Scandinavian Control Library -- Programming and Documentation Rules for Subroutine Libraries

Designed for the SCL

Tyssø, Arne; Elmqvist, Hilding; Wieslander, Johan

1976

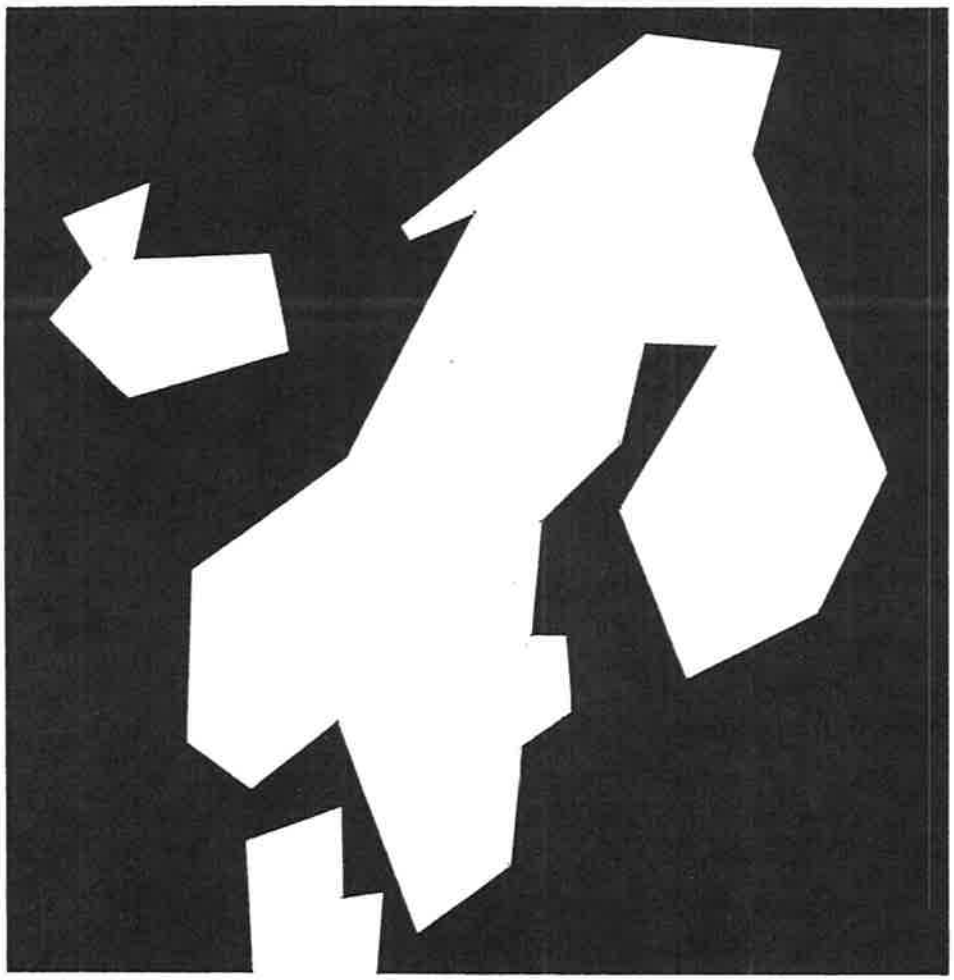
Document Version:
Publisher's PDF, also known as Version of record

SCL

Scandinavian Control Library

Programming and documentation rules for subrutine libraries - designed for the SCL

Edited by Elmqvist, Tyssø, Wieslander



NORDFORSK
The Scandinavian Council for Applied Research

First Edition 1976-06-01

Programming and documentation rules for subrutine libraries – designed for the SCL

The NORDFORSK project-group on "Computer Aided Design of Dynamical Systems"

First Edition 1976-06-01

CONTENTS

Preface

NORDFORSK, the Scandinavian Council for Applied Research, aims at initiating and organizing Scandinavian cooperation in scientific and industrial research and in utilization of research results. NORDFORSK is a joint body of the nine technical research councils and academies in the five nordic countries.

One of the objectives of this NORDFORSK-project "Computer aided design of dynamical systems" is to establish a Scandinavian program and subroutine. library "Scandinavian Control Library". This report contains the common rules for programming and documentation that will be used for the programs in this library.

A report concerning the development and organization of the "Scandinavian Control Library" will be published later in 1976.

This work is a result of a joint effort by the following persons:
Leif Andersson, Hilding Elmqvist, Tommy Essebo, Claes Källström, Tomas Schönthal, Johan Wieslander and Karl-Johan Åström, Lund Institute of Technology, Sweden and Arne Tyssø, Oddvar Hallingstad, The Norwegian Institute of Technology, Norway.

1. INTRODUCTION

It is difficult to make general rules in the field of program design because software and hardware facilities differ quite a lot from one computer system to another. Especially such things as different programming languages and peripherial equipment make it difficult to standardize?

We will concentrate on the design part that is independent of the type of computer system. General input/output handling such as use of graphical display and files is dependent of the computer and will therefore not be treated in the rules. However, regarding exchange of complete program systems it should be stressed that all computer-dependent subroutines are strictly specified and well documented as proposed in this report.

It is generally agreed that all programs and subroutines should be written in Standard Fortran IV as given in USA ANS X3.9 1966. Fortran IV is chosen because Fortran Compilers are available at most computer systems, and many of the already existing subroutines are written in Fortran. However, one should be aware that there exists many different versions of Fortran IV, and some of the features that are accepted by one computer is not allowed by another.

Special features will be accepted to some extent if it is possible to modify the subroutines with a reasonably good editor system. If possible these special features should be confined to special purpose subroutines. More complicated programming features which are not mentioned in the standard should be avoided. Some of these things are discussed in chapter 2.

This report deals especially with the problem of dynamical allocation of matrices and how a subroutine/program should be documented. Also some proposals to symbol use and communication with subroutines are presented.

As a basic rule the following could be set up:

The programs should be well structured and well documented, and they should be written in Fortran IV.

The documentation and all comment statements in the program should be written in English.

2. DESIGN OF PROGRAMS AND SUBROUTINES

This report does not discuss elementary questions about program design. A good introduction to the problems can be found in "Kerrighan-Plauger: The elements of programming style. Mc. Graw-Hill 1974". Some of the rules they introduce are listed below.

Write clearly - don't be too clever.
Say what you mean, simply and directly.
Write clearly - don't sacrifice clarity for "efficiency".
Let the machine do the dirty work.
Replace repetetive expressions by calls to a common function.
Parenthesize to avoid ambiguity.
Choose variable names that won't be confused.
Avoid the Fortran arithmetic IF.
Avoid unnecessary branches.
Don't use conditional branches as a substitute for a logical expression.
If a logical expression is hard to understand, try transforming it.
Modularize. Use subroutines.
Don't patch bad code - rewrite it.
Write and test a big program in small pieces.
Make sure all variables are initialized before use.
Check some answers by hand.
Make it right before you make it faster.
Make it fail-safe before you make it faster.
Make it clear before you make it faster.
Let your compiler do the simple optimizations.
Don't strain to re-use code; reorganize instead.
Make sure comments and code agree.
Don't just echo the code with comments.
Don't comment bad code - rewrite it.
Use variable names that mean something.
Format a program to help the reader understand it.

2.1. Matrix handling

Many results in automatic control theory are formulated using matrix notation. This fact implies that in a subroutine library for automatic control problems there must be a way of handling matrices. When using FORTRAN this is a problem, mainly because of lack of dynamic allocation mechanism. This chapter describes how matrices are stored in FORTRAN, a way of handling work areas and a set of subroutines for basic matrix operations.

Arrays in FORTRAN

Two basic rules for the library are:

1 The subroutines in the library should be independent of problem size (dimension).

2 The subroutines in the library should be constructed in such a way that it is possible to write a calling program which can handle different problem dimensions.

These rules imply that:

3 The actual dimensions of the arrays in the subroutine call must be transferred to the subroutine.

4 Information about how the arrays in the subroutine call are dimensioned must be transferred to the subroutine.

In order to calculate the memory location of an array element, the location of the first element of the array and all except the last maximum dimensions are needed. All except the last maximum dimensions should thus be transferred to subroutines via so called dimension parameters. Thus a vector does not need any dimension parameters. A matrix needs one and a three dimensional array needs two dimension parameters.

Example:

Consider the following main program and subroutine

```
C       MAIN PROGRAM
        DIMENSION A(3,4)
        DATA IA/3/
C
        READ (5,100) N1A,N2A
        ----
        READ (5,110)((A(I,J), J = 1,N2A),I = 1,N1A)
        ----
        CALL SUB1(A,N1A,N2A,IA,...)
        ----
        END

        SUBROUTINE SUB1(A,N1A,N2A,IA,...)
        DIMENSION A(IA,1)
        ----
        DO 10 I = 1,N1A
        DO 10 J = 1,N2A
        X = 0.5*A(I,J)
        ----
   10   CONTINUE
        ----
        END
```

The dimension parameter of A is IA = 3 since A is dimensioned (3,4). The actual dimension of A is read into N1A and N2A. When the matrix A is read, it is stored as illustrated in figure 1. The dimension parameter is used in the dimension statement in the subroutine, and implicitly when using the element A(I,J) in the subroutine.

There is in fact a way of avoiding the dimension parameters. If a subroutine with the dimension statement DIMENSION A(N1A,1) reads in the matrix, then the matrix would be stored using consecutive memory locations (compare figure 1). The subroutine SUB1 then should have the same dimension statement, and the parameter IA could be dropped.

This method, however, has the serious draw back that all arrays must be packed as above,which means that the routines can not be used together with routines from other libraries not using the same method. This method should thus not be used.

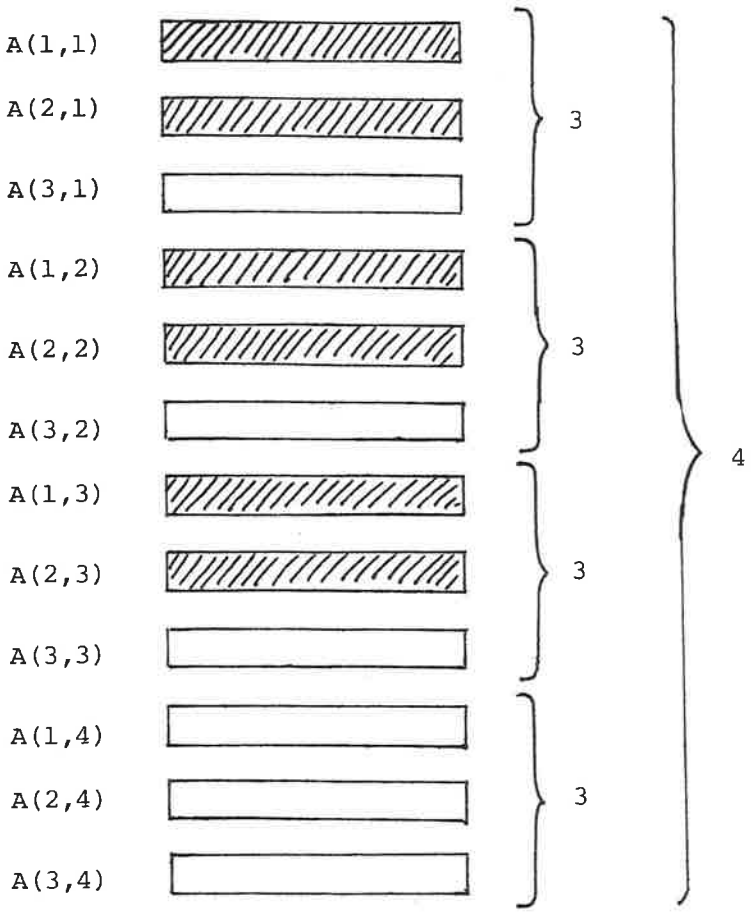


Figure 1. Illustrates how the matrix A is stored. The actual 2x3 matrix is stored in the shaded elements.

Work areas

FORTRAN does not include dynamic allocation of arrays. In order to make the subroutines independent of problem size it is thus required that

5 All work areas with problem dependent size should be transferred via the subroutine call.

The method of using a COMMON-block as allocation area has some serious drawbacks and should not be used.

Work areas for a subroutine does not contain relevant information outside the subroutine and will not be manipulated there. The work areas then do not need any dimension parameters.

6 A work area should be dimensioned in the subroutine using actual dimensions.

Example:

```
SUBROUTINE SUB2(A,N1A,N2A,IA,W1,W2,....)
DIMENSION A(IA,1)
DIMENSION W1(N1A,1), W2(N1A,1)
```

The work areas of subroutines at a low level in a subroutine hierarchy must be included in the call to a subroutine at a high level. This means that the number of work areas in a subroutine call can be large.

Two work areas which do not hold information at the same time in a subroutine but have different shapes, must both be included in the call. They can, however, be equivalenced in the call, which means that the actual parameters are the same.

The equivalence structure can, however, be rather complicated, and if the user of the subroutine does not make equivalence, then the program will be unnecessarily large.

One way of making the use of the subroutines easier in these cases, is to introduce an interface routine and allocation vectors. The allocation vector is partioned, using indices, into a number of work areas, and then the original subroutine is called using these indices.

Example:

```
      SUBROUTINE SUB3(N,... W1,W2,W3,W4)
C     This subroutine does the real work
C     W1 - work area, size(N)
C     W2 - work area, size(N,3)
C     W3 - work area, size(N,2)
C     W4 - work area, size(N,2)
C
C     W1,W2  and  W3,W4  can share space
C
      DIMENSION W1(1),W2(N,1),W3(N,1),W4(N,1)
      ---
      END

      SUBROUTINE SUB3I(N,...,W)
C     This is the interface routine, which only allocates work space
C     W - allocation vector, size(4*N)
C
      DIMENSION W(1)
      KW1 = 1
      KW2 = KW1 + N
      KW3 = KW1
      KW4 = KW3 + 2*N
C
      CALL SUB3(N,...,W(KW1),W(KW2),W(KW3),W(KW4))
      RETURN
      END
```

The user of the subroutine now only has to care about one work area. The equivalence structure is also taken care of in the interface routine.

Using an allocation vector requires a certain programming feature outside the Fortran IV Standard: The number of indices of a formal parameter may be greater than one even if the number of indices of the actual parameter is one.

This straight forward method of handling work areas has one drawback. It introduces a new small subroutine for all subroutines which are intended to be called by users, and which have many work areas. There are, however, two ways of reducing the number of work areas in a subroutine call.

7 Work areas to lower level subroutines can be allocated from an allocation vector in the calling subroutine.

Example:

```
SUBROUTINE SUB4(N,...,WA,WB,W)
DIMENSION WA(N,1),WB(1)
DIMENSION W(1)
KW1 = 1
KW2 = KW1 + N*N
---
CAN SUB41(WA,WB,N,...,W(KW1),W(KW2))
---
END
```

WA and WB are work areas in the subroutine SUB4 and W1 and W2 are work areas in the subroutine SUB41.

The other way of reducing the number of work areas is:

8 If a work area is used only to transfer data from one lower level subroutine to another, then this work area can be allocated in the subroutine, and the pointer used in the subroutine calls.

Example:

```
SUBROUTINE SUB5(...,N,:..,W)
DIMENSION W(1),...
KA = 1
KB = KA + N*N
CALL SUB51(...,W(KA),W(KB),N,...)
---
CALL SUB52(W(KA),W(KB),...,N,...)
---
END
```

The subroutine SUB51 stores its results in the formal parameters A and B which actually corresponds to the areas in W which begin at KA resp. KB. The subroutine SUB52 uses these results via its formal parameters.

The above discussion is summarized in the rule.

9 If the number of work areas is large or there is a complicated equivalence structure, then the subroutine should be rewritten according to rules *7* and *8* or an interface routine should be constructed.

The following rules are also introduced.

10 The allocation should be done explicitly by computing pointers (indices).

11 The allocation of a work area of a certain type should be done from an allocation vector of the same type.

12 Only one allocation vector of each type should be used at each subroutine level.

13 If the required size of an allocation vector is determined by a complicated formula, the user should have the possibility to enter the actual size of the allocation vector. The subroutine should test if the actual size is sufficiently large and use an error parameter if it is too small.

Example:

```
      SUBROUTINE SUB6(N,....IERR,W,NW)
      IERR = 0
      KW1 = 1
      KW2 = KW1 + N
      KW3 = KW2 + 2*N
      K1 = KW2 + N
      KW4 = KW2
      KW5 = KW4 + N*N
      K2 = KW5 + N
      IF(MAX0(K1,K2)-1.GT.NW) GOTO 900
      ---
  900 IERR = 1
      RETURN
      END
```

Basic matrix operations

The basic matrix operations, addition, subtraction and multiplication tend to be hard to read when implented using inline code with DO-loops. The alternative is to use subroutines for these operations. The total object code then becomes smaller and the execution time can sometimes be reduced if the subroutines are implemented in assembly language.

Four subroutines for basic matrix operations (MADD,MULT,RMULT and MMOVE) are documented in the appendix,A2.

The subroutine MADD performes matrix addition and subtraction. There are switches to determine if the operands should be transposed. The subroutine MULT performes matrix multiplication. It has switches to determine if the operands should be transposed or if the result should be made symmetric. The subroutine RMULT multiplies a scalar with a matrix and MMOVE moves a matrix.

14 The subroutines MADD, MULT, RMULT and MMOVE should be used when this is not unnatural.

2.2. Communication with the subroutine

15 If possible the argument list should be ordered in the following way.

- o operands (input data)
- o results (output data)
- o descriptions of operands and results (for example actual dimensions of matrices)
- o other information (for example test quantity, error indicator)
- o dimension parameters
- o work arrays and allocation vectors (and their sizes)

16 Common blocks should generally be avoided. However, they can be used to return information which may be of interest to a user.

17 Operands ought not to be destroyed in a subroutine.

The documentation should tell when arguments may be equivalenced.

18 Subroutines on a basic level in the program library should have different dimension parameters for all matrices. Groups of matrices, in the argument list of higher level subroutines, which always have the same actual dimension and which can be expected to have been dimensioned equally, should have a common dimension parameter.

19 The user is allowed to put a test quantity equal to zero. A computer dependent default value is then used. If the subroutine uses more than one test quantity they should be placed in a vector. It shall be possible to individually determine if default values should be used by setting resp elements of the vector equal to zero. The variables for test quantities must not be altered in the subroutine. The computer dependent default values should be computed using the functions RMACON and IMACON (See appendix, A2).

2.3. Some rules for the program code

20 Input - Output should not be performed in a computational routine. If I/O is desirable (as might be e.g. in function minimazation routines) it should be confined to special I/O subroutines, specified as external routines in the subroutine call. They should be documented as a "User supplied subroutine", but in many cases it would be natural to expect suitable I/O routines to be part of the library.

21 Specification statements should appear before all executable statements and FORMAT statements and in the order:

- o type statements
- o dimension statements
- o common statements
- o equivalence and external statements
- o data statements

This is actually required by some FORTRAN compilers.

22 Computer dependent constants should be obtained through the functions IMACON and RMACON (See appendix A2).

23 The numbers defining labels should appear in increasing order and be preceeded by one space.

24 Character strings and Hollerith data should be avoided in numerical subprograms.
The internal representation of Hollerith data is dependent of the computer word-length.

It is a good habit to use comment statements in the programs. These can in one way replace flow charts, and it becomes easier to read the programs and if necessary, to modify them.

How much of comment-statements to be used is dependent of the degree of documentation required. These things are discussed in next chapter.

25 Comment statements should be written in English.

2.4. Use of symbols

It is of great importance that the use of symbol names is following a certain standard. This will simplify the reading of program listings. It is well known that almost every institute has its own standard more or less, and it is difficult to come to an agreement on this matter.

However, in the following we will give some recommendations with alternatives.

2.4.1. General use of symbols

Fortran has automatic type declaration: every symbolic name with a first letter of I, J, K, L, M and N implies type integer and any other letter

implies type real. This should be followed as a general rule, but logical variables should begin with letter L. Identifiers should have at most 6 characters.

Dimension of arrays

There exist many proposals how to solve this standardizing problem. Here is one example:

26 The names of variables for actual dimension should begin with the letter N. If a vector, matrix, etc. has an own single variable for the actual dimension it shall be named Nx , where x is a string of characters indicating the name of the vector, etc. If a matrix, etc. has more than one own variables for actual dimensions they should be named N1x, N2x, etc.

Example:

A(NA), B(NB,NB), C(N1C,N2C)

27 The dimension parameter should follow rule 26 and begin with the letter I or M.

Error indicator, warning indicator

28 An error- or warning-indicator should be named IERR. Zero (0) indicates "no errors" while small positive values (1, 2, 3,...) indicates different warnings (smaller values) or different error conditions (larger values).

The indication of one of several possible successful calls should be done with a strictly positive integer named IND. (1, 2, 3.....). Larger positive values can be used for error indication.

Work arrays - allocation vectors

29 The name of allocation vector should be W and if it is an integer, JW. As an alternative could be used H and JH.
The name of work arrays should begin with W or H.

Example: H1,H2,W1,WC.

30 The pointers to work areas should begin with the letter K.

A summary of the general symbols is given in Table 1.

Symbol	Description
Ix, Mx	dimension parameter for x (x is a string of characters)
Nx	actual dimension for x
NW	size of real allocation vector
NJW	size of integer allocation vector
Wx, Hx	real work areas
JWx, JHx	integer work areas
KWx	index for work areas
W, H	real allocation vector
JW, JH	integer allocation vector
IERR	error indication
IND	indicator
EPS	test quantity
I, J, K, M, N	integers
L	logical

Table 1. General symbol table.

2.4.2. Special symbol names to be used

In Table 2 there is given a summary of proposed names to be used when making programs and subroutines for design of control systems and parameter identification.
As a general recommendation the following letters in the ending of a name, indicate:

E: estimated value

P: predicted value

I: initial value

F: final value

S: simulated value

M: mean value

As symbols for time and time increment should be used T and DT.

Proposed symbol	Usually used in text-books and literature	Description
A	A,F	System matrix
B	B,G	Control matrix
C		Process disturbance matrix
D,C	D,C,H	Measurement matrix
E,D	E,D	Measurement-control matrix
G,RL	G,C,L	Feedback gain
RK	K	Estimator gain
FI,AD	Φ	Discrete version of
DE,BD	Δ,Γ	A,B,C
OM,CD	Ω,Θ	
PP,Q2	P,B,Q2	Weighting matrix, control vector
QQ,Q1	Q,A,Q_1	Weighting matrix, state vector
PQ,Q12	Q_{12}	Weighting matrix, mixed term
V	$\underline{v}$	Process noise vector
W,E	$\underline{w},\underline{e}$	Measurement noise vector
U	$\underline{u}$	Control vector
X	$\underline{x}$	State vector
Y	$\underline{y},\underline{z}$	Measurement vector
XX,P,S	ΔX,P,S	State vector, error covariance matrix
VV,R1	V,R_1	Covariance matrix, process noise
WW,R2	W,R_2	Covariance matrix, measurement noise
VW,R12	R_{12}	Covariance matrix, correlated term
EP	ε	Innovation process
RR	R	Covariance matrix of the innovation process

Table 2. Special symbol names to be used.

Examples:

Using the special symbols mentioned above we then have:

XI:	$\underline{x}(t_o)$	Initial statevector
YE:	$\hat{\underline{y}}$	Estimated measurement vector
XXP:		Predicted error covariance matrix

3. RULES FOR DOCUMENTATION

These rules on documentation are intended to facilitate the interchange of programs. Because different groups of people may not use the same standard notations although they work in the same field, documentation and definition of the problem solved are important.

These rules apply to two different situations.

a) Documentation in a program library binder

In this case the reader is interested in:

i) ease of finding a given routine.

ii) ease of reading, especially mathematical formulae, and expressions should be in a easily readable form.

iii) Completeness.

Point i) implies that at least the first page of documentation of a routine should have an easily recognized form.

Point ii) dictates typographical freedom including the possibility to use figures.

Point iii) tells us not to impose size restrictions by specifying special forms to be completed. (The conflict between i) and iii) is resolved later on).

b) Documentation in the programlisting

It is highly recommended that the program listing contains the same information in the same order as the program library binder. This information is concentrated to the beginning of the program listing, called the program head. This ensures that a given library routine within itself always contains adequate

information on how it is supposed to work. In fact, good programming practice is to start development of a new subroutine with the definition of the program head, thus defining the intended function of the code being created. In this way, the program head is a good temporary documentation and quite conceivably, part of it can be used in the final documentation in the library binder.

The fixed-format part of the library binder documentation

The information in the library binder concerning an individual subroutine will start with a fixed-format header, see Fig. 2. The items in the header are:

Name: The name of the subprogram. The name should give a hint of the use of the subprogram.

Number: The number of the subprogram concentrated from the section number in the systematic library catalogue followed by a sequence number. Example:

2.3.4.n

Subtitle: A short sentence indicating the purpose of the subprogram and possibly explaining its name, max 2 lines.

Language: The name of the programming language.

Key words: A few words associated with the subprogram.

Implementor: The name of the person who initially implemented the subprogram.

Date: Date of first implementation.

Institute: Name of Institute

Accepted: Date of acceptance in the library.

Version: Version number. Must be changed when the subprogram is revised.

The free-format part

In this part of the documentation, the different sections may have variable length. The sections are indicated by block letter headings which all are compulsory. The sections are divided into subsections. The heading of an empty subsection is omitted. The sections and the subsections are:

SCANDINAVIAN CONTROL LIBRARY

Program documentation

NAME:	NUMBER:
SUBTITLE:	
LANGUAGE:	
KEYWORDS:	
IMPLEMENTOR:	DATE:
INSTITUTE:	
ACCEPTED:	VERSION:

Figure 2. The fixed-format header

PURPOSE

The description of the action of the program. The formulas associated with the routine should be included, at least in the program library description. The connections between the ordinary problem notations and the names of the subroutine arguments should be stated.

Comments:

Information which guides the user of the library to select subroutines. Relations with other subroutines in the library and alternative subroutines could be given.

USAGE

This is a common section heading for the following:

Program type:

SUBROUTINE, FUNCTION, type FUNCTION or PROGRAM

Arguments:

The argument list is given in the form:

SUB(ARG1, ARG2, ...)

The arguments are ordered as described in rule 15.
Each of the arguments is described in the format:

arg - use, size, dimensioned, input-output,
description

The actual dimension of vectors and matrices etc. are given in the form:

size(NA), size(N1B, N2B), etc.

The dimension parameter of the matrices etc. are given in the form:

dimensioned(IB,.) , dimensioned(I1C, I2C,.) etc.

The input-output clause has three forms:

(I) Input parameter

(O) Output "

(I/O) Input and output parameter

This indication should be used for all arguments except for work areas.

A function identifier is described first in the same format.

Example:

NA - Dimension of matrix, (I)
A - Dynamics matrix, size(NA,NA), dimensioned(IA,.), (I)
PARAM - Parameter vector, size(3), (I)
(1): ...
(2): ...
(3): ...
SUB - Name of subroutine which computes ... (I)
IERR - Error indicator (O)
0: Success
1:

Common:

The common block statement is given followed by a description of the variables in the same way as for arguments.

User supplied subroutines:

The description of user written subroutines and functions shall have the format:

Program-type, program-name, argument-list,
purpose of the routine,
description of the arguments.

Example

SUBROUTINE FUNC(N,X,Y)

Defines the function Y = F(X) used in ...

N - Dimension of the vectors (I)

X - Argument, size (N), (I)

Y - Returned function value, size (N), (O)

The actual subroutine name must be declared EXTERNAL and entered in the argument list of ...

Read - Write:

Description of read and written variables including their format.

Notes:

Further information about the subroutine. Information about arguments which might be equivalenced.

Comments:

Less important information about the subroutine.

Examples:

Valuable if the use of the subprogram is exemplified.

METHOD

A short description of the method or the theory behind the subprogram. In the library documentation, formulas etc. should be given in common scientific notation.

References:

References if any ought to be given.

CHARACTERISTICS

Execution time:

Execution time for a specified problem on a specified computer.

Size:

Approximate storage requirements, code and internal variables, for the subroutine itself and its called internal subroutines (see below). The number of memory cells needed is given in decimal form.
The computer system should be specified.

Library subroutines required:

A list of called subroutines and functions, documented in the binder.

Internal subroutines required:

Routines required that are not part of the documentation binder.

Revisions:

The revisions of the subroutine are documented as:

1. name, date
   action

2. ...

An example of documentation is given in appendix A1.

Appendix A1

Example of documentation

SCANDINAVIAN CONTROL LIBRARY

Program documentation

NAME: .EIGEN	NUMBER:
SUBTITLE: Calculates the eigenvalues and eigenvectors of a real matrix	
LANGUAGE: FORTRAN IV	
KEYWORDS: Eigenvalues - eigenvectors - real matrix	
IMPLEMENTOR: O. Hallingstad	DATE: 1976-04-23
INSTITUTE: Eng. Cybernetics, NTH	
ACCEPTED:	VERSION: 1

PURPOSE

EIGEN calls subroutines which calculate the eigenvalues and eigenvectors of a real matrix. EIGEN then performs ordering of the eigenvalues with respect to the real parts: $Re\{\lambda_1\} \geq Re\{\lambda_2\} \geq \ldots Re\ \lambda_n$, and normalizes the eigenvectors. In order to avoid complex matrices in case of complex conjugated eigenvalues, the eigenvector matrix Z has a special form.

USAGE

Program type: SUBROUTINE

Arguments:

EIGEN(A,Z,ER,EI,N,IERR,MA,MZ,JWI,JWL,JWH,W)

A - Real matrix, size (N,N), dimensioned (MA, .),(I)

Z - Eigenvector matrix, size (N,N),dimensioned (MZ, .),(O)
If the i-th eigenvalue is real the i-th column of Z is the corresponding real eigenvector. If eigenvalues i and i+1 are a complex conjugated pair the columns i and i+1 contain the real and imaginary part of the eigenvector corresponding to the eigenvalue with positive imaginary part.

ER - Vector containing the real parts of the eigenvalues, size (N), (O)

EI - Vector containing the imaginary parts of the eigenvalues, size (N), (O)

N - Actual dimension of matrices and vectors, (I)

IERR - Error indicator, (O)

0 : Success

J : If a eigenvalue has not been determined.

MA - Dimension parameter of A

MZ - Dimension parameter of Z

JWI - Work vector, size (N)

JWL - Work vector, sixe (N)

JWH - Work vector, size (N)

W - Work vector, size (N)

METHOD

This algorithm is based upon several works presented by Num. Math. in the Handbook Series of Linear Algebra. [1] , [2] , [3].

It has been pointed out in a work by Osborne [4] that eigenvalue program results have errors at least of order $\varepsilon \cdot ||A||$, where ε is the machine precision and $||A||$ is the Euclidean norm of the given matrix A. Hence he recommends that one precedes the calling of such a routine by certain diagonal similarity transformations of A, designed to reduce its norm. This is done by calling the routine BALAN. BALAN [1] also has the property of detecting isolated eigenvalues. After the balancing subroutine ELMHE is called to obtain the upper Hessenberg transformed matrix, [2].

The subroutines ELTRA and HQR2 find the eigenvalues and eigenvectors of a real upper Hessenberg matrix by the QR-method [3]. The subroutine BALBA forms the eigenvectors of a real general matrix by back transforming those of the corresponding balanced matrix determined by BALAN.

The eigenvalues are then ordered after decreasing values of the real parts. At last each eigenvector is normalized so that the greatest element in each vector is equal to 1.0.

In order to avoid complex matrices in each case of complex conjugated eigenvalues, the eigenvalue matrix Z has the special form described in [5]. Its effect on the A matrix when used in a similarity transformation is demonstrated in the following example [5].

Example:

Let A has the eigenvalues λ_1, $\lambda_2=\sigma+j\omega$, $\lambda_3=\sigma-j\omega$, λ_4. Where λ_1 and λ_4 are real and λ_2 and λ_3 are complex conjugated. $\lambda_1 \geq \sigma \geq \lambda_4$

Then:

$$Z^{-1}AZ = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \sigma & \omega & 0 \\ 0 & -\omega & \sigma & 0 \\ 0 & 0 & 0 & \lambda_4 \end{bmatrix}$$

References

[1] Parlett, B.N. and C. Reinch: Balancing a Matrix for Calculation of Eigenvalues and Eigenvectors.
Numer. Math. 13, 293-304 (1969)

[2] Martin, R.S. and Wilkinson, J.H.: Similarity Reduction of a General Matrix to Hessenberg Form.
Numer. Math. 12, 349-368 (1968)

[3] Peters, G. and Wilkinson, J.H.: Eigenvectors of Real and Complex Matrices by LR and QR triangularization.
Numer. Math. 16, 180-204 (1970)

[4] Osborne, E. E.: On preconditioning of matrices Jour. ACM7, 338-345 (1960)

[5] Ogata, K. : State Space Analysis of Control Systems.
Prentice Hall (1967)

[6] Smith, Boyle, Garbow, Ikebe, Klema and Moler: Matrix Eigensystem Routines - EISPACK Guide.
Springer-Verlag (1974)

CHARACTERISTICS

Execution_time: Tested only on a NORD-10 computer in "Time-sharing" operation. With N = 15 the time used was less than 5 sec.

Size: The storage requirement for the subroutine and called subroutines which are not a part of the subroutine library, is 4930 cells on NORD-10 (16 bits wordlength).

Internal_subroutines_required: BALAN, ELTRA, HQR2, BALBA.

Appendix A2

Documentation of computer dependent subroutines:
MADD, MULT, RMULT, MMOVE, RMACON and IMACON.

SCANDINAVIAN CONTROL LIBRARY

Program documentation

NAME: MADD	NUMBER:
SUBTITLE: Matrix addition and subtraction	
LANGUAGE:	
KEYWORDS:	
IMPLEMENTOR:	DATE:
INSTITUTE:	
ACCEPTED:	VERSION:

PURPOSE

Computes C=A+B or C=A-B, where A,B and C are matrices or vectors.
Transpose(A) and/or transpose(B) may be used instead of A and/or B.

USAGE

Program Type SUBROUTINE

Arguments

MADD(A,B, C, M,N, MINUS,IAT,IBT, IA,IB,IC)

A - Matrix, dimensioned(IA,.), (I)
B - Matrix, dimensioned(IB,.), (I)
C - Matrix, size(M,N), dimensioned(IC,.), (O)
M - Number of rows in C, (I)
N - Number of columns in C, (I)
MINUS - Operation switch, (I). If non-zero A-B computed, else A+B.
IAT - Function switch, (I). If non-zero transpose(A) used instead of A.
IBT - Function switch, (I). If non-zero transpose(B) used instead of B.
IA - Declared first dimension of A, (I)
IB - Declared first dimension of B, (I)
IC - Declared first dimension of C, (I)

Notes

1) A may be the same matrix as C in the call if IAT=0 and B may be the same matrix as C if IBT=0.

CHARACTERISTICS

Execution Time

UNIVAC 1108: 33 + M*(5+N*5) µs

PDP 15: 245 + M*(35+N*40) µs

Size

UNIVAC 1108: 51

PDP 15: 90

SCANDINAVIAN CONTROL LIBRARY

Program documentation

NAME: MULT	NUMBER:
SUBTITLE: Matrix multiplication	
LANGUAGE:	
KEYWORDS:	
IMPLEMENTOR:	DATE:
INSTITUTE:	
ACCEPTED:	VERSION:

PURPOSE

Computes the matrix product C=A*B, where A,B and C are matrices or vectors. Transpose(A) and/or transpose(B) may be used instead of A and/or B.

USAGE

Program Type SUBROUTINE

Arguments

MULT(A,B, C, L,M,N, IAT,IBT,ISYM, IA,IB,IC)

A - Matrix, dimensioned(IA,.), (I)

B - Matrix, dimensioned(IB,.), (I)

C - Matrix, size(L,N), dimensioned(IC,.), (O)

L - Number of rows in C, (I)

M - Number of terms in scalar product, (I)

N - Number of columns in C, (I)

IAT - Function switch, (I). If non-zero transpose(A) used instead of A.

IBT - Function switch, (I). If non-zero transpose(B) used instead of B.

ISYM - Function switch, (I). If non-zero only the lower left triangular half of C is computed and C is symmetrized by copying the lower triangular half of C into the upper half.

IA - Declared first dimension of A, (I)

IB - Declared first dimension of B, (I)

IC - Declared first dimension of C, (I)

CHARACTERISTICS

Execution Time

UNIVAC 1108: 30 + N*(5+ L*(8+M*9)) µs

PDP 15: 250 + N*(35+ L*(55+M*35)) µs

Size

UNIVAC 1108: 92

PDP 15: 150

SCANDINAVIAN CONTROL LIBRARY

Program documentation

NAME: RMULT	NUMBER:
SUBTITLE: Real number multiplied with matrix	
LANGUAGE:	
KEYWORDS:	
IMPLEMENTOR:	DATE:
INSTITUTE:	
ACCEPTED:	VERSION:

PURPOSE

Computes B=R*A, where R is a scalar and A and B are matrices or vectors.

USAGE

Program Type SUBROUTINE

Arguments

RMULT(R,A, B, M,N, IA,IB)

R - Scalar operand, (I)
A - Matrix operand, size(M,N), dimensioned(IA,.), (I)
B - Matrix result, size(M,N), dimensioned(IB,.), (O)
M - Number of rows in A and B, (I)
N - Number of columns in A and B, (I)
IA - Declared first dimension of A, (I)
IB - Declared first dimension of B, (I)

Notes

1) A and B may be the same matrix in the call.

CHARACTERISTICS

Execution Time

UNIVAC 1108: 16 + N*(4+M*6) µs

PDP 15: 220 + N*(45+M*40) µs

Size

Univac 1108: 26

PDP 15: 85

SCANDINAVIAN CONTROL LIBRARY

Program documentation

NAME: MMOVE	NUMBER:
SUBTITLE: Move a matrix	
LANGUAGE:	
KEYWORDS:	
IMPLEMENTOR:	DATE:
INSTITUTE:	
ACCEPTED:	VERSION:

PURPOSE

Moves A to B or transpose(A) to B, where A and B are matrices or vectors.

USAGE

Program Type SUBROUTINE

Arguments

MMOVE(A, B, M,N, IAT, IA,IB)

A - Matrix operand, dimensioned(IA,.), (I)

B - Matrix result, size(M,N), dimensioned(IB,.), (O)

M - Number of rows in B, (I)

N - Number of columns in B, (I)

IAT - Function switch, (I). If non-zero (A)transpose moved instead of A.

IA - Declared first dimension of A, (I)

IB - Declared first dimension of B, (I)

CHARACTERISTICS

Execution Time

UNIVAC 1108: 15 + N*(4.5+M*1.5) µs

PDP 15: 220 + N*(25+M*20) µs

Size

UNIVAC 1108: 22

PDP 15: 90

SCANDINAVIAN CONTROL LIBRARY

Program documentation

NAME: IMACON	NUMBER:
SUBTITLE: Integer machine dependent constants	
LANGUAGE:	
KEYWORDS:	
IMPLEMENTOR:	DATE:
INSTITUTE:	
ACCEPTED:	VERSION:

PURPOSE

Returns machine-dependent integer constants.

USAGE

Program Type INTEGER FUNCTION

Arguments

IMACON(I)

IMACON - Returned integer constant.

I - Constant selector, (I)

1: Largest integer allowed

2: Largest single precision exponent allowed

3: Largest double precision exponent allowed

4: Integer word length (bits)

5: Single precision mantissa word length (except sign bit)

6: Single precision exponent word length (except sign bit)

7: Double precision mantissa word length (except sign bit)

8: Double precision exponent word length (except sign bit)

SCANDINAVIAN CONTROL LIBRARY

Program documentation

NAME: RMACON	NUMBER:
SUBTITLE: Real machine dependent constants	
LANGUAGE:	
KEYWORDS:	
IMPLEMENTOR:	DATE:
INSTITUTE:	
ACCEPTED:	VERSION:

PURPOSE

Returns machine-dependent real constants.

USAGE

Program Type REAL FUNCTION

Arguments

RMACON(I)

RMACON - Returned real constant

I - Constant selector, (I)

1: Relative single precision accuracy

2: Largest single precision number allowed

3: Smallest single precision number allowed (except 0.0)

4: Relative double precision accuracy