Interactive Programs

General Guide

Wieslander, Johan

1980

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-3156)/1-30/(1980)

Interactive Programs

-General Guide

JOHAN WIESLANDER

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA
1980

INTERACTIVE PROGRAMS
-
GENERAL GUIDE

JOHAN WIESLANDER

Organization
LUND INSTITUTE OF TECHNOLOGY
Department of Automatic Control
Box 725
S-220 07 Lund 7 SWEDEN

Document name
Project report STU

Date of issue
February 1980

CODEN: LUTFD2/(TFRT-3156)/1-30/(1980)

Author(s)
Johan Wieslander

Sponsoring organization
Swedish Board for Technical Development (STU), contract No 77-3548

Title and subtitle

Interactive Programs - General Guide

A4 A5

Abstract

The report is a general guide to a family of interactive programs, which use a common interactive language. Command interaction is the basic form of communication with the program. The general syntax rules are described here. A short description of the macro facility is included as is a list of general purpose commands available. The standardized file format is finally discussed.

Key words A4 A5

Classification system and/or index terms (if any)

Supplementary bibliographical information

Language
English

ISSN and key title

ISBN

Recipient's notes

Number of pages
30

Price

Security classification

Distribution by (name and address)

CONTENTS

1. INTRODUCTION

This general guide describes a family of interactive programs for data analysis, system identification, system analysis, and system controller design. They are intended to be powerful tools in the hands of the experienced user.

Thus, talking to the beginner that reads this guide for the first time: You will probably find it somewhat akward to get started using these programs. But have hope, it won't take very long until you too are experienced.

The programs usually interact with the user via a graphic terminal, i.e. outputs diagrams and text on a screen, and accepts keyboard input in the form of command_lines. This command method of interaction has many advantages:

a) it is consise,

b) it is predictable, i.e. the user can prepare himself,

c) it allows the user to improvise at any time,

d) it allows macros,

e) it is good for the programmer.

The drawback is that it leaves the beginner without support.

This general guide will give some background on how data is organized and used by the programs and how the 'interactive language' is constructed and interpreted. A deep knowledge on the organization of data is generally not necessary unless the user attempts some non-standard operation, but some background might prove useful. This guide will also give sufficient information on the general aspects of the command language to enable the user to understand and use the appropriate user's guide. The macro facility of the interactive language is however not treated in detail here. The more active user is therefore strongly recommended to study the Intrac language manual [1].

The abilities of the different programs are summarized below. To each program belongs a User's Guide giving information on the available operations and the accompanying command syntaxes. Comments on the related methods as well as hints on the use of the commands are frequently given, but a working knowledge of the practical aspects of the underlying theory is assumed.

The User's Guides available so far are [2], [3], and [4].

Idpac

Idpac is aimed at problems encountered in analysis of time series. Typical sources of the time series would be recorded measurements and control signals from industrial processes. Examples where Idpac has been applied includes paper and pulp industy, power systems, chemical industry and ship building. Non-industrial applications are found e.g. in biomedicine and econometrics.

The types of questions Idpac may answer are:
a) What do the signals look like?
b) What is the size of the disturbancies?
c) Where can we find them?
d) What are their properties?
e) How can the process be described?
f) How does the process react to specific inputs?

In order to answer such questions, Idpac contains modules for data conversion, scaling and calibration. A comprehensive plotting facility is also included in order to visualize the data.

The analysis tools available in Idpac are:
a) Correlation analysis.
b) Spectral analysis (Fourier transform of correlation function as well as DFT).
c) Parametric model fitting (maximum likelihood and least squares method).
d) Normality and independence tests.
e) Simulation

Modpac

Modpac is intended to be an interface between other program packages. This need arises because Idpac e.g. gives as one of its results a system description on transfer function form, while Synpac is aimed at a system description on state space form.

Modpac resolves this conflict by offering a transformation between these two forms of system descriptions. In short, Modpac will serve as a tool in handling different forms of model descriptions. Today Modpac contains the following facilities:

a) Transformation in both directions between continuous time and discrete time state space systems.
b) Transformation in both directions between multi input - single output transfer functions and state space system descriptions.
c) Transformation of state space descriptions to diagonal, balanced and Hessenberg forms.

d) Computation of the frequency response of a state space system representation.
e) Plotting of frequency responses in Bode, Nichols and Nyquist diagrams.
f) Computation of zeroes for scalar polynomials.
g) Definition and handling of polynomials and matrices and evaluation of matrix expressions.

As indicated above, Modpac is intended as a glue between different program packages. It is therefore of course compatible with them, both regarding its user interaction and its data stuctures.

Synpac

Synpac is aimed at the design of control algorithms for multi input - multi output dynamical systems on state space form. Both continuous time and discrete time systems can be handled.

The intended design criteria are the shape and speed of time responses as well as pole/zero configurations and frequency responses. Also the influence of noise on the system may be a design criterion. The main but not only design tool is linear quadratic methods, used both for regulator and observer design. The 'design knobs' are in these cases the assignment of values for the respective quadratic criteria. This design scheme has proved intuitively natural and is easy to learn. It is supported with a function that helps in transforming the criteria matrices to a suitable form. Methods also exist to adapt the resulting design to a suboptimal, but maybe more realistic one.

To achieve these goals, Synpac contains the following facilities:
a) Matrix definition, handling and operations.
b) System definition.
c) Eigenvalue (pole) computations.
d) Frequency response evaluation.
e) Simulation.
f) Graphic output of time- and frequency responses.
g) Solution of the stationary Riccati equation for optimal regulators and observers.
h) Design of reduced order regulators and observers.
i) Generation of deterministic and stochastic test signals.

2. GENERAL DESCRIPTION OF COMMANDS

The interactive programs described here are command driven. The input of a command and the subsequent decoding is done by a special set of subroutines with the collective name Intrac. Intrac is common to several program packages and is fully described in a separate report [1]. The following is a short introduction.

2.1 Command Modes

Normally, a program action is initiated by the user by giving a command. When the program is in normal mode (or command mode), the commands are entered from the keyboard on the user's terminal. A ready sign (>) is output on the left hand margin when the program awaits a new command. In normal mode essentially any command is legal though not necessarily meaningful.

The second program mode is the macro mode. In this case the commands are read from a special macro file, thus allowing the user easy access to commonly used command sequences. A special set of commands, meaningful only in macro mode, allow looping, testing, and jumps to be done.

In the macro mode, a special input command (READ) is available. This command together with the ones previously mentioned, give the possibility of a macro designed to prompt the user to take certain actions or to make certain decisions. In this way the classical 'question & answer' dialogue between user and program may be realised.

Some commands may require a more detailed specification than the one given in the command line. In such a case, a subcommand sequence is entered. This is indicated by the ready sign appearing a few steps to the right of the lefthand margin.

In the subcommand sequence, only a restricted special set of commands depending on the main one, and those implemented within Intrac, are legal. The commands in the subcommand sequence may be entered either in normal mode or in macro mode. The sequence is terminated either with the comand X (execute) or KILL (ignore previous subcommands, perform no action).

2.2 Generic Description of Commands

A command has the generic form shown below:

$$\text{CMND LARG}_1 \ldots \text{LARG}_{NL} < \text{RARG}_1 \ldots \text{RARG}_{NR}$$

The first item is the command name or the command identifier. After that follows the argument list, separated into two parts by the left arrow (<). The left hand arguments and the right hand arguments represent results and inputs of the command respectively. In some commands one of these parts is missing, then the left arrow is also omitted.

In many commands some of the arguments are optional. In the detailed command description found in the User's Guides [2], [3], and [4], this is indicated by enclosing optional arguments in brackets ([]). When the number of arguments is optional, this is indicated with a series of dots (...). Mutually excluding arguments or options are separated by a slash (/). An argument may be replaced by a comma. If so, the corresponding argument in the previous command is used (short-hand feature). In some cases, slashes (/) are used in the command line to delimit file identification flags.

Comments are preceeded by a double quote (") and may end any command line. An empty command line or one containing only a comment is legal.

2.3 Arguments and Variables

The arguments of a command line can be of different types. They may be integer numbers, real numbers, some special characters (+, -, * etc) and Hollerith strings. An argument is recognized as a Hollerith string when its first character is alphabetic, the remaining characters being alphanumeric. One form of a Hollerith string is a flag. In this case a certain value of the argument specifies a special action to be taken. In the detailed description, these values are denoted by quotes (e.g. 'HP'). An unused flag is omitted or the string 'VOID' is used.

Other forms of Hollerith strings are names. The normal use is as a name of a certain data set, e.g. a signal, a frequency response, or a dynamic system. Another instance is as names of variables. Variables may be either local or global. Global variables are described below. Local variables may be used in a macro. Within a macro, names may also be used as formal arguments.

One or more arguments may be enclosed in parentheses. This indicates that they are attributes to the previous argument. Examples are column numbers in a data file or section name in a system file.

2.4 Global Variables

Some variables may be referenced both in command mode and in macro mode. They are called global variables. A reference to a global variable is constructed in the following way:

NAME.[EXT]

Evidently, it consists of a name followed by a dot. Optionally a second name follows as an extension.

The progam package contains a table of values to be associated with these references. These values are of certain types as described in the previous section. Whenever a global variable reference is found in the command line, it is substituted by the corresponding value and type by the Intrac routines. Thus, e.g. a required integer argument in a command line may be replaced by any global variable reference provided that its corresponding value is of integer type.

Values are assigned to the global variables e.g. via the LET-command. Also some commands may deliver results to global variables, see e.g. the command STAT.

2.5 Reserved Global Variables

In order to make some commands shorter and easier to remember, some variables or flags are left out of the command line. They are instead implemented as a set of reserved and pre-defined global variables. These variables and flags are typically problem dependent and are seldom altered during a run. It is therefore good practice to begin a session with one of the programs by assigning suitable values to such global variables. Note that this operation is a typical use of a macro.

The names, meaning, and default values of the reserved global variables are The list is in alphabetical order. Default values are given within parenthesis. When they are installation dependent, only their type is given. Note that not all global variables are included in all programs. Note also that the Intrac command WRITE when used without any arguments will produce a complete list of global variables including their respective values.

AEPS. (real)
is a test quantity used for absolute tests in some commands.

AMP. (1.0)
is the desired amplitude of a signal generated by the commmand INSI.

CEPS. (real)
is a test quantity used for convergence tests. Normally, it is larger than REPS.

DELTA. (1.0)
is the sample interval of the problem in seconds. Continuous time problems have DELTA. = 0.0.

FTEST. (0)
is the result of the file existence test command FTEST.

IFP. (1)
is the sample point number where a signal generated by INSI will start.

INIML. (0)
is a flag used by the command ML indicating whether initial values should be estimated or not.

ITML. (20)
is the iteration limit used by the command ML.

LIML. (0)
is a flag indicating whether to limit the residuals or not, used by the comomand ML.

NITER. (100)
is an iterative limit used by some commands.

NPLX. (100)
is the length of the horizontal axis in the command PLOT.

NOF. (100)
is the number of frequency points to be computed by some commands giving frequency response results.

NU. (integer)
is the state of the random number generator.

PRIML. (0)
is a flag controlling the printed output from the command ML.

PRINT. (0)
is a flag controlling the printing of results by some commands. Generally: PRINT. = 0 means no printout, PRINT. ≠ 0 means that some output will be generated.

PSEPS. (real)
is a test quantity used in pseudo inversions, e.g. in MATOP.

REPS. (real)
is a test quantity used for relative tests in some commands.

SCALES. (1)
is a flag controlling the allowable scales used by PLOT.

TICK. (1.0)
is the time quantum of the clock giving the sample interval. The following must be true: DELTA. = n*TICK. where n is an integer.

WMAX. (100.0)
is an upper limit for the angular frequency used by some commands.

WMIN. (0.01)
is a lower limit for the angular frequency used by some commands.

YMAX. (0.0)
is an optional upper scaling limit used by PLOT.

YMIN. (0.0)
is an optional lower scaling limit used by PLOT.

3. FACILITIES IN INTRAC

As mentioned earlier, the complete facilities of Intrac should be studied in [1]. In addition to the decoding rules and the handling of global variables described in previous sections, there are two other facilities the user should be aware of. One is the macro facility, the other is the Intrac-implemented commands.

3.1 The Macro Facility

The macro facility gives the user the possibility to store a certain series of commands for later and repeated use. This could be of interest in a number of situations.

A. The same sequence of commands is in a given application used several times with only minor modifications. Storing this sequence as a macro effectively defines a new special purpose command with greater efficiency and ease of use as a result.

B. Defining problem dependent parameters and saving or restoring data files at the beginning resp. end of session is a typical use of a macro. Cf. the section on global variables.

C. The form and the amount of interaction needed to solve problems of a certain kind may be changed through the use of a set of macros. The programs may in this way be adapted to new user categories.

A macro is implemented as a symbolic file. It may be generated either with the editor or with the command MACRO. (In either case, this command is the first one stored in the file.) Whenever a command line contains a command that is not found in the internal table of standard Idpac commands, it is first assumed to be a macro name and the memory is searched for a file with that name. If none is found, an error message 'ILLEGAL COMMAND' is given, otherwise execution of commands from that file is started.

A macro call may contain arguments like other commands. The code within the macro may contain not only calls to application commands but also calls to other macros and to structuring commands allowing looping and conditional execution. Thus a macro corresponds in operation to subroutines or procedures in high level programming languages.

When executing a macro, it may become suspended; either because of the command SUSPEND, actions taken in response to the READ command, or because of an error detected in which case the errouneous line is displayed. For hints as how to

proceed, refer to the commands END, GOTO, READ, and RESUME. When a macro is suspended , the program is in normal mode and the user may use any command he wishes. A natural choice after an error would be a correct form of the erroneous command.

3.2 Intrac Commands

Intrac implements a set of commands of a general and application independent nature. These commands are described in full in Chapter 4 of [1]. Here only a very brief account will be given.

MACRO NAME [ARG1 ...]
This command defines the following command sequences to be a macro with name NAME. The macro may contain formal arguments. At time of call, a list of corresponding actual arguments should be given, but under certain circumstances, the number of arguments need not agree, although this is the normal case.

FORMAL ARG1 [ARG2 ...]
This command extends the list of formal arguments within a macro.

END
This concludes the definition of a macro. If END is entered from the terminal while a macro is suspended, all active macros will be aborted.

LET VAR = expression
The variable VAR is given the value of the expression. Some legal forms are:
LET A = 0
LET P = 3 * 5.5
LET G1. = 2 + P
LET DATA = FILE1

DEFAULT VAR = ARG
The variable VAR is given the value of ARG, but only if VAR previously was unassigned or non-existent.

LABEL L
L is defined as a label, i.e. it is a name of the following command which therefore may be referenced in a GOTO.

GOTO L
The effect is that the next command to be executed is the one named L (by a LABEL). GOTO may be used to resume a suspended macro at a specified point.

IF relation GOTO L
The effect is a GOTO as above, provided that the relation is true. Examples of legal relations are:
A GT 2.5
B EQ FILE
The relational operators available are: EQ, NE, GE, LT, GT, and LT.

FOR V = BEGIN TO FINISH [STEP INCR]
The following commands will be executed for a sequence of values of the variable V. The first value will be BEGIN, the last will be less than or equal to FINISH. V will be incremented by INCR if given, the default is 1.

NEXT V
This signifies the end of a FOR V = ... sequence. The name of the variable (here V) indicates the proper FOR - NEXT pair.

WRITE [([dev][form])] [VAR/STRING ...]
This command will cause output on the device named dev ∈ {DIS,TP,LP} (display, teleprinter, line printer) with line advance form ∈ {LF,FF} (line feed, form feed). The defaults are DIS and LF. Following this information, there is a list of variables or strings to be output according to the respective type. Strings are delimited by string quotes. If the list is empty, the default action is to output all global variables. Example:
WRITE 'The value of VAR is:' VAR

READ V1 T1 [[V2 T2] ...]
This command demands new values for the variables V_i from the terminal keyboard. The arguments T_i specifies the required type of the answer. Under certain circumstances, the user need not give values to all arguments and may also suspend the macro.

SUSPEND
An executing macro is suspended when this command is encountered.

RESUME
Resumes a suspended macro.

SWITCH switch state
The switch ∈ {EXEC, ECHO, LOG, TRACE} will receive the state ∈ {ON, OFF}. The default values are OFF. The effect of the ON state is:
EXEC a macro is executed while being defined.
ECHO a macro is echoed while being executed.
LOG application commands are logged on the line printer.
TRACE the log also includes Intrac commands.

FREE [GVAR ...]
   User-defined global variables are deallocated.

STOP
   Execution of the program is stopped.

4. FILE FORMATS

The data sets operated upon in the commands, and referenced by name in the command arguments, are generally treated as sequential files. This format has proved sufficiently flexible for most application while at the same time providing good prospects for program portability.

In some implementations, these files are from reasons of efficiency actually not handled by the normal file system. This should be invisible by the user, but special actions may have to be taken to save and restore information of a more permanent nature.

The following sections describe the file conventions used. Note that file names need to be unique only within its own group, i.e. a file name FN can be used simultaneously for a data file, a system file, and an aggregate file.

4.1 Data Files and the File Head

Data files is the common name of files of binary form. They are used as one of the main forms of storing objects in the data base of the interactive programs developed in Lund.

To be able to read a binary file, one has to know in detail how it was written. To solve this problem a flexible standardized file structure has been introduced. This has been done in the following manner. Binary files, jointly termed data files, contain a first record of fixed length, the file head. The file head specifies the number of records to follow, and their lengths which are constant within a file. Figure 4.1 shows a detailed description of the file head. I1, I2 and I3 describes the structure of the file. Note that I2 specifies the record length, thus it is possible to correctly read the file, once the file head has been read and decoded.

1	I1 (number of rows)
2	I2 (number of columns)
3	I3 (time dimension)
4	Sampling interval in time units
5	Date recorded
6	Time recorded
7	If zero, the record length is constant
8	Fingerprint (number of the generating command)
9	File type
10	Skip count

Figure 4.1. The file head format.

Other information in the file head is the sampling interval wich is relevant if the file contains either measurement data or parameters etc. of a discrete time model. The date & time information in integers 5 & 6 is valuable as they give an identity to measurements and to results derived from them. The 7:th integer serves as an escape function as it allows non standard files and provides a means to stop reading such a file before any harm is done. The 8:th integer is a fingerprint in the sense that all commands that generates a file puts its command number there. This can be used to check compatibility requirements. It may also serve as a debugging aid.

The 9:th integer is used to indicate the logical contents of a file. Examples of where and why this is useful is given in the special sections on data files etc. below.

The 10:th integer, finally, specifies a skip count, i.e. a number of records to be passed before the file can be treated in the standard fashion. This too is a kind of escape facility and is primarily intended as a way to extend the file head. In Simnon this is used to indicate variable names and associated system names in a STORE file, to allow reference by name to the variables in a subsequent SHOW command.

4.2 Access Rules for Data files

There has to be some rules for controlling the use of data files. A few examples will illustrate this need. Assume that DATOP is a command taking as input a column of the input file and that the output will be a column in the output file. The input and output could be thought of as a time series. Thus a simple example would be:

a) DATOP OUTFIL < INFIL (3)

Here column number 3 (i.e. signal #3) in INFIL is read, operated upon, and the result is a new single column output file OUTFIL. We have met the first rule:

1) If an output file name but no column number is given, a new file is generated. Any old file with the same name as the new output file is lost.

If on the other hand we want to keep old information in an existing output file, we can do so:

b) DATOP OUTFIL (2) < INFIL (3)

In this case only column 2 in the output file is changed while all others are kept as before. As all files are accessed sequentially, this implies that the old version of the output file is read and copied to a new file with

modifications made to, in this case, column 2. This is governed by the second rule which reads:

2) If an output file with a column number is given, the new column must replace an old column or be number N+1 if N is the number of previously existing columns.

There is a shorthand description for the case where the input and output files have the same name:

c) DATOP < INFIL (3)

The rule describing this case is:

3) If the output file name is omitted, the input file name is assumed. If a column number is given for the input file, it is also used for the output.

It should be noted that these general rules are implemented in the command decoding part of the command modules. They may be augmented by other rules, specific for a single command or for a group of commands.

4.3 Specific Examples of Data Files

Some specific details on how the data actually are stored in data files within the Lund programs will be given here.

Time Series

Refer to Figure 4.2. A time series file is implemented as a standard data file with I1 equal to the number of time points, I2 equal to the number of measured signals (and the record length) and I3=1. The file type number of a Time Series file is 0.

Frequency Responses

A frequency response is stored in three columns of a two dimensional array. The frequency values are stored in the first column and the amplitude and phase in the 2:nd and 3:rd.

The main feature wich distinguishes a frequency response file from a time series file, is thus that data is recorded in groups of three columns. Some commands (like FROP, BODE and ASPEC, see Appendix) should recognize this. Therefore this kind of file has a special file code, namely 1.

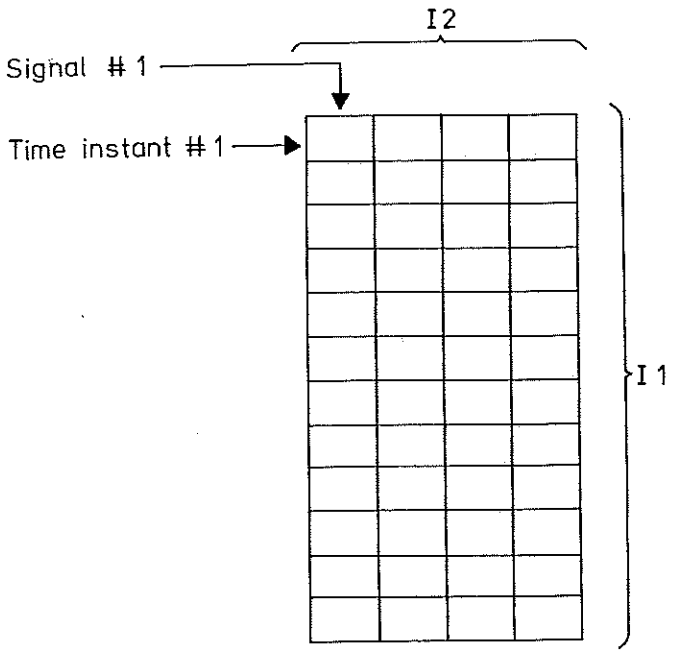

Figure 4.2. The format of a time series file.

Loci

A locus file is used to store sets of complex numbers associated with a real parameter value, (i.e. one parameter value per set). The use is to store eigenvalues, polynomial zeroes and the like. It is possible to include several sets to represent the dependance of the zeroes etc. on the parameter.

The file format for a locus file is that of a data file with the parameter in the first column. The complex eigenvalues or polynomial zeroes are stored with their real and imaginary parts in the following 2n-1, 2n columns, n=1,2 .... . Thus a single row contains the parameter value and the accompanying set of complex numbers. Several rows will store a locus. The file code is 2.

Matrices and Vectors

Matrices and vectors are easily stored within a data file. It is natural to store a matrix row wise, i.e. I2 is the number of columns in the same way as for a time series. The number of rows is stored in I1. A vector is stored as a n*1 or 1*n matrix.

The time dimension I3 is used as a way to store time varying matrices (vectors). For each time instant (sampled data systems), the matrix is stored as above. The number of different time points is in I3. In the time invariant case I3=1. The file code is 3.

Polynomials

A polynomial matrix is represented with the matrix coefficients for various powers of the independant variable stored in the same way as time varying matrices. Thus a 2*3 polynomial matrix of degree 3 is stored with I1=3, I2=2 and I3=4, I3 being the number of coefficients. The file code is 4.

Note that a scalar polynomial of degree N is stored as I1=1, I2=1 and I3=N+1. According to convention, the highest degree coefficient is always included explicitly and is stored first in the file.

4.4 Aggregates

An aggregate file is the concatenation of several individual files of formats described above, into a single sequential file. The individual files must be of the same type. Figure 4.3 illustrates an aggregate file. It consists of a file head with standard format. It is flagged as an aggregate by the file code being 100 in excess of the file code of the individual files it is made up of. The number of concatenated files is recorded in I1 which is also the record count for the records immediately following the aggregate file head. These records contain the file names of the constituent files. These files are then included sequentially in the order of their names, and are preceded by their file heads in the usual manner.

The advantage of this scheme is that the file administration overhead in the computer system is paid for only once for a large set of related files. On the other hand, the advantage of the ability to handle these files separately, as in checking and modifying the data, is not lost. The program modules that do these operations are made to allow a specified file to be a member of an aggregate. The time

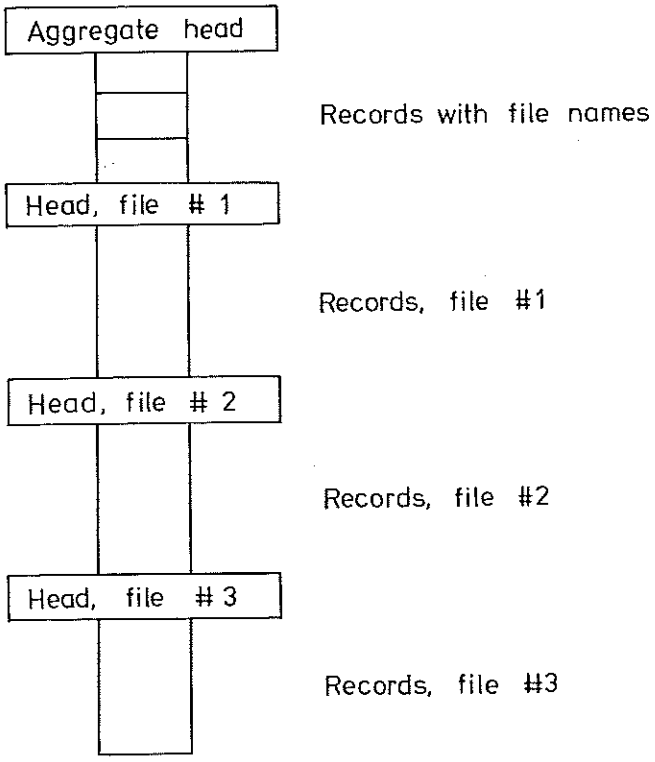

Figure 4.3. The format of an aggregate file.

penalty for this is usually small since reading/writing past other files in the aggregate is a fast operation compared to the file administration.

Note that aggregate files not only serve as a means of increasing efficiency in accessing the data base, they also give the possibility of a nice naming convention. For the user, this is the most important aspect. As an example, the matrices are not required to have distinct names. They may be given standard names as found in literature such as A, B, C etc., functioning as 'forenames'. Only the name of their aggregate has to be distinct like a 'family name'. An example is the command ALTER AG:B where the matrix B in the aggregate AG is altered.

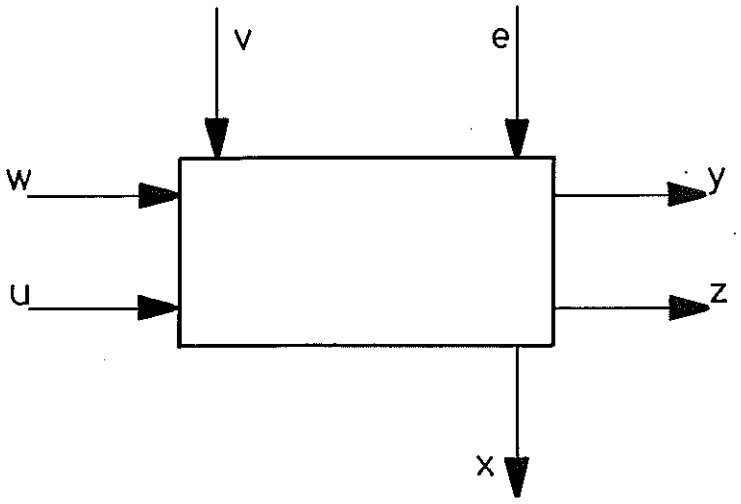

Figure 4.5. General input and output signals of a system.

y is the measured output from the system, i.e. the signal is actually available for feedback but may be corrupted through the disturbance e.

z is the desired output, i.e. signals by which the performance of the system will be judged.

x is the state of the system.

These distinctions are intended for the benefit of the user, helping in the formulation and solution of problems, e.g. in controller and observer design. All the distinctions are not possible to maintain in all representation forms.

Each section must begin with the keyword BEGIN, optionally followed by the section name. The next line must contain the section heading, specifying the kind of system representation used in this section. The section ends with a line with the keyword END. Comments may be freely used throughout the section. They start with a double quote ("). Other information between BEGIN and END will depend on the representation type as described below.

4.5 System Files

A system can be represented in a number of ways, with different types of data of varying structure, matrices, polynomials etc.

Therefore, system descriptions are based on text files. They give a greater freedom of structure since information is easily tagged with keywords and the recordlength is no major problem. Great amounts of data, e.g. matrix elements, polynomial coefficients etc., are stored in data files as described above; in the system file only the appropriate file names are given, sometimes within a keyword structure; see the example in Figure 4.4. If matrices, polynomials etc are available as parts of aggregates, the aggregate filename is included. Different representations may be included in a system file using standardized keywords. If there are more than one type of system representation present in the system file, they are enclosed within a pair of keywords BEGIN - END. Such sections within a system file are named separately, the name appearing after BEGIN.

An example of a section of a system file is given in Figure 4.4. After BEGIN the section name appears, in this case 'cont'. Then the type of system representation is specified by a sequence of keywords: CONTINUOUS STATE SPACE REPRESENTATION. The initial state of the system is specified to be stored in a file with name x0vec. (In the example, filenames are written in lower case letters, keywords are in upper case).

After this, the state space system equations are specified following the keywords DYNAMICS and AGGREGATE. The aggregate name for the system matrices is given. The equations are written in full. They contain elements recognizable by the program such as DX/DT, *X, *U, Y= etc. These constructs serve to delimit filenames where the appropriate matrices are stored, a, bu etc. A matrix together with its key construct may be omitted and is then assumed to be zero, e.g. if dw*W is absent, then dw is assumed zero.

```
BEGIN cont
CONTINUOUS STATE SPACE REPRESENTATION
INITIAL STATE VECTOR: x0vec
DYNAMICS, AGGREGATE: sysagg,
DX/DT= a*X + bu*U + bw*W + bv*V
    Y= c*X + du*U + dw*W + de*E
    Z= g*X + hu*U + hw*W
END
```

Figure 4.4. A section within a system file.

4.6 Access Rules for System Files

In the case of data files, the generation of a new file with a given name implies that any existing file with that name is lost and replaced by the new version. In the case of a system file however, the corresponding situation is when a new section within an existing file is generated. Of course, the deletion of the entire old system file is out of question, it may contain much useful information in other sections. Therefore the old file is merely copied with the new section added at the beginning. This can only be allowed to happen when the operaton that generates the new section (system representation) is such that the new representation is just another way of representing the system.

The access rules are as follows:

a) A command that generates a new system description will check that there is no already existing file with the output name.

b) If a command is used to transform system representations, a new section name must be specified if the output file name is the same as the input file name (or is omitted). If the output name differs, rule a) is applied.

c) A new section is placed first in the file. Any old section with that name is retained but may be accessed only through the text editor, rule d).

d) Only the first section with a given name is accessible through commands operating on system files.

4.7 Specific Forms of System File Sections

A system can be described using different representation forms, sometimes simultaneously, resulting in a system file with several sections. The form of the sections must follow certain conventions described in the following paragraphs.

The following signal name conventions will be used, compare Figure 4.5.

u is a control input, i.e. an input we may manipulate in controller design.

w is a known disturbance input, i.e. we know it but can't affect it.

v is an unknown, i.e. stochastic input.

e is an unknown, i.e. stochastic disturbance on the measured output.

DISCRETE MISO TRANSFER FUNCTION
(polynomial image form)

This section describes a linear, discrete time, multiple input - single output dynamic system on the general form

$$y(t) = \sum \frac{B_i(q^{-1})}{A_i(q^{-1})} u_i(t) + \sum \lambda_i \frac{C_i(q^{-1})}{D_i(q^{-1})} e_i(t).$$

The normal form used in Idpac is, however,

$$y(t) = \sum \frac{B_i(q^{-1})}{A(q^{-1})} u(t) + \lambda \frac{C(q^{-1})}{A(q^{-1})} e(t).$$

(Refer to the description of individual commands to see which form is used.)

The section heading must read:

DISCRETE MISO TRANSFER FUNCTION

The section <u>must</u> contain the following statements:

- sample interval definition
- one A-polynomial
- one B- or C-polynomial
- at least one λ definition if C-polynomials are present.

This section <u>may</u> contain:

- several A-, B-, C-, and D-polynomials
- initial values for the output
- uncertainties of parameter estimates
- loss function value
- Akaike's test quantity
- covariance matrix of parameter estimates
- comments preceeded by a double quote (")
- blank lines (not between parts of a polynomial description).

If there are several polynomials of the same type, they must be enumerated increasingly from 1 on.

Specifics

Sample interval: SAMPLE INTERVAL t s
where t is the sample interval in seconds (integer or real)

Polynomials: XPOLYNOMIAL j
$Q^{K}*(c_1 Q^{i_1} + c_2 Q^{i_2} +...+ c_n Q^{i_n})$

where X ∈ {A,B,C,D}; j = 1,2,.. or omitted
$K, i_1, i_2, .. \leq 0$ integers

- the leading power of Q and the parenthesis are optional
- the multiplication signs are optional
- c_i may be written in free format, i.e. 10, 10. and 1.0E1 are all treated as 10.0
- c_i is treated as $c_i Q^0$
- Q^i is treated as $1.0Q^i$
- terms equal to zero may be omitted
- the order between the terms is not essential
- a polynomial specification may be written over several lines, but there must not be any blank lines or comment lines in between
- there must not be two terms of the same order of Q
- the maximum order of any polynomial is 25 + the order of the time delay
- a C-polynomial definition must be followed by a lambda definition

Noise standard deviation: LAMBDA λ
where λ is the noise standard deviation (integer or real)

Loss function value: LOSS FUNCTION v
where v is the value of a loss function (integer or real)

Akaike's test quantity: AIC v
where v is a test quantity computed by some identification routines

Initial values for the output:

INITIAL VALUES FOR THE OUTPUT

$y_0 + y_{-1} Q^{-1} + \ldots$

cf. polynomial definitions

Standard deviations of parameter estimates:

UNCERTAINTIES

$Q^K*(s_1 Q^i_1 + s_2 Q^i_2 + \ldots)$

cf. polynomial definitions

Covariance matrix of parameter estimates:

COVARIANCE MATRIX

c_{11} c_{12} ... c_{1m}

c_{21}

.
.
.

c_{n1} c_{nm}

blank line

c_{1m+1} c_{1n}

.
.
.

c_{nm+1} c_{nn}

```
Example:
(This is a system file produced by the ML command)

BEGIN
DISCRETE MISO TRANSFER FUNCTION

"MAXIMUM-LIKELIHOOD ESTIMATION OF ORDER 2
"FROM THE DATA FILE WRK

"INPUT(S): COLUMN(S)  1
"OUTPUT: COLUMN  2

SAMPLE INTERVAL  1.00 S

APOLYNOMIAL
           1.0000   Q^-0  -  1.3684   Q^-1 + 0.45712   Q^-2

BPOLYNOMIAL
Q^-1*( 0.85633   Q^-0 + 0.062352   Q^-1 )

CPOLYNOMIAL
           1.0000   Q^-0 + 0.00000    Q^-1 + 0.00000   Q^-2

LAMBDA  0.97626 +- 3.98557E-02
LOSS FUNCTION  142.96
AIC  548.95

END
```

DISCRETE MISO TRANSFER FUNCTION
(polynomial file form)

In the polynomial image form described above, the coefficient values for the polynomials were included in the symbolic text. This is the form used in Idpac, i.e. the result from the identification commands may be listed directly. In other program packages, the polynomials are assumed directly available as polynomial files. The DISCRETE MISO TRANSFER FUNCTION section should then have a slightly different form. The keywords APOLYNOMIAL, BPOLYNOMIAL etc. should be followed by a name of a polynomial file, e.g.

```
APOLYNOMIAL aname
```

and the polynomial coefficients should be absent. A new optional keyword AGGREG followed by the name of an aggregate file for the polynomials may also be used.

STATE SPACE REPRESENTATION

The state space representation section allows the optional inclusion of all signals found in Figure 4.5. Both continuous time and discrete time systems can be used as given by the headings

CONTINUOUS STATE SPACE REPRESENTATION

or

DISCRETE STATE SPACE REPRESENTATION

In the latter case, a definition of the sample interval in seconds must be included:

SAMPLE INTERVAL v S

The matrices of the state space representation are given through the equations in symbolic form. First a keyword is given:

DYNAMICS,

or

DYNAMICS, AGGREGATE: dynag,

where the second form names the aggregate file where the matrices may be found. The equations have the form:

```
DX/DT =  a*X + bu*U + bw*W + bv*V
   Y  =  c*X + du*U + dw*W + de*E
   Z  =  g*X + hu*U + hw*W
```

The following rules apply:

a) For a discrete time representation DX/DT is replaced by XNEW.

b) The first equation must always be present.

c) At least one of Y = or Z = must be present.

All matrix definitions need not be included, in which case the corresponding keyword is absent. A null matrix of appropriate dimension is then assumed. I.e., if there is no bw matrix, *W is omitted. The rule is:

d) Enough matrices must be included to be able to determine the number of states, inputs and outputs.

Note that the distinction between e.g. U, W, and V or between Y and Z depends on the operation performed. Many commands will interpret bu, bw, and bv as blocks of a single B-matrix.

The initial state of the system may optionally be given by (and should be included at the end of the dynamics aggregate):

```
INITIAL STATE VECTOR: xovec
```

For linear quadratic problems, a loss function, an extended loss function, and a covariance function may be specified. For their use, refer to the Synpac commands OPTFB, PENLT, and KALFI. They may optionally contain an aggregate specification, given within square brackets below. Their respective form is:

```
LOSS FUNCTION, [AGGREGATE: lagg,]
Q0: q0,  Q1: q1,  Q12: q12,  Q2: q2

EXTENDED LOSS FUNCTION, [AGGREGATE: elagg,]
EQ0: eq0,  EQ1: eq1,  EQ12: eq12,  EQ2: eq2,
EQ3: eq3,  EQ4: eq4,  EQ5: eq5

COVARIANCE FUNCTION, [AGGREGATRE: cfag,]
R0: r0,  R1: r1,  R12: r12,  R2: r2
```

In all three cases, a matrix that is not included will be assumed zero.

Note in the use of aggregates, that matrices should be included in the order shown above, from the left to the right.

POLYNOMIAL FORM

This representation resembles in many respects the DISCRETE MISO TRANSFER FUNCTION, but is the representation used throughout POLPAC. The required section heading is

CONTINUOUS POLYNOMIAL FORM

or

DISCRETE POLYNOMIAL FORM

In the latter case, the specification SAMPLE INTERVAL v S must be included. The system equation is also here given on symbolic form with an optional aggregate name:

```
DYNAMICS, [AGGREGATE dagg,]
a*Y = b*U + be*E + bw1*W1 + bw2*W2 + ...
```

The polynomials a, b, be, bw1 etc. all have scalar coefficients and bw1,...,be are optional. Note that the case where only a and b are present gives the transfer function representation of classical control theory. The inclusion of several inputs W_i and E allows the formulation of feedforward design and minimum variance controller design problems.

The section allows the inclusion of the keyword FACTOR TABLES. After this keyword, a series of comment lines, generated by the command POLSYS, may follow giving the factored form of the polynomials.

5. REFERENCES

[1] J. Wieslander, H. Elmqvist: Intrac - A Communication Module for Interactive Programs - Language Manual. Report TFRT-3149, Dept. of Automatic Control, Lund Institute of Technology, Sweden.

[2] J. Wieslander: Idpac Commands - User's Guide. Report TFRT-3157, Dept. of Automatic Control, Lund Institute of Technology, Sweden.

[3] J. Wieslander: Modpac Commands - User's Guide. Report TFRT-3158, Dept. of Automatic Control, Lund Institute of Technology, Sweden.

[4] J. Wieslander: Synpac Commands - User's Guide. Report TFRT-3159, Dept. of Automatic Control, Lund Institute of Technology, Sweden.