one matrix is in full storage mode and the other is in variable storage mode. The transfer can be in either direction as determined by an input argument.

"X1":	Input/output real array in full storage mode. It is allocated (NR-by-NC).
"X2":	Input/output real array in variable storage mode. It is allocated (NDIM-by-NDIM) but contains an array sized (NR-by-NC). Note that "NDIM" is the row dimension specification of /MAIN1/.
"NR":	Input integer scalar row dimension.
"NC":	Input integer scalar column dimension.
"ITX":	Input integer scalar controlling the direction in which the matrix transfer takes place. For ITX=1, X2 is the input, X1 is the output, and the (NR-by-NC) sub-array of X2 is stored in X1. For ITX=2, X1 is the input, X2 is the output, and the matrix X1 is stored as an (NR-by-NC) sub-array in X2.

MATLST (A, NR, NC, NT, KDEV)

Routine 'MATLST' is used to output arrays in full storage mode. A name is printed specifying the array.

"A":	Input real array in full storage mode.
"NR":	Input integer scalar row dimension of $\underline{A}$.
"NC":	Input integer scalar column dimension of $\underline{A}$.
"NT":	Input integer scalar with an array name of three or fewer characters.
"KDEV":	Input integer scalar output device number.

NDSCRT (A, N, NTERMS)

Routine 'NDSCRT' computes the number of terms to be used in computing a state transition matrix using a series expansion. It uses a method suggested in Reference 11 (but with a maximum of 30 terms in the expansion because a temporary vector in DSCRT has its dimension fixed at 30). The number of terms is selected to achieve a truncation error of less than 1.E-6.

"A":	Input real array.
"N":	Input integer scalar dimension of $\underline{A}$.
"NTERMS":	Output integer scalar specifying number of terms to be used in expansion approximating $e^{\underline{A}T}$.

RQWGTS (W, ND, NP)

Routine 'RQWGTS' is used to enter the diagonal elements of the quadratic weighting matrices or noise covariance matrices. Elements are specified using a single index for the diagonal element and the value of that element. The index is tested for being in array bounds, and negative entries for diagonal elements are not accepted. For either error a message is written. A diagonal index of zero signals that entry is complete. Elements are tested for proper sign according to argument "NP".

"W":	Input/output real array whose diagonal elements are to be set.
"ND":	Input integer scalar specifying the row dimension of the array within which $\underline{W}$ is stored.
"NP":	Input integer scalar used to determine sign test for diagonal elements. If zero, diagonal elements may be greater than or equal to zero. If NP is non-zero, diagonal elements must be positive.

DVCTOR (N, A, V)

Routine 'DVCTOR' extracts the diagonal elements of an array and stores them in a vector.

"N":	Input integer scalar dimension of input array.
"A":	Input real array.
"V":	Output real vector of diagonal elements of $\underline{A}$.

POLES (A, N, ITYPE, ZM1, ZM2)

Routine 'POLES' computes the eigenvalues of the input matrix using 'EIGEN' of LIBRARY. For the design, command, and truth models it computes the poles of the continuous-time system model. For the PI and filter closed-loop systems, it computes the discrete-time poles then calls 'MAPOLE' to map them to the primary strip in continuous-time. The continuous or pseudo-continuous-time poles are printed along with a title identifying the system.

"A": Input real array.

"N": Input integer scalar dimension of $\underline{A}$.

"ITYPE": Input integer scalar indicating system represented by $\underline{A}$. Values of 1 to 5 refer to the design, command, or truth models, the closed-loop PI, or filter systems, respectively.

"ZM1", "ZM2": Input real arrays used for temporary storage.

MAPOLE (N, ZR, ZI, T)

Routine 'MAPOLE' is used to map the poles of a discrete-time system to the primary strip in the continuous domain (Ref 28). Denote the real (σ) and imaginary (ω) parts of a discrete-time pole as z_R and z_I respectively. MAPOLE uses the following equations:

$$z_m = \sqrt{z_R^2 + z_I^2} \quad \text{(A-73a)}$$

$$\sigma = \text{LOG}_e(z_m)/T \quad \text{(A-73b)}$$

$$\omega = \text{TAN}^{-1}(z_I/z_R)/T \quad \text{(A-73c)}$$

where T is the controller-filter sample period and σ and ω are the corresponding mapped real and imaginary parts of the pole. These computations are performed for each system pole.

"N":	Input integer scalar number of eigen-values.
"ZR":	Input/output real vector of real components of poles (z_R).
"ZI":	Input/output real vector of imaginary components of poles (z_I).
"T":	Input real scalar controller-filter sample period (T).

LADDR (NR, I, J)

Function routine 'LADDR' computes the single index address of an element specified by a (row, column) address within an array. That index value is stored in function name LADDR.

"NR":	Input integer scalar row dimension of array within which an address is sought.
"I":	Input integer scalar element row address.
"J":	Input integer scalar element column address.

FTMTX (X, Y, NR, NC)

Routine 'FTMTX' transfers one array to storage in another when both are in full storage mode.

"X": Input real array whose elements are to be stored elsewhere.

"Y": Output real array containing same elements as X.

"NR": Input integer scalar row dimension of X,Y.

"NC": Input integer scalar column dimension of X,Y.

FMMUL (X, Y, NR1, NC1, NC2, Z)

Routine 'FMMUL' computes the product of two matrices. All matrices are in full storage mode.

"X": Input real array dimensioned (NR1-by-NC1).

"Y": Input real array dimensioned (NC1-by-NC2).

"NR1": Input integer scalar row dimension of X.

"NC1": Input integer scalar column dimension of X and row dimension of Y.

"NC2": Input integer scalar column dimension of Y.

"Z": Output real array formed as product of X and Y and dimensioned (NR1-by-NC2).

FTMUL (X, Y, NR1, NC1, NC2, Z)

Routine 'FTMUL' computes the product of one matrix with the transpose of another. All arrays are in full storage mode.

"X":   Input real array dimensioned (NR1-by-NC1).

"Y":   Input real array dimensioned (NR1-by-NC2).

"NR1", "NC1", "NC2": Input integer scalar dimensions.

"Z":   Output real array formed as product of $\underline{X}^T$ with $\underline{Y}$; it is dimensioned (NC1-by-NC2).

FMADD (X, Y, NR, NC, Z)

Routine 'FMADD' computes the sum of two matrices. All matrices are in full storage mode. Either input matrix can be equivalent to the output matrix.

"X":   Input real array dimensioned (NR-by-NC).

"Y":   Input real array dimensioned (NR-by-NC).

"NR", "NC": Input integer scalar dimensions.

"Z":   Output real array formed as the sum of $\underline{X}$ and $\underline{Y}$ and dimensioned (NR-by-NC).

ZPART (A, NR, NC, ND)

Routine 'ZPART' is used to store zeros in a partition of a matrix which is itself in full storage mode.

"A":   Input/output real array of row dimension "ND"; the first element of $\underline{A}$ is the starting location of the partition to be zeroed.

"NR": Input integer scalar row dimension of the partition.

"NC": Input integer scalar column dimension of the partition.

"ND": Input integer scalar row dimension of the input matrix $\underline{A}$.

SUBI (A, NR, ND)

Routine 'SUBI' is used to subtract an identity matrix of appropriate dimension from a square partition of a larger matrix in full storage mode.

"A": Input/output array of row dimension "ND"; the first element of $\underline{A}$ is the starting location of the square partition.

"NR": Input integer scalar dimension of the square partition.

"ND": Input integer scalar row dimension of the input matrix $\underline{A}$.

WPLOTF (V, N)

Routine 'WPLOTF' writes a vector to the PLOT file.

"V": Input real vector.

"N": Input integer scalar dimension of $\underline{V}$.

RPLOTF (V, N, IERR)

Routine 'RPLOTF' reads a vector from the PLOT file. If an "end-of-file" is encountered in the read an error flag is set.

"V": Output real vector.

"N": Input integer scalar dimension of V.

"IERR": Output integer scalar error flag. IERR is non-zero if an error occurred.

STRPLT (A, V, NS, NV, NP, NVO)

Routine 'STRPLT' extracts specific elements from an input vector and stores them in an output vector. It is used in preparing sets of variables for plotting.

"A": Output real vector into which elements are stored.

"V": Input real vector some of whose elements are extracted for storage in A.

"NS": Input integer vector of addresses where variables are to be stored within A.

"NV": Input integer vector of element addresses of variables in V which are to be extracted.

"NP": Input integer scalar specifying number of variables to be extracted from V.

"NVO": Input integer scalar length of vector V. It also locates the storage of the time variable in V (time is the last element of V).

PLOTLP (N, M, A, IPSC, ISCL, LPTERM, NDEV, ITITLE)

Routine 'PLOTLP' creates line printer plots. As many as five dependent variables may be plotted with respect to a single independent variable. Every sample of the independent variable is plotted, and runs lengthwise on the output listing. The dependent variables are plotted over a field either 50 or 100 print positions in width and may be unscaled, scaled individually, or scaled separately. Each dependent variable is plotted with an integer identifier (1 to 5). The range of the plot is printed with subdivisions, and if independent scaling is used multiple ranges are printed and marked in correspondence to the plot symbol of the variable to which it pertains. Header comments in the source listing define all arguments explicitly. Those descriptions will not be repeated here.

VARSCL (XMIN, XMAX, SCALE, RSPACE, ISCL)

Routine 'VARSCL' is used by routine PLOTLP to achieve scaling of the plot variables. It can give either exact scaling so that the full range of the variable is used or a "nice" scaling with upper and lower values of the range and the scale increment all simple numbers. In the former case maximum resolution is achieved but computation of intermediate values in the range involve numbers that require many digits to specify. In the latter case, resolution may be lessened but the computations to determine intermediate values are simpler. Equal scaling is achieved by scaling over the combined range of all variables.

"XMIN":	Input/output real scalar giving the minimum value of the variable.
"XMAX":	Input real scalar giving the maximum value of the variable.
"SCALE":	Output real scalar giving the scale size of each print position in the range.
"RSPACE":	Input real scalar specifying the number of print positions in the plot range.
"ISCL":	Input integer scalar indicating if exact or "nice" scaling is to be used. ISCL non-zero gives "nice" scaling.

A.12 LIBRARY Routines

Many routines of LIBRARY are called by CGTPIF. Many others are invoked by those which are explicitly called. For descriptions of all the LIBRARY routines see Reference 24. Some general considerations in using these routines will be discussed here.

In essence, the LIBRARY package of routines assumes that arrays used in its computations are in variable storage mode within larger square arrays of dimension NDIM (NDIM is an element of /MAIN1/). Because of the method of array storage in FORTRAN (column-major storage) in most cases only the allocated row dimension of all arrays involved in computations must be identical. During some operations involving matrix transposes, the allocation must actually be square for the matrix which is transposed.

Thus, in all execution of CGTPIF other than 'MAIN', NDIM at any specific time is set to the row dimension of the allocation in which relevant arrays are effectively stored for the computations currently using LIBRARY routines. The variable "NDIM1" of /MAIN1/ is simply the value of NDIM plus one. Both NDIM and NDIM1 are used by the routines of LIBRARY to locate specific elements of arrays.

Sometimes arrays involved in LIBRARY calls are row dimension compatible in their existing storage mode. At other times some arrays must be moved to a variable storage mode of row dimension equal to that of the largest array to be used so that all arrays involved in a computation are effectively stored in arrays of equal allocation dimensions.

CGTPIF does a great deal of array manipulation. The routines of the LIBRARY provide very useful capabilities and should be used when possible. However, the programmer should be very careful to deal properly with array storage in attempting modification of CGTPIF or calls to LIBRARY from optional routines. It is easy to be correct, but it is also easy to be incorrect since programmers typically are unaccustomed to the manner in which FORTRAN stores arrays.

A.13 Array Starting Addresses

Throughout CGTPIF arrays are referenced in terms of single index addresses. These may be the starting addresses of arrays within larger vectors or may be addresses of specific elements of arrays. With the specific exception of the variables in /DESIGN/, essentially all variables used in CGTPIF conform to the following convention: variable names beginning with the character "L" refer to array address indexing. Many such index variables are of temporary use only and can be evaluated in the context of the source code where they occur.

The starting addresses of all arrays preserved in Common storage are stored in variables of associated Commons. These starting address Commons are described below. In all cases, arrays are stored in the associated vector storage area in the same order in which their starting addresses occur in the corresponding address Commons. In identifying array addresses below, the equation number in which each array is defined is given in parentheses to the right of the array name.

/LOCD/ LAP, LGP, LPHI, LBD, LEX, LPHD, LQ, LQN, LQD,
LC, LDY, LEY, LHP, LR

Address Common /LOCD/ is associated with /DSNMTX/ (see Sections A.6.4.1 and A.7.1) and specifies starting addresses within vector "DM" as follow:

"LAP":	$\underline{A}_a$	(A-19a)
"LGP":	$\underline{G}_a$	(A-19c)
"LPHI":	$\underline{\Phi}$	(A-22a)
"LBD":	$\underline{B}_d$	(A-22b)
"LEX":	$\underline{E}_{x_d}$	(A-22a)
"LPHD":	$\underline{\Phi}_n$	(A-22a)
"LQ":	$\underline{Q}$	(A-3a)
"LQN":	$\underline{Q}_n$	(A-3b)
"LQD":	$\underline{Q}_{a_d}$	(A-21c)
"LC":	$\underline{C}$	(A-23c)
"LDY":	$\underline{D}_y$	(A-23c)
"LEY":	$\underline{E}_y$	(A-23c)
"LHP":	$\underline{H}_a$	(A-24)
"LR":	$\underline{R}$	(A-24)

/LOCC/ LPHC, LBDC, LCC, LDC

Address Common /LOCC/ is associated with /CMDMTX/ (see Sections A.6.4.1 and A.7.3) and specifies array starting addresses within vector "CM" as follow:

"LPHC":	$\underline{\Phi}_m$	(A-25a)
"LBDC":	$\underline{B}_{m_d}$	(A-25b)
"LCC":	$\underline{C}_m$	(A-26a)
"LDC":	$\underline{D}_m$	(A-26b)

/LOCT/ LPHT, LBDT, LQDT, LHT, LRT, LTDT, LTNT

Address Common /LOCT/ is associated with /TRUMTX/ (see Sections A.6.4.1 and A.7.2) and specifies starting addresses within vector "TM" as follow:

"LPHT":	$\underline{\Phi}_t$	(A-28a)
"LBDT":	$\underline{B}_{t_d}$	(A-28b)
"LQDT":	$\underline{Q}_{t_d}$	(A-28c)
"LHT":	$\underline{H}_t$	(A-7b)
"LRT":	$\underline{R}_t$	(A-8b)
"LTDT":	$\underline{T}_{DT}$	(A-7c)
"LTNT":	$\underline{T}_{NT}$	(A-7d)

/LCNTRL/ LPI11, LPI12, LPI21, LPI22, LPHDL, LBDL

Address Common /LCNTRL/ is associated with /CONTROL/ (see Sections A.6.4.2 and A.11.3) and specifies starting addresses within vector "CTL" as follow:

"LPI11":	$\underline{\pi}_{11}$	(A-29b)
"LPI12":	$\underline{\pi}_{12}$	(A-29b)
"LIP21":	$\underline{\pi}_{21}$	(A-29b)
"LIP22":	$\underline{\pi}_{22}$	(A-29b)
"LPHDL":	$\underline{\Phi}_{\delta}$	(A-30a)
"LBDL":	$\underline{B}_{\delta}$	(A-30b)

/LREGPI/ LXDW, LUDW, LPHCL, LKX, LKZ

Address Common /LREGPI/ is associated with /CREGPI/ (see Sections A.6.4.3 and A.11.4) and specifies starting addresses within vector "RPI" as follow:

"LXDW":	$\underline{Y}$	(A-32a)
"LUDW":	$\underline{U}_c$	(A-34)
"LPHCL":	$\underline{\Phi}_{\delta CL}$	(A-44)
"LKX":	$\underline{K}_x$	(A-48a)
"LKZ":	$\underline{K}_z$	(A-48b)

/LCGT/ LA11, LA13, LA21, LA23, LA12, LA22, LKXA11,
LKXA12, LKXA13

Address Common /LCGT/ is associated with /CCGT/ (see Sections A.6.4.4 and A.11.5) and specifies starting addresses within vector "CGT" as follow:

"LA11":	$\underline{A}_{11}$	(A-51a)
"LA13":	$\underline{A}_{13}$	(A-51c)
"LA21":	$\underline{A}_{21}$	(A-51d)
"LA23":	$\underline{A}_{23}$	(A-51f)
"LA12":	$\underline{A}_{12}$	(A-51b)
"LA22":	$\underline{A}_{22}$	(A-51e)
"LKXA11":	$\underline{K}_{x_m}$	(A-53a)
"LKXA12":	$\underline{K}_{x_u}$	(A-53b)
"LKXA13":	$\underline{K}_{x_n}$	(A-53c)

/LKF/ LEADSN, LFLTRK, LFCOV

Address Common /LKF/ is associated with /CKF/ (see Sections A.6.4.5 and A.11.7) and specifies starting addresses within vector "FLT" as follow

"LEADSN":	$\underline{\Phi}_a$	(A-21a)
"LFLTRK":	$\underline{\bar{K}}_F$	(A-59)
"LFCOV":	$\sqrt{\underline{\bar{P}}_a(i,i)}$	(A-58)

AD-A115 511   AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH   SCHOO--ETC   F/G 1/3
DESIGN OF ADVANCED DIGITAL FLIGHT CONTROL SYSTEMS VIA COMMAND G--ETC(U)
DEC 81   R M FLOYD
UNCLASSIFIED   AFIT/GE/EE-81-20-VOL-2   NL

2 OF 3
AD A115511

Appendix B

CGTPIF User's Guide

B.1 Introduction

CGTPIF is an interactive program for designing "Command Generator Tracker" control systems. It provides three design options: (1) design of a Proportional-plus Integral (PI) regulator; (2) design of an open-loop (CGT) or closed-loop (CGT/PI) Command Generator Tracker controller; and (3) design of a Kalman filter. These provide the component designs for the final controller, to be implemented as a Command Generator Tracker, with an inner-loop Proportional-plus-Integral regulator, and a Kalman filter for state estimation (CGT/PI/KF). Corresponding to each design option is a set of routines for evaluation of the quality of the design. During program execution, any of the design paths can be pursued in any order and as often as desired.

This "User's Guide" discusses CGTPIF as an existing program (as it executes under CYBER INTERCOM) and with the intention of providing information appropriate to successful execution when applied to the user's design problem. It discusses program operation from the input/output (I/O) perspective: the specific input and output of each design/evaluation path and the terminology employed in each input/output item. It also discusses what the user must do both before and immediately following program execution. Users interested in more detailed information about the operation of the program should refer to the "CGTPIF Programmer's Guide" (Appendix A).

B.2 Preparation Prior to Program Execution

B.2.1 Determine Dynamics Models.

CGTPIF employs three dynamics models for the system design: a "design" model, a "truth" model, and a "command" model. It is necessary that the user determine the dimensions and parameters of these models prior to execution of the program. The specific models needed by each design vary, and only those needed to execute the design paths of interest need be known.

At a minimum, the design model must be known in order to execute any of the designs. The truth model is required for evaluation of the Kalman filter (to perform a covariance analysis) and is optional for evaluation of the PI regulator or CGT and CGT/PI controllers. The command model must be known in order to effect either CGT or CGT/PI designs.

The dynamics models are entered into the program during execution as needed and under input prompting provided by CGTPIF. The models will be discussed in detail in the next section of this user's guide.

B.2.2 Define Objectives and Specifications. Before embarking upon design of the controller, the designer should define: (1) the objectives which are to be sought, and (2) appropriate specifications and constraints to apply to the controller. These may be rather loosely defined initially, then become more specific and firm as the design progresses.

Objectives will vary with the problem under consideration but might be exemplified by formulation of a desired controlled output response behavior. For example, one's objective might consist of achieving decoupled, first-order responses with specified characteristics for each controlled output of a given multi-input multi-output (MIMO) system.

Specifications and constraints derive from the problem application and from the objectives for the design. Typical considerations include time-delay, overshoot, and settling time of the response, and input magnitude and rate limits.

B.2.3 Determine Appropriate Initial Quadratic Weights. Execution of the PI regulator design entails entry of quadratic weights for the optimal cost function. Such weighting matrices are required for the outputs, the input magnitudes, and the input rates (Ref 32; see also Section 3.4.2 of this thesis). For these, only diagonal elements are required as input since CGTPIF assumes them to be diagonal matrices. However, after CGTPIF computes the resulting augmented state and input magnitude weighting matrix (equation (A-33)), the user may modify any element of it to achieve design goals.

Although final selection of appropriate quadratic weighting values to achieve design requirements is achieved in an iterative (trial-and-error, hopefully with insight) fashion, it is possible to make initial choices which are plausible. A common method for determining initial quadratic weights involves inverse square weighting of maximum deviations of outputs and inputs to achieve regulation for an assumed perturbation of the system (Refs 2; 29; 32). For example, the diagonal output weighting matrix element Y_{ii} would be Y_{ii}=1./(maximum allowable $y_i)^2$.

Beginning with the initial set of quadratic weights, the PI design path is executed repeatedly with changes in the choice of weighting elements until the design is satisfactory. CGTPIF provides information during execution which allows the design to be evaluated and iteration of the design to be pursued effectively.

For open-loop CGT designs and Kalman filter designs, preparation consists of defining the various dynamics models. The open-loop CGT design depends only on the design and command models. The initial execution of the Kalman filter design path depends only on the design model (and truth model for evaluation). Further refinements to either design are achieved through modification of the appropriate dynamics model (command or design).

B.3 Definition of the Dynamics Models

Each of the dynamics models entered into CGTPIF is represented by a set of continuous-time state differential equations. A summary description of each model is given here, while more detail appears in Appendix A. The names used in the equations to follow are exactly those used by CGTPIF in reference to these same dimensions and arrays in its I/O. Note that here each name is a single character, possibly subscripted, while in its I/O, CGTPIF incorporates subscripts into the name (e.g., A_t becomes "AT"). Constraints on the models that are mentioned below are tested by CGTPIF and if not satisfied, a message is written to the user terminal and execution is aborted.

B.3.1 Design Model.

$$\dot{\underset{\sim}{x}}(t) = \underline{A}\underset{\sim}{x}(t) + \underline{B}\underline{u}(t) + \underline{E}_x\underset{\sim}{n}_d(t) + \underline{G}\underset{\sim}{w}(t) \qquad \text{(B-1a)}$$

$$\dot{\underset{\sim}{n}}_d(t) = \underline{A}_n\underset{\sim}{n}_d(t) + \underline{G}_n\underset{\sim}{w}_d(t) \qquad \text{(B-1b)}$$

$$\underset{\sim}{y}(t) = \underline{C}\underset{\sim}{x}(t) + \underline{D}_y\underline{u}(t) + \underline{E}_y\underset{\sim}{n}_d(t) \qquad \text{(B-1c)}$$

$$\underset{\sim}{z}(t_i) = \underline{H}\underset{\sim}{x}(t_i) + \underline{H}_n\underset{\sim}{n}_d(t_i) + \underset{\sim}{v}(t_i) \qquad \text{(B-1d)}$$

and

$$E\{\underset{\sim}{w}(t)\underset{\sim}{w}^T(t+\tau)\} = \underline{Q}\delta(\tau) \qquad \text{(B-2a)}$$

$$E\{\underset{\sim}{\underline{w}}_d(t)\underset{\sim}{\underline{w}}_d^T(t+\tau)\} = \underline{Q}_n\delta(\tau) \qquad \text{(B-2b)}$$

$$E\{\underset{\sim}{\underline{v}}(t_i)\underset{\sim}{\underline{v}}^T(t_j)\} = \underline{R}\delta_{ij} \qquad \text{(B-2c)}$$

In these equations, $\underset{\sim}{\underline{x}}$, $\underline{u}$, $\underset{\sim}{\underline{n}}_d$, $\underset{\sim}{\underline{y}}$, and $\underset{\sim}{\underline{z}}$ are the system state, input, disturbance state, output, and measurement vectors, respectively. In input prompts, CGTPIF refers to the diagonal elements of the noise covariance matrices as

$\underline{Q}$: "state noise strengths" (B-3a)

$\underline{Q}_n$: "disturbance noise strengths" (B-3b)

$\underline{R}$: "measurement noise strengths" (B-3c)

Note that $\underline{Q}$, $\underline{Q}_n$, and $\underline{R}$ are all assumed to be diagonal matrices. The dimensions of the model are

n = number of system states

r = number of system inputs

p = number of system outputs

m = number of state measurements

d = number of disturbance states

w = number of independent system noises

w_D = number of independent disturbance noises

(B-4)

and the dimensions of the matrices of the model are

$\underline{A}$:	n-by-n
$\underline{B}$:	n-by-r
$\underline{E}_x$:	n-by-d
$\underline{G}$:	n-by-w
$\underline{Q}$:	w-by-w
$\underline{C}$:	p-by-n
$\underline{D}_y$:	p-by-r
$\underline{E}_y$:	p-by-d
$\underline{H}$:	m-by-n
$\underline{H}_n$:	m-by-d
$\underline{R}$:	m-by-m
$\underline{A}_n$:	d-by-d
$\underline{G}_n$:	d-by-w_D
$\underline{Q}_n$:	w_D-by-w_D

(B-5)

CGTPIF requires that the numbers of design systems inputs and outputs be equal: r=p. Also, the number of system states may not be less than the number of disturbance states, due to the computational setup used for the CGT solution (see Section A.7.1). The dimensions n, r, and p must be non-zero; any of the other dimensions may be zero. If m is zero or if w and w_D are both zero, the Kalman filter design path cannot be pursued. Matrices having either dimension zero, do not exist and are not entered.

B.3.2 Truth Model.

$$\dot{\underset{\sim}{x}}_t(t) = \underline{A}_t \underset{\sim}{x}_t(t) + \underline{B}_t \underline{u}_t(t) + \underline{G}_t \underset{\sim}{w}_t(t) \qquad \text{(B-6a)}$$

$$\underset{\sim}{z}_t(t) = \underline{H}_t \underset{\sim}{x}_t(t_i) + \underset{\sim}{v}_t(t_i) \qquad \text{(B-6b)}$$

$$\underset{\sim}{x}'(t) = \underline{T}_{DT} \underset{\sim}{x}_t(t) \qquad \text{(B-6c)}$$

$$\underset{\sim}{n}_d'(t) = \underline{T}_{NT} \underset{\sim}{x}_t(t) \qquad \text{(B-6d)}$$

and

$$E\{\underset{\sim}{w}_t(t)\underset{\sim}{w}_t^T(t+\tau)\} = \underline{Q}_t \delta(\tau) \qquad \text{(B-7a)}$$

$$E\{\underset{\sim}{v}_t(t_i)\underset{\sim}{v}_t^T(t_j)\} = \underline{R}_t \delta_{ij} \qquad \text{(B-7b)}$$

Note that $\underline{Q}_t$ and $\underline{R}_t$ are both assumed to be diagonal matrices. In these equations, $\underset{\sim}{x}_t$, $\underline{u}_t$, and $\underset{\sim}{z}_t$ are the truth model system state, input, and measurement vectors, respectively. The vectors $\underset{\sim}{x}'$ and $\underset{\sim}{n}_d'$ are as defined for the design model.

The dimensions of the truth model are

n_T = number of system states

r_T = number of system inputs

m_T = number of system measurements

w_T = number of independent noises driving system dynamics

(B-8)

and the dimensions of the matrices of the model are

$\underline{A}_t$: n_T-by-n_T

$\underline{B}_t$: n_T-by-r_T

$\underline{G}_t$: n_T-by-w_T

$\underline{Q}_t$: w_T-by-w_T

$\underline{H}_t$: m_T-by-n_T

$\underline{R}_t$: m_T-by-m_T

$\underline{T}_{DT}$: n-by-n_T

$\underline{T}_{NT}$: d-by-n_T (B-9)

CGTPIF requires that the numbers of inputs and of measurements for the truth and design model be the same: r_T=r and m_T=m. If the number of driving noises (w_T) is zero, evaluation of a Kalman filter design is not pursued (since a covariance analysis with no truth model driving noise would not be very informative). Matrices having either dimension zero, do not exist and are not entered.

B.3.3 Command Model.

$$\underline{\dot{x}}_m(t) = \underline{A}_m\underline{x}_m(t) + \underline{B}_m\underline{u}_m(t) \qquad \text{(B-10a)}$$

$$\underline{y}_m(t) = \underline{C}_m\underline{x}_m(t) + \underline{D}_m\underline{u}_m(t) \qquad \text{(B-10b)}$$

In these equations, $\underline{x}_m$, $\underline{u}_m$, and $\underline{y}_m$ are the command model state, input, and output vectors, respectively.

The dimensions of the command model are

n_M = number of model states

r_M = number of model inputs

p_M = number of model outputs (B-11)

and the dimensions of the matrices are

$$\underline{A}_m\text{:}\quad n_M\text{-by-}n_M$$
$$\underline{B}_m\text{:}\quad n_M\text{-by-}r_M$$
$$\underline{C}_m\text{:}\quad p_M\text{-by-}n_M$$
$$\underline{D}_m\text{:}\quad p_M\text{-by-}r_M \qquad \text{(B-12)}$$

CGTPIF requires that the number of outputs of the command and design models are equal: $p_M=p$. Also, the number of system states of the command model (n_M) cannot be greater than the number of system states of the design model (n). This constraint is due to the setup for computation of the CGT solution (see Section A.7.3).

B.4 File Usage

In addition to the input/output (I/O) communication directly with the user terminal, CGTPIF employs four disk files for I/O purposes. One of these ('PLOT') is for temporary use by CGTPIF only. The other three files ('SAVE', 'DATA', and 'LIST') benefit the user by providing continuity between distinct executions of the program (SAVE, DATA) or provide supplementary design output data (LIST).

B.4.1 SAVE and DATA Files. CGTPIF preserves information for use in distinct executions of the program through use of the SAVE and DATA files. During program execution, the dynamics models as well as the PI regulator gains (if available) may be written to the SAVE file.

Following execution, the user may wish to catalog SAVE as a permanent file. In subsequent executions, CGTPIF may (at the user's option) read any of the dynamics models or PI gains from the file named DATA.

Both files are rewound prior to and following program execution. Letting the abbreviations "BE" and "AE" mean before and after execution, respectively, typical operations on these files include:

1. Catalog SAVE file:

   a.
   ```
   BE:REQUEST,SAVE,*PF
   AE:CATALOG,SAVE,pfn
   ```

   b.
   ```
   AE:REQUEST,DUM,*PF
   AE:COPYBF,SAVE,DUM
   AE:CATALOG,DUM,pfn
   ```

2. Attach DATA file:

   ```
   BE:ATTACH,DATA,pfn
   ```

3. Reuse SAVE file as new DATA file

   ```
   AE:RETURN,DATA
   BE:COPYBF,SAVE,DATA                    (B-13)
   ```

None of these operations are required; they are simply useful operations in the event the user chooses to employ the files to streamline repeated executions of a given design problem. Note that SAVE and DATA are local file names and that the permanent file names are represented here by the abbreviation "pfn". Other operations (e.g., PURGE) and other combinations of operations are possible as for any files, and the usual rules for these operations apply here as well. The essential points to understand are that the SAVE file is created by CGTPIF and is an output file only, and that DATA is a previously SAVE'd file under a new local file name and is an input file only. During a single program execution the two files are distinct and these roles cannot be changed.

B.4.2 LIST File. During program execution, extensive design information is output to the LIST file. After execution is complete, the user may wish to route LIST to a line printer for listing (or it may be "PAGED" at the user terminal). The file is rewound before and after execution. To send LIST to a line printer, the following command is used after execution:

```
ROUTE,LIST,DC=PR,TID=nn,ST=CSB,FID=abc       (B-14)
```

in which "nn" is the identification number of the terminal to which the file is to be sent (for AFIT, nn=91), and "abc" is any three character output banner for the listing.

B.4.3 PLOT File. The PLOT file is used internally by CGTPIF for temporary storage of the variables generated by time response evaluations of the controller or filter. If desired, it may be eliminated following execution using the command:

```
RETURN,PLOT                                  (B-15)
```

B.5 CGTPIF Execution

B.5.1 Overview. An important feature of CGTPIF is that it follows appropriate paths through execution automatically, prompting the user for input as necessary. The basic design paths are selected by the user under prompting, but within a given path, only information needed to execute the specific design and evaluation is requested by the program. The user thus does not need any predetermined sequence of command entries to the program, nor are the commands coded in any way.

Figure B-1 gives a general flowchart of CGTPIF. The first direct input into the program is the sample period (in seconds) of the digital controller. Each of the decision blocks (diamond shaped) represent a prompted request for input to choose the design to be pursued. Each rectangular block with an alphabetic character ("A" through "G") in the lower right corner represents a "computational element" of CGTPIF and is discussed individually.

The block labeled "Establish Design Model [A]" is a specific instance of the usage of a set of routines employed in establishing all three of the dynamics models. The command model is established in the design path of the CGT controller. The truth model is established just prior to the controller or filter evaluation blocks. Although the specific I/O messages differ in content for each model established by this computational element, the kinds of I/O are the same.

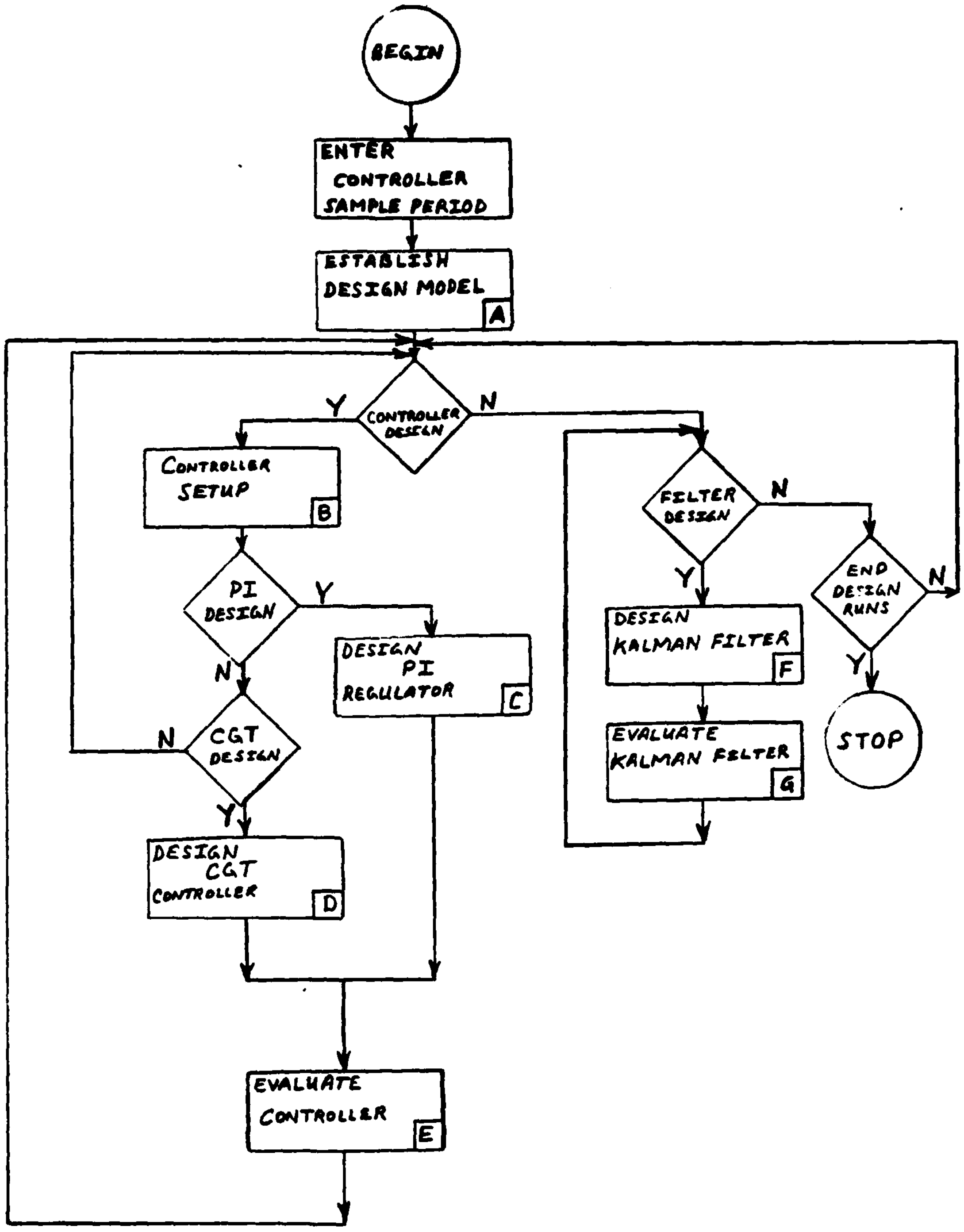


Fig. B-1. CGTPIF General Flowchart

The subsections which follow discuss the I/O of each computational element. In identifying items of I/O, reference will sometimes be made to array names and equations which are specified in the "Programmer's Guide" (equation references will be in parentheses following the array name). All prompts for input define the input that is being requested and the manner in which the entry should be given. Since the actual prompts are themselves understandable, they will not be quoted here. Instead, flowcharts will be used to show where prompts occur and how execution depends on user entries. Blocks involving I/O will be identified by function: an "I" block will signify prompted input from the terminal; an "OT" or "OL" block will signify output to the user terminal or LIST file, respectively. All output to LIST is separated and identified according to the computational element which generated it.

B.5.2 Types of Entries. Required inputs may entail entry of a decision logic value, a single numerical or character value, or multiple numerical values for arrays or vectors. In all cases, CGTPIF prompts the user with messages identifying the nature of the input requested and each prompt ends with the character ">".

B.5.2.1 Decision Logic. All requests for decisions affecting execution are framed as questions requiring a YES ("Y") or NO ("N") response. The user entry is read as character input. Execution proceeds according to a default "YES" assumption: all decision tests assume that if the answer is not "NO", then it is "YES".

B.5.2.2 Single Entry. Requests requiring single entry responses always specify the variable requested. If there are constraints on acceptable input they are indicated in the prompt and adherence is tested in the program after entry. If the entry is not "valid," a message is written to the terminal and the prompt is repeated.

B.5.2.3 Multiple Entry. All requests for entry of vector or array elements specify the name of the array in question and its actual dimensions. Entries for vector elements include an integer specifying the index of the element, and a real specifying its value. For most arrays, all elements may be given values, while for some square matrices, only diagonal elements may be set. In the usual case, elements are entered into arrays by specifying two integers for the [row,column] address and a real for the value of that element. In cases in which only diagonal elements can be specified, entry is the same as for vectors, with the matrix diagonal considered a vector.

As many entries as desired may be made and any entry can be repeated (e.g., to correct previous erroneous entries). Entry is terminated by specifying a row index of zero. Each entry is tested to verify that it lies within the [row,column] bounds of the array (vector). If an index is not "valid", a message is written to the terminal indicating the error and the initial prompt with the array dimensions is given again (previous valid entries are not affected, only the specific invalid entry is rejected). If an entry is valid, the element value is set and the next entry is awaited without additional prompting. For example, if it is desired to set a square matrix of dimension three to an identity matrix, then according to whether the specific matrix is to be entered in [row, column] or diagonal form, entries would be as follow:

1. For [row,column] entry format

```
1 1 1.    (enter)
2 2 1.    (enter)
3 3 1.    (enter)
0/
```

2. For diagonal element entry format

```
1 1.      (enter)
2 1.      (enter)
3 1.      (enter)
0/
```

Items of information may be separated by one or more blanks or by a comma. These entries set specific elements of the matrix to non-zero values, where it has been assumed (as is generally the case) that the matrix was initialized automatically with all elements zeroed.

B.5.3 <u>Establishing Dynamics Models ("A")</u>. All three dynamics models (design, truth, and command) are entered under the control of a single set of routines. The options for entry and the type of I/O involved for each is of identical format, but prompts and output employ terminology specific to each model to identify items of I/O.

Figures B-2a,b,c give flowcharts of the I/O involved in entry of the models. Note that any of the dynamics models may be entered in any of the following ways:

1. The dimensions and array elements may be read from the DATA file.

2. The dimensions and array elements may be entered from the user terminal as prompted by CGTPIF.

3. The dimensions and array elements may be determined by user-provided subroutines.

These modes of entry are offered by CGTPIF in the order above with option 3 assumed selected if options 1 and 2 are declined. If option 1 is selected, the reading of the model is automatically performed. If the model is found not to exist in the DATA file, the other options are offered. For option 3, if the subroutines needed to define the model are not loaded, options 1 and 2 are offered again. This logic is illustrated in Figure B-2a.

Prior to entry of the model matrices, all matrix elements are initialized to zero. Using option 1, all

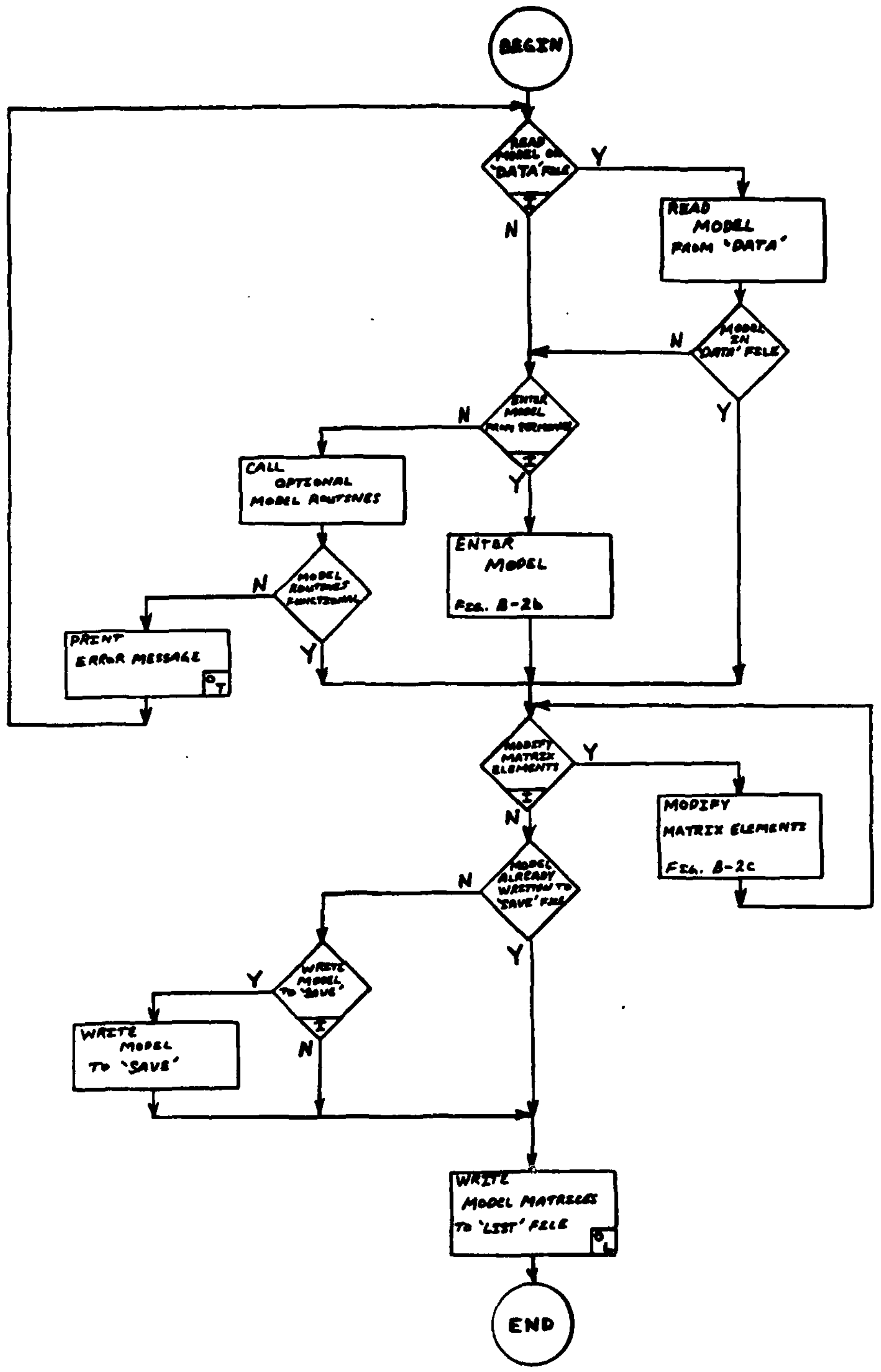


Fig. B-2a. Dynamics Model Entry (Executive)

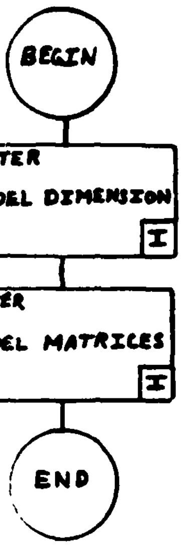


Fig. B-2b. Enter Dynamics Model from Terminal

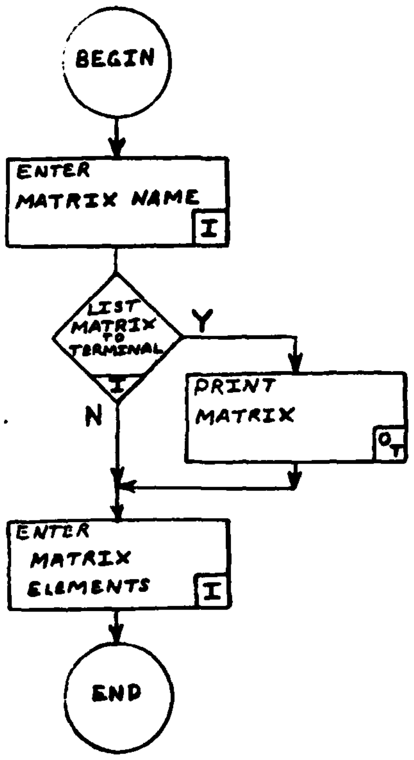


Fig. B-2c. Modify/List Model Arrays

array elements are then read automatically from the DATA file. For options 2 and 3 only the non-zero array elements must be established.

Figure B-2b illustrates model definition under option 2. Entry of the dimensions and arrays is according to prompts by CGTPIF. The dimensions are requested first by the names and in the order of equation (B-4), (B-8), or (B-11), as appropriate. The arrays are then requested, also by name and in the order of equation (B-5), (B-9), or (B-12), as appropriate. An array is not requested if its dimension is zero. Each prompt includes the actual dimensions of the array according to the model dimensions previously entered. Elements of arrays are entered by address, and value by giving the [row,column] address and element value as a three item input. Entry of a zero row address terminates entry of the array.

For option 3, each model requires two user-provided routines of prescribed names, argument lists, and characteristics. These must be compiled with the main routine of CGTPIF and a segmented executable object file created. The "Programmer's Guide" describes the specific requirements for the subroutines and the necessary procedure to obtain an executable CGTPIF program. The job control sequence giving a segmented object file is shown in Appendix E.

After a model is defined using any of the three entry options, the user may list any matrix and modify any array elements, again under prompting by CGTPIF. If a modification/list is desired, the names of the model's matrices are listed at the terminal and the user specifies the array of interest by name. Elements are entered by address and value as described previously. Figure B-2c illustrates the I/O involved in modifying/listing model arrays.

When the model has been defined to the user's satisfaction, it may be written to the SAVE file by CGTPIF if the user chooses. In the course of design iteration, the truth and command models may be redefined if desired, but only a single copy of any model may be written to the SAVE file during a given execution of the program (CGTPIF will not offer additional opportunities after a given model has been SAVE'd).

For each model, the discrete-time representation is computed for the controller sample period specified. Later computations do not depend on the continuous-time dynamics models, so the arrays defining them are not retained.

Arrays defining both the continuous-time and discrete-time models are given in output to the LIST file. The specific output items, their names, and the reference equations are listed below for each model (note that equations (A-*) are from Appendix A):

Design Model

Continuous-time model matrices as listed in equation (B-5) discretized model matrices as:

"PHI":	$\underline{\Phi}$	(A-22a)
"BD":	$\underline{B}_d$	(A-22b)
"QD":	$\underline{Q}_{a_d}$	(A-21c)
"HA":	$\underline{H}_a$	(A-24)
"EXD":	$\underline{E}_{x_d}$	(A-22a)
"PHN":	$\underline{\Phi}_n$	(A-22a)

Command Model

Continuous-time model matrices as listed in equation (B-12) discretized model matrices as:

"PHM":	$\underline{\Phi}_m$	(A-26a)
"BDM":	$\underline{B}_{m_d}$	(A-26b)
"CM":	$\underline{C}_m$	(B-12)
"DM":	$\underline{D}_m$	(B-12)

Truth Model

Continuous-time model matrices as listed in equation (B-9) discretized model matrices as:

"PHT":	$\underline{\Phi}_t$	(A-28a)
"BDT":	$\underline{B}_{t_d}$	(A-28b)
"QDT":	$\underline{Q}_{t_d}$	(A-28c)

In addition, the eigenvalues of the system matrices of each model ($\underline{A}$, $\underline{A}_m$, $\underline{A}_t$) are computed and output both to the user terminal and the LIST file. The system model is identified by type (design, command, truth). Eigenvalues of the corresponding discretized system matrices are not computed.

B.5.4 Controller Setup ("B"). The "controller setup" routines perform computations needed for the controller designs. No input is required of the user and the output is to the LIST file only. The output is, the matrix $\underline{\Pi}$:

"PI": $\underline{\Pi}$ (A-29)

B.5.5 PI Design ("C"). Execution of the PI design path entails user entry of quadratic weighting matrices defining the costs associated with output deviations, control magnitudes, and control rates (see Figure B-[illegible]):

"OUTPUT DEVIATIONS":	$\underline{Y}$	(A-32a)
"CONTROL MAGNITUDES":	$\underline{U}_y$	(A-32b)
"CONTROL RATES":	$\underline{U}_c$	(A-34)

For each of these matrices, only the diagonal elements are entered. On the first execution of the PI design, all weighting matrices are initialized to zero. Subsequent iterations preserve the elements of these matrices so only desired changes in specific weighting elements need be entered. Weights on output deviations should be non-negative, while weights on control magnitudes and

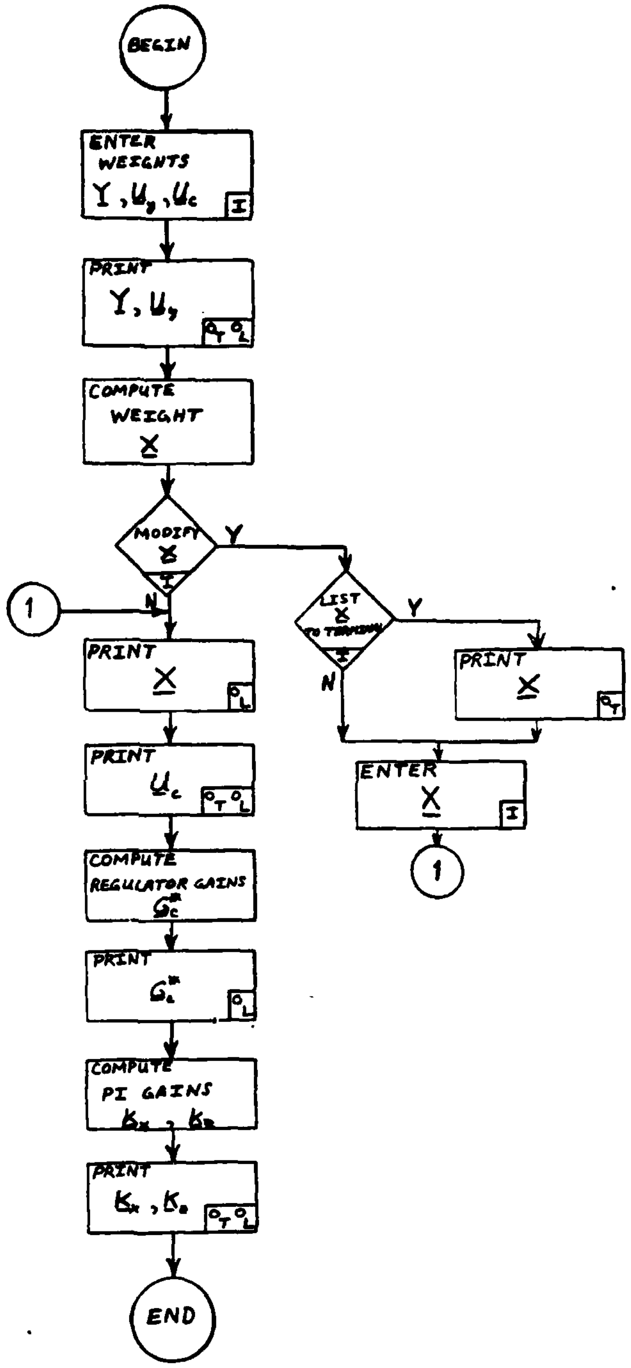


Fig. B-3. PI Regulator Design

rates should be positive. Entries are tested for positive or non-negative values as appropriate. If an entry is not valid, a message is written to the user terminal and that entry is not accepted.

Matrices $\underline{Y}$ and $\underline{U}_y$ are used to compute a quadratic weighting matrix on the state deviations (using an augmented state vector composed of system state and control magnitude perturbations from nominal). This new matrix is referred to as "X" (equations (A-32) and (A-33)). The user may then modify any element of $\underline{X}$ and/or list it at the terminal. Elements entered into $\underline{X}$ are automatically set symmetrically by CGTPIF but the sign of diagonal elements entered is not tested. $\underline{X}$ is not preserved between PI design iterations, so any desired changes in elements with respect to their values as determined from $\underline{Y}$ and $\underline{U}_y$ must be re-entered each design pass.

The diagonal elements of $\underline{Y}$, $\underline{U}_y$, and $\underline{U}_c$ are printed at the user terminal and the entire $\underline{Y}$, $\underline{U}_y$, $\underline{U}_c$, and $\underline{X}$ matrices are output to the LIST file. Next, the regulator gains and PI gains are computed. The PI gains are printed at the terminal ("KX" and "KZ") and all gains are output to the LIST file.

The outputs to the LIST file are,

"Y":	$\underline{Y}$	(A-32a)
"UM":	$\underline{U}_y$	(A-32b)
"X":	$\underline{X}$	(A-33)
"UR":	$\underline{U}_c$	(A-34)

"REG/PI GAIN MATRIX--GCS": $\underline{G}_c^*$ (A-46)

"KX": $\underline{K}_x$ (A-48a)

"KZ": $\underline{K}_z$ (A-48b)

Note that the mnemonics "UM" and "UR" refer to input magnitude and rate weighting matrices, respectively.

When execution of the PI design computations is complete, the "controller evaluation" set of routines is automatically executed. These are discussed in a later subsection as a separate computational element.

B.5.6 CGT Design ("D"). Execution of the "CGT design" path requires that a command model be established. If desired, a new command model can be established during any iteration of the design. The model is actually entered using the routines described in Section B.5.3 above ("Establishing Dynamics Models").

If PI gains already exist in the program storage, then a closed-loop CGT/PI design is effected automatically. If not, the user may elect to have the program read the PI gains from the DATA file and design a closed-loop CGT/PI controller. However, if the user chooses not to have the gains read from DATA or if the gains are found not to exist on the DATA file, an open-loop CGT design is effected automatically (by setting PI gains to zero), but only if the open-loop system is stable. For either open-or closed-loop CGT design, the matrices $\underline{A}_{ij}$ (equation A-51) are automatically output to the LIST file.

Figure B-4 illustrates the I/O, logic, and computations of the CGT design path. Details involved in entering the command model are given in Section B.5.3 and are indicated in this figure by a block titled "Establish Command Model". Note that since the continuous-time representation of the command model is not preserved, "modification" of the command model actually entails complete redefinition of it. In case the command model exists on the DATA file and only specific elements are to be changed, this can be accomplished readily by reading the model from DATA and then modifying individual arrays (as shown in Figure B-2c).

In establishing the command model, I/O is as described in Section B.5.3. Additional output to the LIST file is,

"A11":	$\underline{A}_{11}$	(A-51a)
"A21":	$\underline{A}_{21}$	(A-51d)
"A12":	$\underline{A}_{12}$	(A-51b)
"A22"	$\underline{A}_{22}$	(A-51e)
"A13"	$\underline{A}_{13}$	(A-51c)
"A23":	$\underline{A}_{23}$	(A-51f)
"KXM":	$\underline{K}_{x_m}$	(A-53a)
"KXU":	$\underline{K}_{x_u}$	(A-53b)
"KXN":	$\underline{K}_{x_n}$	(A-53c)

The controller gains ("KXM", "KXU", "KXN") are also printed directly at the user terminal. Note that arrays $\underline{A}_{13}$, $\underline{A}_{23}$, and $\underline{K}_{x_n}$ exist only if disturbance states are

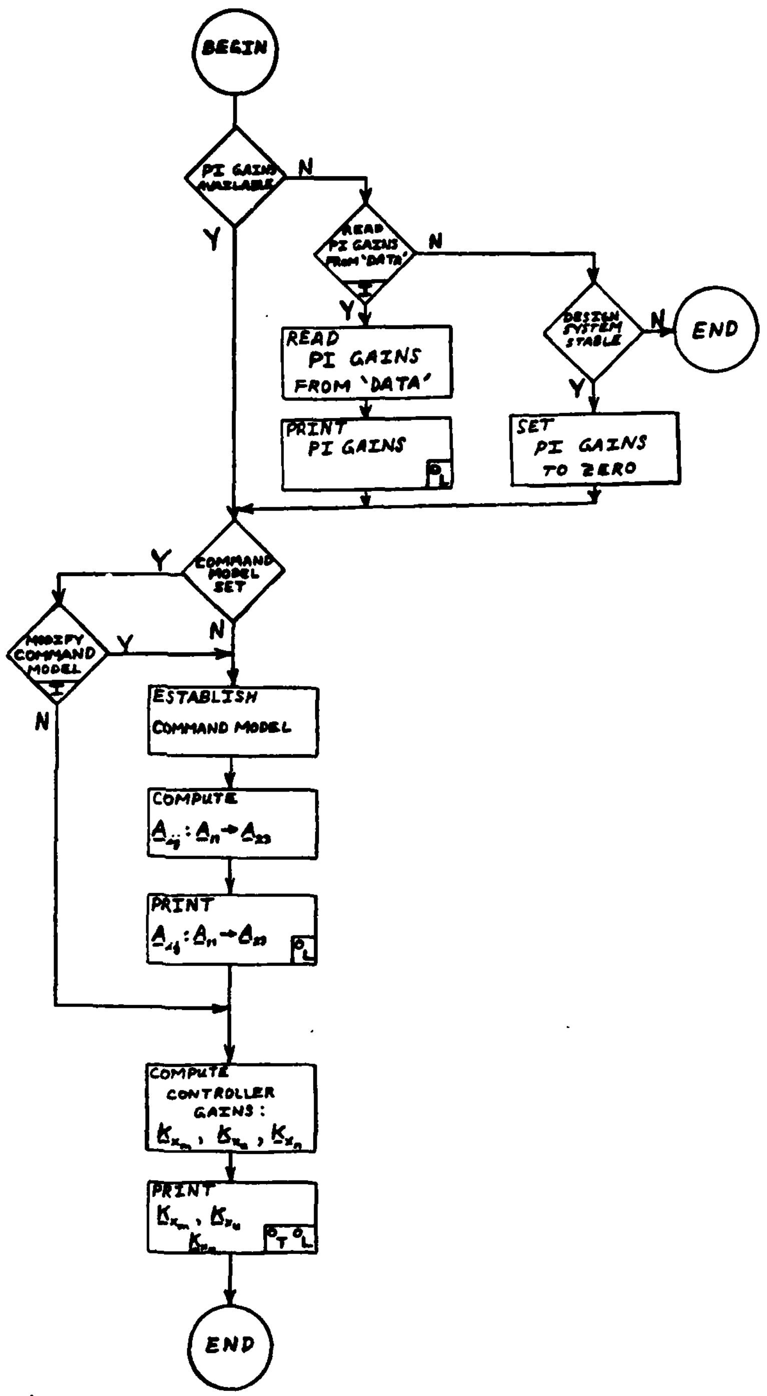


Fig. B-4. CGT Controller Design

specifically modeled in the design model ($\underset{\sim}{n}_d$ of equations (B-1a,b)).

When execution of the CGT design computations is complete, the "controller evaluation" set of routines is automatically executed. These are discussed in the next subsection.

B.5.7 Controller Evaluation ("E"). A single set of routines performs the controller evaluation for both the PI and CGT designs. For the PI controller, the poles of the closed-loop discrete-time system matrix $\underline{\Phi}_{\delta CL}$ (equation (A-44)) are computed and mapped into the primary strip in the continuous-time domain (the z-plane poles are not listed in output). These mapped closed-loop poles are printed both to the user terminal and the LIST file. The primary evaluation tool for both controllers is a time-response simulation. For the PI regulator, the response is taken for non-zero initial conditions (IC's) on the states; for the CGT controller the response is given for a step input on any of the command model's inputs. In either case, the system dynamics can be propagated using the design model or truth model state transition equations. Time response runs may be run repeatedly for a specific controller design.

The I/O, logic, and computations involved in the controller evaluation are shown in Figure B-5. Decision blocks labeled "CGT" test for the type of controller being

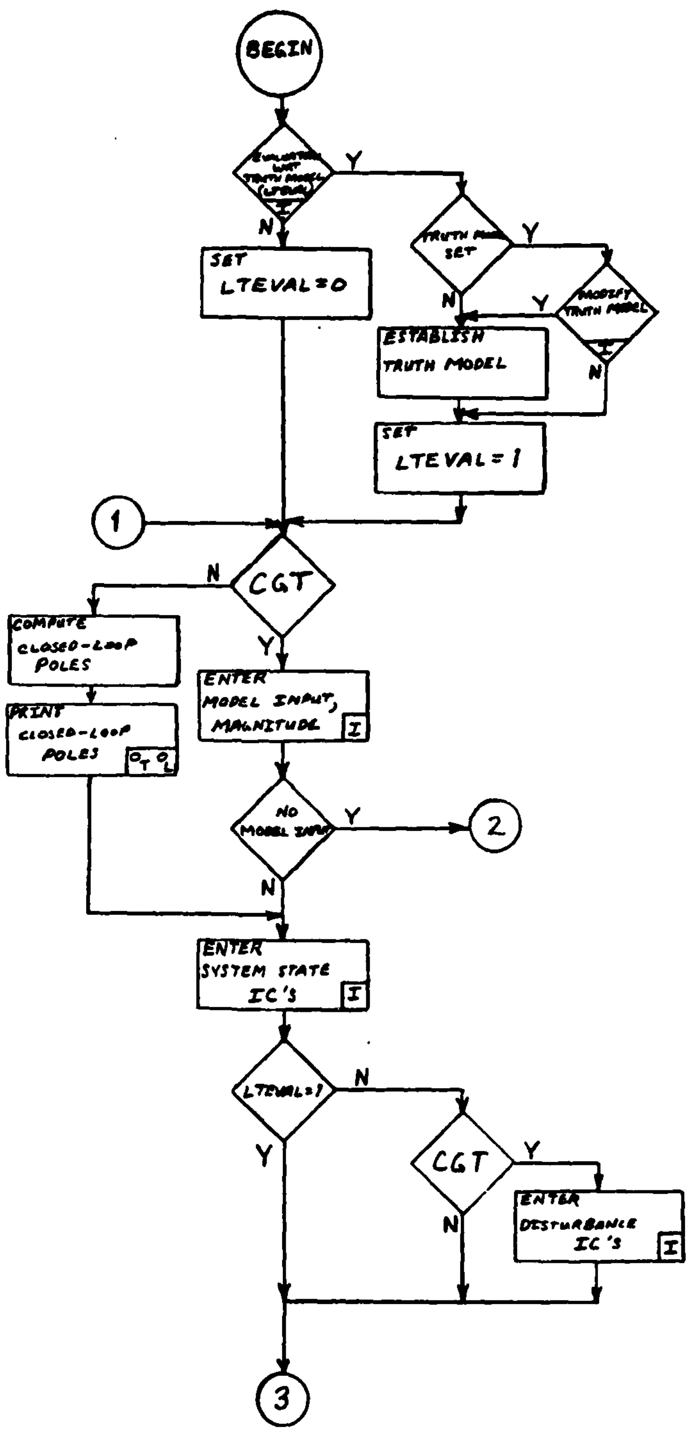


Fig. B-5. Controller Evaluation

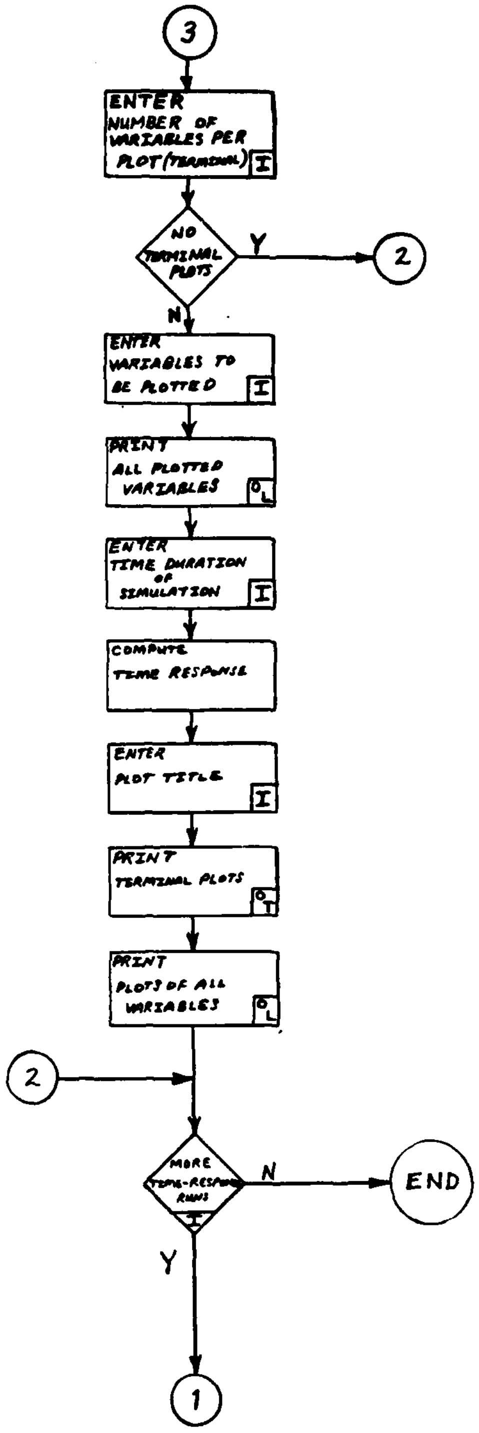

3
ENTER NUMBER OF VARIABLES PER PLOT (TERMINAL) I
NO TERMINAL PLOTS
Y
2
N
ENTER VARIABLES TO BE PLOTTED I
PRINT ALL PLOTTED VARIABLES O L
ENTER TIME DURATION OF SIMULATION I
COMPUTE TIME RESPONSE
ENTER PLOT TITLE I
PRINT TERMINAL PLOTS O T
PRINT PLOTS OF ALL VARIABLES O L
2
MORE TIME-RESPONSE RUNS I
N
END
Y
1

Fig. B-5--Continued

evaluated (CGT or PI). Decision blocks labeled "LTEVAL" test if the truth model is being used to propagate system dynamics (if not, the design model is being used).

The first prompt of the controller evaluation computational element asks if the evaluation is to be conducted with respect to (WRT) truth model dynamics. If yes, the truth model may be established or modified (if previously established) in the manner described in Section B.5.3 above. Note that, since the continuous-time representation of the truth model is not preserved within CGTPIF, "modification" actually entails complete redefinition of the model. In the case that the truth model exists on the data file and only specific array elements are to be modified, it is convenient simply to read the truth model from the DATA file and modify matrix elements as shown in Figure B-2c. If the truth model had been established previously and no modification to it is desired, the existing discrete-time representation of the truth model is used. The design model is used to propagate system dynamics if the truth model evaluation is not selected.

For the CGT evaluation, the first input prompt is for the index of the command input vector to which a step input is to be applied, and the magnitude of that step (only one command input is allowed at a time). CGTPIF tests the input index for validity (within vector length bounds); if it is invalid the prompt is repeated. If the index is zero (or negative) the input is not accepted and the user is queried as to whether time-response runs are desired. If no time-responses are to be run, the controller evaluation routines are exited; otherwise, the prompt requesting command model input specification is repeated. If the CGT controller response is to be run, initial conditions on the system states may be entered. If the design model is used for evaluation and disturbance states exist in the model, they may be given initial conditions also.

For the PI evaluation the first input prompt is for IC's for the system states. The states that are actually given initial values are those of the design or truth model, according to the model used for propagation of dynamics.

Initial conditions are entered for either controller in the same manner. Entry is of the index of the state within its appropriate state vector (design or truth model, or disturbance state vectors) and its initial value. Tests and termination of entry are as described in Section B.5.2 for multiple entries.

Time-response plots are of the "line printer" type and are output both to the terminal and to the LIST file. As many as two plots, each with as many as five dependent variables, may be printed at the user terminal. CGTPIF prompts the user to specify the number of variables for each of the two plots (the user is to enter two integers). If the user enters non-positive integers for both plots, then a prompt queries the user as to whether time-response runs are desired. If no time-responses are to be run, the routines are exited. Note that when no plots are selected for terminal printing, none are output to LIST either and no time-responses are simulated.

In the case that plots are to be printed at the terminal, a series of prompts allow the user to specify exactly which variables shall be included. Variables are selected by specifying a name of the vector type for that variable and its index in two entries for each variable. The names of the vectors are:

"X":	system state vector
"Y":	system output vector
"U":	system input vector
"D":	disturbance state vector
"M":	command model output vector

The system state vector is that of the design or truth models, according to the model used for propagation of dynamics. For example, the pair of entries "U" and "1" specifies that element 1 of the input vector $\underline{U}$ is to be plotted (note that "entry" includes a carriage return). The input prompt includes these definitions and includes only those variables relevant to the controller being evaluated. The model output and disturbance state vectors are only offered for CGT evaluations, and for the latter also only if the disturbance states are explicitly modeled and the design model propagates dynamics of the time-response (since for the truth model the system state vector includes all disturbance states which are considered to act on the system). Each user entry is tested for valid (and relevant) name and for valid index. Prompts specify the plot number and output number for each requested entry.

The user next is prompted to enter the time duration for the simulation. However, the duration actually simulated may be adjusted by the program: a time span that is the nearest integer multiple of 100 times the controller sample period is selected. Plots to the LIST file include the entire time span and use 100 equal time interval samples. Plots to the terminal include 50 time samples selected as follows: if the time duration specified by the user is less than 50 times the controller sample period, the samples plotted are the first 50 time samples from the simulation; otherwise the entire time span is included in 50 equal-interval samples. Thus, for example, with a controller sample period of .02 seconds a user specified time duration of less than 1. second would yield plots to the terminal running from time=0. to time=1. and with .02 seconds between each sample; plots to the line printer would include samples at the same interval but extending to time=2.

After completing the time-response simulation, a prompt requests a title for the plots and prescribes the field width available for the entry (50 characters). The title is applied to all plots generated from the single simulation.

Plots are printed with the independent (time) axis running vertically along the length of the output page with the origin at the top. Each sample time is identified along the left margin of the plot. The dependent axis is horizontal. Each variable is marked with an integer from 1 to 5 at each sample time. Note that since only one character can be printed in each location of the plot field, when two or more variables would occupy a single print position at a sample time, only the symbol of largest value (1 to 5) is printed. For plots to the terminal, if a model output is among the variables of a plot, then all variables in the plot are plotted over a single scale range to facilitate comparisons of actual and desired output responses. In all other cases every variable plotted is scaled over its own range independently in order to achieve greater resolution for each in the plot field. The scale(s) are listed along the bottom of the plot. Rotation of the output page through 90° in a counter-clockwise sense gives the usual abscissa-ordinate orientation. Terminal plots are 50 print positions wide; plots to LIST are 100 print positions wide.

Plots of all relevant variables in a time-response simulation (all states, inputs, and so on) are automatically output to the LIST file if terminal plots are requested. Five variables are included in each plot. A list identifying all the variables by type and index for each plot number and plot symbol is written to LIST prior to the plots.

When all plots have been printed, a prompt queries the user as to whether additional time-response runs are desired. If more are wished, the entire set of plotting options is repeated and the same controller may be evaluated under different conditions and/or different variables may be plotted. If no additional simulations are wished, the controller evaluation routines are exited.

B.5.8 <u>Kalman Filter Design ("F")</u>. The Kalman filter design routines compute the steady-state Kalman filter gains for the design model. Figure B-6 shows the I/O, logic, and computations involved. Note that the first execution of the filter design path bases its filter computations on the noise strengths specified upon initial entry of the design model. In subsequent executions, any of the noise strengths may be modified. The noise strengths are entered as vectors of the matrix diagonals (only diagonal elements may be modified). The matrices are,

"STATE NOISE STRENGTHS": $\underline{Q}$ (B-2a)

"DISTURBANCE NOISE STRENGTHS": $\underline{Q}_n$ (B-2b)

"MEASUREMENT NOISE STRENGTHS": $\underline{R}$ (B-2c)

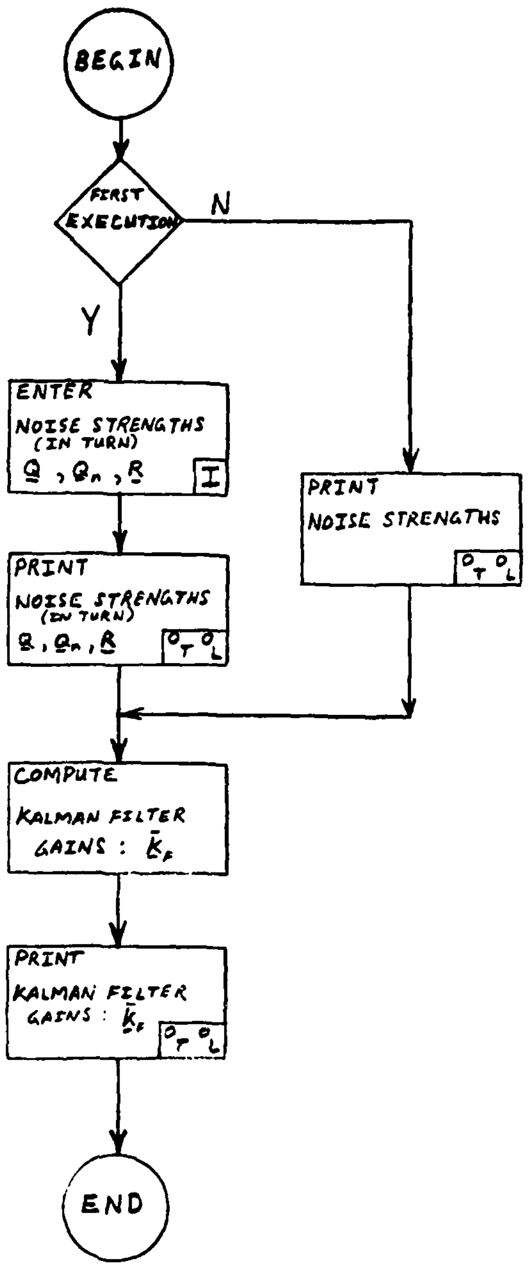

BEGIN
FIRST EXECUTION
N
Y
ENTER
NOISE STRENGTHS (IN TURN)
Q, Qn, R
I
PRINT
NOISE STRENGTHS
OT OL
PRINT
NOISE STRENGTHS (IN TURN)
Q, Qn, R
OT OL
COMPUTE
KALMAN FILTER GAINS : K̄F
PRINT
KALMAN FILTER GAINS : K̄F
OT OL
END

Fig. B-6. Kalman Filter Design

Prompts for state or disturbance noise strengths are given only if the design model specifies driving noises for the respective process dynamics. Negative noise strengths are not accepted.

In each execution of the filter design path, the entire noise strength matrices and Kalman filter gain matrix are output to the terminal and LIST file. However, only the diagonal elements of the noise strength matrices are printed at the terminal.

Following computation of the filter gains, the Kalman filter design routines are exited. Execution proceeds automatically to the filter evaluation computational element described in the next subsection.

B.5.9 Filter Evaluation ("G"). Figure B-7 shows the I/O, logic, and computations of the filter evaluation routines. Execution of the filter evaluation requires that the system truth model be established, since the covariance analysis is performed with respect to the truth model. Comments in Section B.5.7 dealing with establishing the truth model apply equally in this set of routines, except that here use of the truth model is not optional.

Evaluation begins with computation of the eigenvalues of the system-filter matrix $\underline{\Phi}_{KF}$ (equation (A-61)). As for the closed-loop PI regulated system, the discrete-time eigenvalues are mapped to the primary strip in the continuous-time domain. These mapped poles are printed

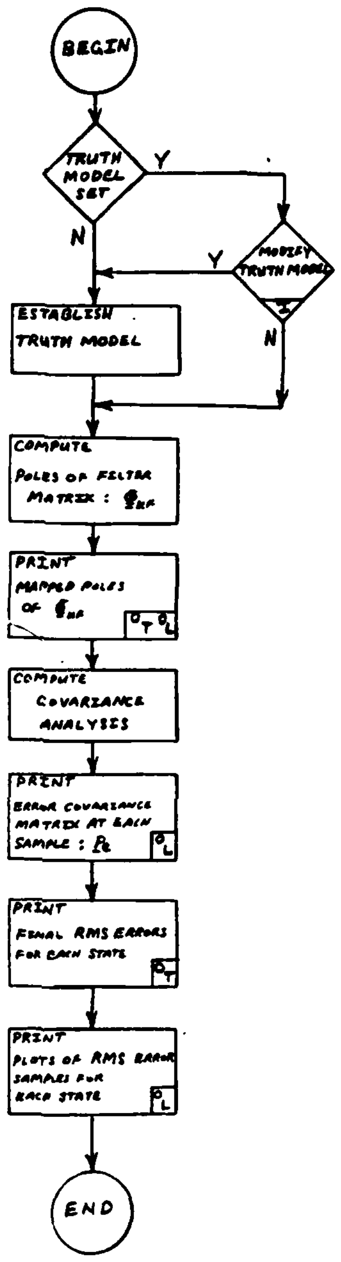

BEGIN
TRUTH MODEL SET
Y
N
MODIFY TRUTH MODEL
I
Y
N
ESTABLISH TRUTH MODEL
COMPUTE POLES OF FILTER MATRIX : Φ_{KF}
PRINT MAPPED POLES OF Φ_{KF}
O_T O_L
COMPUTE COVARIANCE ANALYSIS
PRINT ERROR COVARIANCE MATRIX AT EACH SAMPLE : P_E
O_L
PRINT FINAL RMS ERRORS FOR EACH STATE
O_T
PRINT PLOTS OF RMS ERROR SAMPLES FOR EACH STATE
O_L
END

Fig. B-7. Filter Evaluation

at the user terminal and output to the LIST file (the z-plane eigenvalues are not printed).

The primary evaluation tool applied to the filter design is a steady-state covariance analysis. The covariance matrix of the estimation errors of the filter using measurements of the truth model dynamics is propagated for 50 filter (controller) sample periods. Samples are coincident with the filter's sample times and the total number is fixed at 50. At each time sample the standard deviations of these "true" errors are computed as the square-roots of the diagonal elements of the error covariance matrix $\underline{P}_e$ (equation A-72). The filter's own computed error covariance matrix is $\underline{\bar{P}}_a$ of equation (A-58), which because of the steady-state assumption, is constant for all time samples. Taking the square-roots of the diagonal elements of $\underline{\bar{P}}_a$ then gives the filter's estimate of the standard deviations of its errors in state estimation. Plots for each state are output to the LIST file showing the "true" and "computed" RMS errors for the 50 time samples. A title may be entered to be applied to all plots from the covariance analysis. In addition, the "true" and "computed" RMS errors at the final time sample are printed at the terminal.

This completes the filter evaluation. A new filter design may then be pursued, or any other design option may be selected.

B.6 Program Messages

A variety of messages may be printed at the terminal and/or output to the LIST file during program execution. Some are purely informational, are clear in their meaning, and provide no essential insight into progress of the design or possible difficulties in program execution. Such messages are not discussed here. The remaining messages relate to errors or potential design difficulties and are considered in categories of memory allocation, dimensional errors, or computational problems.

B.6.1 Memory Allocation. CGTPIF uses vectors in named Commons for array storage. These vectors are dimensioned in the main routine and a variable in the Common is set to the value allocated. These vectors are then partitioned within CGTPIF to store individual arrays. Before storing arrays into each vector in Common, the storage needed is computed according to the appropriate equation listed in equations (A-15a) through (A-15j). In the case of temporary storage vectors, at each point in execution at which a new allocation is needed, the particular equation defining that need is used. If more memory will be needed in a vector than has been allocated, a message is written specifying the name of the Common and the necessary minimum allocation. A typical message of this type is,

```
"INSUFFICIENT MEMORY /SYSMTX/, NEED: nnnn"
```

in which the Common /SYSMTX/ has too little storage for the problem and 'nnnn' is the dimension required for the vector in that Common. For the vectors containing the dynamics models, the model with insufficient memory is identified by name. The Commons for the design, truth, and command models are /DSNMTX/, /TRUMTX/, and /CMDMTX/, respectively. After printing such a message to the user terminal, execution is aborted.

In its existing form, CGTPIF will have sufficient vector allocations to deal successfully with problems of many different combinations of dimensions and with system matrices in the range of 10 to 20 states. Since the program will not allow allocated memory to be exceeded, it is reasonable to attempt any given problem and let the program either deal with it successfully, or let it write the appropriate memory message if the problem cannot be accommodated. Section A.9 of the "Programmer's Manual" discusses the steps necessary to obtain a new CGTPIF with different memory allocations.

B.6.2 Dimensional Errors. As each of the dynamics models is established, or just prior to computations which assume relations among the design and command system and disturbance state dimensions (CGT computations), the dimensional constraints mentioned in Section B.3 are tested. In terms of the dimension notation of equations (B-4), (B-8), and (B-11) the constraints are

$$\begin{array}{ll} \text{Design Model:} & p=r \text{ and } n \geq d \\ \text{Truth Model:} & r_T=r \text{ and } m_T=m \\ \text{Command Model:} & p_M=p \text{ and } n \geq n_M \end{array} \qquad \text{(B-16)}$$

If such a constraint is not satisfied, a message is written to the user terminal identifying the problem and execution is aborted. When the constraint affects only a specific section of code, or if redefinition of the model (command or truth) can resolve the error, then only the affected execution path is aborted. In other cases, execution is aborted completely.

Other dimensional tests are made in the Kalman filter design and evaluation computational elements. For filter design, it is necessary that the system state and disturbance state driving noise dimensions not both be equal to zero, and that the number of measurements be non-zero. These are constraints on the design model:

$$\begin{array}{c} \left(\begin{array}{c} w > 0 \\ \text{or} \\ w_D > 0 \end{array} \right) \\ \text{and} \\ m > 0 \end{array} \qquad \text{(B-17)}$$

For filter evaluation, the number of driving noises for the truth model must be non-zero:

$$w_T > 0 \qquad \text{(B-18)}$$

since in this case, the system is deterministic and the covariance analysis would not provide much useful information for evaluation of the filter's performance. If any of these constraints are not satisfied, a message is written to the user terminal identifying the problem and execution of the filter design-evaluation computational elements is aborted.

B.6.3 Computational Problems. In certain of the computations, characteristics of the particular design problem may be identified as having potential impact on the attainment of design objectives. Messages identifying these characteristics may be considered informational. Other messages describe computational problems that are immediately fatal.

In computing the $\underline{\Pi}$ matrix of equation (A-29), a pair of messages may be generated to the LIST file:

```
"PI MATRIX IS RANK DEFECTIVE"
```

and

```
"nr X nc MT RANK mr"
```

in which "nr" and "nc" are the row and column dimensions of $\underline{\Pi}$ and "mr" is its rank. The first message is also printed at the user terminal. The equations employing $\underline{\Pi}$ assume it to be an ordinary matrix inverse. If it is rank defective, the matrix pseudo-inverse is computed instead. Execution of the program continues since the discussion of Reference 32 concerning the $\underline{\Pi}$ matrix suggests that useful results may still be obtained through use of the pseudo-inverse.

Solution of the Riccati equations for the PI regulator (equation (A-43)) and the Kalman filter (equation (A-58)) is achieved using an iterative algorithm (Ref 24) which may generate messages of information or fatal error. The informative message for the PI is,

"RICCATI SOLN IS PSD--RANK mr"

in which "PSD" means positive semi-definite. For the Kalman filter the corresponding message is,

"OBSERVABILITY MATRIX IS nr X nc OF RANK mr"

in which "nr", "nc", and "mr" are the row, column dimensions and the rank, respectively. These messages convey the same information concerning system observability. The message is written in the case of the PI Riccati equation only if the solution is positive semi-definite (rank defective). Both messages are output to the LIST file only.

For both the PI and filter Riccati equations fatal error messages are identical:

"RICCATI NON-CONVERGENT IN nn ITERATIONS"

or

"RICCATI BLOW UP AT ITERATION nn INITIAL N = mm" in which "nn" is the iteration counter at the occurrence of the error and "mm" is the value of a variable set internally and used in achieving initialization of the iterative sequence (the internal variable is not available for modification by the user). The first message indicates that the sequence of iterates did not converge. The second message may indicate numerical difficulties or uncontrollability (unobservability) of the system of the PI (filter) equations. Both messages are output to the LIST file only; a system error exit routine is then called which writes "EXIT" to the user terminal and aborts program execution. Note that in the event of such an abort, the local files SAVE, DATA, and LIST are not rewound automatically.

In computing the CGT controller gains an error may occur in solving for the matrix partitions $\underline{A}_{11}$ or $\underline{A}_{13}$ (see Section A.11.5 of the "Programmer's Guide"). If the iterative refinements to these solutions do not converge to within the established tolerance (1.E-6), then the following message is written both to the terminal and the LIST file:

```
"SOLUTION ERROR FOR 'A' (CGT) AFTER 3 ITERATIONS = nnn"
```

in which "nnn" is the Euclidean norm of the refining matrix solution (residual) at the last iteration. The message is considered to be informational, and execution proceeds normally. However, in case the value of the residual norm is large compared to the convergence tolerance, the CGT design solution can be expected to be invalid.

B.7 Running CGTPIF

This "User's Guide" assumes that a segmented executable object file of CGTPIF exists. If it does not, refer to the "Programmer's Guide" for instructions for obtaining such a file.

For an existing CGTPIF object file the following commands must be entered in INTERCOM to run the program:

```
CONNECT,INPUT,OUTPUT
ATTACH,CGTPIF,pfn
CGTPIF
```

in which 'pfn' is the permanent file on which the object file is cataloged. CGTPIF will then execute as described in Section B.5. Additional commands before and after execution may be appropriate according to one's intended use of the various local files which CGTPIF employs during execution. Refer to Section B.4 for suggested commands relevant to file usage.

Appendix C

CGTPIF Input/Output Listing

The input/output (I/O) listing given here is from a single execution of CGTPIF. It shows two PI regulator designs, a CGT/PI design, and a Kalman filter design for design model AFTI(S3,A2,G3). The regulators and controller, as well as the filter covariance analysis, are all evaluated with respect to the truth model AFTI(S4,A2,G3). The controller design is for the pitch-pointing decoupled control law. Details concerning the design, truth, and command models employed, as well as information about the results of these designs, are given in Chapter VI of this study.

The I/O shown is that obtained directly at the user terminal during execution. The listing is complete and in order, but it has been divided into individual page-sized portions for presentation. During execution, additional extensive output was placed on the 'LIST' file. The LIST file's output is not reproduced here. It extends as continuous listing over about 45 pages and uses an output field width of 125 character positions. However, Section C.2 gives a brief description of the output appearing on LIST for this single execution. Refer to the

"Programmer's Guide" and the "User's Guide" for descriptions of the terminology used to refer to the various items of I/O given here.

C.1 CGTPIF Terminal I/O Listing

C.1.1 Introduction. For this design, both the design model and the truth model are obtained from a 'DATA' file. The command model is of low dimension and is entered directly.

In the listing below, all user entries are identified with an arrow symbol to the immediate right of each entry. Prior to and following program execution, INTERCOM prompts for input are given by "COMMAND-". Within program execution entries occur in two ways: (1) when the entry is on the same line as an input prompt, the entry is bounded on the left by the symbol ">"; (2) in case of multiple entries for a single prompt, entries after the first include the entire line that is identified.

Portions of I/O are discussed within individual numbered paragraphs. Each portion of listing begins on a new page. The specific portions of listing are identified by a number in parentheses at the top center of the page where it begins, and these numbers correspond to the paragraph numbers below.

C.1.2 Summary of Input/Output.

(1) Following "LOGIN", the executable object file 'CGTPIF' is attached, as well as the 'DATA' file containing

the design and truth models. Note that 'CGTPIF' and 'DATA' are local file names while 'THESIS' and 'DESIGN' are the corresponding permanent file names. Next, permanent file space is requested for the local file name 'SAVE'; the SAVE file will be generated during subsequent execution of the program. The "CONNECT" command defines the user terminal as the device that communicates through the FORTRAN standard 'INPUT' and 'OUTPUT' files. Program execution is initiated with the simple command "CGTPIF", which loads the local file CGTPIF and begins execution at its starting address.

(2) Program execution begins with output of an identifying header which includes the current date and time (obtained from calls to system real-time clock routines). The first user entry is the sample period of the controller. Next, the design model is established. The design model is read from the local file DATA. As described in Chapter VI of this thesis, two different controllers were designed for this aircraft dynamics model and for each there were different definitions of the output matrix $\underline{C}$ of the design model. Thus, the $\underline{C}$ matrix is listed in order to verify that the data corresponds to the pitch-pointing design case. Since the $\underline{C}$ matrix is correct, no changes are made to its elements (immediate entry of "0/" when requested to enter element address and value). The design model is then written to the SAVE file. The

poles (eigenvalues) of the design model ($\underline{A}$ matrix) are automatically computed and printed.

(3) The controller design path is then pursued, and a PI regulator design is chosen. Quadratic weights of 200. on outputs 1 and 2, and of 1. on input magnitudes and rates are entered. Weights of the $\underline{X}$ matrix (augmented state and input magnitude weighting matrix) are not modified. The PI gains $\underline{K}_x$ and $\underline{K}_z$ are computed and printed.

(4) The evaluation of the PI regulator is chosen to be with respect to the truth model dynamics. The truth model is read from the DATA file, is not modified, and is written to the SAVE file. The poles of the truth model ($\underline{A}_t$ matrix) are automatically computed and printed.

(5) Evaluation of the PI regulator begins with computation and printout of the continuous-time mapped poles of the closed-loop system matrix ($\underline{\Phi}_{\delta CL}$). In preparation for a time-response simulation of the closed-loop system, initial conditions of 0.01 and -0.01 for states 1 and 2 of the truth model are entered. One plot of 2 variables is printed at the terminal. The variables are selected as outputs 1 and 2 ($\underline{y}(1)$ and $\underline{y}(2)$). A time duration of 0.9 seconds is selected, which will give a plot including all of the first 50 controller sample times and will run for a duration of 1. second at the terminal (here T=0.02, and 50·T=1.). An identifying title is specified for the plot. In the resulting plot the time-axis is vertical, the dependent axis is horizontal, and plot

symbols 1 and 2 identify the plotted variables in order as specified above. Note that a rotation of the plot through 90° in the counter clock-wise sense gives the usual abscissa-ordinate orientation. Both plot variables are scaled individually over ranges of (-0.0090 to 0.0110) and (-0.0050 to 0.0200), respectively.

(6) No additional time-responses are wished, and the design of the PI regulator is repeated. In this iteration, the quadratic weights are not modified (weights on outputs, input magnitudes, and input rates are preserved throughout program execution, unless specifically modified). However, the $\underline{X}$ weighting matrix is modified to include a weight of 50. on state 3 of the design model (weight is value of element $\underline{X}(3,3)$). Note that the $\underline{X}$ matrix is computed anew each iteration from the weighting matrices on the outputs and the input magnitudes. Thus, modifications made explicitly to $\underline{X}$ are not preserved between design iterations. The PI gains $\underline{K}_x$ and $\underline{K}_z$ are computed and printed.

(7) Evaluation of the PI regulator is again taken with respect to the truth model, which is left as it had been previously defined. The evaluation proceeds in the same way as described in Paragraph (5) above. Note the improved damping in the responses of both outputs.

(8) No additional time-responses are selected. Having achieved a satisfactory PI design, a CGT/PI design is pursued next. The command model is entered directly

from the terminal. It is defined as a 2 state, 2 input, and 2 output model. The matrix $\underline{A}_m$ is diagonal with values of -5. for both entries; the matrix $\underline{B}_m$ is diagonal with values of 0.1 for both entries; the matrix $\underline{C}_m$ is diagonal with values of 1. for both entries; finally the matrix $\underline{D}_m$ is the zero matrix. The command model is written to the SAVE file. The poles of the command model ($\underline{A}_m$ matrix) are computed and printed. The equations defining the CGT controller are solved and the CGT/PI feedforward gains $\underline{K}_{x_m}$ and $\underline{K}_{x_u}$ are computed and printed (since the design model does not include disturbance states, the matrix $\underline{K}_{x_n}$ does not exist).

(9) The CGT/PI controller is evaluated with respect to the truth model, as previously defined. A time-response simulation is run for a step of magnitude 1. on command model input 1 ($\underline{u}_m(1)$ = unit step), and no initial conditions are set on the truth model states. Two plots are printed to the terminal of 4 and 2 variables each. The first plot includes (pairwise) outputs 1 and 2 of the truth model and command model (plot symbol 1 is $\underline{y}(1)$, plot symbol 2 is $\underline{y}_m(1)$, and so on). The second plot includes states 5 and 6 of the truth model (these are the control actuator states). Since the first plot includes outputs of the command model, a single scale range is applied to all four variables in the plot. A time duration of 0.9 seconds is specified (which gives the first 50 controller samples again), and a title is entered.

The resulting time-response plots follow.

(10) No additional time-responses of the CGT/PI controller are requested. The Kalman filter design path is then selected. Since this is the first execution of the Kalman filter design, the noise strength matrices ($\underline{Q}$ and $\underline{R}$) specified in the existing definition of the design model are used to compute the Kalman filter gain matrix (note that in subsequent iteration of the Kalman filter design path, the user may modify the diagonal elements of the noise strength matrices). The diagonal elements of the noise strength matrices and the entire filter gain matrix are printed. Next, a title is entered to apply to the plots of state estimation error standard deviations (output to LIST file only). The final values of true and filter-computed RMS errors for the design model state estimates are printed.

(11) The filter design is not repeated, nor are any of the other designs. Upon terminating execution, the PI gains determined previously (Paragraph (6)) are written to the SAVE file. The command "FILES" gives a listing of existing files. Note that files SAVE, LIST, and PLOT have been generated automatically during program execution. The SAVE file (containing the design, truth, and command model data as well as the PI gains $\underline{K}_x$ and $\underline{K}_z$) is then cataloged for future use (as a DATA file). The already existing (attached) DATA file is then returned and the just created SAVE file is copied to the local file named

DATA. Next, the LIST file is sent to a line printer for listing. Finally, the SAVE file is returned, making re-execution of the program feasible (since SAVE had been made a permanent file, it could not be written to in subsequent executions). In a subsequent execution, the various dynamics models and the PI gains would be available from the new DATA file and a new SAVE file would be created (if desired). However, in this case there is no repeated execution of the program, and the user enters a "LOGOUT" from the system.

```
ASD COMPUTER CENTER - INTERCOM 5.1
SYSTEM  CSA
DATE  11/19/81       TIME  09.48.56.

PLEASE LOGIN
LOGIN,D790477 <—
[redacted] ENTER PASSWORD-


11/19/81     LOGGED IN AT  09.49.34.
             WITH USER-ID PD
             EQUIP/PORT 16/051
COMMAND- ATTACH,CGTPIF,THESIS <—
 AT CY= 100 SN=AFFDL
COMMAND- ATTACH,DATA,DESIGN,CY=1 <—
COMMAND- REQUEST,SAVE,*PF <—
COMMAND- CONNECT,INPUT,OUTPUT <—
COMMAND- CGTPIF <—
```

(2)

```
                              * * * CGTPIF * * *
                  PROGRAM TO DESIGN A COMMAND GENERATOR TRACKER
             USING A REGULATOR WITH PROPORTIONAL PLUS INTEGRAL CONTROL
                   AND A KALMAN FILTER FOR STATE ESTIMATION.
                              * * * CGTPIF * * *

      DATE :  11/19/81

      TIME :  09.50.33.


ENTER SAMPLE PERIOD FOR DIGITAL CONTROLLER >.02 <—

READ DESIGN  MODEL FROM 'DATA' FILE (Y OR N) >Y <—

MODIFY MATRIX ELEMENTS (Y OR N) >Y <—
  A     B     EX    G     Q     C     DY    EY    H     HN    R     AN    GN    QN
ENTER MATRIX NAME >C <—
LIST MATRIX TO TERMINAL (Y OR N) >Y <—

C   MATRIX

    1.000        0.           0.           0.           0.           0.
         0.          0.
    1.000       -1.000        0.           0.           0.           0.
         0.          0.
ENTER I,J AND M(I,J)--(0/ WHEN COMPLETE) :  2 BY  8 >0/ <—

MODIFY MATRIX ELEMENTS (Y OR N) >N <—

WRITE DESIGN  MODEL TO 'SAVE' FILE (Y OR N) >Y <—
     DESIGN  MODEL WRITTEN TO 'SAVE' FILE

POLES OF DESIGN  MATRIX

         1.3497279E-03  +J(  0.           )
         1.2965361E+00  +J(  0.           )
        -3.6658859E+00  +J(  0.           )
        -2.0000000E+01  +J(  0.           )
        -2.0000000E+01  +J(  0.           )
        -4.9251429E-01  +J(  0.           )
        -2.2564490E+01  +J(  0.           )
        -4.9251429E-01  +J(  0.           )
```

(3)

```
CONTROLLER DESIGN (Y OR N) >Y <-

DESIGN REG/PI (Y OR N) >Y <-
ENTER WEIGHTS ON OUTPUT DEVIATIONS:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >1 200. <-
2 200. <-
0/ <-
ENTER WEIGHTS ON CONTROL MAGNITUDES:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >1 1. <-
2 1. <-
0/ <-
ENTER WEIGHTS ON CONTROL RATES:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >1 1. <-
2 1. <-
0/ <-

Y   MATRIX

    200.0
    200.0

UM  MATRIX

    1.000
    1.000

MODIFY ELEMENTS OF 'X' MATRIX (Y OR N) >N <-

UR  MATRIX

    1.000
    1.000

KX  MATRIX

   -24.12         19.93         -1.127         1.098        1.3509E-02    .19
75        -67.99         2.4492E-02
    81.36        -83.53          .5534        1.1577E-02    1.034        -.83
15         280.0         6.5388E-03

KZ  MATRIX

    .2172        -.9149
   -2.925         3.051
```

(4)

```
CONTROLLER EVALUATION WRT TRUTH MODEL (Y OR N) >Y  <—

READ TRUTH   MODEL FROM 'DATA' FILE (Y OR N) >Y  <—

MODIFY MATRIX ELEMENTS (Y OR N) >N  <—

WRITE TRUTH   MODEL TO 'SAVE' FILE (Y OR N) >Y  <—
     TRUTH   MODEL WRITTEN TO 'SAVE' FILE

POLES OF TRUTH   MATRIX

      -1.7562050E-02  +J(  1.3462117E-01)
      -1.7562050E-02  +J( -1.3462117E-01)
       1.2486051E+00  +J(  0.            )
      -3.9022986E+00  +J(  0.            )
      -2.0000000E+01  +J(  0.            )
      -2.0000000E+01  +J(  0.            )
      -4.9251429E-01  +J(  0.            )
      -2.2564490E+01  +J(  0.            )
      -4.9251429E-01  +J(  0.            )
```

(5)

```
POLES OF REGPI   MATRIX

       -4.2981128E+00  +J(  0.             )
       -1.0352022E+01  +J(  1.4079013E+01)
       -1.0352022E+01  +J( -1.4079013E+01)
       -2.4472260E+01  +J(  0.             )
       -1.9612664E+01  +J(  0.             )
       -5.4860340E+01  +J(  0.             )
       -5.4957660E+01  +J(  0.             )
       -2.2564493E+01  +J(  0.             )
       -4.9251461E-01  +J(  0.             )
       -4.9251396E-01  +J(  0.             )

ENTER STATE AND IC VALUE (0/ TERMINATES):  9 >1 .01
2 -.01
0/
2 PLOTS OF 5 VARIABLES MAY BE PRINTED AT THE TERMINAL -- SPECIFY NUMBER
FOR EACH (N1,N2) >2 0
ENTER OUTPUTS BY TYPE AND INDEX IN 2 ENTRIES--TYPES ARE
STATE : 'X'
OUTPUT : 'Y'
INPUT : 'U'

PLOT  1
OUTPUT  1 >Y
          >1
OUTPUT  2 >Y
          >2
ENTER TIME DURATION FOR RESPONSE, IN SECONDS >.9
+---------- ENTER TITLE IN GIVEN FIELD ----------+
AFTI(S3,A2,G3) PITCH POINTING PI
```

```
       AFTI(S3,A2,G3) PITCH POINTING PI
   0.00      +         +         +         +         +      1  2
    .02      +         +         +         +         +     21  +
    .04      +         +         +         +        2+     1   +
    .06      +         +         +         +  2      +    1    +
    .08      +         +         +      2  +         +  1      +
    .10      +         +         + 2       +         +1        +
    .12      +         +       2 +         +       1 +         +
    .14      +         +     2   +         +    1    +         +
    .16      +         +   2     +         +1        +         +
    .18      +: : : : :+:2: : : :+: : : :1:+: : : : :+: : : : :+
    .20      +         +2        +    1    +         +         +
    .22      +         2         + 1       +         +         +
    .24      +         2        1+         +         +         +
    .26      +        2+     1   +         +         +         +
    .28      +        2+  1      +         +         +         +
    .30      +       2 1         +         +         +         +
    .32      +       2 +         +         +         +         +
    .34      +     12  +         +         +         +         +
    .36      +   1  2  +         +         +         +         +
    .38      +: 1 :2: :+: : : : :+: : : : :+: : : : :+: : : : :+
    .40      + 1   2   +         +         +         +         +
    .42      +1   2    +         +         +         +         +
    .44      1    2    +         +         +         +         +
    .46      1   2     +         +         +         +         +
    .48      1  2      +         +         +         +         +
    .50      1  2      +         +         +         +         +
    .52      1 2       +         +         +         +         +
    .54      +12       +         +         +         +         +
    .56      + 2       +         +         +         +         +
    .58      +2 1 : : :+: : : : :+: : : : :+: : : : :+: : : : :+
    .60      +2   1    +         +         +         +         +
    .62      +2    1   +         +         +         +         +
    .64      +2      1 +         +         +         +         +
    .66      +2        1         +         +         +         +
    .68      +2        + 1       +         +         +         +
    .70      +2        +   1     +         +         +         +
    .72      +2        +     1   +         +         +         +
    .74      +2        +       1 +         +         +         +
    .76      +2        +         +1        +         +         +
    .78      +:2: : : :+: : : : :+: 1 : : :+: : : : :+: : : : :+
    .80      + 2       +         +    1    +         +         +
    .82      + 2       +         +      1  +         +         +
    .84      +  2      +         +        1+         +         +
    .86      +  2      +         +         +1        +         +
    .88      +   2     +         +         + 1       +         +
    .90      +    2    +         +         +   1     +         +
    .92      +    2    +         +         +    1    +         +
    .94      +     2   +         +         +      1  +         +
    .96      +     2   +         +         +       1 +         +
    .98      +: : : 2 :+: : : : :+: : : : :+: : : :1:+: : : : :+
   1.00      +       2 +         +         +        1+         +

SCALE 1     -.0090    -.0050    -.0010     .0030     .0070     .0110

SCALE 2     -.0050    -.0000     .0050     .0100     .0150     .0200
```

(6)

```
MORE TIME RESPONSE RUNS (Y OR N) >N

CONTROLLER DESIGN (Y OR N) >Y

DESIGN REG/PI (Y OR N) >Y
ENTER WEIGHTS ON OUTPUT DEVIATIONS:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >0/
ENTER WEIGHTS ON CONTROL MAGNITUDES:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >0/
ENTER WEIGHTS ON CONTROL RATES:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >0/

Y   MATRIX

    200.0
    200.0

UM  MATRIX

    1.000
    1.000

MODIFY ELEMENTS OF 'X' MATRIX (Y OR N) >Y
LIST 'X' MATRIX TO TERMINAL (Y OR N) >N
ENTER I,J AND M(I,J)--(0/ WHEN COMPLETE) : 10 BY 10 >3 3 50.
0/

UR  MATRIX

    1.000
    1.000

KX  MATRIX

   -38.87         19.69        -1.687         1.282        5.7119E-02    .18
51        -65.16        2.9382E-02
    78.03        -83.61         .4227        5.5022E-02    1.045        -.83
24         280.0        7.6946E-03

KZ  MATRIX

   7.0350E-03   -.8717
  -2.929         3.068
```

```
CONTROLLER EVALUATION WRT TRUTH MODEL (Y OR N) >Y <—
MODIFY TRUTH MODEL (Y OR N) >N <—

POLES OF REGPI   MATRIX

       -2.9827602E+01  +J(  4.2991187E+01)
       -2.9827602E+01  +J( -4.2991187E+01)
       -1.9616067E+01  +J(  0.            )
       -5.4953050E+01  +J(  0.            )
       -2.2652939E+00  +J(  0.            )
       -7.3649808E+01  +J(  0.            )
       -3.8538426E+00  +J(  0.            )
       -2.2564493E+01  +J(  0.            )
       -4.9251429E-01  +J(  2.3149970E-09)
       -4.9251429E-01  +J( -2.3149970E-09)

ENTER STATE AND IC VALUE (0/ TERMINATES):  9 >1 .01 <—
2 -.01 <—
0/ <—
2 PLOTS OF 5 VARIABLES MAY BE PRINTED AT THE TERMINAL -- SPECIFY NUMBER
FOR EACH (N1,N2) >2 0 <—
ENTER OUTPUTS BY TYPE AND INDEX IN 2 ENTRIES--TYPES ARE
STATE : 'X'
OUTPUT : 'Y'
INPUT : 'U'

PLOT  1
OUTPUT  1 >Y <—
          >1 <—
OUTPUT  2 >Y <—
          >2 <—
ENTER TIME DURATION FOR RESPONSE, IN SECONDS >.9 <—
+---------- ENTER TITLE IN GIVEN FIELD ----------+
AFTI(S3,A2,G3) PITCH POINTING PI' <—
```

```
        AFTI(S3,A2,G3) PITCH POINTING PI'

    0.00      +         +         +         +         +     2   1
     .02      +         +         +         +         + 2      1+
     .04      +         +         +         +     2   +     1   +
     .06      +         +         +         2         +1        +
     .08      +         +         +   2     +   1     +         +
     .10      +         +       2 +      1  +         +         +
     .12      +         +   2    1+         +         +         +
     .14      +         +2 1      +         +         +         +
     .16      +      12 +         +         +         +         +
     .18      +: :1:2: :+: : : : :+: : . : :+: : : : :+: : : : :+
     .20      + 1 2     +         +         +         +         +
     .22      + 12      +         +         +         +         +
     .24      + 2       +         +         +         +         +
     .26      +2  1     +         +         +         +         +
     .28      +2    1   +         +         +         +         +
     .30      2       1 +         +         +         +         +
     .32      2        1+         +         +         +         +
     .34      2         +1        +         +         +         +
     .36      +2        + 1       +         +         +         +
     .38      +2 : : : :+: 1 : : :+: : : : :+: : : : :+: : : : :+
     .40      +2        +   1     +         +         +         +
     .42      +2        +   1     +         +         +         +
     .44      +2        +   1     +         +         +         +
     .46      +2        +   1     +         +         +         +
     .48      + 2       +   1     +         +         +         +
     .50      + 2       +   1     +         +         +         +
     .52      + 2       +   1     +         +         +         +
     .54      + 2       +   1     +         +         +         +
     .56      + 2       +   1     +         +         +         +
     .58      +:2: . : :+: :1: : :+. : : . :+: : : : :+: : : : .+
     .60      + 2       +   1     +         +         +         +
     .62      + 2       +   1     +         +         +         +
     .64      + 2       +   1     +         +         +         +
     .66      + 2       +   1     +         +         +         +
     .68      + 2       +   1     +         +         +         +
     .70      + 2       +    1    +         +         +         +
     .72      +  2      +    1    +         +         +         +
     .74      +  2      +    1    +         +         +         +
     .76      +  2      +    1    +         +         +         +
     .78      +: 2 : : :+: : 1 : :+: : : : :+: : : : :+: : : : :+
     .80      +  2      +    1    +         +         +         +
     .82      +  2      +    1    +         +         +         +
     .84      +  2      +    1    +         +         +         +
     .86      +  2      +    1    +         +         +         +
     .88      +  2      +     1   +         +         +         +
     .90      +  2      +     1   +         +         +         +
     .92      +  2      +     1   +         +         +         +
     .94      +  2      +     1   +         +         +         +
     .96      +   2     +     1   +         +         +         +
     .98      +: :2: : :+: : :1: :+: : : : :+: : : : :+: : : : :+
    1.00      +   2     +     1   +         +         +         +

SCALE 1      -.0050    -.0020     .0010     .0040     .0070     .0100

SCALE 2      -.0030     .0020     .0070     .0120     .0170     .0220
```

(8)

```
MORE TIME RESPONSE RUNS (Y OR N) >N <—

CONTROLLER DESIGN (Y OR N) >Y <—

DESIGN REG/PI (Y OR N) >N <—

DESIGN CGT (Y OR N) >Y <—

READ COMMAND MODEL FROM 'DATA' FILE (Y OR N) >N <—
ENTER COMMAND MODEL FROM TERMINAL (Y OR N) >Y <—
ENTER NM >2 <—
ENTER RM >2 <—
ENTER PM >2 <—

ENTER AM
ENTER I,J AND M(I,J)--(0/ WHEN COMPLETE) :  2 BY  2 >1 1 -5. <—
2 2 -5. <—
0/ <—

ENTER BM
ENTER I,J AND M(I,J)--(0/ WHEN COMPLETE) :  2 BY  2 >1 1 .1 <—
2 2 .1 <—
0/ <—

ENTER CM
ENTER I,J AND M(I,J)--(0/ WHEN COMPLETE) :  2 BY  2 >1 1 1. <—
2 2 1. <—
0/ <—

ENTER DM
ENTER I,J AND M(I,J)--(0/ WHEN COMPLETE) :  2 BY  2 >0/ <—

MODIFY MATRIX ELEMENTS (Y OR N) >N <—

WRITE COMMAND MODEL TO 'SAVE' FILE (Y OR N) >Y <—
     COMMAND MODEL WRITTEN TO 'SAVE' FILE

POLES OF COMMAND MATRIX

      -5.0000000E+00   +J(   0.            )
      -5.0000000E+00   +J(   0.            )

KXM MATRIX

   -9.482        -14.92
   -18.47         61.13

KXU MATRIX

   -.1181        -.1711
  -2.1486E-02     .7287
```

(9)

```
CONTROLLER EVALUATION WRT TRUTH MODEL (Y OR N) >Y <—
MODIFY TRUTH MODEL (Y OR N) >N <—

ENTER MODEL INPUT AND STEP VALUE : 1 >1 1. <—

ENTER STATE AND IC VALUE (0/ TERMINATES):  9 >0/ <—
2 PLOTS OF 5 VARIABLES MAY BE PRINTED AT THE TERMINAL -- SPECIFY NUMBER
FOR EACH (N1,N2) >4 2 <—
ENTER OUTPUTS BY TYPE AND INDEX IN 2 ENTRIES--TYPES ARE
STATE : 'X'
OUTPUT : 'Y'
INPUT : 'U'
MODEL : 'M'

PLOT  1
OUTPUT  1 >Y <—
          >1 <—
OUTPUT  2 >M <—
          >1 <—
OUTPUT  3 >Y <—
          >2 <—
OUTPUT  4 >M <—
          >2 <—

PLOT  2
OUTPUT  1 >X <—
          >5 <—
OUTPUT  2 >X <—
          >6 <—
ENTER TIME DURATION FOR RESPONSE, IN SECONDS >.9 <—
+---------- ENTER TITLE IN GIVEN FIELD ----------+
AFTI(S3,A2,G3) PITCH POINTING CGT/PI <—
```

```
                    AFTI(S3,A2,G3) PITCH POINTING CGT/PI

  0.00      + 4       +         +         +         +         +
   .02      +34  2    +         +         +         +         +
   .04      +341     2+         +         +         +         +
   .06      3 4 1     + 2       +         +         +         +
   .08      3 4    1  +    2    +         +         +         +
   .10      3 4       +1     2  +         +         +         +
   .12      3 4       +   1     2         +         +         +
   .14      3 4       +       1 + 2       +         +         +
   .16      3 4       +         + 1 2     +         +         +
   .18      3:4: : : :+: : : : :+: : 2 : :+: : . : .+: : : : :+
   .20      3 4       +         +      21 +         +         +
   .22      3 4       +         +       2 1         +         +
   .24      +34       +         +        2+ 1       +         +
   .26      +34       +         +         +2  1     +         +
   .28      +34       +         +         + 2  1    +         +
   .30      +34       +         +         +  2 1    +         +
   .32      +34       +         +         +  2  1   +         +
   .34      + 4       +         +         +   2 1   +         +
   .36      + 4       +         +         +    21   +         +
   .38      +:4: : : :+: : : : :+: : : : :+: : :21 :+: : : : :+
   .40      + 4       +         +         +     21  +         +
   .42      + 4       +         +         +      2  +         +
   .44      + 4       +         +         +      21 +         +
   .46      + 4       +         +         +      21 +         +
   .48      + 4       +         +         +       2 +         +
   .50      + 4       +         +         +       21+         +
   .52      + 4       +         +         +        2+         +
   .54      + 4       +         +         +        2+         +
   .56      + 4       +         +         +        21         +
   .58      +:4: : : :+: : : : :+: : : : :+: : : . 21: : : : :+
   .60      + 4       +         +         +         2         +
   .62      + 4       +         +         +         2         +
   .64      + 4       +         +         +         21        +
   .66      + 4       +         +         +         21        +
   .68      + 4       +         +         +         21        +
   .70      + 4       +         +         +         21        +
   .72      + 4       +         +         +         21        +
   .74      + 4       +         +         +         +2        +
   .76      + 4       +         +         +         +2        +
   .78      +:4: : : :+: : : : :+: : : : :+: : : : :+2 : : . :+
   .80      + 4       +         +         +         +2        +
   .82      + 4       +         +         +         +2        +
   .84      + 4       +         +         +         +2        +
   .86      + 4       +         +         +         +2        +
   .88      + 4       +         +         +         +2        +
   .90      + 4       +         +         +         +2        +
   .92      + 4       +         +         +         +2        +
   .94      + 4       +         +         +         +2        +
   .96      + 4       +         +         +         +2        +
   .98      +:4: : : :+: : : : :+: : : : :+: : : : :+2 : : : :+
  1.00      + 4       +         +         +         +2        +

SCALE      -.0010    .0040     .0090     .0140     .0190     .0240
```

```
               AFTI(S3,A2,G3) PITCH POINTING CGT/PI

      0.00     +         +         +   1     +         +     2   +
       .02     +    1    +         +         +         +   2     +
       .04     1         +         +         +         2         +
       .06     +  1      +         +         +     2   +         +
       .08     +         +1        +         + 2       +         +
       .10     +         +         1        2+         +         +
       .12     +         +         +      21+         +         +
       .14     +         +         +   2     +     1   +         +
       .16     +         +         + 2       +         +1        +
       .18     +: : : : :+: : : : :2: : : : :+: : : : :+: : 1 : :+
       .20     +         +        2+         +         +      1  +
       .22     +         +      2  +         +         +       1 +
       .24     +         +    2    +         +         +      1  +
       .26     +         +  2      +         +         +     1   +
       .28     +         + 2       +         +         +    1    +
       .30     +         2         +         +         +  1      +
       .32     +        2+         +         +         + 1       +
       .34     +       2 +         +         +         +1        +
       .36     +      2  +         +         +         1         +
       .38     +: : 2 : :+: : : : :+: : : : :+. : : : 1+: : : : :+
       .40     +     2   +         +         +        1+         +
       .42     +    2    +         +         +         1         +
       .44     +    2    +         +         +         1         +
       .46     +   2     +         +         +         1         +
       .48     +   2     +         +         +         +1        +
       .50     +   2     +         +         +         +1        +
       .52     +  2      +         +         +         + 1       +
       .54     +  2      +         +         +         + 1       +
       .56     +  2      +         +         +         + 1       +
       .58     +: 2 : : :+: : : : :+: : : : :+: : : : :+:1: : : :+
       .60     + 2       +         +         +         + 1       +
       .62     + 2       +         +         +         + 1       +
       .64     + 2       +         +         +         + 1       +
       .66     + 2       +         +         +         + 1       +
       .68     + 2       +         +         +         + 1       +
       .70     + 2       +         +         +         + 1       +
       .72     +2        +         +         +         + 1       +
       .74     +2        +         +         +         + 1       +
       .76     +2        +         +         +         + 1       +
       .78     +2 : : : :+: : : : :+: : : : :+: : : : :+:1: : : :+
       .80     +2        +         +         +         + 1       +
       .82     +2        +         +         +         + 1       +
       .84     +2        +         +         +         + 1       +
       .86     +2        +         +         +         + 1       +
       .88     +2        +         +         +         + 1       +
       .90     +2        +         +         +         + 1       +
       .92     +2        +         +         +         + 1       +
       .94     +2        +         +         +         + 1       +
       .96     +2        +         +         +         + 1       +
       .98     +2 : : : :+: : : : :+: : : : :+: : : : :+:1: : : :+
      1.00     +2        +         +         +         + 1       +

SCALE 1      -.0490     -.0290     -.0090     .0110      .0310      .0510

SCALE 2      -.1400     -.1100     -.0800     -.0500     -.0200     .0100
```

(10)

```
MORE TIME RESPONSE RUNS (Y OR N) >N <—

CONTROLLER DESIGN (Y OR N) >N <—

FILTER DESIGN (Y OR N) >Y <—

Q   MATRIX

    1.000

R   MATRIX

   4.7600E-06
   1.2200E-05
   3.2200E-05

KF  MATRIX

   4.4140E-02   1.6200E-03   9.9108E-03
   1.8745E-02  -2.6979E-02  -7.9667E-03
   6.7043E-02  -2.7613E-03   4.5761E-02
  -6.7599E-92   4.6574E-90   3.4539E-91
   2.2655E-91  -1.5608E-89  -1.1575E-90
  -1.736        34.37         1.423
  -1.4593E-02    .2336        6.9205E-03
  -3.4765E-02   2.068        -.1628
MODIFY TRUTH MODEL (Y OR N) >N <—

POLES OF FILTER  MATRIX

      -2.1736993E+01   +J(  0.              )
      -1.5709760E+01   +J(  0.              )
      -2.3546296E+00   +J(  1.2811239E+00)
      -2.3546296E+00   +J( -1.2811239E+00)
      -5.6279187E-05   +J(  0.              )
      -2.8443847E-01   +J(  0.              )
      -2.0000001E+01   +J(  0.              )
      -2.0000001E+01   +J(  0.              )
+---------- ENTER TITLE IN GIVEN FIELD ----------+
AFTI(S3,A2,G3) KALMAN FILTER <—
```

```
FINAL RMS ERRORS : TRUE =    4.5419573E-04
 (STATE  1)     COMPUTED =    4.7274126E-04

FINAL RMS ERRORS : TRUE =    4.9942017E-04
 (STATE  2)     COMPUTED =    5.3129189E-04

FINAL RMS ERRORS : TRUE =    1.2360643E-03
 (STATE  3)     COMPUTED =    1.2525561E-03

FINAL RMS ERRORS : TRUE =    2.4935549E-92
 (STATE  4)     COMPUTED =    1.1593964E-83

FINAL RMS ERRORS : TRUE =    8.3567084E-92
 (STATE  5)     COMPUTED =    3.8855083E-83

FINAL RMS ERRORS : TRUE =    2.7082759E-01
 (STATE  6)     COMPUTED =    3.0440785E-01

FINAL RMS ERRORS : TRUE =    1.8653740E-03
 (STATE  7)     COMPUTED =    2.0952099E-03

FINAL RMS ERRORS : TRUE =    2.0671737E-02
 (STATE  8)     COMPUTED =    2.2234682E-02
```

```
FILTER DESIGN (Y OR N) >N <-

END DESIGN RUNS (Y OR N) >Y <-
     REG/PI GAINS WRITTEN TO 'SAVE' FILE

PROGRAM EXECUTION STOP
     STOP
     064700 MAXIMUM EXECUTION FL.
     28.723 CP SECONDS EXECUTION TIME.
COMMAND- FILES <-
--LOCAL FILES--
  PLOT       LIST       SAVE      *CGTPIF   *DATA
 $INPUT     $OUTPUT
COMMAND- CATALOG,SAVE,DATAPP <-
 NEWCYCLE CATALOG
 RP = 008 DAYS
 CT ID=  D790477 PFN=DATAPP
 CT CY= 002 SN=AFFDL    00000768 WORDS.:
COMMAND- RETURN,DATA <-
COMMAND- COPYBF,SAVE,DATA <-
COMMAND- ROUTE,LIST,DC=PR,TID=91,ST=CSB,FID=QLL <-

COMMAND- RETURN,SAVE <-
COMMAND- LOGOUT <-
CPA      28.999 SEC.        23.631 ADJ.
IO      118.959 SEC.        35.211 ADJ.
CRUS                        75.329
CONNECT TIME     0 HRS.    24 MIN.
 11/19/81  LOGGED OUT AT 10.13.32.
     <
```

C.2 CGTPIF Output to LIST File

Output written to the LIST file for this execution is identified according to the corresponding paragraph description of Section C.1.2 above. Paragraphs 1 and 11 are not discussed here since they do not involve program execution and therefore do not affect the LIST file.

(2) The first output is a heading with date and time identical to that printed at the terminal. Next the sample period of the controller is identified. A series of outputs related to the design model then follow; these are identified by a heading "DESIGN MODEL". First the matrices defining the continuous-time representation are printed. For this case the matrices are $\underline{A}$, $\underline{B}$, $\underline{G}$, $\underline{Q}$, $\underline{C}$, $\underline{D}_y$, $\underline{H}$, and $\underline{R}$. As for the terminal output, the eigenvalues of $\underline{A}$ are then printed. Finally, the matrices of the discrete-time representation are printed: $\underline{\Phi}$, $\underline{B}_d$, $\underline{Q}_{a_d}$, and $\underline{H}_a$. An additional output is the matrix $\underline{\Pi}$ under a heading of "CONTROLLER SETUP".

(3) Output relating to the design of the PI regulator is identified by a heading of "REG/PI DESIGN". The quadratic weighting matrices $\underline{Y}$, $\underline{U}_m$, $\underline{X}$, and $\underline{U}_R$ are printed, followed by the regulator gain solution $\underline{G}_c^*$. Finally, the PI gains $\underline{K}_x$ and $\underline{K}_z$ are printed.

(4) The truth model description is identified by the heading "TRUTH MODEL" and in this case lists the matrices of the continuous-time system first:

$\underline{A}_t$, $\underline{B}_t$, $\underline{G}_t$, $\underline{Q}_t$, $\underline{H}_t$, $\underline{R}_t$, $\underline{T}_{DT}$. The eigenvalues of the matrix $\underline{A}_t$ are then printed. The matrices of the discrete-time representation are listed: $\underline{\Phi}_t$, $\underline{B}_{t_d}$, $\underline{Q}_{t_d}$.

(5) Outputs due to the controller evaluation routines are identified by a heading of "CONTROLLER EVALUATION" and begin with the mapped eigenvalues of the closed-loop system with PI regulator, $\underline{\Phi}_{\delta CL}$. Time-responses are output in three plots of 5, 5, and 3 variables each. Each plot is labeled with the title specified by the user; the plots include 101 time samples extending from 0. to 2. seconds at the controller sample period of 0.02 seconds; the plot width is 100 character positions in width. The first plot is of states 1 through 5 ($\underline{x}_t(1)$ to $\underline{x}_t(5)$) of the truth model; the second plot is of states 6 through 9 ($\underline{x}_t(6)$ to $\underline{x}_t(9)$) and output 1 ($\underline{y}(1)$) of the truth model; the final plot is of output 2 ($\underline{y}(2)$) and of inputs 1 and 2 ($\underline{u}_t(1)$ and $\underline{u}_t(2)$) of the truth model.

(6) The second execution of the PI regulator design provides the same outputs as described in Paragraph (3) above.

(7) The controller evaluation of the PI regulator design provides the same outputs as described in Paragraph (5) above.

(8) The CGT/PI design path begins with definition of the command model, with relevant output identified by the heading "COMMAND MODEL". The matrices $\underline{A}_m$, $\underline{B}_m$, $\underline{C}_m$, and $\underline{D}_m$ of the continuous-time system are printed, followed

by the eigenvalues of the matrix $\underline{A}_m$. The discrete-time matrices $\underline{\Phi}_m$, $\underline{B}_{m_d}$, $\underline{C}_m$, and $\underline{D}_m$ are then printed. Output due to the CGT design computations is identified by the heading "CGT DESIGN". The matrices $\underline{A}_{11}$, $\underline{A}_{21}$, $\underline{A}_{12}$, and $\underline{A}_{22}$ are printed. Finally, the CGT/PI control gain matrices $\underline{K}_{x_m}$ and $\underline{K}_{x_u}$ are printed.

(9) The evaluation of the CGT/PI controller is identified by the header "CONTROLLER EVALUATION". Three plots are printed with 5, 5, and 5 variables. Characteristics of these plots are the same as described in Paragraph (5) above. The first two plots include the same truth model states and outputs as before. The third plot includes output 2 ($\underline{y}(2)$), and inputs 1 and 2 ($\underline{u}_t(1)$ and $\underline{u}_t(2)$) of the truth model, and outputs 1 and 2 ($\underline{y}_m(1)$ and $\underline{y}_m(2)$) of the command model.

(10) Output due to the Kalman filter design routines is identified by the heading "FILTER DESIGN", and includes the noise strength matrices $\underline{Q}$ and $\underline{R}$ and the Kalman filter gain matrix $\overline{\underline{K}}_F$. The output of the filter evaluation routines is identified by the heading "FILTER EVALUATION". First, the mapped poles of the filter-system matrix $\underline{\Phi}_{KF}$ are printed. During the covariance analysis the full error covariance matrix is printed at each time sample (in this case, from 0. to 1. second each 0.02 seconds). Finally, 8 plots are printed: each plot includes

the standard deviations of the "true" and filter-computed estimation error for each design model state for 50 consecutive time samples taken at the controller/filter sample period.

Appendix D

CGTPIF Program Listing

The following program listing includes all routines of CGTPIF as discussed in the "Programmer's Manual". Routines of the 'LIBRARY' object file are not listed (Ref 24).

CGTPIF is composed of three parts: a 'MAIN' routine, an optional set of user-provided routines, and a large set of invariant routines referred to as 'CGTPIF SUBS'. In this listing, routines 'DSND', 'DSNM', 'TRTHD', and 'TRTHM' are optional routines that are of standard type (see Section A.10 of Appendix A); routines 'ACDATA', 'GUSTS', and 'TBLUP1' are optional routines that are auxiliary to the standard optional routines. These optional routines are used in establishing the design model AFTI(S3,A2,G3) for the pitch-pointing controller, and the truth model AFTI(S4,A2,G3), both as described in Chapter VI of this report. Routines 'CGTXQ' through 'VARSCL' constitute the set of routines CGTPIF SUBS.

```
      PROGRAM MAIN(INPUT=64,OUTPUT=64,LIST=64,
     1 SAVE=64,DATA=64,PLOT=64,
     1 TAPE5=INPUT,TAPE6=OUTPUT,TAPE25=SAVE,TAPE50=DATA,
     2 TAPE99=PLOT,TAPE16=LIST)
      COMMON/MAIN1/NDIM,NDIM1,COM1(400)
      COMMON/MAIN2/COM2(400)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(2125)
      COMMON/ZMTX1/NVZM,ZM1(1225)
      COMMON/ZMTX2/ZM2(1225)
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1750)
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(225)
      COMMON/TRUMTX/NVTM,TM(1725)
      COMMON/CONTROL/NVCTL,CTL(900)
      COMMON/CREGPI/NVRPI,RPI(575)
      COMMON/CCGT/NVCGT,CGT(400)
      COMMON/CKF/NVFLT,FLT(690)
      NDIM=400
      NVSM=2125
      NVZM=1225
      NVDM=1750
      NVCM=225
      NVTM=1725
      NVCTL=900
      NVRPI=575
      NVCGT=400
      NVFLT=690
      KIN=5
      KSAVE=25
      KDATA=50
      KPLOT=99
      KLIST=16
      KTERM=6
      CALL CGTXQ
      STOP
C END MAIN
      END
      SUBROUTINE DSND(ND)
      DIMENSION ND(1)
      ND(1)=8
      ND(2)=2
      ND(3)=2
      ND(4)=3
      ND(5)=0
      ND(6)=1
      ND(7)=0
      RETURN
C END SUBROUTINE DSND
      END




      SUBROUTINE DSNM(A,B,EX,G,Q,C,DY,EY,H,HN,R,AN,GN,QN)
      DIMENSION A(8,8),B(8,2),C(2,8),G(8),DY(2,2),H(3,8),R(3,3)
      DATA GRAVTY,DEGTRD,PI/32.174,.01745329,3.1415927/
      CALL ACDATA(LEVEL,VT,ALT,ALPHA,ZA,ZAD,ZQ,ZU,ZDE,ZDF,
     1 PMA,PMAD,PMQ,PMU,PMDE,PMDF,XA,XAD,XQ,XU,XDE,XDF,
     2 TE,DLX,BSPAN)
   10 ALPHAR=DEGTRD*ALPHA
      U0=VT*COS(ALPHAR)
      W0=VT*SIN(ALPHAR)
      A(1,3)=1.
      A(2,1)=-GRAVTY*SIN(ALPHAR)/U0
      A(2,2)=ZA
      A(2,3)=1.+ZQ
      A(3,2)=PMA
      A(3,3)=PMQ
      A(2,7)=ZA
      A(2,8)=ZQ
      A(3,7)=PMA
      A(3,8)=PMQ
      A(2,4)=ZDE
      A(2,5)=ZDF
      A(3,4)=PMDE
      A(3,5)=PMDF
      A(4,4)=-TE
      A(5,5)=-TE
      B(4,1)=TE
      B(5,2)=TE
      CALL GUSTS(LEVEL,ALT,SLU,SLW,SIGU,SIGW)
      A(6,6)=-VT/SLW
      A(7,6)=(1.-SQRT(3.))*SIGW*SQRT(-A(6,6))/SLW
      A(7,7)=A(6,6)
      A(8,8)=-VT*PI/4./BSPAN
      A(8,6)=-A(8,8)*A(7,6)
      A(8,7)=-A(8,8)*A(7,7)
      G(6)=1.
      G(7)=SIGW*SQRT(3.*VT/SLW)/VT
      G(8)=-A(8,8)*G(7)
      Q=1.
      C(1,1)=1.
      C(2,1)=1.
      C(2,2)=-1.
      H(1,1)=1.
      H(2,2)=1.
      H(3,3)=1.
      H(2,7)=1.
      R(1,1)=4.76E-6
      R(2,2)=1.22E-5
      R(3,3)=3.22E-5
      RETURN
C END SUBROUTINE DSNM
      END



      SUBROUTINE TRTHD(ND)
      DIMENSION ND(1)
      ND(1)=9
      ND(2)=2
      ND(3)=3
      ND(4)=1
      RETURN
C END SUBROUTINE TRTHD
      END



      SUBROUTINE TRTHM(AT,BT,GT,QT,HT,RT,TDT,TNT)
      DIMENSION AT(9,9),BT(9,2),GT(9),HT(3,9),RT(3,3),TDT(8,9)
      DATA GRAVTY,DEGTRD,PI/32.174,.01745329,3.1415927/
      CALL ACDATA(LEVEL,VT,ALT,ALPHA,ZA,ZAD,ZQ,ZU,ZDE,ZDF,
     1 PMA,PMAD,PMQ,PMU,PMDE,PMDF,XA,XAD,XQ,XU,XDE,XDF,
     2 TE,DLX,BSPAN)
 10   ALPHAR=DEGTRD*ALPHA
      U0=VT*COS(ALPHAR)
      W0=VT*SIN(ALPHAR)
      RZAD=1./(1.-ZAD)
      AT(1,3)=1.
      AT(2,1)=-GRAVTY*SIN(ALPHAR)/U0
      AT(2,2)=ZA
      AT(2,3)=1.+ZQ
      AT(2,4)=ZU
      AT(3,2)=PMA
      AT(3,3)=PMQ
      AT(3,4)=PMU
      AT(4,1)=-GRAVTY*COS(ALPHAR)
      AT(4,2)=XA
      AT(4,3)=XQ-WC
      AT(4,4)=XU
      AT(2,5)=ZDE
      AT(2,6)=ZDF
      AT(3,5)=PMDE
      AT(3,6)=PMDF
      AT(4,5)=XDE
      AT(4,6)=XDF
      AT(5,5)=-TE
      AT(6,6)=-TE
      AT(2,8)=ZA
      AT(2,9)=ZQ
      AT(3,8)=PMA
      AT(3,9)=PMQ
      AT(4,8)=XA
      AT(4,9)=XQ
      CALL GUSTS(LEVEL,ALT,SLU,SLW,SIGU,SIGW)
      AT(7,7)=-VT/SLW
      AT(8,7)=(1.-SQRT(3.))*SIGW*SQRT(-AT(7,7))/SLW
      AT(8,8)=AT(7,7)
      AT(9,9)=-VT*PI/4./BSPAN
      AT(9,7)=-AT(9,9)*AT(8,7)
      AT(9,8)=-AT(9,9)*AT(8,8)
      GT(7)=1.
      GT(8)=SIGW*SQRT(3.*VT/SLW)/VT
      GT(9)=-AT(9,9)*GT(8)
      QT=1.
      DO 20 I=1,9
      AT(2,I)=AT(2,I)*RZAD
      AT(3,I)=AT(3,I)+PMAD*AT(2,I)
   20 AT(4,I)=AT(4,I)+XAD*AT(2,I)
      BT(5,1)=TE
      BT(6,2)=TE
      HT(1,1)=1.
      HT(2,2)=1.
      HT(3,3)=1.
      HT(2,8)=1.
      RT(1,1)=4.76E-6
      RT(2,2)=1.22E-5
      RT(3,3)=3.22E-5
      TDT(1,1)=1.
      TDT(2,2)=1.
      TDT(3,3)=1.
      TDT(4,5)=1.
      TDT(5,6)=1.
      TDT(6,7)=1.
      TOT(7,8)=1.
      TOT(8,9)=1.
      RETURN
C END SUBROUTINE TRTHM
      END




      SUBROUTINE ACDATA(LEVEL,VT,ALT,ALPHA,ZA,ZAD,ZQ,ZU,ZDE,ZDF,
     1 PMA,PMAD,PMQ,PMU,PMDE,PMDF,XA,XAD,XQ,XU,XDE,XDF,
     2 TE,DLX,BSPAN)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DATA NENTRY/1/
 5    WRITE 101
      READ*,LEVEL
      IF((LEVEL.GT.3).OR.(LEVEL.LT.1)) GO TO 5
      WRITE 102
      READ*,VT,ALT,ALPHA
      WRITE 103
      READ*,ZA,ZAD,ZQ,ZU,ZDE,ZDF
      WRITE 104
      READ*,PMA,PMAD,PMQ,PMU,PMDE,PMDF
      WRITE 105
      READ*,XA,XAD,XQ,XU,XDE,XDF
      WRITE(KLIST,101)
      WRITE(KLIST,109) LEVEL
      WRITE(KLIST,102)
      WRITE(KLIST,110) VT,ALT,ALPHA
      WRITE(KLIST,103)
      WRITE(KLIST,110) ZA,ZAD,ZQ,ZU,ZDE,ZDF
      WRITE(KLIST,104)
      WRITE(KLIST,110) PMA,PMAD,PMQ,PMU,PMDE,PMDF
      WRITE(KLIST,105)
      WRITE(KLIST,110) XA,XAD,XQ,XU,XDE,XDF
      IF(NENTRY.EQ.0) GO TO 10
      BSPAN=30.
      DLX=13.798
      TE=20.
      RETURN
 10   WRITE 106
      READ*,TE
      WRITE 107
      READ*,DLX
      WRITE 108
      READ*,BSPAN
 101  FORMAT(" ENTER TURBULENCE LEVEL (1,2,3) >")
 102  FORMAT(" ENTER TRIM VELOCITY, ALTITUDE, AND ALPHA >")
 103  FORMAT(" ENTER ZA, ZAD, ZQ, ZU, ZDE, ZDF >")
 104  FORMAT(" ENTER MA, MAD, MQ, MU, MDE, MDF >")
 105  FORMAT(" ENTER XA, XAD, XQ, XU, XDE, XDF >")
 106  FORMAT(" ENTER TIME CONSTANT FOR ELEVATOR >")
 107  FORMAT(" ENTER DISTANCE FROM CG TO ACCELEROMETER >")
 108  FORMAT(" ENTER WING SPAN >")
 109  FORMAT(6X,I1)
 110  FORMAT(6(6X1PE15.7))
      RETURN
C END SUBROUTINE ACDATA
      END




      SUBROUTINE GUSTS(LEVEL,ALT,SLU,SLW,SIGU,SIGW)
      DIMENSION ATRB1(4),ATRB2(4),ATRB3(4),SIGT1(4),SIGT2(4),SIGT3(4)
      DATA ATRB1/2000.,2750.,10000.,30000./
      DATA ATRB2/2000.,2750.,10000.,45000./
      DATA ATRB3/2000.,5000.,20000.,70000./
      DATA SIGT1/4.5,5.,5.,0./
      DATA SIGT2/6.5,10.,10.,0./
      DATA SIGT3/12.,21.,21.,0./
      DATA IT1,IT2,IT3/1,1,1/
      IF(ALT-1750.) 5,15,15
 5    IF(ALT-1000.) 8,10,10
 8    ALTT=ALT
      GO TO 12
 10   ALTT=1000.
 12   SIGW=2.5*FLOAT(LEVEL)
      SIGU=1./(.177+8.23E-4*ALTT)**.4
      SLW=ALTT
      SLU=ALTT*SIGU**3
      SIGU=SIGU*SIGW
      GO TO 100
 15   SLU=1750.
      SLW=1750.
      IF(LEVEL-2) 17,18,16
 16   CALL TBLUP1(ATRB3,SIGT3,4,IT3,ALT,SIGU)
      GO TO 19
 17   CALL TBLUP1(ATRB1,SIGT1,4,IT1,ALT,SIGU)
      GO TO 19
 18   CALL TBLUP1(ATRB2,SIGT2,4,IT2,ALT,SIGU)
 19   SIGW=SIGU
 100  RETURN
C END SUBROUTINE GUSTS
      END
      SUBROUTINE TBLUP1(X,Y,N,IXP,XP,YP)
      DIMENSION X(1),Y(1)
      IF(IXP) 15,15,1
 1    IF(IXP-N) 10,10,5
 5    IXP=N
      GO TO 18
 10   IF(XP-X(IXP)) 12,18,20
 12   IXP=IXP-1
      IF(IXP) 15,15,10
 15   IXP=1
 18   YP=Y(IXP)
      RETURN
 20   IF(IXP-N) 21,18,5
 21   IXPP1=IXP+1
 22   IF(XP-X(IXPP1)) 25,30,30
 25   YP=Y(IXP)+(XP-X(IXP))/(X(IXPP1)-X(IXP))*(Y(IXPP1)-Y(IXP))
      RETURN
 30   IXP=IXPP1
      GO TO 20
C END SUBROUTINE TBLUP1
      END
      SUBROUTINE CGTXQ
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/MAIN2/COM2(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/NDIMT/NNT,NPT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LINT
      COMMON/TRUMTX/NVTM,TM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHCL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      COMMON/LCGT/LA11,LA13,LA21,LA23,LA12,LA22,LKXA11,LKXA12,LKXA13
      COMMON/CCGT/NVCGT,CGT(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CKF/NVFLT,FLT(1)
      DIMENSION LD(15),ND(10)
      DATA NPLTZM/605/
      DATA IEOI,NO/-1,1HN/
      REWIND KLIST
      WRITE(KLIST,115) DATE(DUM),TIME(DUM)
      WRITE(KTERM,115) DATE(DUM),TIME(DUM)
  115 FORMAT("1",27X,"* * * CGTPIF * * *"/14X,
     1 "PROGRAM TO DESIGN A COMMAND GENERATOR TRACKER"/8X,
     2 "USING A REGULATOR WITH PROPORTIONAL PLUS INTEGRAL CONTROL"/16X,
     3 "AND A KALMAN FILTER FOR STATE ESTIMATION."/25X,
     4 "* * * CGTPIF * * *"//11X,"DATE : ",A10//,11X,
     5 "TIME : ",A10////)
      REWIND KSAVE
      REWIND KDATA
      WRITE(KSAVE,112) IEOI,NPLTZM
      DO 10 I=1,10
   10 ND(I)=0
      DO 12 I=1,15
   12 LD(I)=1
      LFLRPI=0
      LFLCGT=0
      LFLKF=0
      LTEVAL=0
      LABORT=0
      IPI=0
      ICGT=0
      ITRU=0
      IFLTR=0
      NVCOM=MIN0(NDIM,NVZM)
      KOUT=KLIST
      KPUNCH=KPLOT
      IF(NVSM.GE.NPLTZM) GO TO 50
      WRITE 101,NPLTZM
      GO TO 1000
50    WRITE 102
      READ*,TSAMP
      IF(TSAMP.LE.0.) GO TO 50
      WRITE(KLIST,103) TSAMP
103   FORMAT("0 SAMPLE PERIOD IS ",F5.3," SECONDS")
      CALL SETUP(ND,LD,ICGT,ITRU,1)
      IF(LABORT) 1000,100,1000
100   LABORT=0
      WRITE 104
104   FORMAT("0CONTROLLER DESIGN (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 500
      LFLKF=0
      CALL PIMTX(IPI)
      IF(LABORT) 1000,125,1000
125   WRITE 105
105   FORMAT("0DESIGN REG/PI (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 150
      CALL SREGPI
      IF(LABORT) 1000,200,1000
150   WRITE 106
106   FORMAT("0DESIGN CGT (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 100
      CALL SETUP(ND,LD,ICGT,ITRU,2)
      IF(ICGT) 155,100,155
155   IF(LABORT) 100,160,1000
160   CALL SCGT
      IF(LABORT) 100,170,1000
170   IF(LFLCGT.LE.0) GO TO 125
200   LABORT=0
      WRITE 107
107   FORMAT("0CONTROLLER EVALUATION WRT TRUTH MODEL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 250
      CALL SETUP(ND,LD,ICGT,ITRU,3)
      IF(LABORT) 200,260,1000
250   LTEVAL=0
260   CALL CEVAL
      GO TO 100
500   LABORT=0
```

```
      WRITE 108
  108 FORMAT("0FILTER DESIGN (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 900
      CALL FLTRK(IFLTR)
      IF(IFLTR.EQ.0) GO TO 900
      IF(LABORT) 1000,510,1000
  510 CALL SETUP(ND,LD,ICGT,ITRU,3)
      IF(LABORT) 500,525,1000
  525 CALL FEVAL
      IF(LABORT) 1000,500,1000
  900 WRITE 109
  109 FORMAT("0END DESIGN RUNS (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 100
      IF(LFLRPI.EQ.0) GO TO 1000
      NPNTS=NRD*NNPR
      ND(1)=NPNTS
      ND(2)=LKX
      ND(3)=LKZ
      CALL WFILFD(4,NPNTS,ND,RPI(LKX))
      WRITE 113
 1000 CONTINUE
      WRITE(KLIST,110)
      REWIND KSAVE
      REWIND KDATA
      REWIND KLIST
      WRITE 110
  101 FORMAT("0INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
  102 FORMAT("0ENTER SAMPLE PERIOD FOR DIGITAL CONTROLLER >")
  110 FORMAT("0PROGRAM EXECUTION STOP")
  111 FORMAT(A3)
  112 FORMAT(2I4)
  113 FORMAT(6X,"REG/PI GAINS WRITTEN TO 'SAVE' FILE")
      RETURN
C END SUBROUTINE CGTXQ
      END
      SUBROUTINE SETUP(ND,LD,ICGT,ITRU,ITYPE)
      DIMENSION ND(1),LD(1)
      IF(ITYPE-2) 10,15,20
 10   CALL SOSN(ND,LD)
      RETURN
 15   CALL SCMD(ND,LD,ICGT)
      RETURN
 20   CALL STRTH(ND,LD,ITRU)
      RETURN
C END SUBROUTINE SETUP
      END




      SUBROUTINE SDSN(ND,LD)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWCD,NPLD,NWPNWD,NNPR
      DIMENSION ND(1),LD(1)
      NSIZE=0
      CALL RSYS(SM,LD,ND,1,NSIZE)
      IF(LABORT.GT.0) RETURN
      NSIZE=NNPR
      IF(NPLD.GT.NSIZE) NSIZE=NPLD
      NSIZE=NSIZE*NSIZE
      IF(NSIZE.LE.NVCOM) GO TO 5
      WRITE 101,NSIZE
 101  FORMAT("0INSUFFICIENT MEMORY /MAIN1/,/MAIN2/,/ZMTX1/,/ZMTX2/, NEED
     1: ",I4)
      LABORT=NSIZE
      RETURN
 5    IF(NRD.EQ.NPD) GO TO 10
      WRITE 102
 102  FORMAT("0NUMBER OF INPUTS AND OUTPUTS MUST BE EQUAL FOR DESIGN")
      LABORT=-1
      RETURN
 10   CALL DSCRTD(LD,ZM1,ZM2)
      RETURN
C END SUBROUTINE SDSN
      END
      SUBROUTINE DSCRTD(LD,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCCV
      COMMON/CKF/NVFLT,FLT(1)
      DIMENSION LD(1),ZM1(1),ZM2(1)
      NDIM=NPLD
      NDIM1=NDIM+1
      CALL POLES(SM,NND,1,ZM1,ZM2)
      DO 1 I=1,NND
1     IF(ZM1(I).GT.0.) LFLCGT=-1
      CALL TFRMTX(SM,DM,NND,NND,2)
      LAP=1
      LGP=LAP+NPLD*NPLD
      IF(NWD.EQ.0) GO TO 5
      CALL TFRMTX(SM(LD(4)),DM(LGP),NND,NWD,2)
5     IF(NDD.EQ.0) GO TO 10
      L1=LADDR(NPLD,NND+1,1)
      L2=LADDR(NPLD,1,NND+1)
      L3=LADDR(NPLD,NND+1,NND+1)
      CALL ZPART(DM(L1),NDD,NND,NPLD)
      CALL TFRMTX(SM(LD(3)),DM(L2),NND,NDD,2)
      CALL TFRMTX(SM(LD(12)),DM(L3),NDD,NDD,2)
      IF(NWD.EQ.0) GO TO 8
      L1=L1+LGP-1
      CALL ZPART(DM(L1),NDD,NWD,NPLD)
8     L2=LADDR(NPLD,1,NWD+1)+LGP-1
      L3=LADDR(NPLD,NND+1,NWD+1)+LGP-1
      CALL ZPART(DM(L2),NND,NWDD,NPLD)
      CALL TFRMTX(SM(LD(13)),DM(L3),NDD,NWDD,2)
10    LPHI=LGP+NPLD*NWPNWD
      LEADSN=1
      CALL NDSCFT(DM,NDIM,NT)
      CALL DSCRT(NPLD,DM,TSAMP,FLT,ZM1,NT)
      LFLTRK=LEADSN+NPLD*NPLD
      CALL TFRMTX(DM(LPHI),FLT,NND,NND,1)
      LBD=LPHI+NND*NND
      CALL TFRMTX(SM,ZM1,NND,NND,1)
      CALL FMMUL(SM,SM(LD(2)),NND,NND,NRD,DM(LBD))
      LEX=LBD+NND*NRD
      IF(NDD.EQ.0) GO TO 15
      L1=LADDR(NPLD,1,NND+1)
      CALL TFRMTX(DM(LEX),FLT(L1),NND,NDD,1)
      LPHD=LEX+NND*NDD
      L1=LADDR(NPLD,NND+1,NND+1)
      CALL TFRMTX(DM(LPHD),FLT(L1),NDD,NDD,1)
      LQ=LPHD+NDD*NDD
      GO TO 20
15    LQ=LEX
20    IF(NWD.EQ.0) GO TO 25
      CALL FTMTX(SM(LD(5)),DM(LQ),NWD,NWD)
      LQN=LQ+NWD*NWD
      GO TO 28
25    LQN=LQ
28    IF(NWDD.EQ.0) GO TO 33
      CALL FTMTX(SM(LD(14)),DM(LQN),NWDD,NWDD)
      LQD=LQN+NWDD*NWDD
      GO TO 35
33    LQD=LQN
      IF(NWPNWD.GT.0) GO TO 35
      LC=LQD
      GO TO 36
35    CALL QDSCRT(DM(LQ),DM(LQN),ZM1,ZM2)
      LC=LQD+NPLD*NPLD
36    LDY=LC+NPD*NND
      LEY=LDY+NPD*NRD
      LHP=LEY+NPD*NDD
      LR=LHP+NMD*NPLD
      L1=LR+NMD*NMD-LC
      CALL FTMTX(SM(LD(6)),DM(LC),L1,1)
      L1=LEY-1
      NODY=1
      DO 40 I=LDY,L1
      IF(DM(I).EQ.0.) GO TO 40
      NODY=0
      GO TO 45
40    CONTINUE
45    NOEY=1
      IF(NDD.LT.1) GO TO 55
      L1=LHP-1
      DO 50 I=LEY,L1
      IF(DM(I).EQ.0.) GO TO 50
      NOEY=0
      GO TO 55
50    CONTINUE
55    CALL MATLST(DM(LPHI),NND,NND,"PHI",KLIST)
      CALL MATLST(DM(LBD),NND,NRD,"BD",KLIST)
      IF(NWPNWD.GT.0) CALL MATLST(DM(LQD),NPLD,NPLD,"QD",KLIST)
      IF(NMD.GT.0) CALL MATLST(DM(LHP),NMD,NPLD,"HA",KLIST)
      IF(NDD.EQ.0) RETURN
      CALL MATLST(DM(LEX),NND,NDD,"EXD",KLIST)
      CALL MATLST(DM(LPHD),NDD,NDD,"PHN",KLIST)
      RETURN
C END SUBROUTINE DSCRTD
      END
      SUBROUTINE QDSCRT(Q,QN,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSMMTX/NVDM,NODY,NOEY,DM(1)
      DIMENSION Q(1),QN(1),ZM1(1),ZM2(1)
      IF(NWD.EQ.0) GO TO 5
      CALL TFRMTX(Q,ZM1,NWD,NWD,2)
 5    IF(NWDD.EQ.0) GO TO 10
      L1=LADDR(NPLD,NWD+1,NWD+1)
      CALL TFRMTX(QN,ZM1(L1),NWDD,NWDD,2)
      IF(NWD.EQ.0) GO TO 10
      L1=LADDR(NPLD,1,NWD+1)
      CALL ZPART(ZM1(L1),NWD,NWDD,NPLD)
      L1=LADDR(NPLD,NWD+1,1)
      CALL ZPART(ZM1(L1),NWDD,NWD,NPLD)
 10   CALL MAT3(NPLD,NWPNWD,DM(LGP),ZM1,ZM2)
      CALL INTEG(NPLD,DM(LAP),ZM2,DM(LQD),TSAMP)
      RETURN
C END SUBROUTINE QDSCRT
      END




      SUBROUTINE SCMD(ND,LD,ICGT)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      DIMENSION ND(1),LD(1)
      DATA NO/1HN/
      WRITE(KLIST,110)
 110  FORMAT(////11X,5("* "),"CGT DESIGN",5(" *")////)
      NEWCM=0
      IF(LFLRPI) 10,5,10
 5    WRITE 102
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 8
      CALL READFS(SM,ND,4,IERR)
      NSIZE=ND(1)
      LKX=ND(2)
      LKZ=ND(3)
      CALL FTMTX(SM,RPI(LKX),NSIZE,1)
      IF(IERR.NE.0) RETURN
      CALL MATLST(RPI(LKX),NRD,NND,"KX",KLIST)
      CALL MATLST(RPI(LKZ),NRD,NRD,"KZ",KLIST)
      LFLRPI=-1
      GO TO 10
 8    IF(LFLCGT.GE.0) GO TO 9
      WRITE 103
 103  FORMAT("0SYSTEM UNSTABLE - - OPEN-LOOP CGT NOT FEASIBLE")
      RETURN
 9    LKX=1
      LKZ=1
      NSIZE=NPD*NND
      CALL ZPART(RPI(LKX),1,NSIZE,1)
 10   IF(ICGT.EQ.0) GO TO 12
      WRITE 106
 106  FORMAT(" MODIFY COMMAND MODEL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) RETURN
 12   CALL RSYS(SM,LD,ND,2,ICGT)
      IF(LABORT.NE.0) RETURN
      NEWCM=1
      CALL POLES(SM,NNC,2,ZM1,ZM2)
      IF(NPC.EQ.NPD) GO TO 15
      WRITE 104
      LABORT=-1
      RETURN
 15   CALL DSCFTC(LD,ZM1)
 102  FORMAT(" READ REG/PI GAINS FROM 'DATA' FILE (Y OR N) >")
 104  FORMAT("0COMMAND AND DESIGN MODEL OUTPUTS NOT EQUAL IN NUMBER")
 111  FORMAT(A3)
      RETURN
C END SUBROUTINE SCMD
      END
      SUBROUTINE DSCRTC(LD,ZM1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      DIMENSION LD(1),ZM1(1)
      NDIM=NNC
      NDIM1=NDIM+1
      CALL NDSCRT(SM,NDIM,NT)
      CALL DSCRT(NDIM,SM,TSAMP,CM,ZM1,NT)
      LPHC=1
      LBDC=LPHC+NNC*NNC
      CALL MMUL(ZM1,SM(LD(2)),NDIM,NDIM,NRC,CM(LBDC))
      LCC=LBDC+NNC*NRC
      LDC=LCC+NPC*NNC
      L1=LDC+NPC*NRC-LCC
      CALL FTMTX(SM(LD(3)),CM(LCC),L1,1)
      NODC=1
      L1=L1+LCC-1
      DO 10 I=LDC,L1
      IF(CM(I).EQ.0.) GO TO 10
      NODC=0
      GO TO 15
 10   CONTINUE
 15   CALL MATLST(CM,NNC,NNC,"PHM",KLIST)
      CALL MATLST(CM(LBDC),NNC,NRC,"BDM",KLIST)
      CALL MATLST(CM(LCC),NPC,NNC,"CM",KLIST)
      CALL MATLST(CM(LDC),NPC,NRC,"DM",KLIST)
      RETURN
C END SUBROUTINE DSCRTC
      END
      SUBROUTINE STRTH(ND,LD,ITRU)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWCD,NPLD,NWPNWD,NWPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      DIMENSION ND(1),LD(1)
      DATA NO/1HN/
      IF(ITRU.EQ.0) GO TO 5
      WRITE 103
  103 FORMAT(" MODIFY TRUTH MODEL (Y OR N) >")
      READ 111,IANS
  111 FORMAT(A3)
      IF(IANS.EQ.NO) GO TO 20
    5 CALL RSYS(SM,LD,ND,3,ITRU)
      IF(LABORT.GT.0) RETURN
      NSIZE=NNT*NNT
      IF(NSIZE.LE.NVCOM) GO TO 8
      WRITE 101,NSIZE
  101 FORMAT("0INSUFFICIENT MEMORY /MAIN1/,/MAIN2/,/ZMTX1/,/ZMTX2/, NEED
     1: ",I2)
      LABORT=NSIZE
      RETURN
    8 IF((NRT.EQ.NRD).AND.(NMT.EQ.NMD)) GO TO 10
      WRITE 102
  102 FORMAT("0INPUTS AND MEASUREMENTS MUST BE EQUAL IN NUMBER FOR DESIG
     1N AND TRUTH MODELS")
      LABORT=-1
      RETURN
   10 CALL POLES(SM,NNT,3,ZM1,ZM2)
      CALL DSCRTT(LD,ZM1)
   20 LTEVAL=1
      RETURN
C END SUBROUTINE STRTH
      END
      SUBROUTINE DSCRTT(LD,ZM1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMC/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TFUMTX/NVTM,TM(1)
      DIMENSION LD(1),ZM1(1)
      NDIM=NNT
      NDIM1=NDIM+1
      CALL NDSCRT(SM,NDIM,NT)
      CALL DSCRT(NDIM,SM,TSAMP,TM,ZM1,NT)
      LPHT=1
      LBDT=LPHT+NNT*NNT
      CALL MMUL(ZM1,SM(LD(2)),NDIM,NDIM,NRT,TM(LBDT))
      LQDT=LBDT+NNT*NRT
      IF(NWT.GT.0) GO TO 10
      LHT=LQDT
      GO TO 15
 10   CALL MAT3(NDIM,NWT,SM(LD(3)),SM(LD(4)),ZM1)
      CALL INTEG(NDIM,SM,ZM1,TM(LQDT),TSAMP)
      LHT=LQDT+NNT*NNT
 15   LRT=LHT+NMT*NNT
      LTDT=LRT+NMT*NMT
      LTNT=LTDT+NND*NNT
      L1=LTNT+NDD*NNT-LHT
      CALL FTMTX(SM(LD(5)),TM(LHT),L1,1)
      CALL MATLST(TM,NNT,NNT,"PHT",KLIST)
      CALL MATLST(TM(LBDT),NNT,NRT,"BDT",KLIST)
      IF(NWT.GT.0) CALL MATLST(TM(LQDT),NNT,NNT,"QDT",KLIST)
      RETURN
C END SUBROUTINE DSCRTT
      END
```

AD-A115 511 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/6 1/3
DESIGN OF ADVANCED DIGITAL FLIGHT CONTROL SYSTEMS VIA COMMAND G--ETC(U)
DEC 81 R M FLOYD
UNCLASSIFIED AFIT/GE/EE-81-20-VOL-2 NL

3 OF 3
AD A115 511

END
DATE
FILMED
7-82
DTIC

```
      SUBROUTINE PIMTX(IPI)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      IF(IPI.EQ.1) RETURN
      WRITE(KLIST,110)
  110 FORMAT(////11X,5("* "),"CONTROLLER SET-UP",5(" *")////)
      NDIM=NNPR
      NSIZE=NDIM*(2*NDIM+NPD)
      IF(NSIZE.LE.NVCTL) GO TO 10
      WRITE 101,NSIZE
  101 FORMAT("0INSUFFICIENT MEMORY /CONTROL/, NEED: ",I4)
      LABORT=NSIZE
      RETURN
   10 NDIM1=NDIM+1
      LPI11=1
      LPI12=LPI11+NND*NND
      LPI21=LPI12+NND*NRD
      LPI22=LPI21+NPD*NND
      LPHDL=LPI22+NPD*NRD
      CALL TFRMTX(DM(LPHI),ZM1,NND,NND,2)
      CALL SUBI(ZM1,NND,NDIM)
      L2=LADDR(NDIM,1,NND+1)
      CALL TFRMTX(DM(LBD),ZM1(L2),NND,NPD,2)
      L3=LADDR(NDIM,NND+1,1)
      CALL TFRMTX(DM(LC),ZM1(L3),NPD,NND,2)
      L4=LADDR(NDIM,NND+1,NND+1)
      CALL TFRMTX(DM(LDY),ZM1(L4),NPD,NRD,2)
      CALL GMINV(NDIM,NDIM,ZM1,ZM2,MR,1)
      IF(MR.EQ.NDIM) GO TO 15
      WRITE 102
      WRITE(KLIST,102)
  102 FORMAT("0PI MATRIX IS RANK DEFECTIVE")
   15 CALL MATLST(ZM2,NNPR,NNPR,"PI",KLIST)
      CALL TFRMTX(CTL(LPI11),ZM2,NND,NND,1)
      CALL TFRMTX(CTL(LPI12),ZM2(L2),NND,NRD,1)
      CALL TFRMTX(CTL(LPI21),ZM2(L3),NPD,NND,1)
      CALL TFRMTX(CTL(LPI22),ZM2(L4),NPD,NRD,1)
      CALL CDIF
      IPI=1
      RETURN
C END SUBROUTINE PIMTX
      END
      SUBROUTINE CDIF
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWCD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      CALL TFRMTX(DM(LPHI),CTL(LPHDL),NND,NND,2)
      L1=LADDR(NDIM,1,NND+1)+LPHDL-1
      CALL TFRMTX(DM(LBD),CTL(L1),NND,NRD,2)
      L1=LADDR(NDIM,NND+1,1)+LPHDL-1
      CALL ZPART(CTL(L1),NRD,NND,NDIM)
      L1=LADDR(NDIM,NND+1,NND+1)+LPHDL-1
      CALL IDNT(NRD,CTL(L1),1.)
      LBDL=LPHDL+NDIM*NDIM
      CALL ZPART(CTL(LBDL),NND,NRD,NDIM)
      L1=LADDR(NDIM,NND+1,1)+LBDL-1
      CALL IDNT(NRD,CTL(L1),1.)
      RETURN
C END SUBROUTINE CDIF
      END
      SUBROUTINE SREGPI
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWCD,NPLD,NWPNWC,NNPR
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHCL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      WRITE(KLIST,110)
110   FORMAT(////11X,5("* "),"REG/PI DESIGN",5(" *")////)
      NSIZE=NRD*(4*NRD+NND)+NNPR*NNPR
      IF(NSIZE.LE.NVRPI) GO TO 5
      WRITE 101,NSIZE
101   FORMAT("0 INSUFFICIENT MEMORY /CREGPI/, NEED: ",I4)
      GO TO 8
5     NSIZE=NNPR*(3*NNPR+NPD)
      IF(NSIZE.LE.NVSM) GO TO 10
      WRITE 102,NSIZE
102   FORMAT("0 INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
8     LABORT=NSIZE
      RETURN
10    LX=1
      LU=LX+NNPR*NNPR
      CALL WXUS(SM(LX),SM(LU),COM1,ZM1,ZM2)
      LUIST=LU+NNPR*NRD
      LPHP=LUIST+NNPR*NNPR
      CALL PXUP(CTL(LPHDL),CTL(LBDL),SM(LX),SM(LU),COM1,ZM2,
     1 SM(LUIST),SM(LPHP),SM(LX),ZM1)
      CALL DRIC(NDIM,SM(LPHP),ZM2,SM(LX),ZM1,RPI(LPHCL))
      CALL GCSTAR(SM(LPHP),CTL(LBDL),SM(LU),ZM1,SM(LUIST),SM(LX),ZM2)
      CALL TFRMTX(ZM1,SM(LX),NRD,NDIM,1)
      CALL FMMUL(ZM1,CTL(LPI11),NRD,NND,NND,RPI(LKX))
      L1=LADDR(NRD,1,NND+1)
      CALL FMMUL(ZM1(L1),CTL(LPI21),NRD,NRD,NND,ZM2)
      NDIM=NRD
      NDIM1=NDIM+1
      CALL MADD1(NRD,NND,RPI(LKX),ZM2,RPI(LKX),1.)
      CALL FMMUL(ZM1,CTL(LPI12),NRD,NND,NRD,RPI(LKZ))
      CALL FMMUL(ZM1(L1),CTL(LPI22),NRD,NRD,NRD,ZM2)
      CALL MADD1(NRD,NRD,RPI(LKZ),ZM2,RPI(LKZ),1.)
      CALL MATLST(RPI(LKX),NRD,NND,"KX",KLIST)
      CALL MATLST(RPI(LKX),NRD,NND,"KX",KTERM)
      CALL MATLST(RPI(LKZ),NRD,NRD,"KZ",KLIST)
      CALL MATLST(RPI(LKZ),NRD,NRD,"KZ",KTERM)
      LFLRPI=1
      LFLCGT=0
      RETURN
```

```
C END SUBROUTINE SREGPI
      END




      SUBROUTINE WXUS(X,U,S,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWCD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LD,LQN,LQD,LC,LDY,LEY,LHP,LK
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHCL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      DIMENSION X(1),U(1),S(1),ZM1(1),ZM2(1)
      DATA NO/1HN/
      IF(LFLRPI) 5,5,10
 5    LXDW=1
      LUDW=LXDW+2*NRD*NRD
      LPHCL=LUDW+NRD*NRD
      LKX=LPHCL+NNPR*NNPR
      LKZ=LKX+NPD*NND
      L1=LPHCL-1
      CALL ZPART(RPI,1,L1,1)
 10   LUX=NRD*NRD+1
      WRITE 101,NPD
 101  FORMAT(" ENTER WEIGHTS ON OUTPUT DEVIATIONS: ",I2)
      CALL ROWGTS(RPI,NPD,0)
      WRITE 102,NRD
 102  FORMAT(" ENTER WEIGHTS ON CONTROL MAGNITUDES: ",I2)
      CALL ROWGTS(RPI(LUX),NRD,1)
      WRITE 103,NRD
 103  FORMAT(" ENTER WEIGHTS ON CONTROL RATES: ",I2)
      CALL ROWGTS(RPI(LUDW),NRD,1)
      CALL MATLST(RPI,NPD,NPD,"Y",KLIST)
      CALL DVCTOR(NPD,RPI,ZM1)
      CALL MATLST(ZM1,NPD,1,"Y",KTERM)
      CALL DVCTOR(NRD,RPI(LUX),ZM1)
      CALL MATLST(ZM1,NRD,1,"UM",KTERM)
      CALL MATLST(RPI(LUX),NRD,NRD,"UM",KLIST)
      NDIM=NNPR
      NDIM1=NDIM+1
      CALL FORMX(RPI,RPI(LUX),DM(LC),DM(LDY),ZM2,ZM1,COM1)
      WRITE(KTERM,104)
 104  FORMAT("( MODIFY ELEMENTS OF 'X' MATRIX (Y OR N) >")
      READ 111,IANS
 111  FORMAT(A3)
      IF(IANS.EQ.NO) GO TO 20
      WRITE(KTERM,105)
 105  FORMAT(" LIST 'X' MATRIX TO TERMINAL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 12
      CALL MATLST(ZM2,NNPR,NNPR,"X",KTERM)
 12   CALL ZMATIN(ZM2,NNPR,NNPR,-1)
 20   CALL MATLST(ZM2,NNPR,NNPR,"X",KLIST)
      CALL MATLST(RPI(LUDW),NRD,NRD,"UR",KLIST)
      CALL DVCTOR(NRD,RPI(LUDW),ZM1)
      CALL MATLST(ZM1,NRD,1,"UR",KTERM)
      T1=.25*TSAMP
      CALL SCALE(ZM1,ZM2,NDIM,NDIM,T1)
      CALL DIAG(NDIM,COM1,CTL(LPHDL),1.,1.)
      CALL MAT3A(NDIM,NDIM,COM1,ZM1,X)
      CALL MAT3A(NRD,NDIM,CTL(LBDL),ZM1,ZM2)
      CALL MAT2A(NDIM,NDIM,COM1,ZM1,ZM1)
      CALL MMUL(ZM1,CTL(LBDL),NDIM,NDIM,NRD,S)
      CALL TFRMTX(RPI(LUDW),ZM1,NRD,NRD,2)
      CALL MADD1(NRD,NRD,ZM2,ZM1,U,TSAMP)
      RETURN
C END SUBROUTINE WXUS
      END




      SUBROUTINE FORMX(QY,RY,C,D,X,Z1,Z2)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      DIMENSION QY(1),RY(1),C(1),D(1),X(1),Z1(1),Z2(1)
      CALL FMMUL(QY,C,NPD,NPD,NND,Z1)
      CALL FTMUL(C,Z1,NPD,NND,NND,Z2)
      CALL TFRMTX(Z2,X,NND,NND,2)
      L1=LADDR(NNPR,NND+1,NND+1)
      CALL TFRMTX(RY,X(L1),NRD,NRD,2)
      L2=LADDR(NNPR,NND+1,1)
      IF(NODY.EQ.0) GO TO 5
      CALL ZPART(X(L2),NRD,NND,NNPR)
      GO TO 15
 5    CALL FTMUL(D,Z1,NPD,NPD,NND,Z2)
      CALL TFRMTX(Z2,X(L2),NRD,NND,2)
      CALL FMMUL(QY,D,NPD,NPD,NRD,Z1)
      CALL FTMUL(D,Z1,NPD,NRD,NRD,Z2)
      L2=0
      DO 12 I=1,NRD
      L1=LADDR(NNPR,NND+1,NND+I)
      DO 12 J=1,NRD
      L1=L1+1
      L2=L2+1
      X(L1)=X(L1)+Z2(L2)
 12   L1=L1+1
 15   DO 20 I=1,NND
      L1=LADDR(NNPR,NND+1,I)
      L2=LADDR(NNPR,I,NND+1)
      DO 20 J=1,NRD
      X(L2)=X(L1)
      L1=L1+1
 20   L2=L2+NNPR
      RETURN
C END SUBROUTINE FORMX
      END
```

```
      SUBROUTINE PXUP(PHIDL,BDEL,X,U,S,BUIBT,UIST,PHIP,XP,ZM1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDC,NPLD,NWPNWC,NNPR
      DIMENSION PHIDL(1),BDEL(1),X(1),U(1),S(1),BUIBT(1),UIST(1),
     1 PHIP(1),XP(1),ZM1(1)
      CALL GMINV(NRD,NRD,U,ZM1,MR,1)
      CALL MAT3(NDIM,NRD,BDEL,ZM1,BUIBT)
      CALL MAT5(ZM1,S,NRD,NRD,NDIM,UIST)
      CALL MMUL(BDEL,UIST,NDIM,NRD,NDIM,ZM1)
      CALL MADD1(NDIM,NDIM,PHIDL,ZM1,PHIP,-1.)
      CALL MMUL(S,UIST,NDIM,NRD,NDIM,ZM1)
      CALL MADD1(NDIM,NDIM,X,ZM1,XP,-1.)
      RETURN
C END SUBROUTINE PXUP
      END
      SUBROUTINE GCSTAR(PHIP,BDEL,U,RK,UIST,GCS,ZM1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWCD,NPLD,NWPNWD,NNPR
      DIMENSION PHIP(1),BDEL(1),U(1),RK(1),UIST(1),GCS(1),ZM1(1)
      CALL MAT3A(NRD,NDIM,BDEL,RK,ZM1)
      CALL MADD1(NRD,NRD,ZM1,U,ZM1,1.)
      CALL GMINV(NRD,NRD,ZM1,U,MR,1)
      CALL MAT5(U,BDEL,NRD,NRD,NDIM,ZM1)
      CALL MAT1(ZM1,RK,NRD,NDIM,NDIM,GCS)
      CALL MMUL(GCS,PHIP,NRD,NDIM,NDIM,ZM1)
      CALL MADD1(NRD,NDIM,ZM1,UIST,GCS,1.)
      WRITE(KLIST,131)
  131 FORMAT("0REG/PI GAIN MATRIX--GCS"/)
      CALL MATIO(GCS,NRD,NDIM,3)
      RETURN
C END SUBROUTINE GCSTAR
      END
      SUBROUTINE SCGT
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NPD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMCN/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      COMMON/LCGT/LA11,LA13,LA21,LA23,LA12,LA22,LKXA11,LKXA12,LKXA13
      COMMON/CCGT/NVCGT,CGT(1)
      IF(NEWCM) 20,20,15
 15   NSIZE=(NND+2*NPD)*(NNC+NRC+NDD)
      IF(NSIZE.LE.NVCGT) GO TO 16
      WRITE 106,NSIZE
      LABORT=NSIZE
      RETURN
 16   IF(NND.GE.NNC) GO TO 17
      WRITE 107
      GO TO 18
 17   IF(NND.GE.NDD) GO TO 19
      WRITE 108
 18   LABORT=-1
      RETURN
 19   LA11=1
      LA13=LA11+NND*NNC
      LA21=LA13+NND*NDD
      LA23=LA21+NPD*NNC
      LA12=LA23+NPD*NDD
      LA22=LA12+NND*NRC
      LKXA11=LA22+NPD*NRC
      LKXA12=LKXA11+NPD*NNC
      LKXA13=LKXA12+NPD*NRC
      CALL CGTA(CGT(LA11),CGT(LA13),CGT(LA21),CGT(LA23),CGT(LA12),
     1 CGT(LA22),ZM1,ZM2)
 20   CALL CGTKX(CGT(LA11),CGT(LA13),CGT(LA21),CGT(LA23),CGT(LA12),
     1 CGT(LA22),CGT(LKXA11),CGT(LKXA12),CGT(LKXA13),RPI(LKX))
      LFLCGT=1
 106  FORMAT("  INSUFFICIENT MEMORY /CCGT/, NEED: ",I4)
 107  FORMAT(" FEWER DESIGN MODEL THAN COMMAND MODEL STATES")
 108  FORMAT("FEWER DESIGN MODEL THAN DISTURBANCE MODEL STATES")
      RETURN
C END SUBROUTINE SCGT
      END
      SUBROUTINE CGTA(A11,A13,A21,A23,A12,A22,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      DIMENSION A11(1),A13(1),A21(1),A23(1),A12(1),A22(1),ZM1(1),ZM2(1)
      NDIM=NND
      NDIM1=NDIM+1
      CALL TFRMTX(CM,ZM1,NNC,NNC,2)
      CALL SUBI(ZM1,NNC,NDIM)
      CALL FMMUL(CTL(LPI12),CM(LCC),NND,NRD,NNC,ZM2)
      CALL SCALE(ZM2,ZM2,NND,NNC,-1.)
      NB=MAX0(NDD,NNC)
      L2=1+NND*NND
      L3=L2+NND*NB
      L4=L3+NND*NB
      L5=L4+NND*NND
      L6=L5+NND*NND
      NSIZE=L6+NPD*NNC-1
      IF(NSIZE.LE.NVSM) GO TO 1
      WRITE 102,NSIZE
102   FORMAT("" INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
      LABORT=NSIZE
      RETURN
1     CALL AXBMXC(CTL(LPI11),NND,ZM1,NNC,ZM2,A11,SM,
     1 SM(L2),SM(L3),SM(L4),SM(L5))
      CALL MMUL(A11,ZM1,NND,NNC,NNC,ZM2)
      CALL FMMUL(CTL(LPI21),ZM2,NPD,NND,NNC,A21)
      CALL FMMUL(CTL(LPI22),CM(LCC),NPD,NRD,NNC,SM(L6))
      CALL FMMUL(A11,CM(LBDC),NND,NNC,NRC,SM)
      CALL FMMUL(CTL(LPI11),SM,NND,NND,NRC,A12)
      CALL FMMUL(CTL(LPI21),SM,NPD,NND,NRC,A22)
      IF(NODC.EQ.1) GO TO 2
      CALL FMMUL(CTL(LPI12),CM(LDC),NND,NRD,NRC,SM(L2))
      CALL FMADD(A12,SM(L2),NND,NRC,A12)
      CALL FMMUL(CTL(LPI22),CM(LDC),NPD,NRD,NRC,SM(L2))
      CALL FMADD(A22,SM(L2),NPD,NRC,A22)
2     IF(NDD.EQ.0) GO TO 15
      CALL MMUL(CTL(LPI11),DM(LEX),NND,NND,NDD,ZM2)
      IF(NOEY.EQ.1) GO TO 5
      CALL FMMUL(CTL(LPI12),DM(LEY),NND,NRD,NDD,ZM1)
      CALL MADD1(NND,NDD,ZM1,ZM2,ZM2,1.)
5     CALL TFRMTX(DM(LPHD),ZM1,NDD,NDD,2)
      CALL SUBI(ZM1,NDD,NDIM)
      CALL AXBMXC(CTL(LPI11),NND,ZM1,NDD,ZM2,A13,SM,
     1 SM(L2),SM(L3),SM(L4),SM(L5))
      CALL MMUL(A13,ZM1,NND,NDD,NDD,ZM2)
      CALL MADD1(NND,NDD,ZM2,DM(LEX),ZM2,-1.)
      CALL FMMUL(CTL(LPI21),ZM2,NPD,NND,NDD,A23)
 15   NDIM=NPD
      NDIM1=NDIM+1
      CALL MADD1(NPD,NNC,A21,SM(L6),A21,1.)
      IF(NOEY.EQ.1) GO TO 20
      CALL MMUL(CTL(LPI22),DM(LEY),NPD,NRD,NDD,ZM1)
      CALL MADD1(NPD,NDD,A23,ZM1,A23,-1.)
 20   CALL MATLST(A11,NND,NNC,"A11",KLIST)
      CALL MATLST(A21,NPD,NNC,"A21",KLIST)
      CALL MATLST(A12,NND,NRC,"A12",KLIST)
      CALL MATLST(A22,NPD,NRC,"A22",KLIST)
      IF(NDD.GT.0) GO TO 25
      WRITE(KLIST,101)
 101  FORMAT("0MATRICES A13 AND A23 ARE ZERO")
      RETURN
 25   CALL MATLST(A13,NND,NDD,"A13",KLIST)
      CALL MATLST(A23,NPD,NDD,"A23",KLIST)
      RETURN
C END SUBROUTINE CGTA
      END




      SUBROUTINE AXBMXC(A,NA,B,NB,C,X,AU,BU,R,Z1,Z2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION A(1),B(1),C(1),X(1),AU(1),BU(1),R(1),Z1(1),Z2(1)
      DATA EMAX,ITMAX/1.E-6,3/
      CALL TRANS1(NA,A,Z1)
      CALL EIGEN(NA,Z1,Z2,Z2(NDIM1),AU,1)
      CALL TRANS1(NA,COM1,Z1)
      CALL EIGEN(NB,B,Z2,Z2(NDIM1),BU,1)
      CALL EQUATE(R,C,NA,NB)
      IT=0
 10   CALL MAT4A(AU,R,NA,NA,NB,Z2)
      CALL MAT1(Z2,BU,NA,NB,NB,R)
      CALL SLVSHR(Z1,NA,COM1,NB,R,NDIM)
      CALL MAT4(R,BU,NA,NB,NB,Z2)
      CALL MAT1(AU,Z2,NA,NA,NB,R)
      IF(IT.GT.0) GO TO 15
      CALL EQUATE(X,R,NA,NB)
      GO TO 30
 15   CALL MADD1(NA,NB,X,R,X,1.)
      CALL ENORM(R,NA,NB,EN)
      IF(EN.LE.EMAX) RETURN
      IF(IT.LT.ITMAX) GO TO 30
      WRITE(KLIST,101) EN
      WRITE(KTERM,101) EN
 101  FORMAT("0SOLUTION ERROR FOR 'A'(CGT) AFTER 3 ITERATIONS = ",1PE15.
     17)
      RETURN
 30   CALL MAT1(A,X,NA,NA,NB,Z2)
      CALL MAT1(Z2,B,NA,NB,NB,R)
      CALL MADD1(NA,NB,X,R,R,-1.)
      CALL MADD1(NA,NB,R,C,R,1.)
      IT=IT+1
      GO TO 10
C END SUBROUTINE AXBMXC
      END




      SUBROUTINE SLVSHR(A,NA,B,NB,C,ND)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      DIMENSION A(ND,1),B(ND,1),C(ND,1),V(16),W(4)
      L=1
 5    LM1=L-1
      DL=1
      IF(L.EQ.NB) GO TO 8
      IF(B(L+1,L).NE.0.) DL=2
 8    LL=LM1+DL
      K=1
 10   KM1=K-1
      DK=1
      IF(K.EQ.NA) GO TO 12
      IF(A(K,K+1).NE.0.) DK=2
 12   KK=KM1+DK
      AKK=A(K,K)
      BLL=B(L,L)
      IF(DL.EQ.2) GO TO 35
      IF(DK.EQ.2) GO TO 20
      IF(L.EQ.1) GO TO 13
      C(K,L)=C(K,L)-AKK*DOT3(LM1,C(K,1),B(1,L))
 13   IF(K.EQ.1) GO TO 18
      DO 15 I=1,KM1
 15   C(K,L)=C(K,L)-A(K,I)*DOT3(L,C(I,1),B(1,L))
 18   V(1)=AKK*BLL-1.
      IF(V(1).EQ.0.) GO TO 99
      C(K,L)=C(K,L)/V(1)
      GO TO 95
 20   IF(L.EQ.1) GO TO 22
      I1=K
      I2=KK
      I3=LM1
      GO TO 24
22    IF(K.EQ.1) GO TO 30
      I1=1
      I2=KM1
      I3=L
24    DO 28 I=I1,I2
      V(1)=DOT3(I3,C(I,1),B(1,L))
      C(K,L)=C(K,L)-A(K,I)*V(1)
28    C(KK,L)=C(KK,L)-A(KK,I)*V(1)
      IF(I1.EQ.K) GO TO 22
30    V(1)=AKK*BLL-1.
      V(2)=A(KK,K)*BLL
      V(3)=A(K,KK)*BLL
      V(4)=A(KK,KK)*BLL-1.
      V(5)=1./(V(1)*V(4)-V(2)*V(3))
      V(6)=V(5)*(C(K,L)*V(4)-V(3)*C(KK,L))
      C(KK,L)=V(5)*(V(1)*C(KK,L)-V(2)*C(K,L))
      C(K,L)=V(6)
      GO TO 95
35    IF(DK.EQ.2) GO TO 50
      IF(L.EQ.1) GO TO 38
      I1=K
      I2=K
      I3=LM1
      GO TO 40
38    IF(K.EQ.1) GO TO 45
      I1=1
      I2=KM1
      I3=LL
40    DO 42 I=I1,I2
      C(K,L)=C(K,L)-A(K,I)*DOT3(I3,C(I,1),B(1,L))
42    C(K,LL)=C(K,LL)-A(K,I)*DOT3(I3,C(I,1),B(1,LL))
      IF(I1.EQ.K) GO TO 38
45    V(1)=AKK*BLL-1.
      V(2)=AKK*B(L,LL)
      V(3)=AKK*B(LL,L)
      V(4)=AKK*B(LL,LL)-1.
48    V(5)=1./(V(1)*V(4)-V(2)*V(3))
      V(6)=V(5)*(C(K,L)*V(4)-V(3)*C(K,LL))
      C(K,LL)=V(5)*(V(1)*C(K,LL)-V(2)*C(K,L))
      C(K,L)=V(6)
      GO TO 95
50    IF(L.EQ.1) GO TO 55
      V(1)=DOT3(LM1,C(K,1),B(1,L))
      V(2)=DOT3(LM1,C(KK,1),B(1,L))
      V(3)=DOT3(LM1,C(K,1),B(1,LL))
      V(4)=DOT3(LM1,C(KK,1),B(1,LL))
      C(K,L)=C(K,L)-AKK*V(1)-A(K,KK)*V(2)
      C(KK,L)=C(KK,L)-A(KK,K)*V(1)-A(KK,KK)*V(2)
      C(K,LL)=C(K,LL)-AKK*V(3)-A(K,KK)*V(4)
      C(KK,LL)=C(KK,LL)-A(KK,K)*V(3)-A(KK,KK)*V(4)
 55   IF(K.EQ.1) GO TO 65
      DO 60 I=1,KM1
      V(1)=DOT3(LL,C(I,1),B(1,L))
      V(2)=DOT3(LL,C(I,1),B(1,LL))
      C(K,L)=C(K,L)-A(K,I)*V(1)
      C(KK,L)=C(KK,L)-A(KK,I)*V(1)
      C(K,LL)=C(K,LL)-A(K,I)*V(2)
 60   C(KK,LL)=C(KK,LL)-A(KK,I)*V(2)
 65   V(1)=AKK*BLL-1.
      V(2)=A(KK,K)*BLL
      V(3)=AKK*B(L,LL)
      V(4)=A(KK,K)*B(L,LL)
      V(5)=A(K,KK)*BLL
      V(6)=A(KK,KK)*BLL-1.
      V(7)=A(K,KK)*B(L,LL)
      V(8)=A(KK,KK)*B(L,LL)
      V(9)=AKK*B(LL,L)
      V(10)=A(KK,K)*B(LL,L)
      V(11)=AKK*B(LL,LL)-1.
      V(12)=A(KK,K)*B(LL,LL)
      V(13)=A(K,KK)*B(LL,L)
      V(14)=A(KK,KK)*B(LL,L)
      V(15)=A(K,KK)*B(LL,LL)
      V(16)=A(KK,KK)*B(LL,LL)-1.
      W(1)=C(K,L)
      W(2)=C(KK,L)
      W(3)=C(K,LL)
      W(4)=C(KK,LL)
      NDS=NDIM
      NDIM=4
      NDIM1=NDIM+1
      CALL DOOLIT(4,V,W,1,ISG)
      NDIM=NDS
      NDIM1=NDIM+1
 95   K=K+DK
      IF(K.LE.NA) GO TO 10
      L=L+DL
      IF(L.LE.NB) GO TO 5
      RETURN
 99   WRITE(KOUT,101)
      RETURN
 101  FORMAT("0* * * ERROR IN CGT SOLUTION: A11->A23")
C END SUBROUTINE SLVSHR
      END
      SUBROUTINE ENORM(A,NR,NC,ENRM)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      DIMENSION A(1)
      ENRM=0.
      NE=NC*NDIM
      DO 10 I=1,NR
      DO 10 J=I,NE,NDIM
   10 ENRM=ENRM+A(J)*A(J)
      ENRM=SQRT(ENRM)
      RETURN
C END SUBROUTINE ENORM
      END
```

```
      SUBROUTINE CGTKX(A11,A13,A21,A23,A12,A22,RKXA11,RKXA12,RKXA13,RKX)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPF
      COMMON/NDIMC/NNC,NRC,NPC
      DIMENSION A11(1),A13(1),A21(1),A23(1),A12(1),A22(1),
     1 RKXA11(1),RKXA12(1),RKXA13(1),RKX(1)
      NDIM=NRD
      NDIM1=NDIM+1
      CALL FMMUL(RKX,A11,NRD,NND,NNC,RKXA11)
      CALL MADD1(NRD,NNC,RKXA11,A21,RKXA11,1.)
      CALL MATLST(RKXA11,NRD,NNC,"KXM",KLIST)
      CALL MATLST(RKXA11,NRD,NNC,"KXM",KTERM)
      CALL FMMUL(RKX,A12,NRD,NND,NRC,RKXA12)
      CALL MADD1(NRD,NRC,RKXA12,A22,RKXA12,1.)
      CALL MATLST(RKXA12,NRD,NRC,"KXU",KLIST)
      CALL MATLST(RKXA12,NRD,NRC,"KXU",KTERM)
      IF(NDD.LT.1) RETURN
      CALL FMMUL(RKX,A13,NRD,NND,NDD,RKXA13)
      CALL MADD1(NRD,NDD,RKXA13,A23,RKXA13,1.)
      CALL MATLST(RKXA13,NRD,NDD,"KXN",KLIST)
      CALL MATLST(RKXA13,NRD,NDD,"KXN",KTERM)
      RETURN
C END SUBROUTINE CGTKX
      END
      SUBROUTINE CEVAL
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      DIMENSION NPLOT(2),NVPLOT(10),NS(6),LSCL(2),ITITLE(5)
      DATA NC/1HN/
      WRITE(KLIST,110)
110   FORMAT(////11X,5("* "),"CONTROLLER EVALUATION",5(" *")////)
      IPOLE=1
10    NVOUT=NRD+NPD+1
      IF(LFLCGT) 17,17,15
15    WRITE 106
      READ*,IUM,VUM
      IF(IUM.LT.1) GO TO 70
      IF(IUM.GT.NRC) GO TO 15
      NVOUT=NVOUT+NPC
      NP=NNC
      GO TO 18
17    IF(IPOLE.EQ.1) CALL POLES(RPI(LPHCL),NNPR,4,ZM1,ZM2)
      NP=0
18    CALL VOUTIC(SM,NVPLOT,NPLOT,NVOUT,LSCL)
      IF(NVOUT.EQ.0) GO TO 70
20    WRITE 108
      READ*,TEND
      IF(TEND) 20,20,25
25    LVX0=NVOUT+1
      LX0=LVX0+NVOUT
      LX1=LX0+NPLD
      LXM0=LX1+NPLD
      LXM1=LXM0+NP
      NP=LXM1+NP
      DO 26 I=LVX0,NP
26    SM(I)=0.
      CALL CTRESP(SM(LVX0),SM,SM(LX0),SM(LX1),SM(LXM0),SM(LXM1),
     1 ZM1,NVOUT,TEND,IUM,VUM,NST)
      WRITE(KTERM,101)
      READ(KIN,102) ITITLE
      M=50*NST
      DO 40 I=1,2
      NS(1)=1
      DO 28 J=2,6
28    NS(J)=NS(J-1)+51
      NP=NPLOT(I)
      IF(NP.EQ.0) GO TO 40
      NPP1=NP+1
      REWIND KPLOT
      NSV=5*I-4
      CALL RPLOTF(ZM1,NVOUT,IERR)
      CALL STRPLT(SM,ZM1,NS,NVPLOT(NSV),NP,NVOUT)
      DO 35 J=1,M
      CALL RPLOTF(ZM1,NVOUT,IERR)
      IF(IERR.EQ.1) GO TO 40
      IF(MOD(J,NST).NE.0) GO TO 35
      DO 30 K=1,NPP1
30    NS(K)=NS(K)+1
      CALL STRPLT(SM,ZM1,NS,NVPLOT(NSV),NP,NVOUT)
35    CONTINUE
      CALL PLOTLP(51,NP,SM,LSCL(I),1,[illegible],KTERM,ITITLE)
40    CONTINUE
      NVM=NVOUT-1
      M=NVM/5
      NE=5*M
      IF(M.EQ.0) GO TO 56
      DO 55 I=1,NE,5
      NS(1)=1
      DO 42 J=2,6
42    NS(J)=NS(J-1)+101
      REWIND KPLOT
      NVS=I-1
      DO 45 J=1,5
45    NVPLOT(J)=NVS+J
      DO 50 J=1,101
      CALL RPLOTF(ZM1,NVOUT,IERR)
      IF(IERR.EQ.1) GO TO 55
      CALL STRPLT(SM,ZM1,NS,NVPLOT,5,NVOUT)
      DO 48 K=1,6
48    NS(K)=NS(K)+1
50    CONTINUE
      CALL PLOTLP(101,5,SM,1,1,1,KLIST,ITITLE)
55    CONTINUE
56    NVM=NVM-NE
      IF(NVM.LT.1) GO TO 70
      NPP1=NVM+1
      NS(1)=1
      DO 57 I=2,6
57    NS(I)=NS(I-1)+101
      DO 58 I=1,NVM
58    NVPLOT(I)=NE+I
      REWIND KPLOT
      DO 65 I=1,101
      CALL RPLOTF(ZM1,NVOUT,IERR)
      IF(IERR.EQ.1) GO TO 70
      CALL STRPLT(SM,ZM1,NS,NVPLOT,NVM,NVOUT)
      DO 60 J=1,NPP1
```

```
 60   NS(J)=NS(J)+1
 65   CONTINUE
      CALL PLOTLP(101,NVM,SM,1,1,1,KLIST,ITITLE)
 70   WRITE 104
      READ 111,IANS
      IF(IANS.EQ.NO) RETURN
      IPOLE=0
      GO TO 10
 101  FORMAT(" +",10("-")," ENTER TITLE IN GIVEN FIELD ",10("-"),"+"/)
 102  FORMAT(5A10)
 104  FORMAT("0MORE TIME RESPONSE RUNS (Y OR N) >")
 106  FORMAT("0ENTER MODEL INPUT AND STEP VALUE : 1 >")
 108  FORMAT(" ENTER TIME DURATION FOR RESPONSE, IN SECONDS >")
 111  FORMAT(A3)
C END SUBROUTINE CEVAL
      END




      SUBROUTINE VOUTIC(VIC,NVPLOT,NPLOT,NVOUT,LSCL)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      DIMENSION NPLOT(1),NVPLOT(1),VIC(1),IOUT(5),LSCL(2)
      DATA IOUT/1HX,1HY,1HU,1HM,1HD/
      IF(LTEVAL) 2,2,5
 2    NVS=NND
      NV=NPLD
      GO TO 8
 5    NV=NNT
      NVS=NV
 8    NVOUT=NVOUT+NV
      DO 9 I=1,NVOUT
 9    VIC(I)=0.
      NVU=NV+NPD
      NVM=NVU+NPD
 10   WRITE 101,NVS
 101  FORMAT("0ENTER STATE AND IC VALUE (0/ TERMINATES): ",I2," >")
 12   READ*,IV,V
      IF(IV.LT.1) GO TO 15
      IF(IV.GT.NVS) GO TO 10
      VIC(IV)=V
      GO TO 12
 15   IF((LFLCGT.LT.1).OR.(LTEVAL.EQ.1).OR.(NDD.LT.1)) GO TO 25
      LD=1
 18   WRITE 102,NDD
 102  FORMAT(" ENTER DISTURBANCE IC VALUE (0/ TERMINATES): ",I2," >")
```

```
20    READ*,IV,V
      IF(IV.LT.1) GO TO 26
      IF(IV.GT.NDD) GO TO 18
      VIC(NND+IV)=V
      GO TO 20
25    LD=0
26    WRITE 103
103   FORMAT(" 2 PLOTS OF 5 VARIABLES MAY BE PRINTED AT THE TERMINAL --
     1SPECIFY NUMBER FOR EACH (N1,N2) >")
      READ*,NPLOT(1),NPLOT(2)
      IF(NPLOT(1).GT.5) NPLCT(1)=5
      IF(NPLCT(2).GT.5) NPLOT(2)=5
      IF((NPLOT(1).GT.0).OR.(NPLOT(2).GT.0)) GO TO 27
      NVOUT=0
      RETURN
27    WRITE 104
104   FORMAT(" ENTER OUTPUTS BY TYPE AND INDEX IN 2 ENTRIES--TYPES ARE"/
     1 " STATE : 'X'"/" OUTPUT : 'Y'"/" INPUT : 'U'")
      IF(LFLCGT) 30,30,28
28    WRITE 105
105   FORMAT(" MODEL : 'M'")
      IF(LD.EQ.1) WRITE 106
106   FORMAT(" DISTURBANCE : 'D'")
30    DO 40 I=1,2
      NC=NPLOT(I)
      IF(NC.LT.1) GO TO 40
      LSCL(I)=1
      NS=5*(I-1)
      WRITE 107,I
107   FORMAT("0PLOT ",I2)
      DO 39 J=1,NC
      NSP=NS+J
31    WRITE 108,J
108   FORMAT(" OUTPUT ",I2," >")
      READ 111,IV
      WRITE 113
113   FORMAT(11X,">")
      READ*,IO
      IF(IV.NE.IOUT(1)) GO TO 32
      IF(IO.GT.NVS) GO TO 38
      NVPLOT(NSP)=IO
      GO TO 39
32    IF(IV.NE.IOUT(2)) GO TO 321
      IF(IO.GT.NPO) GO TO 38
      NVPLOT(NSP)=NV+IO
      GO TO 39
321   IF(IV.NE.IOUT(3)) GO TO 33
      IF(IO.GT.NRD) GO TO 38
      NVPLOT(NSP)=NVU+IO
      GO TO 39
33    IF(LFLCGT.LT.1) GO TO 31
```

```
      IF(IV.NE.IOUT(4)) GO TO 34
      IF(IO.GT.NPC) GO TO 38
      NVPLOT(NSP)=NVM+IO
      LSCL(I)=-1
      GO TO 39
 34   IF(LO.NE.1) GO TO 31
      IF(IV.NE.IOUT(5)) GO TO 31
      IF(IO.GT.NDD) GO TO 38
      NVPLOT(NSP)=NVS+IO
      GO TO 39
 38   WRITE 109
 109  FORMAT(" INDEX TOO LARGE")
      GO TO 31
 39   CONTINUE
 40   CONTINUE
      NVM1=NVOUT-1
      NP=0
      DO 50 I=1,NVM1
      M=MOD((I-1),5)+1
      IF(M.GT.1) GO TO 41
      NP=NP+1
      WRITE(KLIST,110) NP
 110  FORMAT("1PLOT ",I2)
 41   IF(I.GT.NVS) GO TO 42
      IV=IOUT(1)
      IO=I
      GO TO 45
 42   IF(I.GT.NV) GO TO 43
      IV=IOUT(5)
      IO=I-NVS
      GO TO 45
 43   IF(I.GT.NVU) GO TO 441
      IV=IOUT(2)
      IO=I-NV
      GO TO 45
 441  IF(I.GT.NVM) GO TO 44
      IV=IOUT(3)
      IO=I-NVU
      GO TO 45
 44   IV=IOUT(4)
      IO=I-NVM
 45   WRITE(KLIST,112) M,IV,IO
 50   CONTINUE
      RETURN
 111  FORMAT(A1)
 112  FORMAT("    OUTPUT ",I2,": ",A1,I2)
C END SUBROUTINE VOUTIC
      END
      SUBROUTINE CTRESP(VX0,VX1,X0,X1,XM0,XM1,ZM1,NVOUT,TEND,IUM,VUM,
     1 NST)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      DIMENSION VX0(1),VX1(1),X0(1),X1(1),XM0(1),XM1(1),ZM1(1)
      NSTPO=.01*TEND/TSAMP+.5
      NST=2
      IF(NSTPO.GE.1) GO TO 1
      NSTPO=1
      NST=1
1     NSTEPS=100*NSTPO
      IF(LFLCGT.EQ.0) GO TO 2
      LMO=NVOUT-NPC
      IF(NDC.EQ.0) GO TO 4
      LDCGT=1
      GO TO 5
2     LMO=NVOUT
4     LDCGT=0
5     LU=LMO-NPD
      LSO=LU-NPD
      NVX=LSO-1
      IF(LTEVAL)6,6,10
6     DO 7 I=1,NVX
7     X1(I)=VX1(I)
      GO TO 12
10    CALL XFDT(VX1,X1,LDCGT)
12    NNDP1=NND+1
      REWIND KPLOT
      CALL YDSN(X1,VX1(LU),DM(LC),DM(LDY),LDCGT,VX1(LSO))
      IF(LFLCGT.EQ.1) CALL YCMD(XM1,IUM,VUM,CM(LCC),CM(LDC),
     1 VX1(LMO))
      CALL WPLOTF(VX1,NVOUT)
      DO 100 IT=1,NSTEPS
      CALL URPI(RPI(LKX),RPI(LKZ),DM(LC),DM(LDY),X0,X1,VX0(LU),VX1(LU))
      IF(LFLCGT) 20,20,15
15    CALL UCGT(VX0(LU),VX1(LU),XM0,XM1,X0(NNDP1),ZM1,IUM,VUM,IT)
      CALL CUPDAT(XM0,XM1,IUM,VUM)
20    CALL FTMTX(VX1(LU),VX0(LU),NRD,1)
      CALL FTMTX(X1,X0,NPLD,1)
      IF(LTEVAL) 25,25,30
25    CALL DUPDAT(DM(LPHI),DM(LBD),DM(LPHD),DM(LEX),X0,X1,
     1 VX1,VX0(LU),LDCGT,NNDP1)
       GO TO 35
  30   CALL TUPDAT(TM(LPHT),TM(LBDT),VX0,VX1,VX0(LU))
       CALL XFDT(VX1,X1,LDCGT)
  35   IF(MOD(IT,NSTPO).NE.0) GO TO 100
       VX1(NVOUT)=TSAMP*FLOAT(IT)
       CALL YOSN(X1,VX1(LU),DM(LC),DM(LDY),LDCGT,VX1(LSO))
       IF(LFLCGT.EQ.1) CALL YCMD(XM1,IUM,VUM,CM(LCC),CM(LDC),
     1 VX1(LMO))
       CALL WPLOTF(VX1,NVOUT)
 100   CONTINUE
       ENDFILE KPLOT
       RETURN
C END SUBROUTINE CTRESP
       END




       SUBROUTINE DUPDAT(PHI,BD,PHID,EX,X0,X1,VX1,U0,LDCGT,NNDP1)
       COMMON/MAIN1/NDIM,NDIM1,COM1(1)
       COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
       DIMENSION PHI(1),BD(1),PHID(1),EX(1),X0(1),X1(1),VX1(1),U0(1)
       NDIM=NND
       NDIM1=NDIM+1
       CALL FMMUL(BD,U0,NND,NRD,1,X1)
       CALL MMULS(PHI,X0,NND,NND,1,X1)
       IF(LDCGT.EQ.0) GO TO 10
       CALL FMMUL(PHID,X0(NNDP1),NDD,NDD,1,X1(NNDP1))
       CALL MMULS(EX,X1(NNDP1),NND,NDD,1,X1)
 10    CALL FTMTX(X1,VX1,NPLD,1)
       RETURN
C END SUBROUTINE DUPDAT
       END
```

```
      SUBROUTINE CUPDAT(XM0,XM1,IUM,VUM)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      DIMENSION XM0(1),XM1(1)
      NDIM=NNC
      NDIM1=NDIM+1
      CALL FTMTX(XM1,XM0,NNC,1)
      L1=LADDR(NNC,1,IUM)+LBDC-1
      CALL VSCALE(XM1,CM(L1),NNC,VUM)
      CALL MMULS(CM(LPHC),XM0,NNC,NNC,1,XM1)
      RETURN
C END SUBROUTINE CUPDAT
      END



      SUBROUTINE TUPDAT(PHI,BD,VX0,VX1,U0)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      DIMENSION PHI(1),BD(1),VX0(1),VX1(1),U0(1)
      NDIM=NNT
      NDIM1=NDIM+1
      CALL FTMTX(VX1,VX0,NNT,1)
      CALL FMMUL(BD,U0,NNT,NRT,1,VX1)
      CALL MMULS(PHI,VX0,NNT,NNT,1,VX1)
      RETURN
C END SUBROUTINE TUPDAT
      END
      SUBROUTINE XFDT(V,X,LDCGT)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LODT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      DIMENSION V(1),X(1)
      CALL FMMUL(TM(LTDT),V,NND,NNT,1,X)
      IF(LDCGT.EQ.0) RETURN
      CALL FMMUL(TM(LTNT),V,NDD,NNT,1,X(NND+1))
      RETURN
C END SUBROUTINE XFDT
      END




      SUBROUTINE URPI(RKX,RKZ,C,DY,X0,X1,U0,U1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      DIMENSION RKX(1),RKZ(1),C(1),DY(1),X0(1),X1(1),U0(1),U1(1)
      CALL YDSN(X0,U0,C,DY,0,U1)
 10   CALL VSCALE(U1,U1,NRD,-1.)
      CALL MMULS(RKZ,U1,NRD,NRD,1,U0)
      DO 12 I=1,NPLD
 12   X0(I)=X0(I)-X1(I)
      CALL FMMUL(RKX,X0,NRD,NND,1,U1)
      CALL VADD(NRD,1.,U1,U0)
      RETURN
C END SUBROUTINE URPI
      END




      SUBROUTINE UCGT(U0,U1,XM0,XM1,DDIF,ZM1,IUM,VUM,IT)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CFEGPI/NVRPI,RPI(1)
      COMMON/LCGT/LA11,LA13,LA21,LA23,LA12,LA22,LKXA11,LKXA12,LKXA13
      COMMON/CCGT/NVCGT,CGT(1)
      DIMENSION U0(1),U1(1),XM0(1),XM1(1),DDIF(1),ZM1(1)
      CALL YCMD(XM0,IUM,VUM,CM(LCC),CM(LDC),U0)
      IF(IT.GT.1) GO TO 10
      I=LKXA12+LADDR(NPD,1,IUM)-1
      CALL MADD1(NPD,1,U1,CGT(I),U1,VUM)
 10   CALL MMULS(RPI(LKZ),U0,NDIM,NDIM,1,U1)
      DO 12 I=1,NNC
 12   XM0(I)=XM1(I)-XM0(I)
      CALL FMMUL(CGT(LKXA11),XM0,NPD,NNC,1,U0)
      CALL VADD(NDIM,1.,U1,U0)
      IF(NDD.EQ.0) RETURN
      DO 14 I=1,NDD
 14   DDIF(I)=-DDIF(I)
      CALL MMULS(CGT(LKXA13),DDIF,NPD,NDD,1,U1)
      RETURN
C END SUBROUTINE UCGT
      END




      SUBROUTINE YDSN(X,U,C,D,LDCGT,Y)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWCD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      DIMENSION X(1),U(1),C(1),D(1),Y(1)
      NDIM=NPD
      NDIM1=NDIM+1
      CALL FMMUL(C,X,NPD,NND,1,Y)
      IF(NODY.EQ.1) GO TO 10
      CALL MMULS(D,U,NPD,NFD,1,Y)
 10   IF((LDCGT.EQ.0).OR.(NOEY.EQ.1)) RETURN
      CALL MMULS(DM(LEY),X(NND+1),NPD,NDD,1,Y)
      RETURN
C END SUBROUTINE YDSN
      END
      SUBROUTINE YCMD(X,IU,VU,C,D,Y)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      DIMENSION X(1),C(1),D(1),Y(1)
      NDIM=NPC
      NDIM1=NDIM+1
      CALL FMMUL(C,X,NPC,NNC,1,Y)
      IF(NODC.EQ.1) RETURN
      L1=LADDR(NPC,1,IU)
      CALL MADD1(NPC,1,Y,D(L1),Y,VU)
      RETURN
C END SUBROUTINE YCMD
      END
      SUBROUTINE FLTRK(IFLTR)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/MAIN2/COM2(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,L
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CKF/NVFLT,FLT(1)
      IF(NWPNWD.GT.0) GO TO 1
      WRITE(KTERM,108)
108   FORMAT("0NO DRIVING NOISES - - FILTER DESIGN ABORT")
      RETURN
1     IF(NMD.GT.0) GO TO 2
      WRITE(KTERM,109)
109   FORMAT("0NO MEASUREMENTS - - FILTER DESIGN ABORT")
      RETURN
2     WRITE(KLIST,110)
110   FORMAT(////11X,5("* "),"FILTER DESIGN",5(" *")////)
      NSIZE=NPLD*(1+NPLD+NMD)
      IF(NSIZE.LE.NVFLT) GO TO 3
      WRITE 101,NSIZE
101   FORMAT("0INSUFFICIENT MEMORY /CKF/, NEED: ",I4)
      LABORT=NSIZE
      RETURN
3     NDIM=NPLD
      NDIM1=NDIM+1
5     IF(NWD.EQ.0) GO TO 12
      IF(IFLTR.LE.0) GO TO 6
      WRITE 105,NWD
105   FORMAT(" ENTER STATE NOISE STRENGTHS: ",I2)
      CALL RQWGTS(DM(LQ),NWD,0)
6     CALL DVCTOR(NWD,DM(LQ),ZM1)
      CALL MATLST(ZM1,NWD,1,"Q",KTERM)
10    CALL MATLST(DM(LQ),NWD,NWD,"Q",KLIST)
12    IF(NWDD.EQ.0) GO TO 18
      IF(IFLTR.LE.0) GO TO 13
      WRITE 106,NWDD
106   FORMAT(" ENTER DISTURBANCE NOISE STRENGTHS: ",I2)
      CALL RQWGTS(DM(LQN),NWDD,0)
13    CALL DVCTOR(NWDD,DM(LQN),ZM1)
      CALL MATLST(ZM1,NWDD,1,"QN",KTERM)
15    CALL MATLST(DM(LQN),NWDD,NWDD,"QN",KLIST)
18    CALL QDSCRT(DM(LQ),DM(LQN),ZM1,ZM2)
      IF(IFLTR.LE.0) GO TO 19
      WRITE 107,NMD
107   FORMAT(" ENTER MEASUREMENT NOISE STRENGTHS: ",I2)
```

```
      CALL RQWGTS(DM(LR),NMD,?)
19    CALL DVCTOR(NMD,DM(LR),ZM1)
      CALL MATLST(ZM1,NMD,1,"R",KTERM)
20    CALL MATLST(DM(LR),NMD,NMD,"R",KLIST)
25    CALL TFRMTX(DM(LHP),SM,NMD,NDIM,2)
      CALL TRANS2(NMD,NDIM,SM,ZM1)
      LFCOV=LFLTRK+NDIM*NMD
      CALL DVCTOR(NMD,DM(LR),FLT(LFCOV))
      CALL KFLTR(NDIM,NMD,FLT,ZM1,DM(LQD),FLT(LFCOV),ZM2,
     1 FLT(LFLTRK),SM)
      CALL TFRMTX(SM,COM2,NDIM,NDIM,2)
      IA=1
      DO 30 I=1,NPLD
      FLT(LFCOV-1+I)=SQRT(ZM2(IA))
30    IA=IA+NDIM1
      CALL MATLST(FLT(LFLTRK),NDIM,NMD,"KF",KLIST)
      CALL MATLST(FLT(LFLTRK),NDIM,NMD,"KF",KTERM)
      IFLTR=1
      LFLKF=1
111   FORMAT(A3)
      RETURN
C END SUBROUTINE FLTRK
      END
      SUBROUTINE FEVAL
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/MAIN2/COM2(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,L
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CKF/NVFLT,FLT(1)
      DIMENSION ITITLE(5),NS(3),NVPLOT(2)
      IF(NWT.GT.0) GO TO 1
      WRITE(KTERM,108)
  108 FORMAT("0NO TRUTH MODEL DRIVING NOISE - - FILTER EVALUATION ABORTE
     1D")
      RETURN
    1 WRITE(KLIST,110)
  110 FORMAT(////11X,5("* "),"FILTER EVALUATION",5(" *")////)
      CALL FMMUL(COM2,FLT(LEADSN),NPLD,NPLD,NPLD,SM)
      CALL POLES(SM,NPLD,5,ZM1,ZM2)
      NA=NNT+NPLD
      NSIZE=NA*NA
      IF(NSIZE.LE.NVSM) GO TO 8
      WRITE 101,NSIZE
  101 FORMAT("0INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
      GO TO 9
    8 IF(NSIZE.LE.NVZM) GO TO 10
      WRITE 103,NSIZE
  103 FORMAT("0INSUFFICIENT MEMORY /ZMTX1/,/ZMTX2/, NEED: ",I4)
    9 LABORT=NSIZE
      RETURN
   10 CALL ZPART(SM,1,NSIZE,1)
      NDIM=NPLD
      NDIM1=NDIM+1
      CALL TFRMTX(TM(LRT),ZM1,NMD,NMD,2)
      CALL MAT3(NPLD,NMD,FLT(LFLTRK),ZM1,COM1)
      NVOUT=2*NPLD+1
      REWIND KPLOT
      CALL DACOV(SM,FLT(LFCOV),ZM1,ZM2,NA,NVOUT,0.)
      DO 20 IT=1,50
      TIME=TSAMP*FLOAT(IT)
      CALL ACOVUD(SM,TM(LQDT),COM1,TM(LPHT),FLT(LEADSN),
     1 COM2,ZM1,ZM2)
      CALL DACOV(SM,FLT(LFCOV),ZM1,ZM2,NA,NVOUT,TIME)
```

```
20    CONTINUE
      ENDFILE KPLOT
      WRITE(KTERM,104)
      READ(KIN,102) ITITLE
      DO 50 I=1,NPLD
      REWIND KPLOT
      NS(1)=1
      NS(2)=52
      NS(3)=103
      NVPLOT(1)=I+I-1
      NVPLOT(2)=I+I
      DO 40 J=1,51
      CALL RPLOTF(ZM1,NVOUT,IERR)
      IF(IERR.EQ.1) GO TO 50
      CALL STRPLT(SM,ZM1,NS,NVPLOT,2,NVOUT)
      DO 35 K=1,3
35    NS(K)=NS(K)+1
40    CONTINUE
      WRITE 107,ZM1(NVPLOT(1)),I,ZM1(NVPLOT(2))
107   FORMAT("0FINAL RMS ERRORS : TRUE = ",1PE15.7/"  (STATE",I3,
     1 ")",4X,"COMPUTED = ",1PE15.7)
      CALL PLOTLP(51,2,SM,-1,1,0,KLIST,ITITLE)
      WRITE(KLIST,106) I
106   FORMAT("0    STATE : ",I2//4X,"SYMBOL 1 : TRUE ERROR"/
     1 4X,"SYMBOL 2 : COMPUTED ERROR "/)
50    CONTINUE
      RETURN
104   FORMAT(" +",10("-")," ENTER TITLE IN GIVEN FIELD ",10("-"),"+"/)
102   FORMAT(5A10)
C END SUBROUTINE FEVAL
      END




      SUBROUTINE DACOV(PCA,PC,ZM1,ZM2,NA,NVOUT,TIME)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPC,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      DIMENSION PCA(1),PC(1),ZM1(1),ZM2(1)
      NDIM=NA
      NDIM1=NDIM+1
      CALL TFRMTX(TM(LTDT),ZM1,NND,NNT,2)
      IF(NDD.LT.1) GO TO 5
      IA=LADDR(NA,NND+1,1)
      CALL TFRMTX(TM(LTNT),ZM1(IA),NDD,NNT,2)
5     CALL SCALE(ZM1,ZM1,NPLD,NNT,-1.)
```

```
      IA=LADDR(NA,1,NNT+1)
      CALL IDNT(NPLD,ZM1(IA),1.)
      CALL MAT3(NPLD,NA,ZM1,PCA,ZM2)
      WRITE(KLIST,101) TIME
  101 FORMAT("0'TRUE' DESIGN ERROR COVARIANCE AT TIME = ",F6.4)
      CALL MATIO(ZM2,NPLD,NPLD,3)
      IA=1
      DO 10 I=1,NPLD
      NS=I+I
      ZM1(NS-1)=SQRT(ZM2(IA))
      ZM1(NS)=PC(I)
   10 IA=IA+NDIM1
      ZM1(NVOUT)=TIME
      CALL WPLOTF(ZM1,NVOUT)
      RETURN
C END SUBROUTINE DACOV
      END




      SUBROUTINE ACOVUD(PC,QD,RKRKT,PHIT,PHI,RIMKH,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LINT
      COMMON/TFUMTX/NVTM,TM(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CKF/NVFLT,FLT(1)
      DIMENSION PC(1),QD(1),RKRKT(1),PHIT(1),PHI(1),RIMKH(1),
     1 ZM1(1),ZM2(1)
      L1=LADDR(NDIM,1,NNT+1)
      CALL ZPART(ZM2(L1),NNT,NPLD,NDIM)
      CALL TFRMTX(PHIT,ZM2,NNT,NNT,2)
      L1=LADDR(NDIM,NNT+1,NNT+1)
      CALL TFRMTX(PHI,ZM2(L1),NPLD,NPLD,2)
      L2=LADDR(NDIM,NNT+1,1)
      CALL ZPART(ZM2(L2),NPLD,NNT,NDIM)
      CALL MAT3(NDIM,NDIM,ZM2,PC,ZM1)
      CALL FPADD(ZM1,NDIM,QD,NNT,NNT,1,PC)
      CALL IDNT(NNT,ZM2,1.)
      CALL FMMUL(FLT(LFLTRK),TM(LHT),NPLD,NMD,NNT,ZM1)
      CALL TFRMTX(ZM1,ZM2(L2),NPLD,NNT,2)
      CALL TFRMTX(RIMKH,ZM2(L1),NPLD,NPLD,2)
      CALL MAT3(NDIM,NDIM,ZM2,PC,ZM1)
      CALL FPADD(ZM1,NDIM,RKRKT,NPLD,NPLD,L1,PC)
      RETURN
C END SUBROUTINE ACOVUD
      END
      SUBROUTINE FPADD(X,NX,Y,NRY,NCY,LADDR,Z)
      DIMENSION X(1),Z(1),Y(NRY,NCY)
      CALL FTMTX(X,Z,NX,NX)
      LAM1=LADDR-1
      DO 10 I=1,NCY
      LA1=LAM1+NX*(I-1)
      DO 10 J=1,NRY
      LA1=LA1+1
   10 Z(LA1)=Z(LA1)+Y(J,I)
      RETURN
C END SUBROUTINE FPADD
      END
      SUBROUTINE RSYS(A,L,ND,ITYPE,IWRT)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      DIMENSION A(1),L(1),ND(1),NAD(14,2),IND(7,3),NTYP(2,3),NTITLE(3),
     1 NMAT(14,3)
      DATA NTYP/7,14,3,4,4,8/
      DATA NO/1HN/
      DATA IND/1HN,1HF,1HP,1HM,1HD,1HW,2HWD,2HNM,2HRM,2HPM,4(1H ),
     1 2HNT,2HRT,2HMT,2HWT,1H ,1H ,1H /
      DATA NTITLE/6HDESIGN,7HCOMMAND,5HTRUTH/
      DATA NMAT/1HA,1HB,2HEX,1HG,1HQ,1HC,2HDY,2HEY,1HH,2HHN,1HR,2HAN,
     1 2HGN,2HQN,2HAM,2HBM,2HCM,2HDM,10(1H ),2HAT,2HBT,2HGT,2HQT,2HHT,
     2 2HRT,3HTDT,3HTNT,6(1H )/
      NDM=NTYP(1,ITYPE)
      NAR=NTYP(2,ITYPE)
      NT=NTITLE(ITYPE)
      WRITE(KLIST,110) NT
110   FORMAT(////11X,5("* "),A7," MODEL",5(" *")////)
5     WRITE 101,NT
101   FORMAT(" READ ",A7," MODEL FROM 'DATA' FILE (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 10
      IF=1
      CALL READFS(A,ND,ITYPE,IERR)
      IF(IERR.EQ.0) GO TO 201
10    WRITE 102,NT
102   FORMAT(" ENTER ",A7," MODEL FROM TERMINAL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 15
      IF=2
      DO 12 I=1,NDM
      WRITE 112,IND(I,ITYPE)
112   FORMAT(" ENTER ",A2," >")
12    READ*,ND(I)
      GO TO 201
15    IF=3
      IF(ITYPE-2) 16,17,18
16    CALL DSND(ND)
      GO TO 20
17    CALL CMDD(ND)
      GO TO 20
18    CALL TRTHD(ND)
20    IF(ND(1).GT.0) GO TO 201
      WRITE 114,NT
114   FORMAT(" ",A7," MODEL SUBROUTINE NON-EXISTENT")
      GO TO 5
201   IF(ITYPE-2) 21,22,23
21    CALL DSNDM(ND,NAD)
      GO TO 25
22    CALL CMDDM(ND,NAD)
```

```
      GO TO 25
23    CALL TRTHDM(NC,NAD)
25    IF(LABORT.EQ.0) GO TO 26
      WRITE 103,NT,LABORT
103   FORMAT("0INSUFFICIENT MEMORY FOR ",A7," MODEL, NEED: ",I4)
      RETURN
26    L(1)=1
      DO 30 I=2,NAR
30    L(I)=L(I-1)+NAD(I-1,1)*NAD(I-1,2)
      NPNTS=L(NAR)+NAD(NAR,1)*NAD(NAR,2)-1
      IF(NPNTS.LE.NVSM) GO TO 34
      WRITE 104,NPNTS
104   FORMAT("0INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
      LABORT=NPNTS
      RETURN
34    IF(IF-2) 75,35,50
35    IZ=1
      DO 40 I=1,NAR
      N1=NAD(I,1)
      N2=NAD(I,2)
      IF((N1.EQ.0).OR.(N2.EQ.0)) GO TO 40
      WRITE 113,NMAT(I,ITYPE)
113   FORMAT("0ENTER ",A3)
      CALL ZMATIN(A(L(I)),N1,N2,IZ)
40    CONTINUE
      GO TO 75
50    CALL ZPART(A,1,NPNTS,1)
      IF(ITYPE-2) 55,60,65
55    CALL DSNM(A(L(1)),A(L(2)),A(L(3)),A(L(4)),A(L(5)),A(L(6)),A(L(7)),
     1 A(L(8)),A(L(9)),A(L(10)),A(L(11)),A(L(12)),A(L(13)),A(L(14)))
      GO TO 75
60    CALL CMDM(A(L(1)),A(L(2)),A(L(3)),A(L(4)))
      GO TO 75
65    CALL TRTHM(A(L(1)),A(L(2)),A(L(3)),A(L(4)),A(L(5)),A(L(6)),
     1 A(L(7)),A(L(8)))
75    IZ=0
77    WRITE 105
105   FORMAT("0MODIFY MATRIX ELEMENTS (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 90
      WRITE 106,(NMAT(I,ITYPE),I=1,NAR)
106   FORMAT(1X,14(2X,A3))
78    WRITE 107
107   FORMAT(" ENTER MATRIX NAME >")
      READ 111,IANS
      DO 80 I=1,NAR
      IF(IANS.EQ.NMAT(I,ITYPE)) GO TO 81
80    CONTINUE
      GO TO 78
81    WRITE 116
116   FORMAT(" LIST MATRIX TO TERMINAL (Y OR N) >")
```

```
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 83
      CALL MATLST(A(L(I)),NAD(I,1),NAD(I,2),NMAT(I,ITYPE),KTERM)
 83   CALL ZMATIN(A(L(I)),NAD(I,1),NAD(I,2),IZ)
      GO TO 77
 90   IF(IWRT) 95,92,93
 92   IWRT=1
 93   WRITE 115,NT
 115  FORMAT("CWRITE ",A7," MODEL TO 'SAVE' FILE (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 95
      CALL WFILED(ITYPE,NPNTS,ND,A)
      IWRT=-1
      WRITE 109,NT
 109  FORMAT(6X,A7," MODEL WRITTEN TO 'SAVE' FILE")
 95   DO 100 I=1,NAR
      N1=NAD(I,1)
      N2=NAD(I,2)
      IF((N1.EQ.0).OR.(N2.EQ.0)) GO TO 100
      CALL MATLST(A(L(I)),N1,N2,NMAT(I,ITYPE),KLIST)
 100  CONTINUE
 111  FORMAT(A3)
      RETURN
C END SUBROUTINE RSYS
      END




      SUBROUTINE DSND(ND)
      DIMENSION ND(1)
      ND(1)=0
      RETURN
C END SUBROUTINE DSND
      END
      SUBROUTINE CMOD(ND)
      DIMENSION ND(1)
      ND(1)=0
      RETURN
C END SUBROUTINE CMOD
      END
```

```
      SUBROUTINE TRTHD(ND)
      DIMENSION ND(1)
      ND(1)=0
      RETURN
C END SUBROUTINE TRTHD
      END
```

```
      SUBROUTINE DSNM(A,B,EX,G,Q,C,DY,EY,H,HD,R,AD,GD,QD)
      RETURN
C END SUBROUTINE DSNM
      END
```

```
      SUBROUTINE CMDM(AM,BM,CM,DM)
      RETURN
C END SUBROUTINE CMDM
      END
      SUBROUTINE TRTHM(AT,BT,GT,QT,HT,RT,TDT,TNT)
      RETURN
C END SUBROUTINE TRTHM
      END



      SUBROUTINE DSNDM(ND,NAD)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/NDIMD/NND,NPD,NPC,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      DIMENSION ND(1),NAD(14,2)
      NND=ND(1)
      NRD=ND(2)
      NPD=ND(3)
      NMD=ND(4)
      NDD=ND(5)
      NWD=ND(6)
      NWDD=ND(7)
      NPLD=NND+NDD
      NWPNWD=NWD+NWDD
      NNPR=NND+NRD
      NAD(1,1)=NND
      NAD(2,1)=NND
      NAD(3,1)=NND
      NAD(4,1)=NND
      NAD(5,1)=NWD
      NAD(6,1)=NPD
      NAD(7,1)=NPD
      NAD(8,1)=NPD
      NAD(9,1)=NMD
      NAD(10,1)=NMD
      NAD(11,1)=NMD
      NAD(12,1)=NDD
      NAD(13,1)=NDD
      NAD(14,1)=NWDD
      NAD(1,2)=NND
      NAD(2,2)=NRD
      NAD(3,2)=NDD
      NAD(4,2)=NWD
      NAD(5,2)=NWD
      NAD(6,2)=NND
      NAD(7,2)=NRD
      NAD(8,2)=NDD
      NAD(9,2)=NND
      NAD(10,2)=NDD
      NAD(11,2)=NMD
      NAD(12,2)=NDD
      NAD(13,2)=NWDD
      NAD(14,2)=NWDD
      NSIZE=NPLD*(2*NPLD+NND+NMD+NPD+NWPNWD)+NRD*(NND+NPD)+
     1 NDD*NDD+NMD*NMD+NWD*NWD+NWDD*NWDD
      IF(NSIZE.GT.NVDM) LABORT=NSIZE
      RETURN
C END SUBROUTINE DSNDM
      END




      SUBROUTINE CMDDM(ND,NAD)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      DIMENSION ND(1),NAD(14,2)
      NNC=ND(1)
      NRC=ND(2)
      NPC=ND(3)
      NAD(1,1)=NNC
      NAD(2,1)=NNC
      NAD(3,1)=NPC
      NAD(4,1)=NPC
      NAD(1,2)=NNC
      NAD(2,2)=NRC
      NAD(3,2)=NNC
      NAD(4,2)=NRC
      NSIZE=NNC*(NNC+NRC+NPC)+NPC*NRC
      IF(NSIZE.GT.NVCM) LABORT=NSIZE
      RETURN
C END SUBROUTINE CMDDM
      END
      SUBROUTINE TRTHOM(ND,NAD)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWCD,NPLD,NWPNWD,NNPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/TPUMTX/NVTM,TM(1)
      DIMENSION ND(1),NAD(14,2)
      NNT=ND(1)
      NRT=ND(2)
      NMT=ND(3)
      NWT=ND(4)
      NAD(1,1)=NNT
      NAD(2,1)=NNT
      NAD(3,1)=NNT
      NAD(4,1)=NWT
      NAD(5,1)=NMT
      NAD(6,1)=NMT
      NAD(7,1)=NND
      NAD(8,1)=NDD
      NAD(1,2)=NNT
      NAD(2,2)=NRT
      NAD(3,2)=NWT
      NAD(4,2)=NWT
      NAD(5,2)=NNT
      NAD(6,2)=NMT
      NAD(7,2)=NNT
      NAD(8,2)=NNT
      NSIZE=NNT*(2*NNT+NMD+NRD+NPLD)+NMD*NMD
      IF(NSIZE.GT.NVTM) LABORT=NSIZE
      RETURN
C END SUBROUTINE TRTHOM
      END




      SUBROUTINE ZMATIN(A,NR,NC,IZ)
      DIMENSION A(NR,NC)
      IF(IZ) 10,10,1
 1    DO 5 I=1,NR
      DO 5 J=1,NC
 5    A(I,J)=0.
 10   WRITE 101,NR,NC
 15   READ*,I,J,V
      IF(I.EQ.0) RETURN
      IF((I.LE.NR).AND.(J.LE.NC)) GO TO 20
      WRITE 102
      GO TO 15
 20   A(I,J)=V
      IF(IZ.LT.0) A(J,I)=V
      GO TO 15
 101  FORMAT(" ENTER I,J AND M(I,J)--(0/ WHEN COMPLETE) : "I2," BY "I2,
     1 " >")
 102  FORMAT(" ERROR IN ARRAY INDEX")
C END SUBROUTINE ZMATIN
      END




      SUBROUTINE WFILED(NT,NP,ND,A)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION ND(10),A(NP)
      DATA IEOI/-1/
      BACKSPACE KSAVE
      WRITE(KSAVE,101) NT,NP
      WRITE(KSAVE,101) (ND(I),I=1,10)
      WRITE(KSAVE,102) (A(I),I=1,NP)
      WRITE(KSAVE,101) IEOI,NP
      RETURN
 101  FORMAT(2I4)
 102  FORMAT(E20.10)
C END SUBROUTINE WFILED
      END




      SUBROUTINE READFS(A,ND,NT,IERR)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION A(1),ND(10)
      DATA IEOI/-1/
      REWIND KDATA
 5    READ(KDATA,102) IT,NP
      IF(IT.NE.IEOI) GO TO 10
      WRITE 101
      IERR=1
      RETURN
 10   CALL FARRAY(A,ND,NP)
      IF(IT.NE.NT) GO TO 5
      IERR=0
 101  FORMAT("0DATA NOT IN 'DATA' FILE . . .")
 102  FORMAT(2I4)
      RETURN
C END SUBROUTINE READFS
      END
      SUBROUTINE FARRAY(A,ND,NP)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION A(NP),ND(10)
      READ(KDATA,101) (ND(I),I=1,10)
      READ(KDATA,102) (A(I),I=1,NP)
      RETURN
  101 FORMAT(2I4)
  102 FORMAT(E20.10)
C END SUBROUTINE FARRAY
      END




      SUBROUTINE TERMTX(X1,X2,NR,NC,ITX)
      COMMON/MAIN1/NDIM
      DIMENSION X1(1),X2(1)
      IF(ITX.EQ.2) GO TO 20
      J=NC*NDIM
      KK=0
      DO 10 I=1,J,NDIM
      L=I+NR-1
      DO 10 JJ=I,L
      KK=KK+1
   10 X1(KK)=X2(JJ)
      RETURN
   20 KK=NR*NC+1
      DO 30 I=1,NC
      L=(NC-I)*NDIM+1
      DO 30 J=1,NR
      KK=KK-1
      JJ=L+NR-J
   30 X2(JJ)=X1(KK)
      RETURN
      END
      SUBROUTINE MATLST(A,NR,NC,NT,KDEV)
      DIMENSION A(NR,NC)
      WRITE(KDEV,101) NT
      DO 10 I=1,NR
 10   WRITE(KDEV,102) (A(I,J),J=1,NC)
 101  FORMAT(1H0,A3," MATRIX"/)
 102  FORMAT(1X1P10G13.4)
      RETURN
C END SUBROUTINE MATLST
      END




      SUBROUTINE NDSCRT(A,N,NTERMS)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      DIMENSION A(1)
      NTERMS=IFIX(6.+3.*TSAMP*XNORM(N,A))
      IF(NTERMS.GT.30) NTERMS=30
      RETURN
C END SUBROUTINE NDSCRT
      END




      SUBROUTINE RQWGTS(W,ND,NP)
      DIMENSION W(1)
 10   WRITE 101
 101  FORMAT(" ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >")
 15   READ*,I,V
      IF(I.EQ.0) RETURN
      IF(I.LE.ND) GO TO 20
      WRITE 102
 102  FORMAT(" ERROR IN ARRAY INDEX")
      GO TO 10
 20   IF(V) 25,30,40
 25   WRITE 103
 103  FORMAT(" ELEMENTS MUST BE NON-NEGATIVE")
      GO TO 15
 30   IF(NP) 35,40,35
 35   WRITE 104
 104  FORMAT(" ELEMENTS MUST BE POSITIVE")
      GO TO 15
 40   L1=LADDR(ND,I,I)
      W(L1)=V
      GO TO 15
C END SUBROUTINE RQWGTS
      END
      SUBROUTINE DVCTOR(N,A,V)
      DIMENSION A(1),V(1)
      NP1=N+1
      N2=N*N
      J=0
      DO 10 I=1,N2,NP1
      J=J+1
 10   V(J)=A(I)
      RETURN
C END SUBROUTINE DVCTOR
      END
```

```
      SUBROUTINE POLES(A,N,ITYPE,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION NTYP(5),A(1),ZM1(1),ZM2(1)
      DATA NTYP/6HDESIGN,7HCOMMAND,5HTRUTH,5HREGPI,6HFILTER/
      NDS=NDIM
      NDIM=N
      NDIM1=NDIM+1
      CALL EIGEN(NDIM,A,ZM1,ZM1(NDIM1),ZM2,0)
      IF(ITYPE.LT.4) GO TO 10
      CALL MAPOLE(N,ZM1,ZM1(NDIM1),TSAMP)
 10   WRITE(KLIST,102) NTYP(ITYPE)
      WRITE(KTERM,102) NTYP(ITYPE)
      WRITE(KLIST,101) (ZM1(I),ZM1(NDIM+I),I=1,N)
      WRITE(KTERM,101) (ZM1(I),ZM1(NDIM+I),I=1,N)
      NDIM=NDS
      NDIM1=NDIM+1
 101  FORMAT(6X,1PE15.7,"  +J(",1PE15.7,")")
 102  FORMAT("0POLES OF ",A7," MATRIX"/)
      RETURN
C END SUBROUTINE POLES
      END
      SUBROUTINE MAPOLE(N,ZR,ZI,T)
      DIMENSION ZR(1),ZI(1)
      RT=1./T
      DO 10 I=1,N
      ZM=SQRT(ZR(I)**2+ZI(I)**2)
      SIGMA=RT*ALOG(ZM)
      ZI(I)=RT*ATAN2(ZI(I),ZR(I))
   10 ZR(I)=SIGMA
      RETURN
C END SUBROUTINE MAPOLE
      END
```

```
      FUNCTION LADDR(NR,I,J)
      LADDR=I+NR*(J-1)
      RETURN
C END FUNCTION LADDR
      END
```

```
      SUBROUTINE FTMTX(X,Y,NR,NC)
      DIMENSION X(1),Y(1)
      NE=NR*NC
      DO 10 I=1,NE
   10 Y(I)=X(I)
      RETURN
C END SUBROUTINE FTMTX
      END
      SUBROUTINE FMMUL(X,Y,NR1,NC1,NC2,Z)
      DIMENSION X(NR1,NC1),Y(NC1,NC2),Z(NR1,NC2)
      DOUBLE PRECISION TD
      DO 10 I=1,NR1
      DO 10 J=1,NC2
      TD=0.D0
      DO 5 K=1,NC1
 5    TD=TD+X(I,K)*Y(K,J)
 10   Z(I,J)=TD
      RETURN
C END SUBROUTINE FMMUL
      END
```

```
      SUBROUTINE FTMUL(X,Y,NR1,NC1,NC2,Z)
      DIMENSION X(NR1,NC1),Y(NR1,NC2),Z(NC1,NC2)
      DOUBLE PRECISION TD
      DO 10 I=1,NC1
      DO 10 J=1,NC2
      TD=0.D0
      DO 5 K=1,NR1
 5    TD=TD+X(K,I)*Y(K,J)
 10   Z(I,J)=TD
      RETURN
C END SUBROUTINE FTMUL
      END
```

```
      SUBROUTINE FMADD(X,Y,NR,NC,Z)
      DIMENSION X(1),Y(1),Z(1)
      NE=NR*NC
      DO 10 I=1,NE
 10   Z(I)=X(I)+Y(I)
      RETURN
C END SUBROUTINE FMADD
      END
      SUBROUTINE ZPART(A,NR,NC,ND)
      DIMENSION A(1)
      NE=NC*ND
      DO 10 I=1,NR
      DO 10 J=I,NE,ND
 10   A(J)=0.
      RETURN
C END SUBROUTINE ZPART
      END
```

```
      SUBROUTINE SUBI(A,NR,ND)
      DIMENSION A(1)
      ND1=ND+1
      NE=NR*ND
      DO 10 I=1,NE,ND1
 10   A(I)=A(I)-1.
      RETURN
C END SUBROUTINE SUBI
      END
```

```
      SUBROUTINE WPLOTF(V,N)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION V(N)
      WRITE(KPLOT,101) (V(I),I=1,N)
      RETURN
 101  FORMAT(E20.10)
C END SUBROUTINE WPLOTF
      END
      SUBROUTINE RPLOTF(V,N,IERR)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION V(N)
      READ(KPLOT,101) (V(I),I=1,N)
      IF(EOF(KPLOT)) 5,10
 5    IERR=1
      RETURN
 10   IERR=0
      RETURN
 101  FORMAT(E20.10)
C END SUBROUTINE RPLOTF
      END



      SUBROUTINE STRPLT(A,V,NS,NV,NP,NVO)
      DIMENSION A(1),NS(1),NV(1),V(1)
      A(NS(1))=V(NVO)
      DO 5 I=1,NP
 5    A(NS(I+1))=V(NV(I))
      RETURN
C END SUBROUTINE STRPLT
      END



      SUBROUTINE PLOTLP(N,M,A,IPSC,ISCL,LPTERM,NDEV,ITITLE)
C * * * * * * * * * * *
C * N = NUMBER OF POINTS TO BE PLOTTED
C * M = NUMBER OF OUTPUTS TO BE PLOTTED
C * A = VECTOR OF SAMPLE POINTS FOR PLOTTING : DIMENSION = N*M
C *    ELEMENTS 1 TO N ARE THE INDEPENDENT VARIABLE
C *    ELEMENTS (N+1) TO 2*N, (2*N+1) TO 3*N, AND SO ON ARE
C *    THE DEPENDENT VARIABLES--EACH VARIABLE IS IN CONSECUTIVE
C *    STORAGE WITH CORRESPONDING SAMPLE POINTS FOR EACH
C *    SEPARATED BY MULTIPLES OF N
C * IPSC = -1 => SCALE ALL VARIABLES TOGETHER (1 PLOT)
C *      =  0 => SCALE TOGETHER AND SEPARATELY (2 PLOTS)
C *      = +1 => SCALE SEPARATELY (1 PLOT)
C * ISCL = 0 => PLOT OVER EXACT RANGE OF VARIABLE
C *      = 1 => PLOT USING EVEN SCALING
C * LPTERM = 0 => PLOT IS TO TERMINAL (50 CHARACTERS WIDE)
C *        = 1 => PLOT IS TO LINE PRINTER (100 CHARACTERS WIDE)
C * NDEV = DEVICE NUMBER FOR PLOT OUTPUT
C * ITITLE = VECTOR(DIMENSIONED 5) WITH 50 CHARACTER TITLE
C * * * * * * * * * * *
      DIMENSION YSCAL(6),YMIN(6),IBLNK(6),YPR(11),A(1)
      INTEGER OUT(131),SYMBOL(6),BLANK,PLUS,GRID,ITITLE(5)
      DATA BLANK,PLUS,COLON,SYMBOL/1H ,1H+,1H:,1H1,1H2,1H3,1H4,1H5,1H6/
1     FORMAT(1H )
2     FORMAT(1H1,11X,5A10/)
10    FORMAT(1H ,F11.2,6X,101A1)
12    FORMAT(11H0    SCALE ,A1,1X,11F10.4)
      IPAPER=5*(1+LPTERM)
      ISPAC=10*IPAPER
      RISPAC=FLOAT(ISPAC)
      ISPAC=ISPAC+1
      IPRT1=IPAPER+1
      RMIN=A(N+1)
      RMAX=RMIN
25    DO 41 ISC=1,M
      M1=ISC*N+1
      YL=A(M1)
      YH=YL
      M2=N*(ISC+1)
      DO 40 J=M1,M2
      IF(A(J).LT.YL) GO TO 39
      IF(A(J).GT.YH) YH=A(J)
      GO TO 40
39    YL=A(J)
40    CONTINUE
      IF(YL.LT.RMIN)RMIN=YL
      IF(YH.GT.RMAX) RMAX=YH
      IF(IPSC.GE.0) CALL VARSCL(YL,YH,YSCAL(ISC),RISPAC,ISCL)
41    YMIN(ISC)=YL
      IF(IPSC.LE.0)CALL VARSCL(RMIN,RMAX,SCAL,RISPAC,ISCL)
      IC=2-IABS(IPSC)
      DO 45 IX=1,ISPAC
45    OUT(IX)=BLANK
      DO 100 ICO=1,IC
      WRITE(NDEV,2) (ITITLE(I),I=1,5)
      DO 60 I=1,N
      XPR=A(I)
      IF(MOD(I,10).EQ.0) GO TO 456
      GRID=BLANK
      GO TO 460
456   GRID=COLON
460   DO 461 IX=2,ISPAC,2
461   OUT(IX)=GRID
      DO 46 IX=1,ISPAC,10
46    OUT(IX)=PLUS
      DO 55 J=1,M
      IL=I+J*N
      IF(IPSC) 48,47,49
47    IPSCT=IPSC+ICO
      IF(IPSCT.EQ.2) GO TO 49
48    JP=IFIX((A(IL)-RMIN)/SCAL)+1
      GO TO 50
```

```
 49    JP=IFIX((A(IL)-YMIN(J))/YSCAL(J))+1
 50    OUT(JP)=SYMBOL(J)
 55    IBLNK(J)=JP
       WRITE(NDEV,10) XPR,(OUT(IX),IX=1,ISPAC)
       DO 59 J=1,M
       ITEMP=IBLNK(J)
 59    OUT(ITEMP)=BLANK
 60    CONTINUE
       IF(IPSC) 68,67,72
 67    IF(IPSCT.EQ.2) GO TO 72
 68    YPR(1)=RMIN
       DO 70 I=1,IPAPER
 70    YPR(I+1)=YPR(I)+10.*SCAL
       WRITE(NDEV,12) BLANK,(YPR(I),I=1,IPRT1)
       GO TO 100
 72    DO 76 ISC=1,M
       YPR(1)=YMIN(ISC)
       DO 74 I=1,IPAPER
 74    YPR(I+1)=YPR(I)+10.*YSCAL(ISC)
 76    WRITE(NDEV,12) SYMBOL(ISC),(YPR(IX),IX=1,IPRT1)
 100   WRITE(NDEV,1)
       RETURN
C END SUBROUTINE PLOTLP
       END




       SUBROUTINE VARSCL(XMIN,XMAX,SCALE,RSPACE,ISCL)
       IF(XMAX.EQ.XMIN) XMIN=.9*XMIN-10.
       SCALE=XMAX-XMIN
       IF(ISCL.EQ.0) GO TO 25
       EXP=IFIX(100.+ALOG10(SCALE))-100.
       FACTOR=10.**(1.-EXP)
       XMINT=XMIN*FACTOR
       XMAXT=XMAX*FACTOR
       IF(XMAXT.GE.0.) XMAXT=XMAXT+.9
       IF(XMINT.LE.0.) XMINT=XMINT-.9
       XMINT=AINT(XMINT)
       ISCAL=XMAXT-XMINT
       IF(MOD(ISCAL,5).NE.0) ISCAL=ISCAL+5-MOD(ISCAL,5)
       FACTOR=10.**(EXP-1.)
       XMIN=XMINT*FACTOR
       SCALE=FACTOR*FLOAT(ISCAL)
 25    SCALE=SCALE/RSPACE
       RETURN
C END SUBROUTINE VARSCL
       END
```

Appendix E

CGTPIF Segmentation Job Control

The following listing shows the job control commands and segmentation directives used in obtaining a segmented object file suitable for interactive execution on the CDC CYBER computer system. The job employs three object files: "L", "S", and "A". The routines on each of these files are (see the program description and listing of Appendices A and D, respectively),

"L": 'MAIN' and all optional routines ('MAIN' through 'TBLUP1')

"S": 'CGTPIF SUBS' ('CGTXQ' through 'VARSCL')

"A": 'LIBRARY'

Object files L and S are loaded into memory in order of MAIN then CGTPIF SUBS. The "NOGO" command then completes the load from LIBRARY and system routines in order, but does not initiate execution. Next, the segmentation directives are executed (segmentation directives appear between the pair of "*EOR" lines). When segmentation is complete, the resulting object file is cataloged.

In this listing, the names given the various object files ("L", "S", "A") are arbitrary, and the "ATTACH" commands may occur in any order. The file name ("lfn") given in the "LIBRARY, lfn" command must correspond to the name used in attaching the object file of 'LIBRARY' routines ("A" in this case). The name given to the segmented object file is arbitrary ("CGTPIF" in this case) but must be consistent in the "REQUEST", "SEGLOAD", and "CATALOG" commands. The segmentation directives should not be modified in any way.

As done in this case, it is convenient to maintain distinct object files for each of the three sets of routines. Thus both 'CGTPIF SUBS' and 'LIBRARY' remain invariant in object and LIBRARY object files, respectively. The routine 'MAIN' and any desired user-provided optional routines may then be developed as an independent set, and compiled to obtain the needed object file. Descriptions of 'MAIN' and optional routines are given in the "Programmer's Guide" (Appendix A).

```
RMF. D790477,FLOYD
MAP,FULL.
ATTACH,L,SEGMENT,CY=10.
ATTACH,S,FLOYDL1,CY=10.
ATTACH,A,FLOYDL1,CY=1.
LIBRARY,A.
REQUEST,CGTPIF,*PF.
SEGLOAD(B=CGTPIF)
LOAD(L,S)
NOGO.
CATALOG,CGTPIF,THESIS,CY=100,RP=180,PW=RMF.
*EOR
SETUP    INCLUDE DSCRT
SREGPI   INCLUDE RQWGTS,MLINEQ,FACTOR
FLTRK    INCLUDE RQWGTS,KFLTR,MLINEQ,FACTOR,INTEG
STRTH    INCLUDE DSCRTT,INTEG
SDSN     INCLUDE QDSCRT
CEVAL    INCLUDE PLOTLP,VARSCL,RPLOTF,WPLOTF,STRPLT
FEVAL    INCLUDE PLOTLP,VARSCL,RPLOTF,WPLOTF,STRPLT,DACOV
B1       TREE SETUP-(SDSN,SCMD,STRTH)
B2       TREE PIMTX
B3       TREE SREGPI
B4       TREE SCGT
B5       TREE CEVAL
B6       TREE FLTRK
B7       TREE FEVAL
A        TREE CGTXQ-(B1,B2,B3,B4,B5,B6,B7)
ROOT     TREE MAIN-A
         GLOBAL MAIN1,MAIN2,INOU,DESIGN,FILES,SYSMTX,ZMTX1,ZMTX2,
,NDIMD,LOCD,DSNMTX,NDIMC,LOCC,CMDMTX,NDIMT,LOCT,TRUMTX,LCNTRL,CONTROL,
,LREGPI,CREGPI,LCGT,CCGT,LKF,CKF
         END
*EOR
*EOF
..
```

Bibliography

1. Asseo, S. J. "Application of Optimal Control to Perfect Model Following," Journal of Aircraft, 7: 308-313 (1970).

2. Athans, M. "The Role and Use of the Stochastic Linear-Quadratic-Gaussian Problem in Control System Design," IEEE Trans. Automatic Control, AC-16:529-551 (1971).

3. Barfield, A. F. Aircraft Control Engineer. Unpublished AFTI/F-16 Linear Aerodynamic Data. Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, October 1980.

4. Barraud, A. Y. "A Numerical Algorithm to Solve $A^TXA-X=Q$," IEEE Trans. Automatic Control, AC-22:883-885.

5. Berry, P. W., J. R. Broussard, and S. Gully. "Stability and Control Analysis of V/STOL Type B Aircraft," ONR-CR213-162-1F. Office of Naval Research, Arlington, Virginia, March 1979.

6. Berry, P. W., J. R. Broussard, and S. Gully. "Validation of High Angle-of-Attack Analysis Methods," ONR-CR215-237-3F. Office of Naval Research, Arlington, Virginia, September 1979.

7. Bristol, E. H. "Designing and Programming Control Algorithms for DDC Systems," Control Engineering, 24:24-26 (1977).

8. Broussard, J. R. "Command Generator Tracking," TASC TIM-612-3, The Analytic Sciences Corp., Reading, Masachusetts, March 1978.

9. Broussard, J. R., P. W. Berry, and R. F. Stengel. "Modern Digital Flight Control System Design for VTOL Aircraft," NASA CR-159019. National Aeronautics and Space Administration, Hampton, Virginia, March 1979.

10. Broussard, J. R., Engineer. Personal correspondence. Information & Control Systems, Inc., Hampton, Virginia, May 4, 1981.

11. Cadzow, J. A., and H. R. Martens. Discrete-time and Computer Control Systems. Englewood Cliffs, New Jersey: Prentice-Hall, 1970.

12. Chalk, C. R. et al. "Background Information and User Guide for MIL-F-8785B(ASG) Entitled Military Specification--Flying Qualities of Piloted Airplanes," AFFDL TR 69-72, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, August 1969.

13. CYBER Loader Version 1 Reference Manual. Publication number 60429800. Control Data Corporation, Sunnyvale, California, 1979.

14. Davison, E. J., and S. H. Wang. "Properties and Calculation of Transmission Zeros of Linear Multivariable Systems," Automatica, 10:643-658 (1974).

15. Dorato, P., and A. H. Levis. "Optimal Linear Regulators: The Discrete-Time Case," IEEE Trans. Automatic Control, AC-16:613-620 (1971).

16. Elliott, J. R. "NASA's Advanced Control Law Program for the F-8 Digital Fly-By-Wire Aircraft," IEEE Trans. Automatic Control, AC-22:753-757 (1977).

17. Erzberger, H. "On the Use of Algebraic Methods in the Analysis and Design of Model-Following Control Systems," NASA TN-D-4663. National Aeronautics and Space Administration, Washington, D.C., July 1968.

18. Floyd, R. M. Aircraft Control Engineer. Unpublished AFTI/F-16 Aerodynamic Stability Derivative Data. Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, June 1980.

19. Fortmann, T. E., and K. L. Hitz. An Introduction to Linear Control Systems. New York: Marcel Dekker, Inc., 1977.

20. Gran, R., H. Berman, and M. Rossi. "Optimal Digital Flight Control for Advanced Fighter Aircraft," Journal of Aircraft, 14:32-37 (1977).

21. Heath, R. E. "State Variable Model of Wind Gusts," AFFDL/FGC-TM-72-12, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, July 1972.

22. Kalman, R. E., T. S. Englar, and R. S. Bucy. Fundamental Study of Adaptive Control Systems, ASD-TR-61-77, Wright-Patterson AFB, Ohio, 1961.

23. Kalman, R. E., and T. S. Englar. "User's Manual for the Automatic Synthesis Program," NASA CR-475. National Aeronautics and Space Administration, Washington, D.C., June 1966.

24. Kleinman, D. L. "A Description of Computer Programs Useful in Linear Systems Studies," Tec. Rep. TR-75-4. University of Connecticut, Storrs, Connecticut, October 1975.

25. Kreindler, E. "On the Linear Optimal Servo Problem," _International Journal of Control_, _9_:465-472 (1969).

26. Kreindler, E., and D. Rothschild. "Model-Following in Linear-Quadratic Optimization," _AIAA Journal_, _14_:835-842 (1976).

27. Kriechbaum, G. K. L., and R. W. Stineman. "Design of Desirable Airplane Handling Qualities via Optimal Control," _Journal of Aircraft_, _9_:365-369 (1972).

28. Kuo, B. C. _Digital Control Systems_. Champaign, Illinois: SRL Publishing Company, 1979.

29. Kwakernaak, H., and R. Sivan. _Linear Optimal Control Systems_. New York: Wiley, 1972.

30. McRuer, D., I. Ashkenas, and D. Graham. _Aircraft Dynamics and Automatic Control_. Princeton, New Jersey: Princeton University Press, 1973.

31. Maybeck, P. S. _Stochastic Models, Estimation, and Control_, _Vol. 1_. New York: Academic Press, 1979.

32. Maybeck, P. S. _Stochastic Models, Estimation, and Control_, _Vol. 2_. Unpublished manuscript. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1981.

33. Stein, G., and A. H. Henke. "A Design Procedure and Handling-Quality Criteria for Lateral-Directional Flight Control Systems," AFFDL TR-70-152, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, May 1971.

34. Tyler, J. S., Jr. "The Characteristics of Model-Following Systems as Synthesized by Optimal Control," _IEEE Trans. Automatic Control_, _AC-9_:485-498 (1964).

35. Winsor, C. A., and R. J. Roy. "The Application of Specific Optimal Control to the Design of Desensitized Model Following Control Systems," _IEEE Trans. Automatic Control_, _AC-15_:326-333 (1970).

ILMED

8