NBER WORKING PAPER SERIES

ROSEPACK Document No. 3

Guidelines for Writing Semi-portable FORTRAN

Neil E. Kaden*
Virginia Klema*

Working Paper No. 130

Computer Research Center for Economics and Management Science
National Bureau of Economic Research, Inc.
575 Technology Square
Cambridge, Massachusetts 02139

March 1976

*NBER Computer Research Center. Research supported in part by National Science Foundation Grant #DCR 75-08802 to the National Bureau of Economic Research, Inc.

Abstract

Transferring Fortran subroutines from one manufacturer's machine to another, or from one operating system to another, puts certain constraints on the construction of the Fortran statements that are used in the subroutines. The reliable performance of mathematical software should be unaffected by the host environment in which the software is used or by the compiler from which the code is generated. In short, the reliable performance of the algorithm is to be independent of the computing environment in which it is run.

The subroutines of ROSEPACK (Robust Statistics Estimation Package) are Fortran IV source code designed to be semi-portable where semi-portable is defined to mean transportable with minimum change.* This paper described the guidelines by which ROSEPACK subroutines were written.

Acknowledgements

The authors wish to thank J. Boyle and W. Cody for sharing their internal document on programming conventions; J. Kirsch for his helpful suggestions on documentation of subroutines, John Dennis, Jim Ferris, David Gay, David Hoaglin and Gerald Ruderman for their constructive criticism of this document, and Karen Glennon for her careful typing of the manuscript.

---

*Cody, W.J., "The Construction of Numerical Subroutine Libraries," SIAM Review, Vol. 16, No. 1, pp. 36-46, January (1974)

Table of Contents

Introduction

Semi-portability of Fortran subroutines puts certain constraints on the construction of Fortran statements, the declaration of variables, and the representation of constants that are used in the subroutines. Many of these constraints are needed for Fortran subroutines that are to be imbedded in applications subsystems that are written in another language, say PL/1.

Frequently, the numerical algebra algorithms themselves are strengthened when their performance is unaffected by the arithmetic of the machine on which they are used and the Fortran compiler by which their code is generated.

The rules for structured programming [1,2,3] and structured documentation (see Part Two, Section X of this document) should be followed insofar as possible. The comments within the program or subroutine should be sufficient to inform the user about input parameters, output parameters, temporary storage parameters, error exits, and the algorithm that the program implements. Sample programs are given in the Appendix.

This document presents certain suggestions for programming that will tend toward requirements for semi-portability of ANSI Fortran IV (as described in CACM, Vol. 7, No. 10, October '64) subroutines and programs. We also suggest certain conventions for comments and general formatting of the Fortran code. By "formatting" we mean the spacing and indentation that determine the general appearance and readability of the code. Such formatting is suggested to help the reader or the user understand the algorithm, the program, and the flow of control within the program.

Part One

Some guidelines for programming are

I. CØMMØN storage should not be used for arrays. This is not an ANSI restriction, but driver programs become simpler to write, and the use in a paged environment is enhanced if one does not use CØMMØN. CØMMØN storage requires a particular ordering of declarations so that boundary alignments conform. Arrays in CØMMØN must have fixed dimensions at compile time which may cause excessive storage allocation or a requirement to recompile if a user needs arrays of dimensions higher than those originally defined in the CØMMØN block of storage.

II. All array arguments should have adjustable dimensions. These dimensions should be made explicit in the declarations of the formal parameters for each subroutine. For example,

```
REAL   A(NM,N)
```

not

```
REAL   A(NM,1)
```

The latter construction may be rejected by some compilers.

III. EQUIVALENCE statements should not be used: EQUIVALENCE statements may inhibit automatic conversion between single and double precision.

IV. Certain Fortran compilers do not distinguish more than six characters of an identifier. Hold identifiers to six characters or fewer.

V. Do not use multiple entry points or non-standard returns. Such constructions are not necessarily available for all compilers.

VI. Be sure that the precision of any Fortran library routine or built-in function is explicit in all statements. For example, DABS, not ABS, for absolute value for long precision computing. Do not use mixed mode arithmetic or assume there is an implied conversion anywhere, not even for constants. For example

```
DØUBLE PRECISION X
X = X + 10.0D0
```

not

```
DØUBLE PRECISION X
X = X + 10.0
```

VII. Constants that are used in iterations or convergence criteria should be functions of the machine's precision, i.e., the smallest floating point number, ε, representable in the machine for which the floating representation of $1+\varepsilon \neq 1$. Certainly, a constant that cannot be converted precisely on the machine should never be used. For example, .1, is representative of such numbers. Numbers that can be represented exactly are integer powers of the base of the arithmetic of the computing machine. Such numbers must be within the range of arithmetic of the computing machine.

VIII. Test cases must be devised so that data can be converted uniformly on all target machine. The word length of the machine determines the truncation of the internal representation of floating point numbers, and conversion routines do not treat floating point numbers uniformly. The integers are treated uniformly with respect to conversion so long as they lie within the precision range of arithmetic of the computing machine. One suggestion for portability for test cases used as input numbers is to read them in as integers and then DEFLØAT to get the internal floating point representations. It is extremely important that the internal representation of data for test cases represent the same numbers on all target machines. Different internal representations can create perturbations (on input data) sufficient to cause changes in computed solutions that are awkward to analyze.

IX. Obscure underflows can often produce side effects that give divide checks or overflows. This problem is particularly acute because the range of arithmetic on many machines is not symmetric about zero. For example, the range of arithmetic on the IBM 360/370 machines is about $10^{75} > |x| > 10^{-78}$. On the IBM 360/370 machines, the number 10^{-77} is within the tolerated range of arithmetic; the reciprocal of 10^{-77} is not within the machine's range of arithmetic. That an algorithm will exhibit overflow, underflow, or divide check problems is often not known in advance; moreover, some linear systems routines have this problem when an inner product is formed or when multiple multiplies are encountered. Be prepared to isolate such problems. Different operating systems and different computing machines do not treat underflow interrupts uniformly. The variations in underflow processing range from ignoring the underflow completely to causing a termination from the operating system. One solution to avoid underflow is to reorder arthmetic expressions. Another solution is to resort to extended precision arithmetic for critical sections of code.

X. The usual rules for separate sections for error handling and input-output that are required for applications subsystems are equally applicable for semi-portable Fortran programs. For convenience, the error handling from the subroutines should be uniform. The input-output should be confined to main programs or special I-Ø subroutines; that is to say, computation

subroutines must be I-Ø free. The goal of the error recovery is to permit computation to continue without resorting to system termination, yet give the user information about the performance of the algorithm on his problem. See the example of IERR in the QRF subroutine in the Appendix.

XI. We expect the subroutines in ROSEPACK to be compiled with Fortran compilers with the highest level of optimization. We have not used hand optimization in the subroutines. That we do not use hand optimization is a matter of style that we believe enhances the exposition of the algorithm. The indirection of addressing in QRF is an example of such exposition.

Part Two

The suggestions for formatting are

I. Identifiers

Identifiers, i.e., variable names, should correspond to default declarations in Fortran. However, explicit declarations should be written for each identifier.

Variables from the calling sequence, internal variables, and function names should all be declared separately. For a suggestion on how to accomplish this see X Internal Documentation Section B (PARAMETERS), Section C (LOCAL VARIABLES), and Section D (FUNCTIONS) of the description of the Prologue.

II. Labels

If the order of labels within a subroutine is not linear the convention used should be explicitly described. This ordering of statement labels should be linear and could proceed in multiples of 10 for interior program sections. The next level of program section could proceed as 100, and perhaps the next as 1000.

Do not use unreferenced labels. Warning messages are given by some compilers for unreferenced labels.

Code for error exits should be surrounded by comments and located at the end of the program or subroutine so that error exits are easy to find. The labels for error exists should be 2 or 3 digits, the first of which is 9, the last non-zero. The convention for labels is arbitrary.

One format statement may be used by more than one print statement in a program. Therefore we suggest that all format statements be labeled with 4 digit numbers the last of which is non-zero and placed after the RETURN statement and before the END statement of the program.

Preferably all input-output should be written in subroutine form. We suggest that a DATA statement be used to fix units of I-Ø and that this DATA statement be made particular to a given installation. An example of such a construction in a driver program could be

```
DATA  IØIN /5/
DATA  IØØUT /6/
```

The variables containing this I-Ø unit information should be passed as parameters to all subroutines using these I-Ø units. This device allows a global change of an I-Ø unit without recompiling individual subroutines.

III. Use of blank spaces

There should not be extra blank spaces around dummy variables or constants in DØ loops. Blank spaces should delimit = symbols in assignment statements. This use of blank spaces avoids confusion between the use of = for assignment and for indexing. Blank spaces should be used wherever such use will enhance readability of elements of expressions or statements.

IV. Tab Spacings

Throughout this document we are assuming tab setting in columns 1, 7, 10, 15, 20, 25, etc. The tab spacing is a matter of style and convenience in reading.

V. Continuation Characters

Second and subsequent lines of all continuation statements should be numbered 1 through 9, then A through Z in column 6. The text of each continuation statement should be indented one tab space from the initial line of the statement. The convention for continuation characters and tab spacing is imposed for consistent style.

IV. DØ loops

All DØ loops should be surrounded by comment statements which may be blank. Text comments should follow a blank comment statement. If more than one statement is in the range of a DØ loop, the closing statement of the DO loop should be a CONTINUE. This CØNTINUE should be unambiguous. Statements between DØ and CØNTINUE should be indented one additional tab space to correspond to a block structure.

Inner loops should be indented one tab space to the right of their surrounding outer loop.

For examples of indentation of DØ loops see the examples in the Appendix.

VII. DATA Statements

Data statements should be used to set installation-dependent constants, such as data-set numbers for I/Ø, and machine precision, underflow tolerances, or other machine-dependent constants. See X, Internal Documentation, Section L, for more details.

If a non-numeric character string must be used in a DATA statement, it should be packed as one character per machine word and always stored in an array since different machines allow a varying number of characters to be stored in one machine word.

VIII. Structured programming

The programming and formatting conventions that we describe are similar to structured programming in the following ways:

1. format is for readability and understanding
2. indentation is done for major and minor loops
3. array dimensions are adjustable
4. temporary storage arrays are passed as parameters
5. documentation is structured such that it is contained within the routine.

IX. Printed output

In order to aid in reproduction of the output, all printed output should be formatted such that it is not greater than 8 1/2 inches in width. Most line printers print 10 characters per inch, and 80 characters per line allows ample margins.

X. Internal Documentation

All Fortran programs should be well documented by liberal use of comment statements. Proper documentation will enhance readability and appearance of the code, improve understanding of the algorithm used, help ensure proper use of the program, and aid in future modifications. When semi-portability is also considered, proper documentation serves to isolate those portions of the code which are installation-dependent.

In-line documentation of Fortran programs in ROSEPACK has been considered in two major sections, the Prologue and the Program-flow comments. The latter consists of the comment statements embedded within the code describing how the algorithm is being carried out as the flow of control passes from statement to statement. The former consists of certain non-executable Fortran statements found at the beginning of the subprogram which fully describe the proper use of the software, as well as information concerning its development. Any user familiar with the guidelines has the added advantage of knowing where to find specific information concerning the program. The Prologue also identifies and isolates installation-dependent aspects of the program and thus enhances semi-portability. The Prologue is the major documentation for the use of the program or subroutine.

The Prologue consists of the declarations of the calling sequence and variable names of the subprogram, a number of sections of text on the subprogram, and any DATA statements. It contains a number of headings denoting the different logical sections of the Prologue. The headings are comment statements with the character "*" in columns 7 through 11 and the heading name beginning in solumn 12 and followed by a colon (:). A blank comment statement should not immediately follow a heading. If the section denoted by the heading line is empty, the heading should be followed by a comment statement containing "NONE" in columns 7 through 10 and then a blank comment statement.

The different headings, in the order they should appear, are:

-PARAMETERS
-LOCAL VARIABLES
-FUNCTIONS
-PURPOSE
-PARAMETER DESCRIPTION
-APPLICATION AND USAGE RESTRICTIONS
-ALGORITHM NOTES
-REFERENCES
-HISTORY
-GENERAL
-BODY OF PROGRAM

Six delimiter lines, consisting of two blank comment statements, two comment statements consisting of the special character colon (:) in columns 7 through 72, and then two more blank comment statements, should occur immediately before the PURPOSE heading and immediately after the GENERAL section. All lines between these delimiters should be comment statements. When columns 73 through 80 of each line contain serialization of identification characters, this gives a box-like appearance to the part of the Prologue containing text.

An example of two programs following these guidelines is in the Appendix.

What follows is a brief description of each section of the Prologue. Note that no blank comment statements should occur until after the FUNCTIONS section.

A. CALLING SEQUENCE

The SUBRØUTINE or FUNCTIØN statement should be the first line of the subprogram. Blanks should be used to enhance readability.

B. PARAMETERS

Declaration statements should be grouped by type, i.e., first INTEGER, then REAL, then DØUBLE PRECISION, or REAL*8,

then REAL*16, then CØMPLEX, CØMPLEX*32, then LØGICAL. Within each type grouping, variable names should be listed in the order they occur in the calling sequence. By parameters, we mean all of the variable names appearing in the calling sequence.

C. LOCAL VARIABLES

As with the preceding section, declaration statements are grouped by type, and in the same order. Within each type, variable names should be listed alphabetically.

ALL variables used in the program should be explicitly declared.

D. FUNCTIONS

All functions called by the program should be explicitly declared. Declaration statements are grouped by type.

E. PURPOSE

Briefly describe the purpose of this subprogram. Give references when necessary. More detail can be given in later sections.

F. PARAMETER DESCRIPTION

This section contains 3 subsections. The first describes input parameters, the second describes output parameters, and the third subsection describes parameters used for temporary storage by the subprogram. If the contents of any parameter variable can be changed by the subprogram, it should be considered an output parameter. See the examples in the Appendix for the format, keywords, punctuation and indentation used in this section.

G. APPLICATION AND USAGE RESTRICTIONS

If any other programs in this package can call this subprogram, or are called by it, they should be described here. If this subprogram is part of a group of programs which are called in some specified order, this should also be included. Give references except when a reference is implicit, as with another member of the same package.

Also included in this section are any warnings about special cases or possible errors which can occur if there are errors in the subprogram call. Warnings about misuse of tolerance parameters belong here. The entry in PARAMETER DESCRIPTION should refer the reader to this section where applicable.

H. ALGORITHM NOTES

Anything special about the algorithm used or its implementation should be listed here. Any special conventions regarding statement labeling or commenting should be mentioned. If there is anything special about error handling which has not yet been mentioned, it should be described here.

I. REFERENCES

References from elsewhere in the documentation, as well as any other references pertaining to the subprogram, should be listed.

J. HISTORY

The author of this subprogram, we well as the date and place of origin should be listed. If the subprogram is a translation of a program in another language or is based on another program, a reference should be given. If the program has been modified since it was written, the date and person making the modification whould be noted. If this subprogram has been released as part of a subroutine library, the current release data of the library, and machine version, should be given.

K. GENERAL

If this subprogram was developed under research supported by a grant requiring acknowledgement, the required information should occur here. The person to contact concerning comments and problems with the subprogram should have his address in this section.

L. DATA Statements

Following the second occurrence of delimiting comment statements (two blank comment statements, two comment statements with colons in columns 7 through 72, and two more blank comment statements) is where all DATA statements should occur. If a DATA statement contains an installation-dependent constant, comment statements explaining its value and mentioning the installation's designation, should precede the DATA statement. Those comment statements should conform to the standards of program-flow comments.

M. BODY OF PROGRAM

This heading denotes the end of the Prologue and the beginning of the program body.

Program-flow comments should be delimited by special characters to enhance their readability and appearance. In ROSEPACK the colon (:) is used. Such comments should also follow the rules for statement indentation described elsewhere in this document.

In most cases, the text of the comment should be preceded and followed by a string of 10 special characters (colons). At least one blank space, but not more than three blank spaces, should be put between the special character strings and the text of the comment. If this method is used for a comment extending over several lines, all lines whould have a "C" in column 1, and all but the first line should be indented one additional tab (beyond the current level of indentation).

Building on the suggestions of Boyle and Cody, an alternative method of delimiting comments, recommended for important comments or those extending over several lines, is to surround them by a "box" of special characters. The following is an example of what is meant by a "box":

```
C        ::::::::::::::::::::::::::
C        ::                       ::
C        ::                       ::
C        ::  THIS IS A COMMENT    ::
C        ::                       ::
C        ::::::::::::::::::::::::::
```

Blank comment statements (i.e., comment statements containing blanks in columns 2 through 72) may be used wherever their use enhances the readability of the program.

The statement immediately before the END statement should always be a comment statement delimiting the end of the program and containing the name of the program. The consistent wording can help in separating several subroutines in one file. An example follows:

```
         RETURN
C        ::::::::::    LAST CARD OF (NAME OF SUBROUTINE) ::::::::::
         END
```

References

[1] Dahl, O.H, Dijkstra, E.W., Hoare, C.A.R., Structured Programming, Academic Press, (1972).

[2] Kerningham, B.W., Plauger, P.J., "Programming Style: Examples and Counter-Examples," ACM Computing Surveys, Vol. 6, No. 4, pp. 303-319, December, (1974)

[3] Kerninghan, B.W., Plauger, P.J., "The Elements of Programming Style," Bell Telephone Laboratories (1974)

Appendix

This appendix contains listings of two subroutines that are samples of candidates for inclusion in ROSEPACK. The reader is reminded that we are relying on Fortran compiler optimization of sub-expressions within loops.

```
      SUBROUTINE QRF(NM,M,N,QR,ALPHA,IPIVOT,IERR,IHTTC,Y,SUM)
C     *****PARAMETERS:
      INTEGER NM,M,N,IPIVOT(N),IERR,IHTTC(N)
      REAL*8  QR(NM,N),ALPHA(N),Y(N),SUM(N)
C     *****LOCAL VARIABLES:
      INTEGER I,J,JBAR,K,K1,MINUM
      REAL*8  ALPHAK,BETA,QRKK,QRKMAX,SIGMA,TEMP,UFETA,UFTOL
C     *****FUNCTIONS:
      INTEGER MINO
      REAL*8  DABS,DSQRT
C
C
C     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
C
C     *****PURPOSE:
C
C     THIS SUBROUTINE DOES A QR-DECOMPOSITION ON THE M X N MATRIX QR,
C        WITH AN OPTIONALLY MODIFIED COLUMN PIVOTING, AND RETURNS THE
C        UPPER TRIANGULAR R-MATRIX, AS WELL AS THE ORTHOGONAL VECTORS
C        USED IN THE TRANSFORMATIONS.
C
C     *****PARAMETER DESCRIPTION:
C     ON INPUT:
C
C        NM MUST BE SET TO THE ROW DIMENSION OF THE TWO DIMENSIONAL
C             ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C             DIMENSION STATEMENT;
C
C        M MUST BE SET TO THE NUMBER OF ROWS IN THE MATRIX;
C
C        N MUST BE SET TO THE NUMBER OF COLUMNS IN THE MATRIX;
C
C        QR CONTAINS THE REAL RECTANGULAR MATRIX TO BE DECOMPOSED;
C
C        IHTTC IS USED TO CONTROL PIVOTING. IF IHTTC(J) IS SET TO
C             ZERO FOR ALL J THEN NORMAL COLUMN PIVOTING IS USED.
C             IF IHTTC(J) IS LESS THAN 0 THEN THE J-TH COLUMN OF THE
C             INPUT MATRIX CANNOT BE IN THE FIRST N COLUMNS OF THE
C             QR-DECOMPOSITION.  ALL ELEMENTS OF THIS VECTOR SHOULD
C             BE SET TO ZERO TO INSURE NORMAL PIVOTING;
C
C        Y IS USED FOR TEMPORARY STORAGE FOR THE SUBROUTINE;
C
C        SUM IS USED FOR TEMPORARY STORAGE FOR THE SUBROUTINE.
C
C     ON OUTPUT:
C
C        QR CONTAINS THE NON-DIAGONAL ELEMENTS OF THE R-MATRIX
C             IN THE STRICT UPPER TRIANGLE. THE VECTORS U, WHERE
```

```
C              Q-TRANSPOSE = IDENT - BETA * U * U-TRANSPOSE,
C              ARE IN THE COLUMNS OF THE LOWER TRIANGLE;
C
C         ALPHA CONTAINS THE DIAGONAL ELEMENTS OF THE R-MATRIX;
C
C         IPIVOT REFLECTS THE COLUMN PIVOTING PERFORMED ON THE INPUT
C              MATRIX TO ACCOMPLISH THE DECOMPOSITION. THE J-TH
C              ELEMENT OF IPIVOT GIVES THE COLUMN OF THE ORIGINAL
C              MATRIX WHICH WAS PIVOTED INTO THAT COLUMN DURING THE
C              DECOMPOSITION;
C
C         IERR IS SET TO:
C              0 FOR NORMAL RETURN,
C              K IF NO NON-ZERO PIVOT COULD BE FOUND FOR THE K-TH
C                   TRANSFORMATION, OR
C              -K FOR AN ERROR EXIT ON THE K-TH THANSFORMATION.
C              IF AN ERROR EXIT WAS TAKEN, THE FIRST (K - 1)
C              TRANSFORMATIONS ARE CORRECT.
C
C
C     *****APPLICATIONS AND USAGE RESTRICTIONS:
C     THE FORTRAN SUBROUTINE QRSOL SOLVES A SYSTEM AX=B BY USING
C     QRF.
C
C     *****ALGORITHM NOTES:
C     THIS VERSION OF QRF TRIES TO ELIMINATE THE OCCURRENCE OF
C     UNDERFLOWS DURING THE ACCUMULATION OF INNER PRODUCTS.
C
C     ADAPTED FROM THE ALGOL ROUTINE SOLVE (1).
C
C     *****REFERENCES:
C     (1)      BUSINGER,P. AND GOLUB,G.H., LINEAR LEAST SQUARES
C     SOLUTIONS BY HOUSHOLDER TRANSFORMATIONS, IN WILKINSON,J.H.
C     AND REINSCH,C.(EDS.), HANDBOOK FOR AUTOMATIC COMPUTATION,
C     VOLUME II: LINEAR ALGEBRA, SPRINGER-VERLAG, 111-118 (1971);
C     PREPUBLISHED IN NUMER.MATH. 7, 269-276 (1965).
C
C     *****HISTORY:
C     ROSEPACK RELEASE 0.2      FEBRUARY 1976
C     IBM 360/370 VERSION
C     DOUBLE PRECISION DECK
C
C     WRITTEN BY NEIL E. KADEN (NBER/CRC) 1974
C     MODIFIED 20 APRIL, 1975 BY N. KADEN
C
C     *****GENERAL:
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO:
C        SUPPORT STAFF MANAGER
C        COMPUTER RESEARCH CENTER FOR ECONOMICS AND MANAGEMENT SCIENCE
C        NATIONAL BUREAU OF ECONOMIC RESEARCH
C        575 TECHNOLOGY SQUARE
C        CAMBRIDGE, MASS. 02139.
```

```
C
C     DEVELOPMENT OF THIS PROGRAM SUPPORTED IN PART BY
C     NATIONAL SCIENCE FOUNDATION GRANT GJ-1154X3 AND
C     NATIONAL SCIENCE FOUNDATION GRANT DCR75-08802
C     TO NATIONAL BUREAU OF ECONOMIC RESEARCH, INC.
C
C
C
C     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
C
C     ::::::::::  UFETA IS THE SMALLEST POSITIVE FLOATING POINT NUMBER
C        S.T. UFETA AND -UFETA CAN BOTH BE REPRESENTED.
C        IBM 360/370: UFETA = 16.**-65  ::::::::::
      DATA UFETA /Z0010000000000000/
C
      UFTOL = DSQRT(UFETA)
C
C     *****BODY OF PROGRAM:
      IERR = 0
C
      DO 40 J=1,N
C        ::::::::::J-TH COLUMN SUM::::::::::
         QRKMAX = 0.0D0
C
         DO 10 I=1,M
              IF (DABS( QR(I,J) ) .GT. QRKMAX) QRKMAX = DABS( QR(I,J) )
   10    CONTINUE
C
         SUM(J) = 0.0D0
         IF (QRKMAX .LT. UFTOL)  GO TO 30
C
         DO 20 I=1,M
              IF (DABS( QR(I,J) ) .LE. QRKMAX * UFTOL)  GO TO 20
              TEMP = QR(I,J) / QRKMAX
              SUM(J) = SUM(J) + TEMP * TEMP
   20    CONTINUE
C
         SUM(J) = QRKMAX * QRKMAX * SUM(J)
   30    CONTINUE
         IPIVOT(J) = J
   40 CONTINUE
C
      MINUM = MIN0(M,N)
C
      DO 200 K=1,MINUM
C        ::::::::::K-TH HOUSEHOLDER TRANSFORMATION::::::::::
         SIGMA = 0.0D0
         JBAR = 0
C        ::::::::::FIND LARGEST COLUMN SUM::::::::::
         DO 110 J=K,N
              IF (IHTTC( IPIVOT(J) ) .LT. 0)  GO TO 110
```

```
                IF (SIGMA .GE. SUM(J))  GO TO 110
                SIGMA = SUM(J)
                JBAR = J
  110       CONTINUE
C
            IF (JBAR .EQ. 0)  GO TO 999
            IF (JBAR .EQ. K)  GO TO 130
C           ::::::::::COLUMN INTERCHANGE::::::::::
            I = IPIVOT(K)
            IPIVOT(K) = IPIVOT(JBAR)
            IPIVOT(JBAR) = I
            SUM(JBAR) = SUM(K)
            SUM(K) = SIGMA
C
            DO 120 I=1,M
                SIGMA = QR(I,K)
                QR(I,K) = QR(I,JBAR)
                QR(I,JBAR) = SIGMA
  120       CONTINUE
C           ::::::::::END OF COLUMN INTERCHANGE::::::::::
  130       CONTINUE
C           ::::::::::  SECOND INNER PRODUCT  ::::::::::
            QRKMAX = 0.0D0
C
            DO 140 I=K,M
                IF (DABS( QR(I,K) ) .GT. QRKMAX) QRKMAX = DABS( QR(I,K) )
  140       CONTINUE
C
            SIGMA = 0.0D0
            IF (QRKMAX .LT. UFTOL)  GO TO 998
C
            DO 150 I=K,M
                IF (DABS( QR(I,K) ) .LE. QRKMAX * UFTOL)  GO TO 150
                TEMP = QR(I,K) / QRKMAX
                SIGMA = SIGMA + TEMP * TEMP
  150       CONTINUE
C
            SIGMA = QRKMAX * QRKMAX * SIGMA
C           ::::::::::  END SECOND INNER PRODUCT  ::::::::::
            IF (SIGMA .EQ. 0.0D0)  GO TO 998
            QRKK = QR(K,K)
            ALPHAK = DSQRT(SIGMA)
            IF (QRKK .GE. 0.0D0)  ALPHAK = -ALPHAK
            ALPHA(K) = ALPHAK
            BETA = 1.0D0 / (SIGMA - QRKK*ALPHAK)
            QR(K,K) = QRKK - ALPHAK
            K1 = K + 1
            IF (K1 .GT. N) GO TO 200
C
            DO 170 J=K1,N
                TEMP = 0.0D0
C
                DO 160 I=K,M
                    TEMP = TEMP + QR(I,K)*QR(I,J)
```

```
  160          CONTINUE
C
               Y(J) = BETA * TEMP
  170     CONTINUE
C
C
          DO 190 J=K1,N
C
               DO 180 I=K,M
                    QR(I,J) = QR(I,J) - QR(I,K)*Y(J)
  180          CONTINUE
C
               SUM(J) = SUM(J) - QR(K,J)**2
  190     CONTINUE
C         ::::::::::END OF K-TH HOUSEHOLDER TRANSFORMATION::::::::::
  200 CONTINUE
C
      GO TO 1000
C
C     ::::::::::ERROR EXIT ON K-TH TRANSFORMATION::::::::::
  998 IERR = -K
      GO TO 1000
C
C     ::::::::::NO NON-ZERO ACCEPTABLE PIVOT FOUND::::::::::
  999 IERR = K
C
C     ::::::::::RETURN TO CALLER::::::::::
 1000 RETURN
C     ::::::::::LAST CARD OF QRF::::::::::
      END
```

```
      SUBROUTINE WANDRW(N,U,CONST,SQW)
C     *****PARAMETERS:
      INTEGER N
      REAL*8  U(N),CONST,SQW(N)
C     *****LOCAL VARIABLES:
      INTEGER I
      REAL*8  C1,OFLIM,PI,RKTOL,UFETA,UPLIM,U1
C     *****FUNCTIONS:
      REAL*8  DABS,DSIN,DSQRT
C
C
C     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
C
C     *****PURPOSE:
C     THIS SUBROUTINE PRODUCES THE SQUARE ROOTS OF THE WEIGHTS
C     DETERMINED BY THE INPUT VECTOR U OF PREVIOUSLY COMPUTED
C     SCALED RESIDUALS AND THE ANDREWS WEIGHT FUNCTION. (REFERENCE 1)
C
C     *****PARAMETER DESCRIPTION:
C     ON INPUT:
C
C        N MUST BE SET TO THE NUMBER OF ELEMENTS IN THE VECTORS U AND
C             SQW;
C
C        U CONTAINS THE STANDARDIZED RESIDUALS FROM A PREVIOUS LINEAR
C             FIT.  THAT IS, U(I) = R(I) / S WHERE R(I) IS THE I-TH
C             RESIDUAL FROM A LINEAR FIT, R(I) = Y(I) - YFITTED(I),
C             AND S = S(R) IS A RESIDUAL SCALING FINCTION (E.G. S COULD
C             BE THE OUTPUT OF THE FORTRAN SUBROUTINE SMAD).
C
C        CONST IS THE 'TUNING CONSTANT' FOR THE WEIGHT FUNCTION
C             W(U). IF CONST IS NOT POSITIVE, OR EXCEEDS UPLIM, AN
C             ERROR EXIT WILL BE TAKEN.  (SEE APPLICATION AND USAGE
C             RESTRICTIONS)
C
C
C     ON OUTPUT:
C
C        CONST IS UNCHANGED IF NO ERROR EXIT OCCURRED.
C             IF CONST WAS LESS THAN OR EQUAL TO ZERO, THEN
C             CONST REMAINS UNCHANGED.
C             IF CONST WAS GREATER THAN THE ALLOWABLE UPPER
C             LIMIT, THEN CONST IS SET TO MINUS ONE.
C
C
C        SQW CONTAINS A VECTOR OF THE SQUARE ROOTS OF THE WEIGHTS
C             DETERMINED BY THE SCALED RESIDUALS AND THE WEIGHTING
C             FUNCTION.
C
C     *****APPLICATION AND USAGE RESTRICTIONS:
```

```
C     THE ROOT-WEIGHTS ARE NEEDED FOR THE COMPUTATION OF THE
C     ITERATIVELY REWEIGHTED LEAST SQUARES ESTIMATES USING THE
C     FORTRAN SUBROUTINES MINFIT AND MINSOL.  IN THIS COMPUTATION
C     SQW(I) MULTIPLIES THE CORRESPONDING ROWS OF THE X-MATRIX
C     AND THE Y-VECTOR. (REFERENCE 2)
C
C     THE LARGER THE VALUE OF CONST, THE MORE NEARLY ALL THE VALUES
C     OF W(U) WILL EQUAL UNITY.
C
C     IF CONST IS TAKEN TO BE VERY SMALL IT IS POSSIBLE TO PRODUCE A
C     VECTOR OF ROOT-WEIGHTS ALL OF WHICH EQUAL OR NEARLY EQUAL
C     ZERO, AND THIS WILL BE USELESS AS INPUT TO THE WEIGHTED LEAST
C     SQUARES COMPUTATIONS.
C
C     IF A TUNING CONSTANT VALUE OF 1.339 IS USED, UNDER THE
C     ASSUMPTION OF GAUSSIAN ERRORS, THE RESULTING ESTIMATOR
C     WILL HAVE 95 PERCENT ASYMPTOTIC EFFICIENCY.
C
C     IT IS POSSIBLE TO COMPUTE SQW WHEN CONST EXCEEDS THE UPPER LIMIT
C     IMPOSED, BUT AS THIS REQUIRES ADDITIONAL TESTS FOR UNDERFLOWS AND
C     OVERFLOWS, IT WILL BE IMPLEMENTED WHEN THIS CAPABILITY IS
C     JUSTIFIED.
C
C     WARNING - AS AN ERROR EXIT MAY CAUSE THE VALUE OF CONST TO BE
C     OVERWRITTEN, IF A CONSTANT (RATHER THAN A VARIABLE) WAS PASSED BY
C     THE CALLING PROGRAM, THEN THE CALLER'S CONSTANT TABLE MAY
C     BE CHANGED, CAUSING INCORRECT BEHAVIOR OF THE PROGRAM.
C
C     *****ALGORITHM NOTES:
C     THE INPUT PARAMETERS ARE CHECKED TO AVOID UNDERFLOWS AND
C     OVERFLOWS.
C     FOR SUFFICIENTLY SMALL VALUES OF U(I) AN APPROXIMATION BY POWER
C     SERIES EXPANSION IS USED.
C
C     *****REFERENCES:
C     (1)      ANDREWS,D.F. (1974), TECHNOMETRICS 16, 523-532.
C     (2)      BEATON,A.E. AND TUKEY,J.W.(1974), TECHNOMETRICS 16,
C        147-192.
C
C     *****HISTORY:
C     ROSEPACK RELEASE 0.2      FEBRUARY 1976
C     IBM 360/370 VERSION
C     DOUBLE PRECISION DECK
C
C     WRITTEN BY NEIL KADEN (NBER / COMPUTER RESEARCH CENTER)
C     JUNE 23, 1975.
C     MODIFIED 23 JULY, 1975 BY NEIL KADEN
C     MODIFIED 29 OCTOBER, 1975 BY NEIL KADEN
C
C     *****GENERAL:
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO:
C        SUPPORT STAFF MANAGER
C        COMPUTER RESEARCH CENTER FOR ECONOMICS AND MANAGEMENT SCIENCE
C        NATIONAL BUREAU OF ECONOMIC RESEARCH
```

```
C          575 TECHNOLOGY SQUARE
C          CAMBRIDGE, MASS. 02139.
C
C      DEVELOPMENT OF THIS PROGRAM SUPPORTED IN PART BY
C      NATIONAL SCIENCE FOUNDATION GRANT GJ-1154X3 AND
C      NATIONAL SCIENCE FOUNDATION GRANT DCR75-08802
C      TO NATIONAL BUREAU OF ECONOMIC RESEARCH, INC.
C
C
C      ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C      ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
C
C      ::::::::::  OFLIM IS THE LARGEST POSITIVE FLOATING POINT NUMBER
C         IBM 370/360: OFLIM = (16.**63)*(1. - 16.**-14)  ::::::::::
       DATA OFLIM /Z7FFFFFFFFFFFFFFF/
C
C      ::::::::::  PI = 3.14...  ::::::::::
       DATA PI /3.14159265358979D0/
C
C      ::::::::::  RKTOL IS THE SQUARE ROOT OF THE RELATIVE PRECISION
C         OF FLOATING POINT ARITHMETIC (MACHEP).
C         IBM 360/370: RKTOL = 2.**-26  ::::::::::
       DATA RKTOL /Z3A40000000000000/
C
C      ::::::::::  UFETA IS THE SMALLEST POSITIVE FLOATING POINT NUMBER
C         S.T. UFETA AND -UFETA CAN BOTH BE REPRESENTED.
C         IBM 360/370: UFETA = 16.**-65  ::::::::::
       DATA UFETA /Z0010000000000000/
       UPLIM = OFLIM / PI
C
C      *****BODY OF PROGRAM:
       IF (CONST .LE. 0.0D0)  RETURN
C      ::::::::::  ERROR EXIT IF CONST IS NOT POSITIVE  ::::::::::
       IF (CONST .LE. UPLIM)  GO TO 10
C      ::::::::::ERROR EXIT IF CONST IS TOO LARGE  ::::::::::
       CONST = -1.0D0
       RETURN
C      ::::::::::  CONST IS IN RANGE  ::::::::::
    10 CONTINUE
       C1 = CONST * DSQRT(12.0D0)
C
       DO 100 I=1,N
          U1 = DABS( U(I) )
          IF (U1 .LE. PI * CONST)  GO TO 20
C         ::::::::::  DABS( U(I) ) .GT. PI * CONST  ::::::::::
          SQW(I) = 0.0D0
          GO TO 100
   20     CONTINUE
          IF (U1 .GE. UFETA*C1)  GO TO 30
C         ::::::::::  DIVISION WOULD UNDERFLOW  ::::::::::
          SQW(I) =  1.0D0
          GO TO 100
```

```
   30     CONTINUE
          U1 = U(I) / C1
C         ::::::::::  CHECK IF U1 IS IN RANGE  ::::::::::
          IF (DABS( U1 ) .GT. RKTOL)  GO TO 40
C         ::::::::::  USE ALTERNATE METHOD FOR SMALL VALUES  ::::::::::
          SQW(I) = (0.5D0 + U1 + 0.5D0) * (0.5D0 - U1 + 0.5D0)
          GO TO 100
   40     CONTINUE
C         ::::::::::  FUNCTION CAN BE COMPUTED NORMALLY  ::::::::::
          U1 = U(I) / CONST
          SQW(I) = DSQRT( DSIN(U1) / U1 )
  100 CONTINUE
C
      RETURN
C     ::::::::::  LAST CARD OF WANDRW  ::::::::::
      END
```