296
3/18
AEC RESEARCH AND DEVELOPMENT REPORT

MASTER

THE PROGRAMMER'S HANDBOOK

EDITOR:

R. E. Funderlic

**UNION
CARBIDE**

**UNION CARBIDE CORPORATION
NUCLEAR DIVISION**

Operating the

OAK RIDGE GASEOUS DIFFUSION PLANT • OAK RIDGE NATIONAL LABORATORY
OAK RIDGE Y-12 PLANT • PADUCAH GASEOUS DIFFUSION PLANT

*For the Atomic Energy Commission
Under U.S. Government Contract W7405 eng 26*

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Printed in the United States of America. Available from Clearinghouse for Federal
Scientific and Technical Information, National Bureau of Standards,
U.S. Department of Commerce, Springfield, Virginia 22151
Price: Printed Copy \$3.00; Microfiche \$0.65

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

- A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or
- B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

Report Date: February 9, 1968

Report Number: K-1729

Subject Category: MATHEMATICS
AND
COMPUTERS

THE PROGRAMMER'S HANDBOOK

A Compendium of Numerical Analysis Utility Programs

R. E. Funderlic (editor)

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

UNION CARBIDE CORPORATION
NUCLEAR DIVISION
Computing Technology Center
Scientific Applications Department
Oak Ridge, Tennessee

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DWR 5-24

August 23, 1968

ERRATA FOR "THE PROGRAMMER'S HANDBOOK," K-1729
Subject Category: Mathematics and Computers

1. ALLMAT, ORD 9079

The listing on page 233 has an error which affects some matrices in which the Hessenberg form has zero elements on the subdiagonal.

This error can be corrected by replacing cards ALLM0154 and ALLM0155 by

```
IF(RHØ.NE.Ø.)GØ TØ 60
```

```
TEMP1=A(RP1,RP1)
```

```
TEMP2=A(R+2,RP1)
```

```
GØ TØ 36
```

```
60 CØS=TEMP1/RHØ
```

In the SHARE version AMAT 138 and AMAT 139 are replaced by the above cards.

2. CMATEQ, ORD 9067

In order to compile the subprogram CMATEQ, ORD 9067, on the FORTRAN G compiler or the FORTRAN/360 Compiler at ORNL, the following changes must be made.

The statement in the listing on page 137 labeled CMTQ0001 should be replaced by

```
CØMPLEX FUNCTION CMATEQ*16 (A,B,III,JJJ,ID)
```

The statement labeled CMTQ0008 should be replaced by

```
CØMPLEX*16 A,B,S,D
```

3. RILMAT, ORD 9078

There is an error when the entry EIGS is called in the subroutine RILMAT.

After the statement labeled RILM0013 on page 214 of the listing, the statement

```
1 IF(IRET.EQ.0)GØ TØ 50
```

should be inserted.

The statement number "1" must be removed from the statement labeled RILM0014.

The statement RILM0017 should be numbered "50".

On page 215 after the statement labeled RILM0061

```
IF(IRET.EQ.0)GØ TØ 51
```

should be inserted.

The statement labeled RILM0069 should be numbered "51".

When the input matrix is computationally singular, a divide by zero exception will occur in the subroutine EIGQR. Replacing correspondingly labeled statements on page 220 and page 221 with the statements listed below will prevent the exception. The answers are not changed.

```
43 Z1=(E-AA**2+(F-B)**2-0.25DO*(E*E+F*F)      RILM 339
```

```
      Z2=(G-C)**2+(H-DD)**2-0.25DO*(G*G+H*H)    RILM 340
```

```
      IF(Z1)44,44,47      RILM 341
```

```
44 IF(Z2)45,45,46      RILM 342
```

```
47 IF(Z2)48,48,49      RILM 349
```

4. SIMPUN, ORD 9057

An error exists which affects the "backward integration" option of SIMPUN. This can be corrected by changing the card labeled SIMN0026 to

IF(NX.EQ.IX)GØ TØ 2

5. DSCALE, ORD 9084

There is an error in the writeup and some errors in the comment cards of the listings. On page 271 the first sentence under purpose should be rewritten to read: "DSCALE scales a vector of double precision numbers (REAL*8) by a constant factor so that the magnitude of the largest element is greater than or equal to 1/16 and is less than 1. This is effected in such a way as to cause no rounding errors." This description should also be exhibited in comment cards 5 and 6 of the listing on page 273. Also cards 11 and 12 should be replaced by

* N IS THE INTEGER NUMBER OF
* ELEMENTS IN A.

6. KUTTA, ORD 9060

The card labeled 0059 of the listing on page 83 should be deleted.

7. FLTRN, ORD 9082

The paragraph labeled "Initialization" on page 259 is misleading. The user can not arbitrarily change the constants mentioned there.

8. DLAG, ORD 9051

In the description of the arguments, disregard the last sentence under the argument IEXTX.

Under the heading Coding Information, change NY to M in the last sentence and add the following sentences:

If $NX < M$, M is set equal to NX for interpolation along the x-axis. If $NY < M$, M is set equal to NY for interpolation along y-axis.

Under the heading Restrictions, the sentence should be disregarded.

Program Changes

DLAG0025 should be omitted

After DLAG0049, add the statement

10 M1=M

DLAG0050 should be changed to

IF(M1.LE.NY)GØ TØ 11

DLAG0052 should be omitted.

9. AXTY, ORD 9083

The following change will not affect the execution of AXTY, but will allow the program to be inserted into an overlay, or replaced in a load module.

Card 1 should be replaced by

AXTY CSECT

Card 9 should be replaced by

STM 14,12,12(13)

10. SYMIN1 and SYMIN2, ORD 9068

The listings on pages 143 and 144 should be altered so that the array R does not appear in the double precision type statement, but is dimensioned by 200.

TABLE OF CONTENTS

	PAGE
PREFACE.....	11
<u>Write-ups and Listings of Routines</u>	
A. <u>Interpolation</u>	
ORD 9050 YLAG - Lagrangian interpolation for a function of one variable.....	17
ORD 9051 DLAG - Lagrangian interpolation for a function of two variables.....	23
ORD 9052 FØURA, FØURAS, FØURAC - Fourier approximation.....	29
B. <u>Numerical Differentiation</u>	
ORD 9053 DERIV1, DERIV2 - Derivative of least squares fit.....	39
ORD 9054 DXLAGR - Derivative of Lagrangian fit.....	45
C. <u>Numerical Integration</u>	
ORD 9055 INTGR - Gaussian integration.....	53
ORD 9056 SIMP - Simpson integration.....	57
ORD 9057 SIMPUN - Simpson type integration for uneven spacing.....	61
ORD 9058 LAGUER - Laguerre-Gauss integration.....	67
ORD 9059 HERMIT - Hermite-Gauss integration.....	71

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

D. Ordinary Differential Equations

ORD 9060	KUTTA - Self-adjusting Runge-Kutta.....	77
ORD 9061	DIFEQ - Fixed interval Runge-Kutta.....	85
ORD 9062	DIFE2 - Runge-Kutta for second order systems.....	89

E. Zeros of Functions

ABCISA	Linear fractional statement function.....	95
ORD 9063	CEST - Müller's iteration.....	97
ORD 9064	PØLSØL - Modified Lehmer's method for polynomial equations.....	101
ORD 9065	ZERØIN - Secant method for systems of equations.....	111
ORD 9066	NØNLIN - Secant method for systems of equations.....	119

F. Solution of Matrix Equations and Determinants

ORD 9067	DMATEQ, CMATEQ - Gaussian elimination for real and complex systems.....	131
ORD 9068	SYMIN1, SYMIN2 - Inversion of symmetric matrix.....	139
ORD 9069	BANSØL - Solution of band systems of linear equations.....	147

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

PAGE

ORD 9070	TRISØL - Solution of tridiagonal systems of linear equations.....	153
ORD 9071	CDET - Determinant of complex matrix.....	157
ORD 9072	HDET - Determinant of Hermitian matrix.....	161

G. Solution of Eigensystems

ORD 9073	MATGT, MATG1 - Generates test matrices.....	169
ORD 9074	SYMMAT/BIGMAT - Symmetric matrix eigen- system solver.....	173
ORD 9075	QRVEC - Symmetric matrix eigensystem solver.....	187
ORD 9076	MATDAG - Symmetric matrix eigensystem solver.....	195
ORD 9077	JACØBH - Hermitian matrix eigensystem solver.....	201
ORD 9078	RILMAT - Real matrix eigensystem solver.....	209
ORD 9079	ALLMAT - Complex matrix eigensystem solver.....	225

H. Least Squares

ORD 9080	LSQFIT - Linear least squares.....	239
ORD 9081	ALSQ, BLSQ - Linear least squares.....	243

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

I. Statistics

ORD 9082 Seven distributions of random numbers..... 257

Z. Miscellaneous

ORD 9083 AXTY - Inner product of vectors..... 267

ORD 9084 DSCALE - Scale vectors..... 271

ORD 9085 ARRAY, ARRAYD, ARRAYC, ARAYDC -
Array listers..... 277

ORD 9086 SØRT - Sorting routine..... 285

ORD 9087 NGCD, LCM - Greatest common divisors
and least common multiples..... 289Bibliography..... 293

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

PREFACE

This compendium was written primarily to provide documentation for programs which solve problems that occur frequently in mathematics of computation. These programs have also been helpful in teaching programming and numerical techniques. Emphasis was placed on ease of usage, and every effort was made to provide comments on the theory behind each routine, theoretical and practical limitations, and coding information.

Although all the programs have been checked using an IBM Fortran IV H level compiler for System/360 model 50 (see IBM manual C28-6515), many of the programs (in modified form) date back to our Fortran I days on an IBM 704. Use of tapes 5 and 6, prologue reducers, etc., are from our IBM 7090 days. We hope the next edition will be more in the spirit of the new Fortran. Most of our timing experience is given for an IBM 7090 as we have not had much experience in timing programs on IBM System/360. The precision on the 360 gave us much concern. We finally decided to convert all the programs to double precision because on the larger models of the 360 double precision does not take appreciably more time than single precision. Also, there are fewer chances for programming errors in converting a program from double precision to single precision than vice versa.

This compendium is primarily intended for internal use and our CTC library will honor program deck requests only from AEC installations in Oak Ridge. However, it is easy to keypunch decks from the program lists which are included. Internal requests should be made to

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

the CTC library by referring to the ORD number which appears with each write-up and listing.

Although CTC employees have made significant contributions in optimization, nonlinear least squares, and statistics, I regret that pertinent programs on these subjects and others were not included in this edition. Most of the credit for this report goes to our mathematical development section. Special credit should go to Malcolm Herndon, a summer employee, who checked all the programs and did much of the proofreading. As we all know, errors will still persist, and pertinent criticisms are always welcome. Grateful acknowledgment is made to Marie Cuthbert for a painstaking job of typing.

R. E. Funderlic, Editor

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

A

INTERPOLATION

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Lagrangian interpolation of a function of one variable.

Purpose

Given a set of points, this subprogram calculates the polynomial fitting through them by the Lagrangian method, and then interpolates at any desired location along the curve. The subprogram, YLAG, allows the user 1) to search for table entries, 2) to "center" the indicated number of points about the required point, 3) to detect extrapolation and, 4) to interpolate at a specified location in the table.

Usage

The calling sequence is:

YX = YLAG(XI,X,Y,IND1,N1,IMAX,IEX)

where

YLAG is the double precision function subprogram which returns the function approximation at XI,

XI is the double precision abscissa of the desired interpolated point,

X is the singly-dimensioned array of distinct monotone increasing, double precision abscissas,

Y is the singly-dimensioned double precision array of corresponding ordinates,

IND1 is the integer index of the "central" point of the interpolation formula; if IND1 = 0, a table search of the X-array will occur,

N1 is the integer number of points for the interpolation polynomial (see "Coding Information"),

IMAX is the integer number of elements of the X-array or Y-array,

IEX is an integer calculated in YLAG and returns

- 1 if extrapolation occurred below table,
- 0 if interpolation occurred,
- +1 if extrapolation occurred above table,

YX is the interpolated Y-value.

When possible the point XI should be chosen to lie between abscissas such that the abscissas are symmetric about XI or such that an extra point lies below XI.

Coding Information

YLAG is a Fortran IV double precision function subprogram which can easily be converted to single precision. YLAG must appear in a DOUBLE PRECISION type statement in the calling program.

If the grid is evenly spaced, the IND1 feature should probably be used. This will eliminate the necessity for searching the X-array (see "Usage").

N1 is one more than the degree of the polynomial being used for the interpolation.

Restrictions

See "Method."

Accuracy

See "Method."

Method

The implementation of Lagrangian interpolation is accomplished by determining the unique $n-1$ order polynomial which will pass thru n -points of the desired function and interpolating to the desired point by evaluating the polynomial. The results of this process can be expressed as follows:

$$Y_n(X) = \prod_{i=1}^n (X - X_i) \sum_{k=1}^n \frac{Y_k}{(X - X_k) \prod_{\substack{i=1 \\ i \neq k}}^n (X_k - X_i)}$$

where

X is the independent variable.

Y is the dependent variable.

X_i, Y_i are the coordinates of the i^{th} point of the n -points used for interpolation of the true function, $f(X)$

$Y_n(X)$ is the interpolated value of X

n is the number of points.

The error analysis of the Lagrangian method leads to the following:

$$E_n(X) = |f(X) - Y_n(X)| \leq \frac{f^{(n)}(\hat{X})}{n!} \prod_{i=1}^n (X - X_i)$$

where

$$X_i \leq \hat{X} \leq X_n$$

$$f^{(n)}(\hat{X}) = (d^n f(X)/dX^n)_{X=\hat{X}}$$

Since the error of this method is dependent upon the value of $f^{(n)}(\hat{X})$, the method is satisfactory only when this derivative is suitably small. Since high order derivatives are not usually meaningful in experimental data or rounded function values, the user should be cautious in such cases not to use a needlessly high order. Likewise, the error is reduced if X lies near the center of $[X, X_n]$ and if the spacing between points is small.

References

- [1] Hildebrand, F. B. (1956): Introduction to Numerical Analysis, McGraw-Hill, p. 60.
- [2] Milne, W. E. (1949): Numerical Calculus, Princeton.

Authors

A. A. Brooks and E. C. Long, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

C      DOUBLE PRECISION FUNCTION YLAG(XI,X,Y,IND1,N1,IMAX,IEX)
C      PROGRAM AUTHORS  A. A. BROOKS AND E.C. LONG,
C      COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C      OAK RIDGE, TENN.
C      CTC ORD PROGRAM NO. 5050
C      LAGRANGIAN INTERPOLATION
C      XI IS INTERPOLATED ENTRY INTO X-ARRAY
C      N IS THE ORDER OF LAGRANGIAN INTERPOLATION
C      Y IS ARRAY FROM WHICH YLAG IS OBTAINED BY INTERPOLATION
C      IND IS THE MIN-I FOR X(I).GT.XI
C      IF IND=0,X-ARRAY WILL BE SEARCHED
C      IMAX IS MAX INDEX OF X-AND Y-ARRAYS
C      EXTRAPOLATION CAN OCCUR,IEX=-1 OR +1
C
C      DIMENSION X(1),Y(1)
C      DOUBLE PRECISION P,D,S,XD,XI,X,Y,YLAG
C      IND=IND1
C      N=N1
C      IEX=0
C      IF (N .LE. IMAX) GO TO 1
C      N=IMAX
C      IEX=N
C      1 IF (IND .GT. 0) GO TO 4
C      DO 2 J = 1,IMAX
C      IF (XI-X(J)) 3,13,2
C      2 CONTINUE
C      IEX=1
C      GO TO 7
C      3 IND=J
C      4 IF (IND .GT. 1) GO TO 5
C      IEX=-1
C      5 INL=IND-(N+1)/2
C      IF (INL .GT. 0) GO TO 6
C      INL=1
C      6 INU=INL+N-1
C      IF (INU .LE. IMAX) GO TO 8
C      7 INL=IMAX-N+1
C      INU=IMAX
C      8 S=0.
C      P=1.
C      DO 11 J=INL,INU
C      P=P*(XI-X(J))
C      D=1.
C      DO 10 I=INL,INU
C      IF (I .NE. J) GO TO 9
C      XD=XI
C      GO TO 10
C      9 XD=X(J)
C      10 D=D*(XD-X(I))

```

```
11 S=S+Y(J)/D
   YLAG=S*P
12 RETURN
13 YLAG=Y(J)
   GO TO 12
END
```

```
YLAG0051
YLAG0052
YLAG0053
YLAG0054
YLAG0055
YLAG0056
```

Identification

Two variable Lagrangian interpolation.

Purpose

Given a set of points $f(x,y)$, where x and y are their given independent coordinate values, this function subprogram gives an interpolation approximation to $f(xx,yy)$ where xx and yy are the given independent coordinates of the position where the approximation to f is desired.

Usage

The calling sequence is:

$F = \text{DLAG}(XX,YY,X,Y,Z,NX,NY,M,IEXTX,IEXTY,ID)$

where

DLAG is the double precision valued function subprogram which returns the interpolated function approximation,

XX is the double precision x-coordinate of the desired interpolated point,

YY is the double precision y-coordinate of the desired interpolated point,

X is the singly-dimensioned double precision array of x-coordinates,

Y is the singly-dimensioned double precision array of y-coordinates,

Z is the doubly-dimensioned double precision array of function values, given by

$$Z(I,J) = f(X(I),Y(J)),$$

NX is the integer number of elements in the X-array,
NY is the integer number of elements in the Y-array,
M is an integer which indicates that M^2 points are to
be considered in the interpolation,
IEXTX is an integer calculated in DLAG and returns
1 if extrapolation occurred above X table,
0 if interpolation occurred,
-1 if extrapolation occurred below X table,
M if $M > NX$
IEXTY is the same for the Y-array as IEXTX.
ID is the first dimension of Z in the calling program.

Coding Information

DLAG is a Fortran IV double precision function subprogram which can easily be converted to single precision. DLAG must appear in a DOUBLE PRECISION type statement.

DLAG calls the auxiliary subroutine INTER which sets up the interpolation polynomial and calculates the interpolated point. $NY \leq 100$ unless dimension of XLAG is changed.

Restrictions

The arguments NX and NY must not be less than M.

Method

The method is based on one variable interpolation [1]. There are M Lagrangian one-variable polynomials of degree M-1 fit in the x-direction. Each of these is evaluated at a distinct (XX,Y(I))

point and then the appropriate Lagrangian one-variable polynomial of degree $M-1$ is evaluated at (XX,YY) .

References

- [1] Hildebrand, F. B. (1956): Introduction to Numerical Analysis, McGraw-Hill, pp. 60-64.
- [2] Milne, W. E., et.al. (1958): Multivariate Interpolation, WADC Tech. Report 57-556.

Author

F. D. Hammerling, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.


```

      DOUBLE PRECISION FUNCTION DLAG(XX,YY,X,Y,Z,NX,NY,M,IEXTX,IEXTY,ID)DLAG0001
C   PROGRAM AUTHOR    F. D. HAMMERLING,          DLAG0002
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV., DLAG0003
C   OAK RIDGE, TENN.          DLAG0004
C                               DLAG0005
C   CTC ORD PROGRAM NO. 9051          DLAG0006
C   TWO VARIABLE LAGRANGIAN INTERPOLATION. DLAG0007
C                               DLAG0008
      DIMENSION Z(ID,1),X(1),Y(1),XLAG(100) DLAG0009
      DOUBLE PRECISION Z,X,Y,XLAG,XX,YY,XLAG,DLAG,YLAG DLAG0010
      LOGICAL QUITX,QUITY DLAG0011
C                               DLAG0012
C   INITIALIZE          DLAG0013
C                               DLAG0014
      QUITX=.FALSE. DLAG0015
      QUITY=.FALSE. DLAG0016
      M1=M DLAG0017
      IEXTX=0 DLAG0018
      IEXTY=0 DLAG0019
C                               DLAG0020
C   FIND THE RANGE OF INTERPOLATION ALONG X DLAG0021
C                               DLAG0022
      IF(M1.LE.NX)GO TO 1 DLAG0023
      M1=NX DLAG0024
      IEXTX=M1 DLAG0025
1   DO 2 I=1,NX DLAG0026
      IF(XX-X(I))3,9,2 DLAG0027
2   CONTINUE DLAG0028
      IEXTX=1 DLAG0029
      GO TO 8 DLAG0030
3   MIDX=I DLAG0031
      IF(MIDX.GT.1)GO TO 4 DLAG0032
      IEXTX=-1 DLAG0033
      GO TO 5 DLAG0034
4   IF(DABS(XX-X(MIDX)).GE.DABS(XX-X(MIDX-1)))GO TO 5 DLAG0035
      ISXPT=MIDX-(M1/2) DLAG0036
      GO TO 6 DLAG0037
5   ISXPT=MIDX-((M1+1)/2) DLAG0038
6   IF(ISXPT.GT.0)GO TO 7 DLAG0039
      ISXPT=1. DLAG0040
7   IEXPT=ISXPT+M1-1 DLAG0041
      IF(IEXPT.LE.NX)GO TO 10 DLAG0042
8   ISXPT=NX-M1+1 DLAG0043
      IEXPT=NX DLAG0044
      GO TO 10 DLAG0045
9   QUITX=.TRUE. DLAG0046
C                               DLAG0047
C   FIND THE RANGE OF INTERPOLATION ALONG Y DLAG0048
C                               DLAG0049
10  IF(M1.LE.NY)GO TO 11 DLAG0050

```

M1=NY	DLAG0051
IEXTY=M1	DLAG0052
11 DO 12 J=1,NY	DLAG0053
IF(YY-Y(J))13,19,12	DLAG0054
12 CONTINUE	DLAG0055
IEXTY=1	DLAG0056
GO TO 18	DLAG0057
13 MIDY=J	DLAG0058
IF(MIDY.GT.1)GO TO 14	DLAG0059
IEXTY=-1	DLAG0060
GO TO 15	DLAG0061
14 IF(DABS(YY-Y(MIDY)).GE.DABS(YY-Y(MIDY-1)))GO TO 15	DLAG0062
ISYPT=MIDY-(M1/2)	DLAG0063
GO TO 16	DLAG0064
15 ISYPT =MIDY-((M1+1)/2)	DLAG0065
16 IF(ISYPT.GT.0)GO TO 17	DLAG0066
ISYPT=1	DLAG0067
17 IEYPT =ISYPT+M1-1	DLAG0068
IF(IEYPT.LE.NY)GO TO 20	DLAG0069
18 ISYPT=NY-M1+1	DLAG0070
IEYPT=NY	DLAG0071
GO TO 20	DLAG0072
19 QUITY=.TRUE.	DLAG0073
20 IF(QUITX.AND.QUITY)GO TO 27	DLAG0074
IF(QUITX.AND..NOT.QUITY)GO TO 24	DLAG0075
IF(.NOT.QUITX.AND.QUITY)GO TO 23	DLAG0076
C	DLAG0077
C INTERPOLATE ALONG X	DLAG0078
C	DLAG0079
21 DO 22 K=ISYPT,IEYPT	DLAG0080
22 CALL INTER(XX,Z(1,K),X,ISXPT,IEXPT,XLAG(K))	DLAG0081
GO TO 26	DLAG0082
23 CALL INTER(XX,Z(1,J),X,ISXPT,IEXPT,XLAG(J))	DLAG0083
DLAG=XLAG(J)	DLAG0084
GO TO 28	DLAG0085
C	DLAG0086
C INTERPOLATE ALONG Y	DLAG0087
C	DLAG0088
24 DO 25 L=ISYPT,IEYPT	DLAG0089
25 XLAG(L)=Z(I,L)	DLAG0090
26 CALL INTER(YY,XLAG,Y,ISYPT,IEYPT,YLAG)	DLAG0091
DLAG=YLAG	DLAG0092
GO TO 28	DLAG0093
27 DLAG=Z(I,J)	DLAG0094
28 RETURN	DLAG0095
END	DLAG0096
SUBROUTINE INTER(PT,FUN,ORD,ISPT,IEPT,ALAG)	DLAG 97
C	DLAG 98
DIMENSION FUN(1),ORD(1)	DLAG 99
DOUBLE PRECISION FUN,ORD,SUM,PROD,PT,DENOM,D,ALAG	DLAG 100

```
SUM=0.0
PROD=1.
DO 3 J=ISPT,IEPT
  PROD=PROD*(PT-ORD(J))
  DENOM=1.
  DO 2 K=ISPT,IEPT
    IF(K.NE.J)GO TO 1
    D=PT
    GO TO 2
1 D=ORD(J)
2 DENOM=DENOM*(D-ORD(K))
3 SUM=SUM+FUN(J)/DENOM
ALAG=SUM*PROD
RETURN
END
```

```
DLAG 101
DLAG 102
DLAG 103
DLAG 104
DLAG 105
DLAG 106
DLAG 107
DLAG 108
DLAG 109
DLAG 110
DLAG 111
DLAG 112
DLAG 113
DLAG 114
DLAG 115
```

Identification

Fourier approximation subroutines.

Purpose

To evaluate a sin-cos series approximation of the form

$$f(x) \approx a_0 + \sum_{k=1}^n (a_k \cos kx + b_k \sin kx) ,$$

a sine series approximation of the form

$$G(x) \approx \sum_{k=1}^{n-1} b_k \sin kx ,$$

and a cosine series approximation of the form

$$H(x) \approx a_0 + \sum_{k=1}^n a_k \cos kx ,$$

where we are to determine the coefficients a_0 , a_k , and b_k .

Usage

The subroutines are called by the following sequences:

CALL FØURA(F,A0,A,B,N)

CALL FØURAS(G,B,N)

CALL FØURAC(H,A0,A,N)

where

F is a double precision, singly-dimensioned input array containing $2N+1$ function values assumed equally spaced over the interval $(-\pi, \pi)$.

G is a double precision, singly-dimensioned array containing $N-1$ function values assumed equally spaced over the interval $(0, \pi)$.

- H is a double precision, singly-dimensioned input array containing $N+1$ function values assumed equally spaced over the interval $(0, \pi)$.
- A0 is a double precision variable which will contain a_0 upon returning from the subroutine.
- A is a double precision, singly-dimensioned array which will contain the N coefficients a_k after returning from the subroutine.
- B is a double precision, singly-dimensioned array which will contain the $N-1$ coefficients b_k after returning from the subroutine.
- N is the integer number of harmonics one wishes to retain, where N is the same as n in purpose.

The three subroutines named above evaluate the sin-cos, sine, and cosine approximations respectively.

Coding Information

FØURA, FØURAS and FØURAC are Fortran IV double precision programs which can easily be converted to single precision.

Accuracy

The subroutines gave exact answers except for those errors due to roundoff for the approximations tested.

Method

For the sin-cos approximation, we suppose that $f(x)$ is of period 2π , but that its values are known only at a discrete set of equally

spaced points in a period interval, say at the $2N+1$ points, x_r ,

$$-\pi, -\frac{(N-1)\pi}{N}, \dots, -\frac{\pi}{N}, 0, \frac{\pi}{N}, \dots, \frac{(N-1)\pi}{N}, \pi$$

of the interval $(-\pi, \pi)$. Since $f(-\pi) = f(\pi)$ from the assumed periodicity, we then have $2N$ independent data, which may be expected to serve to determine the coefficients of $2N$ terms of an approximation of the form

$$f(x) = A_0 + \sum_{k=1}^n (A_k \cos kx + B_k \sin kx).$$

By the fact that the set is orthogonal under summation over the set

$$(-N+1, -N+2, \dots, -1, 0, 1, \dots, N-1, N),$$

and applying the least squares criterion

$$\sum_{r=-N+1}^N [f(x_r) - A_0 - \sum_{k=1}^n (A_k \cos kx_r + B_k \sin kx_r)]^2 = \min,$$

we find that

$$A_0 = \frac{1}{2N} \sum_{r=-N+1}^N f(x_r),$$

$$A_k = \frac{1}{N} \sum_{r=-N+1}^N f(x_r) \cos kx_r,$$

$$A_n = \frac{1}{2N} \sum_{r=-N+1}^N f(x_r) \cos Nx_r, \quad \text{and}$$

$$B_k = \frac{1}{N} \sum_{r=-N+1}^N f(x_r) \sin kx_r.$$

For the purpose of numerical calculation, it is convenient to resolve $f(x)$ into its even and odd components, by introducing the auxiliary functions

$$H(x) = \frac{1}{2} [f(x) + f(-x)] , \text{ and}$$

$$G(x) = \frac{1}{2} [f(x) - f(-x)]$$

so that

$$f(x) = H(x) + G(x) .$$

If we note that $x_{-r} = -x_r$ and that $x_0 = 0$, we find that the above equations for the coefficients may be reduced to the forms

$$A_0 = \frac{1}{N} \left(\frac{1}{2} H_0 + H_1 + H_2 + \dots + H_{N-1} + \frac{1}{2} H_N \right) ,$$

$$A_k = \frac{2}{N} \left(\frac{1}{2} H_0 + H_1 \cos kx_1 + H_2 \cos kx_2 + \dots + H_{N-1} \cos kx_{N-1} + \frac{1}{2} H_N \cos kx_N \right) ,$$

$$A_N = \frac{1}{N} \left(\frac{1}{2} H_0 - H_1 + H_2 - \dots + (-1)^{N-1} H_{N-1} + (-1)^N \frac{1}{2} H_N \right) , \text{ and}$$

$$B_k = \frac{2}{N} (G_1 \sin kx_1 + G_2 \sin kx_2 + \dots + G_{N-1} \sin kx_{N-1}) .$$

A complete derivation may be found in [1]. The sine and cosine approximations follow from this and may be found in [1] and [2]. New versions of these routines for unevenly spaced points and more general intervals are expected. The user should also refer to programs in the IBM SSP based on the work of Cooley and Tukey [3].

References

- [1] Hildebrand, F. D. (1956): "Approximations of Various Types,"
Introduction to Numerical Analysis, McGraw-Hill, pp. 368-377.
- [2] Lanczos, C. (1965): Applied Analysis, Prentice-Hall, pp. 215-219.
- [3] Cooley, J. W., and Tukey, J. W. (1965): "An algorithm for the
machine calculation of complex Fourier series," Math. Comp.
19, 90, pp. 297-301.

Author

F. D. Hammerling, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.


```

      SUPROUTINE FOURA(F,A0,A,B,N)
C   PROGRAM AUTHOR   F. D. HAMMERLING,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC. ORD PROGRAM NO. 9052.1
C   EVALUATES A 2N+1 POINT SINE COSINE SERIES APPROXIMATION TO A FUNCTION
C   F(X) OVER THE RANGE -PI LESS THAN X LESS THAN PI, THROUGH HARMONICS
C   OF ORDER NOT EXCEEDING N.
C
      DIMENSION F(2),A(2),B(2),X(50),H(50),G(50),SUM2(50),SUM3(50)
1     ,COSINE(50),SINE(50)
      DOUBLE PRECISION F,A,B,X,H,G,SUM2,SUM3,COSINE,SINE,PI,SUM,A0
      L=2*N+1
      M=N+1
      M2=N-1
      PI = 3.141592653589793
      H(1)=F(1)/2.
      H(M)=F(M)/2.
      DO 1 I=2,N
      I2=(L-I)+1
      H(I)=(F(I)+F(I2))/2.
1-   G(I-1)=(F(I)-F(I2))/2.
      SUM=0.0
      DO 2 I=1,M
2     SUM=SUM+H(I)
      AC=SUM/DFLOAT(N)
      DO 3 K=1,N
      X(K)=((DFLOAT(K)-1.)*PI)/DFLOAT(N)
      COSINE(K)=DCOS(X(K))
3     SINE(K)=DSIN(X(K))
      DO 4 I=1,M2
      SUM2(I)=0.0
      DO 4 K=1,M
      ISUB=I*(K-1)
      L2=MOD(ISUB,N)+1
      SUM2(I)=SUM2(I)+H(K)*(-1.)**(ISUB/N)*COSINE(L2)
      A(I)=SUM2(I)*2./DFLOAT(N)
4     CONTINUE
      SUM=0.0
      DO 5 K=1,M
5     SUM=SUM+(-1.)**(K-1)*H(K)
      A(N)=SUM/DFLOAT(N)
      DO 7 I=1,M2
      SUM3(I)=0.0
      DO 6 K=1,M2
      ISUR2=I*K
      L3=MOD(ISUR2,N)+1
6     SUM3(I)=SUM3(I)+G(K)*(-1.)**(ISUR2/N)*SINE(L3)
7     B(I)=SUM3(I)*2./DFLOAT(N)

```

```

FORA0001
FORA0002
FORA0003
FORA0004
FORA0005
FORA0005
FORA0007
FORA0008
FORA0009
FORA0010
FORA0011
FORA0012
FORA0013
FORA0014
FORA0015
FORA0016
FORA0017
FORA0018
FORA0019
FORA0020
FORA0021
FORA0022
FORA0023
FORA0024
FORA0025
FORA0026
FORA0027
FORA0028
FORA0029
FORA0030
FORA0031
FORA0032
FORA0033
FORA0034
FORA0035
FORA0036
FORA0037
FORA0038
FORA0039
FORA0040
FORA0041
FORA0042
FORA0043
FORA0044
FORA0045
FORA0046
FORA0047
FORA0048
FORA0049
FORA0050

```

```

      RETURN
      END
      SUBROUTINE FOURAS(G,B,N)
C   PROGRAM AUTHOR   F. D. HAMMERLING,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC ORD PROGRAM NO. 9052.2
C   EVALUATES A N-1 POINT SINE SERIES APPROXIMATION TO A FUNCTION G(X)
C   OVER THE HALF RANGE 0 LESS THAN X LESS THAN PI, THROUGH HARMONICS
C   OF ORDER NOT EXCEEDING N-1.
C
      DIMENSION G(2),B(2),X(25),SINE(25),SUM(25)
      DOUBLE PRECISION G,B,X,SINE,SUM,PI
      PI = 3.141592653589793
      M2=N-1
      DO 1 K=1,N
      X(K)=((DFLOAT(K)-1.)*PI)/DFLOAT(N)
1  SINE(K)=DSIN(X(K))
      DO 3 I=1,M2
      SUM(I)=0.0
      DO 2 K=1,M2
      ISUB=I*K
      L3=MOD(ISUB,N)+1
2  SUM(I)=SUM(I)+G(K)*(-1.)**(ISUB/N)*SINE(L3)
3  B(I)=SUM(I)*2./DFLOAT(N)
      RETURN
      END
      SUBROUTINE FOURAC(H,AO,A,N)
C   PROGRAM AUTHOR   F. D. HAMMERLING,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC ORD PROGRAM NO. 9052.3
C   EVALUATES A N+1 POINT COSINE SERIES APPROXIMATION TO A FUNCTION H(X)
C   OVER THE HALF RANGE 0 LESS THAN X LESS THAN PI, THROUGH HARMONICS
C   OF ORDER NOT EXCEEDING N.
C
      DIMENSION H(2),A(2),X(25),COSINE(25),SUM2(25)
      DOUBLE PRECISION H,A,X,COSINE,SUM2,PI,SUM,AO,SUM3
      M=N+1
      M2=N-1
      PI = 3.141592653589793
      H(1) = H(1)/2.
      H(M) = H(M)/2.
      SUM=0.0
      DO 1 I=1,M
1  SUM=SUM+H(I)
      AO=SUM/DFLOAT(N)
      DO 2 K=1,N

```

```

FORA0051
FORA0052
FORS0001
FORS0002
FORS0003
FORS0004
FORS0005
FORS0006
FORS0007
FORS0008
FORS0009
FORS0010
FORS0011
FORS0012
FORS0013
FORS0014
FORS0015
FORS0016
FORS0017
FORS0018
FORS0019
FORS0020
FORS0021
FORS0022
FORS0023
FORS0024
FORS0025
FORS0026
FORC0001
FORC0002
FORC0003
FORC0004
FORC0005
FORC0006
FORC0007
FORC0008
FORC0009
FORC0010
FORC0011
FORC0012
FORC0013
FORC0014
FORC0015
FORC0016
FORC0017
FORC0018
FORC0019
FORC0020
FORC0021
FORC0022

```

X(K)=((DFLOAT(K)-1.)*PI)/DFLOAT(N)	FORC0023
2 COSINE(K)=DCOS(X(K))	FORC0024
DO 3 I=1,M2	FORC0025
SUM2(I)=0.0	FORC0026
DO 3 K=1,M	FORC0027
ISUB=I*(K-1)	FORC0028
L2=MOD(ISUB,N)+1	FORC0029
SUM2(I)=SUM2(I)+H(K)*(-1.)**(ISUB/N)*COSINE(L2)	FORC0030
A(I)=SUM2(I)*2./DFLOAT(N)	FORC0031
3 CONTINUE	FORC0032
SUM3=0.0	FORC0033
DO 4 K=1,M	FORC0034
4 SUM3=SUM3+(-1.)**(K-1)*H(K)	FORC0035
A(N)=SUM3/DFLOAT(N)	FORC0036
RETURN	FORC0037
END	FORC0038

B

NUMERICAL
DIFFERENTIATION

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Differentiation of an empirical function.

Purpose

Given a set of abscissas and corresponding ordinates, these function subprograms approximate the derivative at a specified point. The subprograms treat unevenly and evenly spaced grids separately. For the case of evenly spaced points, the set of equations to be solved is simplified analytically and the number of calculations reduced considerably.

Usage

For evenly spaced points the calling sequence is:

ANS = DERIV1 (DELT, Y, I, N)

where

DERIV1 is the double precision valued function subprogram
which returns the approximate derivative,

DELT is the difference between consecutive abscissas
(double precision),

Y is the singly-dimensioned double precision array of
function values,

I indicates that the derivative will be calculated at
the Ith point (integer),

N is the integer number of points in the Y-array.

For unevenly spaced points the calling sequence is:

ANS = DERIV2 (X, Y, I, N)

where

DERIV2 is the double precision valued function subprogram
which returns the approximate derivative,
X is the singly-dimensioned double precision array of
abscissas,
Y is the singly-dimensioned double precision array of
ordinates,
I indicates that the derivative will be calculated at
the Ith point (integer),
N is the integer number of points in the X and Y arrays.

Coding Information

DERIV1 and DERIV2 are Fortran IV double precision function sub-
programs which can easily be converted to single precision.
DERIV1 and DERIV2 must appear in DOUBLE PRECISION type statements
in the calling program.

Restrictions

It is well known [1] that numerical approximations to derivatives
are often poor and the user is well advised to attempt to formulate
his problem differently.

Method

The method used assumes that the second derivative does not change
significantly for any five consecutive points. Let (x_i, y_i) be the
point where the derivative is to be calculated. Assume that a
parabola of the form $y = ax^2 + bx + c$ is fit by least square

minimization to five consecutive points symmetric about (x_i, y_i) :

We can translate the axis such that $x_i = 0$, in which case the derivative $y'_i = 2ax_i + b$ evaluated at $x_i = 0$ is the parameter b .

This parameter is conveniently obtained by using Cramer's Rule to solve the set of normal equations [1]:

$$\begin{pmatrix} \sum h_j^4 & \sum h_j^3 & \sum h_j^2 \\ \sum h_j^3 & \sum h_j^2 & \sum h_j \\ \sum h_j^2 & \sum h_j & 5 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \sum h_j^2 y_j \\ \sum h_j y_j \\ \sum y_j \end{pmatrix}$$

where $h_j = x_j - x_i$ and $j = i-2, i-1, i, i+1, i+2$.

Accuracy

The accuracy depends largely on how well any five consecutive points can be represented by a parabola. The intervals between points should be relatively small. The error was approximately .5% when the set of points generated by the function $y = e^x$, $x = 1(.1) 2.5$, was tested with a single precision version on an IBM 7090. The derivative calculated at the first two and last two points will generally be slightly less accurate.

Reference

- [1] Lanczos, Cornelius, (1956): Applied Analysis; Prentice Hall, Inc., pp. 321-323.

Author

Miss M. H. Noojin, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.


```

      DOUBLE PRECISION FUNCTION DERIV1(H,F,I,N)
C   PROGRAM AUTHOR   MISS M. NOOJIN,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC ORD PROGRAM NO. 9053.1
      DIMENSION F(1)
      DOUBLE PRECISION F,H
      IF(I.LE.2.OR.I.GE.(N-1)) GO TO 1
      DERIV1=(-2.*F(I-2)-F(I-1)+F(I+1)+2.*F(I+2))/(10.*H)
      GO TO 5
1     IF(I.EQ.1.OR.I.EQ.N) GO TO 3
      IF(I.EQ.2) GO TO 2
      DERIV1=(-11.*F(N)+3.*F(N-1)+7.*F(N-2)+F(N-3))/(20.*H)
      GO TO 5
2     DERIV1=(-11.*F(1)+3.*F(2)+7.*F(3)+F(4))/(20.*H)
      GO TO 5
3     IF(I.EQ.1) GO TO 4
      DERIV1=(-21.*F(N)+13.*F(N-1)+17.*F(N-2)-9.*F(N-3))/(20.*H)
      GO TO 5
4     DERIV1=(-21.*F(1)+13.*F(2)+17.*F(3)-9.*F(4))/(20.*H)
5     RETURN
      END
      DOUBLE PRECISION FUNCTION DERIV2(X,F,I,N)
C   PROGRAM AUTHOR   MISS M. NOOJIN,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC ORD PROGRAM NO. 9053.2
      DIMENSION H(5),FX(5),F(1),X(1)
      DOUBLE PRECISION H,FX,F,X,SHI2FI,SHIFI,SFI,SHI,SHI2,SHI3,SHI4,TEMP
      IF(I.LE.2.OR.I.GE.(N-1)) GO TO 2
      DO 1 K=1,5
        IND=K-3+I
        H(K)=X(IND)-X(I)
1     FX(K)=F(IND)
      GO TO 6
2     IF(I.EQ.2.OR.I.EQ.1) GO TO 4
C
C   CASE WHERE I=N OR I=N-1
C
      DO 3 K=1,5
        IND=N-5+K
        H(K)=X(IND)-X(I)
3     FX(K)=F(IND)
      GO TO 6
C
C   CASE WHERE I=1 OR I=2
C
4     DO 5 K=1,5

```

H(K)=X(K)-X(I)	DER20028
5 FX(K)=F(K)	DER20029
6 SHI2FI=0.	DER20030
SHIFI=0.	DER20031
SFI=0.	DER20032
SHI=0.	DER20033
SHI2=0.	DER20034
SHI3=0.	DER20035
SHI4=0.	DER20036
DO 7 K=1,5	DER20037
SFI=SFI+FX(K)	DER20038
SHI=SHI+H(K)	DER20039
SHIFI=SHIFI+H(K)*FX(K)	DER20040
TEMP=H(K)**2	DER20041
SHI2=SHI2+TEMP	DER20042
SHI2FI=SHI2FI+FX(K)*TEMP	DER20043
TEMP=TEMP*H(K)	DER20044
SHI3=SHI3+TEMP	DER20045
TEMP=TEMP*H(K)	DER20046
SHI4=SHI4+TEMP	DER20047
7 CONTINUE	DER20048
DERIV2=(5.*SHI4*SHIFI+SHI2FI*SHI2*SHI+SHI3*SHI2*SFI	DER20049
1-SHI2**2*SHIFI-5.*SHI3*SHI2FI-SHI4*SHI*SFI)/(	DER20050
25.*SHI4*SHI2+2.*SHI3*SHI*SHI2-SHI2**3-SHI**2*SHI4	DER20051
3-5.*SHI3**2)	DER20052
RETURN	DER20053
END	DER20054

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Differentiation of the Lagrangian interpolate.

Purpose

To find the derivative at one of the points used in a Lagrangian interpolation and hence give a numerical approximation to the derivative of a function. This subroutine has the following features:

1. Checks that the point of the derivative is one of the input points.
2. Centers the point of the derivative amongst the points used when possible.
3. Sets a flag if the point of the derivative is at either extreme end of the point array.

Usage

The calling sequence is:

```
CALL DXLAGR (X,Y,NNP,XI,IFLAG,IIMAX,DX)
```

where

X is the singly-dimensioned array of sequential double precision abscissas,

Y is the singly-dimensioned array of double precision ordinates,

NNP is the integer number of points from the array to be used in the calculation (NNP is equal to n mentioned in Method),

XI is the double precision abscissa at which the derivative is to be evaluated. (XI is x_j mentioned in Method).

IFLAG is an integer calculated in DXLAGR and returns:

-1, if XI is smallest value in the X array.

+1, if XI is largest value in the X array.

2, if XI is not a point in the X array (no calculation).

3, if NNP exceeds the number of points in the array. In this case IIMAX is used instead of NNP.

IIMAX is the integer number of points in the X array.

DX returns the derivative at point XI (DX is $df(x_j)/dx$ mentioned in Method).

Coding Information

DXLAGR is a double precision Fortran IV subprogram which may be easily converted to single precision.

Method

The derivative at point $(x_j, f(x_j))$ of an $(n - 1)$ order polynomial passing through (n) points (one of which is $(x_j, f(x_j))$), and calculated by Lagrangian interpolation [2] is:

$$\frac{df(x_j)}{dx} = \prod_{\substack{i=1 \\ i \neq j}}^n (x_j - x_i) \sum_{\substack{i=1 \\ i \neq j}}^n \frac{y_i}{(x_j - x_i) \prod_{\substack{k=1 \\ k \neq i}}^n (x_i - x_k)} +$$

$$\sum_{\substack{i=1 \\ i \neq j}}^n \frac{y_j}{(x_j - x_i)},$$

where:

$y_i = f(x_i)$ and (x_i, y_i) are the (i^{th}) coordinates of the (n) points used to derive the interpolation polynomial $f(x)$.

(x_j, y_j) are the coordinates in the array of (n) points at which the derivative is to be evaluated. $df(x_j)/dx$ is the approximate derivative of $f(x)$ at coordinates (x_j, y_j) .

Accuracy

The error analysis of this method is quite similar to that of the Lagrangian interpolation [2]. It can be expressed as follows:

$$E = \left| \frac{F^n(\hat{x})}{n!} \sum_{\substack{i=1 \\ i \neq j}}^n (x_j - x_i) \right|,$$

where $F^n(\hat{x})$ is the n^{th} derivative of the true curve being approximated: $(x_i < \hat{x} < x_n)$.

Note: Though the differences between the calculated and true curves may be slight, the differences in slope may be appreciable, especially if there are observation and rounding errors. Also, when the point of the derivative is the smallest or largest x value of the array, the accuracy is quite often poor.

References

- [1] Milne, W. E., (1949): Numerical Analysis, Princeton, p. 93.
- [2] Hildebrand, F. B., (1956): Introduction to Numerical Analysis, McGraw-Hill, pp. 60-70.

Author

J. A. Gillcrist, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE DXLAGR (X,Y,NNP,XI,IFLAG,IIMAX,DX)
C  PROGRAM AUTHOR   J. A. GILLCRIST,
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C  CTC ORD PROGRAM NO. 9054
C  DX=DERIVATIVE OF LAGRANGIAN INTERPOLATION AT PT XI.
C  X=ARRAY OF DISTINCT SEQUENTIAL VALUES.
C  Y=ARRAY VALUES CORRESPONDING TO X.
C  NNP=NUMBER OF POINTS USED IN CALCULATION (NNP-1=ORDER OF INTER-
C  POLATION POLYNOMIAL).
C  IIMAX=MAX NUMBER OF POINTS IN X AND Y ARRAYS.
C  IFLAG=-1 MEANS  XI  IS SMALLEST X USED IN CALC POLYNOMIAL.
C  IFLAG=+1 MEANS  XI  IS LARGEST X IN CALC POLYNOMIAL.
C  IFLAG=2  MEANS  XI  NOT IN X ARRAY. CALCULATION SUPPRESSED.
C  IFLAG=3  MEANS NNP LARGER THAN IIMAX. NNP MADE EQUAL TO IIMAX.
C
      DIMENSION X(1),Y(1)
      DOUBLE PRECISION X,Y,XI,XSUBI,PRODA,SUMB,SUMC,PRODD,D,DX
      NP=NNP
      IMAX=IIMAX
      XSUBI=XI
      IFLAG=0
      IF(NP.LE.IMAX) GO TO 1
      IFLAG=3
      NP=IMAX
1    DO 2 J=1,IMAX
      IF(XSUBI.EQ.X(J)) GO TO 3
2    CONTINUE
      IFLAG=2
      RETURN
3    IU=J+(NP+1)/2-1
      IL=IU-(NP-1)
      IF(IU.LT.IMAX) GO TO 4
      IU=IMAX
      IL=IU-(NP-1)
4    IF(IL.GE.1) GO TO 5
      IL=1
      IU=IL+(NP-1)
5    IF(J.NE.IU) GO TO 6
      IFLAG=1
      GO TO 7
6    IF(J.NE.IL) GO TO 7
      IFLAG=-1
7    PRODA=1.
      SUMB=0.
      SUMC=0.
      DO 11 K=IL,IU
      IF (J.EQ.K) GO TO 11
      PRODA=PRODA*(X(J)-X(K))
      SUMC=SUMC+Y(J)/(X(J)-X(K))
      DXLG0001
      DXLG0002
      DXLG0003
      DXLG0004
      DXLG0005
      DXLG0006
      DXLG0007
      DXLG0008
      DXLG0009
      DXLG0010
      DXLG0011
      DXLG0012
      DXLG0013
      DXLG0014
      DXLG0015
      DXLG0016
      DXLG0017
      DXLG0018
      DXLG0019
      DXLG0020
      DXLG0021
      DXLG0022
      DXLG0023
      DXLG0024
      DXLG0025
      DXLG0026
      DXLG0027
      DXLG0028
      DXLG0029
      DXLG0030
      DXLG0031
      DXLG0032
      DXLG0033
      DXLG0034
      DXLG0035
      DXLG0036
      DXLG0037
      DXLG0038
      DXLG0039
      DXLG0040
      DXLG0041
      DXLG0042
      DXLG0043
      DXLG0044
      DXLG0045
      DXLG0046
      DXLG0047
      DXLG0048
      DXLG0049
      DXLG0050

```



```
8 PRODD=1.  
DO 10 L=IL,IU  
IF (L.EQ.K) GO TO 9  
D=X(K)  
GO TO 10  
9 D=X(J)  
10 PRODD=PRODD*(D-X(L))  
SUMB=SUMB+Y(K)/PRODD  
11 CONTINUE  
DX=PRODA*SUMB+SUMC  
RETURN  
END
```

```
DXLG0051  
DXLG0052  
DXLG0053  
DXLG0054  
DXLG0055  
DXLG0056  
DXLG0057  
DXLG0058  
DXLG0059  
DXLG0060  
DXLG0061  
DXLG0062
```

C

NUMERICAL
INTEGRATION

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Integration by Gaussian quadrature.

Purpose

This subroutine uses successively the three point Gaussian quadrature formula to evaluate the integral of a function.

Usage

The calling sequence is:

```
CALL INTGR (XL, XU, N, ANS, FUN)
```

where

XL is the double precision lower limit of integration,
XU is the double precision upper limit of integration,
N is the integer number of subintervals that [XL, XU]
is to be subdivided into (the integrand function will
be evaluated at 3N points between XL and XU),
ANS is the double precision approximation to the integral.
FUN is the name of a double precision function subprogram
which defines the integrand function. FUN is a
function of one double precision variable. Of course,
the user may specify some other name besides FUN.

Coding Information

INTGR is a Fortran IV double precision subroutine which may easily be converted to single precision. An EXTERNAL statement with the names of the function subprograms which define functions (integrands) to be integrated must be included in the calling program. Any

function called must be included in a DOUBLE PRECISION type statement in the calling program. INTGR calls the double precision function subprogram GAUSS which is included as part of the deck.

Restrictions

Slow or no convergence may occur when the integrand possesses singularities on the interval of integration or at points in the complex plane fairly near the interval of integration [1]. For example, the integral of $\sqrt{x}$ on the interval [0, 1] gave only 4 places of accuracy for $N = 10$ on an IBM 360. This is because the derivative of $\sqrt{x}$ does not exist at 0. In contrast, the integral of e^x on the interval [0, 1] gave 12 places of accuracy for $N = 10$.

Method

INTGR uses the three point Gaussian quadrature formula [1], [2] to give the approximate integral of $f(x)$:

$$\int_{-1}^1 f(x) dx \approx f[5(-\sqrt{15}/5) + 8f(0) + 5f(\sqrt{15}/5)]/9$$

The actual formulas used in INTGR were obtained by a transformation of the limits of integration. The three point formula is used for each of the N subintervals of the interval of integration, $[XL, XU]$. If the integral exists, it is not necessary that the integrand be defined at the end points XL or XU since INTGR does not ask that the function defined by "FUN" be evaluated at the end points.

Accuracy

In each of the N subintervals, the three point approximation used will be algebraically exact whenever the integrand is a polynomial of degree not greater than 5. Assuming no errors in INTGR, the error [1] in a subinterval is $f^{(6)}(x)/15750$ where x is some point in the subinterval. Also see Restrictions.

References

- [1] Hildebrand, F. B. (1956): Introduction to Numerical Analysis, McGraw-Hill, pp 323, 324, 329.
- [2] Krylov, V. I. (1962): Approximate Calculation of Integrals, (translated from Russian), Macmillan.

Author

John Clawson, formerly with Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE INTGR(XL,XU,N,ANS,FUN)
C  PROGRAM AUTHOR      J. CLAWSON, FORMERLY WITH
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC GRD PROGRAM NO. 9055
      DOUBLE PRECISION ANS,H,XU,XL,GAUSS,FUN
      EXTERNAL FUN
      ANS = 0.
      H = (XU-XL)/DFLOAT (N)
      DO 1 I=1,N
1  ANS=ANS+GAUSS(XL+(DFLOAT(I)-1.)*H,XU-(DFLOAT(N)-DFLOAT(I))*H,FUN)
      RETURN
      END
      DOUBLE PRECISION FUNCTION GAUSS(XL,XU,FUNCTN)
      DOUBLE PRECISION XL,XU,T,E,C,GAUSS,FUNCTN
      T = XU + XL
      E = XU - XL
C  THE CONSTANT IN THE FOLLOWING STATEMENT IS (1 - X)/2 WHERE X IS
C  THE ZERO = SQRT(15)/5 OF THE LEGENDRE POLYNOMIAL OF DEGREE 3
      C = .1127016653792583 * E
      GAUSS =E/18.*(5.*FUNCTN (C+XL)+8.*FUNCTN (T/2.)+5.*FUNCTN (XU-C))
      RETURN
      END

```

INGR0001
 INGR0002
 INGR0003
 INGR0004
 INGR0005
 INGR0006
 INGR0007
 INGR0008
 INGR0009
 INGR0010
 INGR0011
 INGR0012
 INGR0013
 INGR0014
 INGR 15
 INGR 16
 INGR 17
 INGR 18
 INGR 19
 INGR 20
 INGR 21
 INGR 22
 INGR 23
 INGR 24

Identification

Numerical integration by Simpson's rule.

Purpose

The subroutine SIMP uses Simpson's rule to give a numerical approximation to a function that is specified at evenly spaced points.

Usage

The limits of integration are from $X(1)$ to $X(NX)$.

The calling sequence is:

```
CALL SIMP (XX, FX, AX, NX)
```

where

XX is a monotone, strictly increasing, double precision, singly-dimensioned array of abscissas at which the integrand function was evaluated,

FX is a double precision, singly-dimensioned array of the integrand values which were evaluated at $XX(I)$, $I = 1, 2, \dots, NX$,

NX is the integer number of points at which the integrand is evaluated (must not be less than 3),

AX is a double precision singly-dimensioned array which contains the value of the integral from $XX(1)$ to $XX(I)$, $I = 1, 2, 3, \dots, NX$.

Coding Information

SIMP is a Fortran IV double precision subprogram which can easily be converted to single precision. The arrays entered in the

argument list of SIMP must be dimensioned not less than NX in the calling program, but the dimensions in SIMP are irrelevant.

Strictly speaking, Simpson's rule is only for meshes with an even number of intervals, but SIMP allows for an odd number (see Method).

Accuracy

Assuming an even number of intervals and no rounding errors in SIMP, the error is given by

$$[XX(1) - XX(NX)] h^4 f^{(4)}(x)/180$$

for some x between XX(1) and XX(NX) and h being the interval size [1],[2]. Also if there is an even number of intervals, SIMP will integrate exactly any cubic polynomial.

Method

For an even number of intervals with interval size h, the following formula [1],[2] is used to approximate the integral:

$$h[FX(1) + 4FX(2) + 2FX(3) + 4FX(4) + \dots + 4FX(NX - 1) + FX(NX)]/3$$

If the number of intervals is odd, the approximation to the integral of the last interval is added to the foregoing:

$$h[5FX(NX) + 8FX(NX - 1) - FX(NX - 2)]/12$$

References

- [1] Hildebrand, F. B. (1956): Introduction to Numerical Analysis, McGraw-Hill, pp 64-82.

- [2] Conte, S. D. (1965): Elementary Numerical Analysis, McGraw-Hill, p. 135.

Author

R. Mashburn, formerly with the Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

SUBROUTINE SIMP(XX,FX,AX,NX)	SIMP0001
C PROGRAM AUTHOR R. MASHBURN, FORMERLY WITH	SIMP0002
C COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	SIMP0003
C OAK RIDGE, TENN.	SIMP0004
C	SIMP0005
C CTC ORD PROGRAM NO. 9056	SIMP0006
DIMENSION XX(2),FX(2),AX(2)	SIMP0007
DOUBLE PRECISION XX,FX,AX,DELX	SIMP0008
DELX = XX(2)-XX(1)	SIMP0009
AX(1)=0.0	SIMP0010
DO 1 IX = 2,NX,2	SIMP0011
IF (NX .EQ. IX) GO TO 2	SIMP0012
AX(IX)=DELX*(5.0*FX(IX-1)+8.0*FX(IX)-FX(IX+1))/12.0+AX(IX-1)	SIMP0013
1 AX(IX+1)=DELX*(FX(IX-1)+4.0*FX(IX)+FX(IX+1))/3.0+AX(IX-1)	SIMP0014
GO TO 3	SIMP0015
2 AX(NX)=DELX*(-FX(NX-2)+8.0*FX(NX-1)+5.0*FX(NX))/12.0+AX(NX-1)	SIMP0016
3 RETURN	SIMP0017
END	SIMP0018

Identification

Quadratic integration over unevenly spaced points.

Purpose

SIMPUN is a combination of quadratic integration and the trapezoidal rule over unevenly spaced points. SIMPUN is a suggested alternative and often superior to trapezoidal integration. The numerical approximation of the integral may be specified as forward or backward along the interval of integration. SIMPUN saves the accumulated value of the integral along subintervals. This may be valuable in applications in which the integral is required as a function. Also functions exist for which backward integration is much more accurate, e.g. some integrands which are small with respect to the integral on the left portion of the interval of integration.

Usage

The calling sequence is:

```
CALL SIMPUN (XX, FX, NX, L, AX)
```

where

XX is a strictly increasing, double precision, singly-dimensioned array of abscissas at which the integrand was evaluated,

FX is a double precision, singly-dimensioned array of integrand values which were evaluated at XX(I),
I=1,2,...,NX,

NX is the integer number of points at which the integral was evaluated,

L is an integer which determines the direction of integration. If L is non-negative, e.g. +1, integration is forward. If L is negative, e.g. -1, integration is backwards (see next argument),

AX is a double precision, singly-dimensioned array which contains the values of the integral on subintervals:

- (a) $L \geq 0$ means that AX(I) will contain the approximation to the integral with lower limit XX(1) and upper limit XX(I), $I=1,2,\dots,NX$.
- (b) $L < 0$ means that AX(I) will contain the approximation to the integral with lower limit XX(I) and upper limit XX(NX), $I=NX,NX-1,\dots,1$.

Coding Information

SIMPUN is a Fortran IV, double precision subprogram which can be easily converted to single precision. The arrays entered in the argument list of SIMPUN must be dimensioned not less than NX in the calling program, but the dimensions in SIMPUN are irrelevant.

Accuracy

For an even number of subintervals SIMPUN will integrate exactly any quadratic polynomial function. For evenly spaced intervals SIMPUN is equivalent to Simpson's rule.

Method

The method in the case of an even number of intervals is a generalization of Simpson's rule [1] and was derived as suggested by Milne [2]:

For three points set the integral

$$\int_{x_1}^{x_3} f_i(X) dX = a_1 f_i(x_1) + a_2 f_i(x_2) + a_3 f_i(x_3)$$

and let $f_i(X)$ take the value 1, X and X^2 to solve for the undetermined coefficients. The result is

$$a_3 = (x_3 - x_1) \left[\frac{1}{2} - \frac{x_3 - x_1}{6(x_3 - x_2)} \right]; a_2 = \frac{(x_3 - x_1)^3}{6(x_2 - x_1)(x_3 - x_2)}$$

and $a_1 = x_3 - x_1 - a_2 - a_3$

For two points set the integral

$$\int_{x_1}^{x_2} f_i(X) dX = a_1 f_i(x_1) + a_2 f_i(x_2)$$

and let $f_i(X)$ take the values 1 and X to obtain

$$a_1 = a_2 = \frac{x_2 - x_1}{2} \text{ which is the trapezoided rule.}$$

Hence the basic quadrature formula for each three point panel is $a_1 f(x_1) + a_2 f(x_2) + a_3 f(x_3)$ where f is the integrand and the x 's are successive points.

References

- [1] Hildebrand, F. B. (1956): Introduction to Numerical Analysis, McGraw-Hill, pp. 64-82, p. 75.

- [2] Milne, W. E. (1949): Numerical Calculus, Princeton,
pp. 104-116.

Author

Joseph Barish, Computing Technology Center, Union Carbide Corporation,
Nuclear Division, Oak Ridge, Tennessee.

SUBROUTINE SIMPUN(XX,FX,NX,1,AX)	SIMN0001
C PROGRAM AUTHOR J. BARISH,	SIMN0002
C COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	SIMN0003
C OAK RIDGE, TENN.	SIMN0004
C	SIMN0005
C CTC ORD PROGRAM NO. 9057	SIMN0006
DIMENSION XX(2),FX(2),AX(2)	SIMN0007
DOUBLE PRECISION XX,FX,AX,D1,D2,D3,A2,A3	SIMN0008
IF (I .LT.0) GO TO 3	SIMN0009
AX(1)=0.0	SIMN0010
DO 1 IX=2,NX,2	SIMN0011
D1=XX(IX)-XX(IX-1)	SIMN0012
AX(IX)=AX(IX-1)+D1/2.0*(FX(IX)+FX(IX-1))	SIMN0013
IF (NX .EQ. IX) GO TO 2	SIMN0014
D2=XX(IX+1)-XX(IX-1)	SIMN0015
D3=D2/D1	SIMN0016
A2=D3/6.0*D2**2/(XX(IX+1)-XX(IX))	SIMN0017
A3=D2/2.0-A2/D3	SIMN0018
1 AX(IX+1)=AX(IX-1)+(D2-A2-A3)*FX(IX-1)+A2*FX(IX)+A3*FX(IX+1)	SIMN0019
2 RETURN	SIMN0020
3 AX(NX)=0.0	SIMN0021
DO 4 IX=2,NX,2	SIMN0022
IC=NX+1-IX	SIMN0023
D1=XX(IC+1)-XX(IC)	SIMN0024
AX(IC)=AX(IC+1)+D1/2.0*(FX(IC+1)+FX(IC))	SIMN0025
IF (NX .EQ.0) GO TO 2	SIMN0026
D2=XX(IC+1)-XX(IC-1)	SIMN0027
D3=D2/(XX(IC)-XX(IC-1))	SIMN0028
A2=D3/6.0*D2**2/D1	SIMN0029
A3=D2/2.0-A2/D3	SIMN0030
4 AX(IC-1)=AX(IC+1)+(D2-A2-A3)*FX(IC-1)+A2*FX(IC)+	SIMN0031
A3*FX(IC+1)	SIMN0032
RETURN	SIMN0033
END	SIMN0034

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Numerical integration by Laguerre-Gaussian quadrature.

Purpose

To give a numerical approximation to the integral $\int_0^{\infty} e^{-\alpha x} f(x) dx$ by a 5, 10, or 15 point Laguerre-Gaussian quadrature formula.

Usage

The calling sequence is:

CALL LAGUER (NOPT, ALPHA, ANS, FUNX)

where

NOPT is 5, 10, or 15 for a 5, 10 or 15 point quadrature respectively (integer),

ALPHA is the double precision value of α (see Purpose),

ANS is the double precision approximation to the integral,

FUNX is the name of the double precision function subprogram in which the function to be evaluated is defined, i.e., the $f(x)$ portion of the integrand (see Purpose).

Coding Information

LAGUER is a Fortran IV subprogram which can easily be converted to single precision, but this would necessitate changing the constants in the DATA statements to single precision. NOPT and ALPHA are input arguments from the calling program. There must be an EXTERNAL statement with the names of the function subprograms in which functions, $f(x)$, (see Purpose) are defined, in the calling program, e.g.

EXTERNAL FUNX1, FUNX2

where FUNX1 and FUNX2 are double precision function subprograms defining functions, $f(x)$.

Accuracy

If $f(x)$ can be expressed as a polynomial of degree no greater than $2(\text{NOPT})-1$, this routine gives exact answers except for those errors due to roundoff. The exact form of the error term which involves the derivative which is a function of NOPT can be found in [1, 2].

Method

The Laguerre-Gauss quadrature, for the case when the weighting function

$$w(x) = e^{-\alpha x}$$

is used over the semi-infinite interval $(0, \infty)$, becomes

$$\int_0^{\infty} e^{-\alpha x} f(x) dx \approx \frac{1}{\alpha} \sum_{k=1}^n W_k f\left(\frac{x_k}{\alpha}\right)$$

where W_k is the weight corresponding to a particular abscissa x_k .

A table of weights and abscissas is found in [2].

References

- [1] Hildebrand, F. B. (1956): Introduction to Numerical Analysis, McGraw-Hill, pp. 325-327.
- [2] Krylov, V. I, (1962): Approximate Calculation of Integrals, Macmillan, pp. 347-350.

Author

F. D. Hammerling, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUPROUTINE LAGUER(NN,ALPH,VALUE,FUCTN)
C   PROGRAM AUTHOR  F. D. HAMMERLING,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC ORD PROGRAM NO. 9058
      DIMENSION X1(5),X2(10),X3(15),W1(5),W2(10),W3(15)
      DIMENSION X(15,3),W(15,3)
      EQUIVALENCE (X1,X(1,1)),(X2,X(1,2)),(X3,X(1,3)),(W1,W(1,1)),(W2,
1W(1,2)),(W3,W(1,3))
      DOUBLE PRECISION K,X,ALPH,FUCTN,VALU,FX,VALUE,X1,X2,X3,W1,W2,W3
      DATA  X1/.263560319718,1.413403059107,3.596425771041,7.08581000,
15859,12.640800844276/, X2/.137793470540,.729454549503,1.8083429,
201740, 3.401433697855,5.552496140064,8.330152746764,11.84378583790,
30,16.279257831378,21.996585811981,29.920597012274/, X3/.0933078,
412017,.492691740302,1.215595412071,2.269949526204,3.667622721751,5,
5.425336627414,7.565916226613,10.120228568019,13.130282482176,16.65,
64407708330,20.776478899449,25.623894226729,31.407519169754,38.5306,
783306486,48.026085572686/
      DATA  W1/.521755610583,.398666811083,.759424496817D-1,.36117586,
17992D-2,.233699723858D-4/, W2/.308441115765,.401119929155,.2180,
268287612,.620674560987D-1,.950151697518D-2,.753008388586D-3,.28259,
32334960D-4,.424931398496D-6,.183956482398D-8,.991182721961D-12/,
4 W3/.218234885940,.342210177923,.263027577942,.126425818106,.4020,
568649210D-1,.856387780361D-2,.121243614721D-2,.111674392344D-3,.64,
65992676202D-5,.222631690710D-6,.422743038498D-8,.392189726704D-10,
7.145651526407D-12,.148302705111D-15,.160059490621D-19/
      M=NN/5
      VALU=0.0
      DO 1 I=1,NN
      FX=W(I,M)*FUCTN(X(I,M)/ALPH)/ALPH
      VALU=VALU+FX
1 CONTINUE
      VALUE=VALU
      RETURN
      END

```

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Numerical integration by Hermite-Gaussian quadrature.

Purpose

To give a numerical approximation to the integral $\int_{-\infty}^{\infty} e^{-\alpha^2 x^2} f(x) dx$ by a 5, 10, or 15 point Hermite-Gaussian quadrature formula.

Usage

The calling sequence is:

```
CALL HERMIT (NOPT, ALPHSQ, ANS, FUNX)
```

where

NOPT is 5, 10, or 15, for a 5, 10, or 15 point quadrature respectively (integer),

ALPHSQ is the double precision coefficient of the exponential of x^2 (see Purpose),

ANS is the double precision approximation to the integral,

FUNX is the name of a double precision function subprogram in which the function to be evaluated is defined, i.e. the $f(x)$ portion of the integrand (see Purpose).

Coding Information

HERMIT is a Fortran IV subprogram which can easily be converted to single precision, but this would necessitate changing the constants in the DATA statements to single precision. NOPT and ALPHSQ are input arguments from the calling program. There must be an EXTERNAL statement with the names of the function subprograms in which functions, $f(x)$, (see Purpose) are defined in the calling program, e.g.

EXTERNAL FUNX1, FUNX2

where FUNX1 and FUNX2 are double-precision function subprograms defining functions, $f(x)$.

Accuracy

If $f(x)$ can be expressed as a polynomial of degree no greater than $2(\text{NOPT})-1$, this routine gives exact answers except for those errors due to roundoff. The error term [1] is

$$M! \sqrt{\pi} f^{(2M)}(x) / [2^M (2M)!]$$

where $M = \text{NOPT}$ and x is some real number [1].

Method

The Hermite-Gauss quadrature, for the case when the weighting function

$$w(x) = e^{-\alpha^2 x^2}$$

is used over the infinite interval $(-\infty, \infty)$, becomes

$$\int_{-\infty}^{\infty} e^{-\alpha^2 x^2} f(x) dx = \frac{1}{\alpha} \sum_{k=1}^N w_k \cdot f(x_k/\alpha)$$

where w_k is the weight corresponding to a particular abscissa x_k .

A table of weights and abscissas may be found in [2].

References

- [1] Hildebrand, F. B. (1956): Introduction to Numerical Analysis, McGraw-Hill, pp. 327-330.
- [2] Krylov, V. I. (1962): Approximate Calculation of Integrals, Macmillan, pp. 347-350.

Author

F. D. Hammerling, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.


```

      SUBROUTINE HERMIT(NN,ALPHA,VALUE,FUCTN)
C   PROGRAM AUTHOR   F. D. HAMMERLING,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC ORD PROGRAM NO. 9059
      DIMENSION X1(5),X2(10),X3(15),W1(5),W2(10),W3(15)
      DIMENSION X(15,3),W(15,3)
      EQUIVALENCE (X1,X(1,1)),(X2,X(1,2)),(X3,X(1,3)),(W1,W(1,1)),(W2,
1W(1,2)),(W3,W(1,3))
      DOUBLE PRECISION W,X,ALPH,VALU,FX,ALPHA,VALUE,FUCTN,X1,X2,X3,
1W1,W2,W3
      DATA      X1/-.958572464613819,-2.020182870456086,0.000,.9585724646HERM0001
113819,2.020182870456086/,      X2/-.342901327223705,-1.0366108297895HERM0002
214,-1.756683649299882,-2.532731674232790,-3.436159118837738,0.3429HERM0003
301327223705,1.036610829789514,1.756683649299882,2.532731674232790,HERM0004
43.436159118837738/,      X3/-.56506958325558,-1.13611558521092,-1.71HERM0005
5999257518649,-2.32573248617386,-2.96716692790560,-3.66995037340445HERM0006
6,-4.49999070730939,0.000,.56506958325558,1.13611558521092,1.719992HERM0007
757518649,2.32573248617386,2.96716692790560,3.66995037340445,4.4999HERM0008
89070730939/
      DATA      W1/.3936193231522,.1995324205905D-1,.9453087204825,.39361HERM0009
193231522,.1995324205905D-1/,      W2/.6108626337353,.2401386110823,.HERM0010
23387439445548D-1,.1343645746781D-2,.7640432855233D-5,.610862633735HERM0011
33,.2401386110823,.3387439445548D-1,.1343645746781D-2,.764043285523HERM0012
43D-5/,      W3/.4120286874989,.1584889157959,.3078003387255D-1,.2778HERM0013
5068842913D-2,.1000044412325D-3,.1059115547711D-5,.1522475804254D-8HERM0014
6,.5641003087264,.4120286874989,.1584889157959,.3078003387255D-1,.2HERM0015
7778068842913D-2,.1000044412325D-3,.1059115547711D-5,.1522475804254HERM0016
8D-8/
      M=NN/5
      ALPH= DSQRT(ALPHA)
      VALU=0.0
      DO 1 I=1,NN
      FX=W(I,M)*FUCTN(X(I,M)/ALPH)/ALPH
      VALU=VALU+FX
1  CONTINUE
      VALUE=VALU
      RETURN
      END

```

D
ORDINARY
DIFFERENTIAL
EQUATIONS

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Solution of simultaneous first order differential equations.

Purpose

KUTTA integrates a system of first order differential equations $dY_i/dX=F_i(X,Y_1,Y_2,\dots,Y_{NE})$ over some interval $[x_0,x_1]$ using a self-adjusting step size.

Usage

The calling sequence is:

```
CALL KUTTA(X0, X1, Y, NE, DEL, ACCURC, IMAX, EQUA)
```

where

X0 is the initial double-precision abscissa of the interval,

X1 is the final double-precision abscissa of the interval,

Y is a singly-dimensioned, double-precision array of initial values of the dependent variable, i.e. Y_i evaluated at X0. Upon return to the calling program, Y contains the values of the dependent variable at X1,

NE is the integer number of differential equations to be solved,

DEL is the initial double-precision estimate of the step size and upon return to the calling program, DEL contains the final step size, i.e. the original DEL is, in general, overwritten.

ACCURC is a double-precision preassigned accuracy which is used in adjusting the step size. Values from 10^{-6} to 10^{-8} seem to work very well for most problems,

IMAX is the maximum integer number of times the step size is to be halved (see Method and Coding Information),

EQUA is the name of a subroutine called by KUTTA which, of course, may be specified by some other name in the calling program (see Coding Information).

Coding Information

KUTTA is a Fortran IV double-precision subprogram which can easily be converted to single precision. When calling KUTTA an EXTERNAL statement with the name(s) of subroutine(s) defining the system(s) of equations must appear in the calling program. If more than 10 equations are to be solved, the dimensions in KUTTA can easily be changed. The user should set IMAX so that DEL does not underflow. If the limit IMAX is reached in the interval halving, KUTTA returns control to the calling program.

It is required that the user of KUTTA write a subroutine defining the differential equations to be evaluated, e.g. EQUA (X,Y,YP)

where

X is the double-precision independent variable

Y is the double-precision singly-dimensioned array of dependent variables.

YP is the double-precision, singly-dimensioned array
defining the differential equations:

$$YP_i = dY_i/dX = F_i(X, Y_1, \dots, Y_{NE}).$$

Method

The differential equations are integrated by a one-step Runge-Kutta-Merson technique which is a modification of fourth order Runge-Kutta integration [1]. The formulae are:

$$(1) \quad Y_i^{n+1} = Y_i^n + (K1_i + 4 \cdot K4_i + K5_i)/2 + O(h^5)$$

where

$$K1_i = h \cdot F_i(X^n, Y_1^n, Y_2^n, \dots, Y_{NE}^n)/3$$

$$K2_i = h \cdot F_i[X^n + (h/3), Y_1^n + K1_1, Y_2^n + K1_2, \dots, Y_{NE}^n + K1_{NE}]/3$$

$$K3_i = h \cdot F_i[X^n + (h/3), Y_1^n + (K1_1/2) + (K2_1/2), Y_2^n + (K1_2/2) + (K2_2/2), \dots, Y_{NE}^n + (K1_{NE}/2) + (K2_{NE}/2)]/3$$

$$K4_i = h \cdot F_i[X^n + (h/2), Y_1^n + (3 \cdot K1_1/8) + (9 \cdot K3_1/8), Y_2^n + (3 \cdot K1_2/8) + (9 \cdot K3_2/8), \dots, Y_{NE}^n + (3 \cdot K1_{NE}/8) + (9 \cdot K3_{NE}/8)]/3$$

$$K5_i = h \cdot F_i[X^n + h, Y_1^n + (3 \cdot K1_1/2) - (9 \cdot K3_1/2) + 6 \cdot K4_1, \dots, Y_{NE}^n + (3 \cdot K1_{NE}/2) - (9 \cdot K3_{NE}/2) + 6 \cdot K4_{NE}]/3$$

$O(h^5)$ indicates that the quantity on the right side of (1) provides an approximation to the left side correct to terms of order h^5 as $h \rightarrow 0$. This is not used in the actual calculation [2].

In the above equations, the subscript i refers to the i^{th} differential equation, the superscript n refers to the previous point just calculated, while $n+1$ is the point to be calculated.

The advantage of this method is that an estimate of the truncation error, e , is given by

$$e_i = K1_i - (9 \cdot K3_i / 2) + 4 \cdot K4_i - (K5_i / 2)$$

which is accurate enough for automatic step-size adjustment.

If $\text{MAX}(e_i)$ is greater than the preassigned accuracy, the interval, h , is halved and the computation for the step is begun again; but if $\text{MAX}(e_i)$ is less than $1/32$ of the preassigned accuracy, the interval is doubled and the calculation for the next step is begun. This method requires five function evaluations per step.

Timing

This routine was found to be faster than the standard Runge-Kutta method because of the fact that one usually underestimates the step-size for his desired accuracy. Also this routine eliminates the time-consuming procedure of choosing a very small mesh in order to obtain the desired accuracy at a few particular intervals and hence, having to use this same mesh throughout the calculation.

With this routine the mesh is changed automatically when the need arises.

References

- [1] Lance, G. N. (1960): Numerical Methods for High Speed Computers, Iliffe & Sons, pp. 54-57.
- [2] Conte, S. D. (1965): Elementary Numerical Analysis, McGraw-Hill, pp. 114-117.

Author

F. D. Hammerling, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.


```

      SUBROUTINE KUTTA(XL,XU,Y,NE,DEL,ACCURC,IMAX,EQUA)
C  PROGRAM AUTHOR   F. D. HAMMERLING,
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9060
      DIMENSION Y(1),YI(10),YN(10),K1(10),K2(10),K3(10),
1      K4(10),K5(10),F(10),E(10),F1(10)
      DOUBLE PRECISION Y,F,E,F1,XN,XL,DEL,XU,H,TEST,ACCURC,YN,YI,K1,
1K2,K3,K4,K5
      LOGICAL QUIT
      ITTER=0
      N=NE
      XN=XL
      H=DEL
      QUIT=.FALSE.
      DO 1 I=1,N
1  YN(I)=Y(I)
2  IF(XN+H.LT.XU)GO TO 3
      DEL=H
      H=XU-XN
      QUIT=.TRUE.
3  CALL EQUA(XN,YN,F1)
4  DO 5 I=1,N
      K1(I)=H*F1(I)/3.
5  YI(I)=YN(I)+K1(I)
      CALL EQUA(XN+H/3.,YI,F)
      DO 6 I=1,N
      K2(I)=H*F(I)/3.
6  YI(I)=YN(I)+K1(I)/2.+K2(I)/2.
      CALL EQUA(XN+H/3.,YI,F)
      DO 7 I=1,N
      K3(I)=H*F(I)/3.
7  YI(I)=YN(I)+3.*K1(I)/8.+9.*K3(I)/8.
      CALL EQUA(XN+H/2.,YI,F)
      DO 8 I=1,N
      K4(I)=H*F(I)/3.
8  YI(I)=YN(I)+3.*K1(I)/2.-9.*K3(I)/2.+6.*K4(I)
      CALL EQUA(XN+H,YI,F)
      TEST=0.0
      DO 9 I=1,N
      K5(I)=H*F(I)/3.
      E(I)=(K1(I)-9.*K3(I)/2.+4.*K4(I)-K5(I)/2.)/5.
      TEST=DMAX1(TEST,DABS(E(I)))
9  CONTINUE
      IF(TEST.LT.ACCURC)GO TO 10
      ITTER=ITTER+1
      IF(ITTER.GE.IMAX)GO TO 13
      H=H/2.
      QUIT=.FALSE.

```

```

KUTT0001
KUTT0002
KUTT0003
KUTT0004
KUTT0005
KUTT0006
KUTT0007
KUTT0008
KUTT0009
KUTT0010
KUTT0011
KUTT0012
KUTT0013
KUTT0014
KUTT0015
KUTT0016
KUTT0017
KUTT0018
KUTT0019
KUTT0020
KUTT0021
KUTT0022
KUTT0023
KUTT0024
KUTT0025
KUTT0026
KUTT0027
KUTT0028
KUTT0029
KUTT0030
KUTT0031
KUTT0032
KUTT0033
KUTT0034
KUTT0035
KUTT0036
KUTT0037
KUTT0038
KUTT0039
KUTT0040
KUTT0041
KUTT0042
KUTT0043
KUTT0044
KUTT0045
KUTT0046
KUTT0047
KUTT0048
KUTT0049
KUTT0050

```

```
      GO TO 4
10 DO 11 I=1,N
11 YN(I)=YN(I)+(K1(I)+4.*K4(I)+K5(I))/2.
    XN=XN+H
    IF(TEST.LT.ACCURC/32.)H=2.*H
    IF(.NOT.QUIT)GO TO 2
    DO 12 I=1,N
12 Y(I)=YN(I)
    IMAX = ITTER
    GO TO 14
13 DEL=H
14 RETURN
    END
```

```
KUTTC051
KUTTC052
KUTTC053
KUTTC054
KUTTC055
KUTTC056
KUTTC057
KUTTC058
KUTTC059
KUTTC060
KUTTC061
KUTTC062
KUTTC063
```

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

First order differential equations solver.

Purpose

To integrate a system of N first order differential equations

$dy_i/dx = F_i(x, y_1, y_2, \dots, y_n)$ over some interval $[x_0, x_1]$.

Usage

The calling sequence is:

```
CALL DIFEQ(X0,X1,Y,N,K,EQUA)
```

where

X0 is the initial double precision abscissa of the interval.

X1 is the final double precision abscissa of the interval.

Y is a double precision singly dimensioned array of initial values of the dependent variable, i.e. the Y_i 's evaluated at X0. Upon return to the calling program, Y contains the values of the dependent variable at X1.

N is the integer number of equations to be solved.

K is the integer number of subintervals into which $[X0;X1]$ is divided.

EQUA is the name of a subroutine called by DIFEQ which, of course, may be specified by some other name in the calling program (see Coding Information).

Coding Information

DIFEQ is a Fortran IV double precision subprogram which can easily be converted to single precision. When calling DIFEQ, an EXTERNAL statement must appear in the calling program with the name(s) of the subprogram(s) which define the system(s) of equations. The Y array must be dimensioned in the calling program. If the system is of order greater than 10, the arrays in DIFEQ and EQUA must be redimensioned accordingly.

It is required that the user of DIFEQ write a subroutine defining the differential equations to be evaluated, e.g., EQUA (X,Y,YP)

where

X is the double precision independent variable,
 Y is the double precision singly-dimensioned array
 of dependent variables,
 YP is the double precision singly-dimensioned array
 defining the differential equations:

$$YP_i = dY_i/dX = F_i(X, Y_1, \dots, Y_N).$$

Method

The interval $[X_0, X_1]$ is divided into K equal sub-intervals. With this sub-interval used as a step size, the equations are integrated over the interval $[X_0, X_1]$ using the Runge-Kutta fourth order method. The following formulae [1] are used in the calculation (assuming N=1):

$$Y^{n+1} = Y^n + (K_1 + 2K_2 + 2K_3 + K_4)/6$$

where

$$K_1 = h \cdot F(X^n, Y^n)$$

$$K_2 = h \cdot F(X^n + h/2, Y^n + K_1/2)$$

$$K_3 = h \cdot F(X^n + h/2, Y^n + K_2/2)$$

$$K_4 = h \cdot F(X^n + h, Y^n + K_3)$$

where the superscript $(n+1)$ indicates the $(n+1)^{th}$ point being calculated while the n^{th} point is the previous calculated point, i.e. when only one interval is considered, the $(n+1)^{th}$ point is X_1 while the n^{th} point is X_0 (this assumes $K=1$).

The application of these formulae requires four substitutions into the differential equations. The h in the above formulae is the length of one of the K sub-intervals.

Reference

- [1] Hildebrand, F. B. (1956): Introduction to Numerical Analysis, McGraw-Hill.

Author

J. L. Cox, formerly with the Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

	SUBROUTINE DIFEQ(X,XE,Y,NN,I,EQUA)	DIFQ0001
C	PROGRAM AUTHOR J. L. COX, FORMERLY WITH	DIFQ0002
C	COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	DIFQ0003
C	OAK RIDGE, TENN.	DIFQ0004
C		DIFQ0005
C	CTC ORD PROGRAM NO. 9061	DIFQ0006
	DIMENSION Y(10), YI(10), YF(10), AM1(10), AM2(10), AM3(10), AM4(10)	DIFQ0007
	DOUBLE PRECISION Y, YI, YF, AM1, AM2, AM3, AM4, X, XE, FI, H, XF	DIFQ0008
	XF=X	DIFQ0009
	FI=I	DIFQ0010
	H=(XE-XF)/FI	DIFQ0011
	N=NN	DIFQ0012
	DO1J=1,N	DIFQ0013
1	YF(J)=Y(J)	DIFQ0014
	DO5J=1,I	DIFQ0015
	CALL EQUA(XF,YF,AM1)	DIFQ0016
	XF=XF+H/2.0	DIFQ0017
	DO2K=1,N	DIFQ0018
2	YI(K)=YF(K)+AM1(K)*H/2.0	DIFQ0019
	CALL EQUA(XF,YI,AM2)	DIFQ0020
	DO3K=1,N	DIFQ0021
3	YI(K)=YF(K)+AM2(K)*H/2.0	DIFQ0022
	CALL EQUA(XF,YI,AM3)	DIFQ0023
	XF=XF+H/2.0	DIFQ0024
	DO4K=1,N	DIFQ0025
4	YI(K)=YF(K)+AM3(K)*H	DIFQ0026
	CALL EQUA(XF,YI,AM4)	DIFQ0027
	DO5K=1,N	DIFQ0028
5	YF(K)=H/6.0*(AM1(K)+AM4(K)+2.0*(AM2(K)+AM3(K)))+YF(K)	DIFQ0029
	DO6K=1,N	DIFQ0030
6	Y(K)=YF(K)	DIFQ0031
	RETURN	DIFQ0032
	END	DIFQ0033

Identification

Solution of simultaneous second order differential equations.

Purpose

To evaluate a system of second order differential equations over some interval $[X_0, X_1]$.

Usage

The calling sequence is:

```
CALL DIFE2 (X0,X1,Y,YP,K,N,EVAL)
```

where

X0	is the initial double precision abscissa of the interval.
X1	is the final double precision abscissa of the interval.
Y	is the singly-dimensioned, double precision array of initial values of the dependent variable. Upon return to the calling program, it contains the final values of the dependent variable at X1.
YP	is the singly-dimensioned, double precision array of initial values of the first derivative of the functions with respect to the independent variable.
K	is the integer number of differential equations in the system to be solved.
N	is the integer number of subintervals into which the interval $[X0, X1]$ is to be divided.

EVAL is the name of the subroutine defining the system of equations to be solved.

The user of this subroutine is required to write a subroutine defining the system of equations to be evaluated. For this subprogram, the calling sequence is:

CALL EVAL (X,Y,YP,D2YDX2)

where

X is the double precision independent variable,
 Y is the singly-dimensioned, double precision array of dependent variables,
 YP is the same as YP above,
 D2YDX2 is the singly-dimensioned, double precision array of values for the second derivative of the functions with respect to the independent variable defined:

$$D2YDX2 \equiv \frac{d^2 y_i}{dx^2} = f_i(x, y_1, \dots, y_n, \frac{dy_1}{dx}, \dots, \frac{dy_n}{dx})$$

Coding Information

DIFE2 is a Fortran IV double precision subprogram which may be easily converted to single precision. An EXTERNAL type statement must occur in the calling program with the name(s) of the subprogram(s) defining the system(s) of equations to be solved. DIFE2 is dimensioned to accommodate systems up to 10 equations but obvious changes may be made for larger systems.

Method

The system of differential equations is solved by applying the second order Runge-Kutta approximations on an evenly spaced interval [1].

Reference

- [1] Scarborough, J. B. (1950): Numerical Mathematical Analysis, Oxford, pg. 301.

Author

Mark Elson, formerly a summer employee with Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE DIFE2(XO,X1,Y,YP,K,N,EQU2)
C  PROGRAM AUTHOR      M. ELSON, FORMERLY WITH
C  COMPUTER TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9062
      DIMENSION Y(10),YP(10),Q1(10),Q2(10),Q3(10),Q4(10),Y1(10),YP1(10),
1 F(10)
      DOUBLE PRECISION Y,YP,Q1,Q2,Q3,Q4,Y1,YP1,FN,H,A,XP,XO,F,FL,X1
      FN=N
      H=(X1-XO)/FN
      A=H**2/2.
      XP=XO
      DO 5 L=1,N
      CALL EQU2(XP,Y,YP,F)
      DO 1 I=1,K
      Q1(I)=A*F(I)
      Y1(I)=Y(I)+H*YP(I)/2.+Q1(I)/4.
1  YP1(I)=YP(I)+Q1(I)/H
      CALL EQU2(XP+H/2.,Y1,YP1,F)
      DO 2 I=1,K
      Q2(I)=A*F(I)
2  YP1(I)=YP(I) +Q2(I)/H
      CALL EQU2(XP+H/2.,Y1,YP1,F)
      DO 3 I=1,K
      Q3(I)=A*F(I)
      Y1(I)=Y(I)+YP(I)*H+Q3(I)
3  YP1(I)=YP(I)+2.*Q3(I)/H
      CALL EQU2(XP+H,Y1,YP1,F)
      DO 4 I=1,K
      Q4(I)=A*F(I)
      Y(I)=Y(I)+YP(I)*H+(Q1(I)+Q2(I)+Q3(I))/3.
4  YP(I)=YP(I)+(Q1(I)+2.*Q2(I)+2.*Q3(I)+Q4(I))/(3.*H)
      FL=L
5  XP = XO+H*FL
      RETURN
      END

```

```

DIF20001
DIF20002
DIF20003
DIF20004
DIF20005
DIF20006
DIF20007
DIF20008
DIF20009
DIF20010
DIF20011
DIF20012
DIF20013
DIF20014
DIF20015
DIF20016
DIF20017
DIF20018
DIF20019
DIF20020
DIF20021
DIF20022
DIF20023
DIF20024
DIF20025
DIF20026
DIF20027
DIF20028
DIF20029
DIF20030
DIF20031
DIF20032
DIF20033
DIF20034
DIF20035
DIF20036
DIF20037

```

E

ZEROS OF FUNCTIONS

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Linear fractional iteration for a zero of a real function.

Purpose

Given points x_1 , x_2 , and x_3 which are estimates of a zero of a function, $f(x)$, and the function values $f(x_1)$, $f(x_2)$ and $f(x_3)$, this statement function gives a "better" estimate of the zero using a linear fractional approximate.

Usage

ABCISA is a "statement function," formerly called arithmetic statement function. The calling sequence is:

$$X4 = \text{ABCISA} (X3, X2, X1, FX3, FX2, FX1)$$

where

$X4$ is the new estimate of the zero of f ;

$X1$, $X2$, and $X3$ are x_1 , x_2 , and x_3 ;

$FX1$, $FX2$, and $FX3$ are $f(x_1)$, and $f(x_2)$, and $f(x_3)$ (see Purpose).

Coding Information

The statement function

$$\begin{aligned} \text{ABCISA} (X3, X2, X1, Y3, Y2, Y1) = & X3 + ((Y1*Y3 - Y2*Y3)* \\ & (X3 - X1)*(X2 - X3)) / (Y1*Y2*(X1 - X2) + Y1*Y3*(X3 - X1) + \\ & Y2*Y3*(X2 - X3)) \end{aligned}$$

is placed before the first executable statement in the main program or subprogram in which you desire to find a zero of a function.

Restrictions

As in most methods discontinuities will cause difficulties. Also oscillating functions exist which will not converge using most any method.

Method

A function of the form $(x - a)/(bx + c)$ is fitted through the 3 data points, and a is used as the new estimate [1]. The order of convergence [1] is 1.84 which is not quite as good as Newton's method. However, this is only an asymptotic property, and we have found that linear fractional iteration often takes fewer iterations than Newton's method. Also it does not suffer from having to obtain and evaluate derivatives. Hence, it often takes less time even if it takes more iterations than other schemes.

Reference

- [1] Jarratt, P., and Nudds, D., (1965): "The use of rational functions in the iterative solution of equations on a digital computer," Computer J. 8, 1. pp. 62-65.

Author

J. E. Murphy, formerly with Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee, suggested linear fractional iteration as a very effective method in 1958. Coding: Anon.

Identification

Müller's iteration for a zero of a complex function.

Purpose

Given z_1, z_2, z_3 which are three distinct complex estimates for a zero of a complex function $f(z)$, and given the function values $f(z_1), f(z_2)$, and $f(z_3)$, this function subprogram gives a "better" approximation to a zero of $f(z)$.

Usage

The calling sequence is:

$Z4 = \text{CEST} (Z3, Z2, Z1, F3, F2, F1)$

where

CEST is the complex, double precision (COMPLEX*16) valued function subprogram which returns the new estimate of the zero of f .

Z4 is the new estimate of the zero of f .

Z1, Z2,

and Z3 are z_1, z_2 , and z_3 .

F1, F2,

and F3 are $f(z_1), f(z_2)$, and $f(z_3)$ (see Purpose).

Method

The method is an elegant implementation of finding a zero of the quadratic Lagrangian interpolate through the points $(z_1, f(z_1))$, $(z_2, f(z_2))$, and $(z_3, f(z_3))$. The zero of the quadratic lying closest to z_3 is the new and we hope better approximation, z_4 .

Reference

- [1] Müller, D. E. (1956): "A method for solving algebraic equations using an automatic computer," Mathematical Tables and Aids to Comp., Vol. 10, pp. 208-215.

Author

J. A. Gillcrist, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

C      FUNCTION CEAT(XI,XI1,XI2,FXI,FXI1,FXI2)
C      PROGRAM AUTHOR   J. A. GILLCRIST
C      COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C      OAK RIDGE, TENN.
C      CTC ORD PROGRAM NO. 9063
C      XI,XI1,XI2 ARE THREE EARLIER APPROXIMATIONS OF THE ROOT OF FUNCTION
C      FXI,FXI1,FXI2 ARE THE CORRESPONDING FUNCTION VALUES.
C      THIS SUBPROGRAM RETURNS A BETTER APPROXIMATION AS ITS ANSWER.
C
C      IMPLICIT COMPLEX*16 (A-H,O-Z)
C      HI=XI-XI1
C      HI1=XI1-XI2
C      EI=HI/HI1
C      DI=1.+EI
C      GI=FXI2*EI**2-FXI1*DI**2+FXI*(EI+DI)
C      TEMP=CDSQRT(GI**2-4.*FXI*DI*EI*(FXI2*EI-FXI1*DI+FXI))
C      DENOM=GI+TEMP
C      GIM=GI-TEMP
C      IF(CDABS(DENOM).LT.CDABS(GIM)) DENOM=GIM
C      IF(CDABS(DENOM).EQ.0.) GO TO 2
C      E=-2.*FXI*DI/DENOM
C
C      1 H=E*HI
C      CEAT=XI+H
C      RETURN
C
C      2 E=(1.,0.)
C      GO TO 1
C
C      END

```

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Polynomial equation solver.

Purpose

PØLSØL is a subroutine to find the roots $r_1, r_2, \dots, r_n$ of the polynomial equation

$$f(z) = a_1 + a_2 z + \dots + a_n z^{n-1} + a_{n+1} z^n = 0$$

by a variant of Lehmer's method.

Usage

The calling sequence is:

CALL PØLSØL (A,R,F,FP,N)

where

- A is a singly dimensioned, double precision, complex array which contains the coefficients of the polynomial, i.e., $A(I) = a_i, i=1, 2, \dots, |N|+1$.
- R is a singly dimensioned, double precision, complex array which on return contains the approximate roots r_i .
- F is a singly dimensioned, double precision, complex array which on return contains $f(r_i)$.
- FP is a singly dimensioned, double precision, complex array which on return contains $f'(r_i)$.
- N is an integer whose absolute value is the degree of the polynomial. If the value of N is negative, then the final purification step is skipped (see Method).

Method

The method is a combination of a variant of Lehmer's method and Newton's iteration. Lehmer's method [2] uses an algorithm of Cohn [1] to isolate a zero of $f(z)$ in a sequence of circles of decreasing radii. When a zero has been found to sufficient accuracy, it is divided out and the process restarted with the reduced polynomial. In PØLSØL, as soon as a zero has been isolated in a circle, an attempt is made to find it using Newton's method. If the iteration converges to a zero in the circle, that zero is divided out and the process restarted with the reduced polynomial. If the iteration fails, the usual Lehmer procedure is continued. After N approximate zeros have been found, they are purified by Newton's iteration in the original polynomial.

Accuracy

The accuracy to which PØLSØL finds a given zero will in general depend on how well that zero is determined by the coefficients of the polynomial. A well-conditioned zero which is not very sensitive to small perturbations in the coefficients will usually be calculated to high accuracy: a sensitive zero will be calculated inaccurately. In particular, repeated or closely clustered zeros tend to be ill-conditioned. Some measure of the accuracy of a given approximation r to a zero may be obtained by examining the ratio $f(r)/f'(r)$.

Timing

The Lehmer process is the most time-consuming part of the program; hence, the time required to find a zero will depend on how long the Lehmer process is continued before the zero is located by Newton's method. In general, it seems that a well-conditioned zero will be found almost immediately while a poorly conditioned zero will take a good deal more time.

Coding Information

PØLSØL is a Fortran IV double-precision complex subprogram. PØLSØL uses the following subroutines:

FUNDER evaluates a polynomial and its derivative.
SHIFT produces a polynomial with shifted zeros.
SCALE produces a polynomial with scaled zeros.
ANULUS finds an annulus in which a polynomial has a zero.
RØØT determines if a polynomial has a root in or on the unit circle.

The constant NEWTØN limits the number of Newton's iterations in any circle. The constant EPS is a relative convergence criterion for the Newton's iteration.

Restrictions

In the course of execution PØLSØL may produce a large number of floating point underflows. Hence, this program must be used with a subroutine which suppresses the printing out of underflow error messages.

References

- [1] Cohn, A. (1922): Über die Anzahl der Wurzeln einer algebraischen Gleichung in einem Kreise," Math. Z. 14, 110-148.
- [2] Lehmer, D. H. (1961): "A machine method for solving polynomial equations," J. ACM 8 151-162.

Author

G. W. Stewart III, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE POLSOL(AA,ZZ,P,DP,NN)
C  PROGRAM AUTHOR  G. W. STEWART, III
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9064
      COMPLEX*16 AA(1),ZZ(1),P(1),DP(1),A(51),B(51),C(51),Z,Q,ZI,
1      F,FP,DZ
      REAL*8 RS,R1,P2,RI
      COMPLEX*16 UNIT8(8)/(1.0000000000000000,0.0000000000000000),
1      (0.7071067811865472,-.7071067811865472),
2      (0.7071067811865472,0.7071067811865472),
3      (0.0000000000000000,-1.0000000000000000),
4      (0.0000000000000000,1.0000000000000000),
5      (-.7071067811865472,-.7071067811865472),
6      (-.7071067811865472,0.7071067811865472),
7      (-1.0000000000000000,0.0000000000000000)/
      REAL*8 CONQ/1.625D0/,CONRI/0.875D0/,
1      EPS/1.D-12/
      INTEGER*4 NEWTON/10/
C
C  INITIALIZATION
C
100  N = IABS(NN)
      NK = N
      DO 110 I=1,N
110  A(I) = AA(I)/AA(N+1)
      A(N+1) = 1.
      RS = 0.
C
C  LOCATE AN ANNULUS ABOUT ZERO CONTAINING A ROOT
C
200  Z = 0.
      IF(RS.GT.0.) GO TO 210
      R1 = 1.1*CDABS(A(1))*(1./FLOAT(NK))
      GO TO 220
210  R1 = -RS
220  CALL ANULUS(A,R1,F2,NK)
      RS = R1
      IF(R1.EQ.0.) GO TO 500
C
C  COVER THE ANNULUS WITH CIRCLES AND FIND ONE WITH A ROOT IN IT
C
300  Q = CONQ*R1
      IF(CDABS(Z).EQ.0.) GO TO 310
      Q = -Q*(Z/CDABS(Z))
      GO TO 320
310  IF(NK.EQ.N) GO TO 320
      IF(CDABS(ZZ(N-NK)).NE.0.) Q = Q*(DCONJG(ZZ(N-NK))/CDABS(ZZ(N-NK)))
320  RI = CONRI*R1

```

```

PLSL0001
PLSL0002
PLSL0003
PLSL0004
PLSL0005
PLSL0006
PLSL0007
PLSL0008
PLSL0009
PLSL0010
PLSL0011
PLSL0012
PLSL0013
PLSL0014
PLSL0015
PLSL0016
PLSL0017
PLSL0018
PLSL0019
PLSL0020
PLSL0021
PLSL0022
PLSL0023
PLSL0024
PLSL0025
PLSL0026
PLSL0027
PLSL0028
PLSL0029
PLSL0030
PLSL0031
PLSL0032
PLSL0033
PLSL0034
PLSL0035
PLSL0036
PLSL0037
PLSL0038
PLSL0039
PLSL0040
PLSL0041
PLSL0042
PLSL0043
PLSL0044
PLSL0045
PLSL0046
PLSL0047
PLSL0048
PLSL0049
PLSL0050

```


DO 330 I=1,6	PLSL0051
ZI = Z + Q*UNIT8(I)	PLSL0052
CALL SHIFT(A,B,ZI,NK)	PLSL0053
CALL SCALE(B,C,R1,NK)	PLSL0054
CALL ROOT(C,KKK,NK)	PLSL0055
GO TO (340,330),KKK	PLSL0056
330 CONTINUE	PLSL0057
R1 = 2.*R2	PLSL0058
LLL = 2	PLSL0059
GO TO 400	PLSL0060
340 LLL = 1	PLSL0061
R1 = P1	PLSL0062
Z = ZI	PLSL0063
C	PLSL0064
C ITERATE FOR THE ROOT, IF IT CANNOT BE FOUND, EITHER FIND A NEW	PLSL0065
C ANNULUS OR GET OUT.	PLSL0066
C	PLSL0067
400 ZI = Z + R1*(0.,.01)	PLSL0068
DO 410 I=1,NEWTON	PLSL0069
CALL FUNDER(A,B,ZI,F,FP,NK)	PLSL0070
DZ = -F/FP	PLSL0071
ZI = ZI + DZ	PLSL0072
IF(CDABS(ZI-ZI).GT.R1) GO TO (430,500),LLL	PLSL0073
IF(CDABS(DZ/ZI).LT.EPS) GO TO 420	PLSL0074
410 CONTINUE	PLSL0075
GO TO (430,420),LLL	PLSL0076
420 Z = ZI	PLSL0077
GO TO 500	PLSL0078
430 CALL SHIFT(A,B,Z,NK)	PLSL0079
CALL ANULUS(B,R1,P2,NK)	PLSL0080
IF(R1.EQ.0.) GO TO 500	PLSL0081
GO TO 300	PLSL0082
C	PLSL0083
C DEFLATE THE POLYNOMIAL AND TEST TO SEE IF ALL THE ROOTS HAVE	PLSL0084
C SEEN FOUND	PLSL0085
C	PLSL0086
500 CALL FUNDER(A,B,Z,F,FP,NK)	PLSL0087
510 ZZ(N-NK+1) = Z	PLSL0088
DO 520 I=1,NK	PLSL0089
520 A(I) = B(I)	PLSL0090
NK = NK-1	PLSL0091
IF(NK.NE.1) GO TO 200	PLSL0092
ZZ(N) = -A(1)	PLSL0093
C	PLSL0094
C PURIFY THE ROOTS IN THE ORIGINAL POLYNOMIAL	PLSL0095
C	PLSL0096
600 IF(NN.LT.0) GO TO 650	PLSL0097
DO 620 K=1,N	PLSL0098
DO 610 I=1,5	PLSL0099
CALL FUNDER(AA,B,ZZ(K),P(K),DP(K),N)	PLSL0100

```

      DZ = -P(K)/DP(K)
      IF(CDABS(DZ/ZZ(K)).LT.EPS) GO TO 620
610  ZZ(K) = ZZ(K) + DZ
620  CONTINUE
      RETURN
650  DO 660 I=1,N
660  CALL FUNDER(AA,B,ZZ(I),P(I),DP(I),N)
      RETURN
      END
      SUBROUTINE FUNDER(A,B,Z,F,FP,N)
C
C      SUBROUTINE TO EVALUATE THE N-TH DEGREE POLYNOMIAL WITH
C      COEFFICIENTS IN A AND ITS DERIVATIVE AT Z. THE VALUE IS RETURNED
C      IN F AND THE DERIVATIVE IN FP. THE COEFFICIENTS OF THE DEFLATED
C      POLYNOMIAL ARE RETURNED IN B.
C
      COMPLEX*16 A(1),B(1),Z,F,FP
      F = A(N+1)
      FP = 0.
      DO 10 I=1,N
      II = N-I+1
      B(II) = F
      F = A(II) + Z*F
10   FP = B(II) + Z*FP
      RETURN
      END
      SUBROUTINE SCALE(A,B,R,N)
C
C      SUBROUTINE TO SCALE THE ROOTS OF A POLYNOMIAL. GIVEN THE N-TH
C      DEGREE POLYNOMIAL WITH COEFFICIENTS IN A, THE SUBROUTINE RETURNS
C      IN B THE COEFFICIENTS OF THE POLYNOMIAL WHOSE ROOTS ARE DIVIDED
C      BY R.
C
      COMPLEX*16 A(1),B(1)
      REAL*8 R,RR,MAX,BIG/1.060/
      N1 = N+1
      MAX = 0.
      DO 5 I=1,N1
5     MAX = DMAX1(MAX,CDABS(A(I)))
      RR = BIG/MAX
      IF(R.GT.1.) GO TO 20
      DO 10 I=1,N1
      B(I) = RR*A(I)
10    RR = R*RR
      RETURN
20    DO 30 I=1,N1
      II = N1-I+1
      B(II) = RR*A(II)
30    RR = RR/R
      RETURN

```

```

PLSL0101
PLSL0102
PLSL0103
PLSL0104
PLSL0105
PLSL0106
PLSL0107
PLSL0108
PLSL0109
PLSL0110
PLSL0111
PLSL0112
PLSL0113
PLSL0114
PLSL0115
PLSL0116
PLSL0117
PLSL0118
PLSL0119
PLSL0120
PLSL0121
PLSL0122
PLSL0123
PLSL0124
PLSL0125
PLSL0126
PLSL0127
PLSL0128
PLSL0129
PLSL0130
PLSL0131
PLSL0132
PLSL0133
PLSL0134
PLSL0135
PLSL0136
PLSL0137
PLSL0138
PLSL0139
PLSL0140
PLSL0141
PLSL0142
PLSL0143
PLSL0144
PLSL0145
PLSL0146
PLSL0147
PLSL0148
PLSL0149
PLSL0150

```

```

END
SUBROUTINE SHIFT(A,B,Z,N)
C
C SUBROUTINE TO SHIFT THE ROOTS OF A POLYNOMIAL. GIVEN THE N-TH
C DEGREE POLYNOMIAL WITH COEFFICIENTS IN A, THE ROUTINE RETURNS IN B
C THE COEFFICIENTS OF THE POLYNOMIAL WHOSE ROOTS HAVE BEEN
C SHIFTED BY -Z.
C
COMPLEX*16 A(1),B(1),Z
N1 = N+1
DO 10 I=1,N1
10 B(I) = A(I)
DO 20 I=1,N
DO 20 J=I,N
JJ = N-J+1
20 B(JJ) = B(JJ) + Z*B(JJ+1)
RETURN
END
SUBROUTINE ANNULUS(A,R1,R2,N)
C
C SUBROUTINE TO FIND AN ANNULUS IN WHICH THE N-TH DEGREE POLYNOMIAL
C WITH COEFFICIENTS IN A HAS AT LEAST ONE ROOT. UPON ENTRY R1 MUST
C CONTAIN A STARTING RADIUS. IF R1 IS POSITIVE, THE PROGRAM ASSUMES
C THAT THERE IS A ROOT IN THE CIRCLE Z.LE.R1. ON RETURN R1 AND R2
C CONTAIN THE INNER AND OUTER RADII OF THE ANNULUS. IF THE
C POLYNOMIAL HAS A ROOT AT Z=0, THEN ON RETURN R1=R2=0.
C
COMPLEX*16 A(1),P(51)
REAL*8 R1,R2,FAC/2.7
IF(CDABS(A(1)).NE.0.) GO TO 10
R1 = 0.
R2 = 0.
RETURN
10 IF(R1.GT.0.) GO TO 100
R1 = -R1
20 R2 = R1
CALL SCALE(A,B,R1,N)
CALL ROOT(B,KKK,N)
GO TO (100,200),KKK
100 R2 = R1
R1 = R1/FAC
CALL SCALE(A,B,R1,N)
CALL ROOT(B,KKK,N)
GO TO (100,400),KKK
200 R1 = R2
R2 = FAC*R1
CALL SCALE(A,B,R2,N)
CALL ROOT(B,KKK,N)
GO TO (400,200),KKK
400 RETURN

```

PLSL0151
PLSL0152
PLSL0153
PLSL0154
PLSL0155
PLSL0156
PLSL0157
PLSL0158
PLSL0159
PLSL0160
PLSL0161
PLSL0162
PLSL0163
PLSL0164
PLSL0165
PLSL0166
PLSL0167
PLSL0168
PLSL0169
PLSL0170
PLSL0171
PLSL0172
PLSL0173
PLSL0174
PLSL0175
PLSL0176
PLSL0177
PLSL0178
PLSL0179
PLSL0180
PLSL0181
PLSL0182
PLSL0183
PLSL0184
PLSL0185
PLSL0186
PLSL0187
PLSL0188
PLSL0189
PLSL0190
PLSL0191
PLSL0192
PLSL0193
PLSL0194
PLSL0195
PLSL0196
PLSL0197
PLSL0198
PLSL0199
PLSL0200

END	PLSL0201
SUBROUTINE ROOT(AA, KKK, N)	PLSL0202
C	PLSL0203
C SUBROUTINE TO DETERMINE IF THE POLYNOMIAL OF DEGREE N WITH	PLSL0204
C COEFFICIENTS IN AA HAS A ROOT IN THE UNIT CIRCLE. THE SUBROUTINE	PLSL0205
C RETURNS WITH	PLSL0206
C KKK = 1, YES	PLSL0207
C KKK = 2, NO	PLSL0208
C	PLSL0209
COMPLEX*16 AA(1), A(51), MULT, TEMP	PLSL0210
REAL*8 R, AN, A0	PLSL0211
N1 = N+1	PLSL0212
KM = 1	PLSL0213
R = 0.	PLSL0214
DO 10 I=1, N1	PLSL0215
10 R = DMAX1(R, CDABS(AA(I)))	PLSL0216
M1 = 0	PLSL0217
DO 20 I=1, N1	PLSL0218
II = N1-I+1	PLSL0219
A(II) = AA(II)/R	PLSL0220
20 IF(CDABS(A(II)).NE.0. .AND. M1.EQ.0) M1 = II	PLSL0221
KKK = 2	PLSL0222
30 AN = CDABS(A(M1))	PLSL0223
A0 = CDABS(A(1))	PLSL0224
IF(AN.GE.A0) GO TO 50	PLSL0225
IF(M1.EQ.2) GO TO 60	PLSL0226
MULT = A(M1)/DCONJG(A(1))	PLSL0227
KM = KM+1	PLSL0228
A(1) = A(1) - MULT*DCONJG(A(M1))	PLSL0229
M1 = M1-1	PLSL0230
M2 = 1 + M1/2	PLSL0231
DO 40 I=2, M2	PLSL0232
II = M1-I+2	PLSL0233
TEMP = A(II) - MULT*DCONJG(A(I))	PLSL0234
A(I) = A(I) - MULT*DCONJG(A(II))	PLSL0235
40 A(II) = TEMP	PLSL0236
GO TO 30	PLSL0237
50 KKK = 1	PLSL0238
60 RETURN	PLSL0239
END	PLSL0240

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Solution of non-linear systems of equations.

Purpose

ZEROIN calculates the solution vector of the system of equations

$$F(t) = [f_1(t), f_2(t), \dots, f_n(t)]^T = 0$$

where

$t = (t_1, t_2, \dots, t_n)^T$ is the vector at which F is evaluated.

No gradients are required.

Usage

The calling sequence is:

CALL ZEROIN (N, XO, RO, EPS, ICV, EVAL, IRITE)

where

N is the integer number of variables ($n \leq 20$),

XO is the singly-dimensioned, double precision array containing the first estimate. Upon return, it contains the solution vector.

RO is a double-precision first estimate of $||X - XO||$ where X is the solution. In practice,

$RO = ||XO||/10$ is a good first estimate if the roots are roughly of the same magnitude. RO will contain $||X_{new} - X_{old}||^2 / ||X_{new}||^2$ upon return.

EPS is the double-precision convergence criterion, usually taken to be $5 \cdot 10^{-13}$

Convergence is assumed if

$$||X_{new} - X_{old}||^2 \leq EPS \cdot ||X_{new}||^2.$$

Upon return EPS will contain $|| f(X_0) ||^2$

ICV is an integer convergence flag such that if it

= 0 means convergence but

$$|| f(X_0) ||^2 > 5 \cdot 10^{-11}$$

= 1 means convergence

= 2 means no convergence in the allowed iterations

where the maximum number of iterations is equal
to $\text{MIN}(10 \cdot N, 50)$.

IRITE is an integer output flag such that if it

= 0 no output is given and results are returned
to the main program,

= 1 each iteration is written on data set 6.

EVAL is the name of the subroutine supplied by the user to
evaluate the system at a point. Its calling sequence is:

CALL EVAL(T,F)

where

T is a singly-dimensioned, double precision array
of estimates.

F is a singly-dimensioned, double precision array
of functional values of the system to be solved.

Coding Information

ZEROIN is a Fortran IV double precision subprogram and may be
easily converted to single precision. An EXTERNAL statement must
appear in the calling program with the name(s) of the subprogram(s)
which evaluate the system(s) of non-linear equations.

Every attempt should be made to analyze the problem before blindly using any system solver. If the equations can be scaled to give all roots roughly the same magnitude, the method is more inclined to work.

If a knowledge of the bounds on the partials is known, then the following provides an excellent starting point:

$$X = -(F(0) \cdot Q) / 2$$

where

Q is found by the following procedure:

$$\text{If } \left| \frac{\partial f_p}{\partial x_k} \right| \leq M_{pk} \quad (p, k = 1, \dots, n)$$

$$\text{Let } K = \left[\max_{1 \leq I \leq n} \sum_{p=1}^n M_{pI} \sum_{k=1}^n M_{pk} \right]^{\frac{1}{2}}$$

$$\text{If } \frac{\partial f_p}{\partial x_p} \geq p_p > q_p \quad (p = 1, \dots, n)$$

$$\text{where } q_p = \frac{1}{2} \sum_{\substack{k=1 \\ k \neq p}}^n (M_{pk} + M_{kp}) \text{ and } p_p > 0,$$

$$\text{Let } C = \min_{1 \leq p \leq n} (p_p - q_p).$$

$$\text{Then let } Q = (C + K) / (C \cdot K).$$

For an example of the above technique, see [2].

Method

The iterative solution vectors are generated by $x^{k+1} = x^k + \Delta^{k+1}$

where $\Delta^{k+1} = - [F(x^k, x^{k-1}) \cdot X^{-1}(x^k, x^{k-1})]^{-1} \cdot f(x^k)$

where

$$\begin{aligned} X(x^k, x^{k-1}) &= \|x^k - x^{k-1}\| \cdot H^k \text{ with } H^k \text{ any orthogonal} \\ &\text{matrix and } F(x^k, x^{k-1}) \cdot e_J = \\ &f(x^k + X(x^k, x^{k-1}) \cdot e_J) - f(x^k) \quad 1 \leq J \leq n. \end{aligned}$$

Clearly, $F \cdot X^{-1}$ is just a difference approximation to $J(x^k)$, the Jacobian at x^k being a generalization of the divided difference $f(x^k, x^{k-1}) = [f(x^k) - f(x^{k-1})]/[x^k - x^{k-1}]$ in the one-dimensional secant method.

The matrix H is initially taken to be I and modified when the process begins to converge.

References

- [1] Robinson, S. M. (1966): "Interpolative solution of systems of non-linear equations, SIAM Journal of Numerical Analysis," 3, p. 650.
- [2] Mancino, O. G. (1967): "Resolution by iteration of some non-linear systems," J. Assoc. Comp. Mach., 14, pp. 341-350.

Author

G. W. Westley, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE ZEROIN(N,X0,R0,EPS,ICV,EVAL,IRITE)
C PROGRAM AUTHOR G. W. WESTLEY
C COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C OAK RIDGE, TENN.
C
C CTC ORD PROGRAM NO. 9065
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X0(20),X(20),F(20,20),H(20,20),Y(20),D(20),F0(20),FX(20)
      NUMB=0
      ICV=0
      MAX=10*N
      MAX=AMIN0(MAX,50)
      CT=R0*R0
      N1= N-1
      CALL EVAL(X0,F0)
      NUMB=NUMB+1
      IF(IRITE.EQ.1) WRITE(6,100)
      IF(IRITE.EQ.1) WRITE(6,200) R0,(X0(I),I=1,N)
      IF(IRITE.EQ.1) WRITE(6,300) NUMB, (F0(I),I=1,N)
      DO 1 I=1,N
1    FX(I)=F0(I)
      GO TO 4
2    DO 3 I=1,N
      F0(I)=FX(I)
3    X0(I)=X(I)
C
C      USE I FOR THE ORTHOGONAL MATRIX.
C
4    DO 5 I=1,N
      C= X0(I)
      X0(I)= X0(I)+R0
      CALL EVAL(X0,X)
      NUMB=NUMB+1
      X0(I)= C
      DO 5 J=1,N
5    F(J,I)= X(J) - F0(J)
      IT=1
      GO TO 18
6    DO 7 I=1,N
7    D(I)= -R0*FX(I)
8    CON=0.
      RK= 0.000
      FSUM= 0.
      DO 9 I=1,N
      X(I)= D(I)+ X0(I)
      CON= CON + X(I)**2
      RK= RK + D(I)**2
9    Y(I)= RK
      ZER00001
      ZER00002
      ZER00003
      ZER00004
      ZER00005
      ZER00006
      ZER00007
      ZER00008
      ZER00009
      ZER00010
      ZER00011
      ZER00012
      ZER00013
      ZER00014
      ZER00015
      ZER00016
      ZER00017
      ZER00018
      ZER00019
      ZER00020
      ZER00021
      ZER00022
      ZER00023
      ZER00024
      ZER00025
      ZER00026
      ZER00027
      ZER00028
      ZER00029
      ZER00030
      ZER00031
      ZER00032
      ZER00033
      ZER00034
      ZER00035
      ZER00036
      ZER00037
      ZER00038
      ZER00039
      ZER00040
      ZER00041
      ZER00042
      ZER00043
      ZER00044
      ZER00045
      ZER00046
      ZER00047
      ZER00048
      ZER00049
      ZER00050

```

```

CALL EVAL(X,FX)
NUMB=NUMB+1
CON= RK/CON
ICV=ICV+1
IF(IRITE.EQ.1) WRITE(6,200) CON,( X(I),I=1,N)
IF(IRITE.EQ.1) WRITE(6,300) NUMB, (FX(I),I=1,N)
C
C
C CHECK FOR CONVERGENCE AND NEARNESS TO THE SOLUTION.
C
C
IF(ICV.GT.MAX) GO TO 27
IF(CON.LE. EPS) GO TO 28
IF(RK.GT.CT) GO TO 2
DO 10 I=1,N
F(I,N)= FO(I)-FX(I)
H(I,N)=-D(I)
FO(I)=FX(I)
XO(I) = X(I)
DO 10 J=1,N1
10 H(I,J)=0.0
C
C
C USE THE DIFFERENCE MATRIX AS THE ORTHOGONAL APPROXIMATION.
C
C
DO 11 J=1,N1
K= J+1
CON=DSQRT(RK/(Y(J)*Y(K)))
H(K,J) = -Y(J)*CON
DO 11 I=1,J
11 H(I,J) = CON*D(K)*D(I)
DO 14 J= 1,N1
DO 12 I=1,N
12 X(I)= XO(I) +H(I,J)
CALL EVAL(X,FX)
NUMB=NUMB+1
DO 13 I=1,N
13 F(I,J) = FX(I)-FO(I)
14 CONTINUE
DO 15 I=1,N
15 FX(I)= FO(I)
IT=2
GO TO 18
16 DO 17 I=1,N
D(I)=0.
DO 17 J=1,N
17 D(I)=D(I)- H(I,J)*FX(J)
GO TO 8
C

```

```

ZER00051
ZER00052
ZER00053
ZER00054
ZER00055
ZER00056
ZER00057
ZER00058
ZER00059
ZER00060
ZER00061
ZER00062
ZER00063
ZER00064
ZER00065
ZER00066
ZER00067
ZER00068
ZER00069
ZER00070
ZER00071
ZER00072
ZER00073
ZER00074
ZER00075
ZER00076
ZER00077
ZER00078
ZER00079
ZER00080
ZER00081
ZER00082
ZER00083
ZER00084
ZER00085
ZER00086
ZER00087
ZER00088
ZER00089
ZER00090
ZER00091
ZER00092
ZER00093
ZER00094
ZER00095
ZER00096
ZER00097
ZER00098
ZER00099
ZER00100

```

C		ZER00101
C	SOLVE THE SYSTEM $F \cdot A = F_X$ AND STORE THE RESULT IN F_X . THE SYSTEM	ZER00102
C	IS SOLVED BY GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING.	ZER00103
C		ZER00104
C		ZER00105
	18 DO 24 I=1,N1	ZER00106
	C=0.	ZER00107
	DO 19 J=I,N	ZER00108
	CON=DABS(F(J,I))	ZER00109
	IF(CON.LT.C) GO TO 19	ZER00110
	C=CON	ZER00111
	INDX=J	ZER00112
	19 CONTINUE	ZER00113
	IF(INDX.EQ.I) GO TO 21	ZER00114
	DO 20 J=I,N	ZER00115
	C=F(I,J)	ZER00116
	F(I,J)=F(INDX,J)	ZER00117
	20 F(INDX,J)=C	ZER00118
	C=FX(I)	ZER00119
	FX(I)=FX(INDX)	ZER00120
	FX(INDX)=C	ZER00121
	21 IF(F(I,I).EQ.0.000) GO TO 24	ZER00122
	NXT=I+1	ZER00123
	DO 23 J=NXT,N	ZER00124
	IF(F(J,I).EQ.0.000) GO TO 23	ZER00125
	C=F(J,I)/F(I,I)	ZER00126
	F(J,I)=0.	ZER00127
	DO 22 IP=NXT,N	ZER00128
	22 F(J,IP)=F(J,IP) - F(I,IP)*C	ZER00129
	FX(J)=FX(J) - FX(I)*C	ZER00130
	23 CONTINUE	ZER00131
	24 CONTINUE	ZER00132
	FX(N)=FX(N)/F(N,N)	ZER00133
	DO 26 I=1,N1	ZER00134
	NXT=N-I	ZER00135
	DO 25 J=NXT,N1	ZER00136
	25 FX(NXT)=FX(NXT) - F(NXT,J+1)*FX(J+1)	ZER00137
	26 FX(NXT)=FX(NXT)/F(NXT,NXT)	ZER00138
C		ZER00139
C		ZER00140
C	END GAUSSIAN PORTION AND RETURN TO MAIN PROGRAM.	ZER00141
C		ZER00142
C		ZER00143
	GO TO (6,16),IT	ZER00144
	27 ICV=2	ZER00145
	GO TO 29	ZER00146
	28 ICV=1	ZER00147
	29 DO 30 I=1,N	ZER00148
	30 FSUM= FSUM + FX(I)**2	ZER00149
	GO TO (31,32),ICV	ZER00150

31	IF(FSUM.GT..5D-10)	ICV=0		ZER00151
32	EPS=FSUM			ZER00152
	RC= CON			ZER00153
	DO 33 I=1,N			ZER00154
33	XQ(I)=X(I)			ZER00155
	IF(IRITE.EQ.1)	WRITE(6,400)	ICV	ZER00156
	RETURN			ZER00157
100	FORMAT(1H1)			ZER00158
200	FORMAT(3H X 1PD15.7,2X,7D15.7/(3H	17X,7D15.7))		ZER00159
300	FORMAT(3H * I6,11X,	7D15.7/(3H	17X,7D15.7))	ZER00160
400	FORMAT(4H *** I6)			ZER00161
	END			ZER00162

Identification

Solution of simultaneous non-linear systems of equations by the secant method.

Purpose

NØNLIN is designed to calculate a root of the system of equations $y(x) \equiv (\eta_1(x), \eta_2(x), \dots, \eta_N(x))^T = 0$ where $x^T = (\xi_1, \xi_2, \dots, \xi_N)$. The i^{th} equation of this system is $\eta_i(x) = 0$ and ξ_i is the i^{th} variable in this equation. NØNLIN will usually be successful even if some of the η_i are discontinuous whereas gradient type routines may fail. Also the user is spared of the sometimes tedious task of programming the partial derivatives.

Usage

The calling sequence is:

```
CALL NØNLIN(X,N,ANS,ICONV,IR,EVAL)
```

where

- N is the integer number of equations in the system.
- X is the doubly-dimensioned double precision input matrix of size $N \times (N+1)$ with each of its first $N+1$ columns consisting of a trial solution vector, i.e. $N+1$ initial estimates of the solution are required. Also see Restrictions.
- ANS is a singly-dimensioned double precision array which will contain the solution vector.
- ICONV is an integer output variable indicating the number of iterations required for convergence or if NØNLIN does not converge, ICONV will equal 20N.

IR is the integer first dimension of X. In the calling program, the second dimension of X should be at least XR+1.

EVAL is the name by which NØNLIN calls the subroutine defining the system of equations to be solved.

NØNLIN requires that the user write a subroutine called EVAL which defines $y(x)$, i.e., $\eta_1(x), \eta_2(x), \dots, \eta_N(x)$. The calling sequence is:

```
CALL EVAL(XI,Y)
```

where

XI is x, the singly-dimensioned double precision array, of solution estimates, and

Y is y, the singly-dimensioned double precision array of function values.

Coding Information

NØNLIN is a double precision Fortran IV subprogram which may be easily converted to single precision. An EXTERNAL statement must be included in the calling program with the name(s) of the subroutine(s) called by NØNLIN defining the system(s) of equations to be solved. NØNLIN is programmed to solve not more than ten equations. To solve larger systems than of order 10, all dimensioned arrays (except X) in NØNLIN and MATINV (called by NØNLIN) must be redimensioned (entries of 10(11) would be changed to N(N+1)). In the calling program, ANS must be dimensioned not less than N. The convergence criterion should occasionally be changed from 10^{-28} to a more

lenient one. On a 7090 single precision version this convergence criterion was set at 10^{-14} . It has been the authors' experience that it is often enlightening to print out the arrays XI and Y inside EVAL to show how the iteration is proceeding.

Method

NØNLIN uses the secant method [1] for solving systems of equations.

Let e_i be the i th column of the identity matrix. The input matrix

$X = (x_1, x_2, \dots, x_{N+1})$ is used to generate the matrix A such that

$Ae_i = (\eta_1(x_i), \eta_2(x_i), \dots, \eta_N(x_i), 1)^T$ $i = 1, 2, \dots, N+1$. The

inverse of A is calculated and $w = A^{-1}e_{N+1}$. A new trial vector,

$\bar{x} = Xw$, is calculated and the j th column of X is replaced by $\bar{x}$.

The appropriate j th column is chosen by letting $\max_k \sum_{i=1}^N |\eta_i(x_k)| = \sum_{i=1}^N |\eta_i(x_j)|$.

EVAL is used to calculate $p = (\eta_1(\bar{x}), \dots, \eta_N(\bar{x}), 1)^T$ which replaces

(theoretically) the j th column of A to form $\hat{A}$. In order to obtain

a new vector w, $\hat{A}^{-1}$ must be calculated. This can be accomplished

by the following algorithm (pp 123-124 of [2] can be used in the

derivation; [1] has several misprints in this regard). First calculate

$b = A^{-1}p$. Then if $b = (\beta_1, \beta_2, \dots, \beta_{N+1})^T$,

$$e_j^T \hat{A}^{-1} e_k = e_j^T A^{-1} e_k / \beta_j \quad \text{for all } k$$

and

$$e_i^T \hat{A}^{-1} e_k = e_i^T A^{-1} e_k - e_j^T A^{-1} e_k (\beta_i / \beta_j) \quad \text{for } i \neq j$$

all k

Successive $\bar{x}$'s are compared until

$(||\bar{x}_{\text{new}} - \bar{x}_{\text{old}}|| / ||\bar{x}_{\text{new}}||)^2 < 10^{-28}$ at which time convergence is assumed (the double bars indicate Euclidean length). The initial inverse of A is calculated by MATINV which uses Gaussian elimination with partial pivoting (p. 127 of [2]).

Restrictions

It is necessary that the initial trial points do not essentially lie in a proper subspace S of Euclidean N-space. This can be proved by noting that the sum of the components of w equals one and hence $\bar{x} = Xw$ is in S. The user would be well advised to pick his trial points as the distinct vertices of an equilateral polyhedron within the domain of y. The most common misuse of NØNLIN consists in not following this restriction, e.g. if $N = 2$, users frequently choose the three starting estimates in a straight line. Also the user should check y(ANS) to see if he has converged to a zero.

The choice of NØNLIN's convergence criterion may give trouble if $y(0) = 0$.

Accuracy

Some problems are inherently ill-conditioned and higher precision would be necessary to calculate the $y(x)$ accurately. For instance, on the 7090 a single precision version of NØNLIN was used to solve a system of linear equations $Hx = b$ with H being a Hilbert section of order 6. NØNLIN was as accurate as a single precision Gaussian

elimination program which, of course, was still poor.

Example

The following example shows how to use NONLIN to solve the system:

$$\eta_1(\xi_1, \xi_2) = \xi_1^2 + \xi_1 - \xi_2^2 + 1$$

$$\eta_2(\xi_1, \xi_2) = \xi_2(1 + 2\xi_1)$$

Hence for $N=2$ we must find ξ_1 and ξ_2 such that $\eta_1 = \eta_2 = 0$.

IBM 360/50 RESULTS, SEPTEMBER 1967

M	ξ_1			ξ_2			η_1	η_2
1	-.60000	00000	00000	1.1000	00000	00000	-.4500 0000	-.2200 0000
2	-.30000	00000	00000	1.1000	00000	00000	-.4200 0000	+.4400 0000
3	-.60000	00000	00000	1.4000	00000	00000	-1.200 0000	-.2800 0000
4	-.51605	83941	60584	.92335	76642	33577	-.1023 3150	-.2965 5283
5	-.50334	68489	70612	.87074	08455	13125	-.0081 7842	-.0058 2848
6	-.50047	31416	47820	.86616	43464	40452	-.0002 4045	-.0008 1963
7	-.50005	25239	00848	.86601	79862	15216	+. (4) 1 2850	-. (4) 9 0973
8	-.50000	42004	06078	.86602	47355	36833	+. (5) 1 1575	-. (5) 7 2753
9	-.50000	00212	26510	.86602	54003	13276	+. (8) 6 0122	-. (7) 3 6765
10	-.50000	00000	25290	.86602	54037	80403	+. (11) 6990	-. (10) 4380
11	-.49999	99999	99974	.86602	54037	84444	-. (14) 9326	+. (13) 4577
12	-.50000	00000	00000	.86602	54037	84439	-. (14) 1110	0.0

Of course a different computer or even a slightly different compiler would change the iterates considerably.

A calling program and subroutine EVAL to solve this problem
would be:

```
EXTERNAL EVAL
```

```
DOUBLE PRECISION X(2,3),ANS(2)
```

```
NAMelist/OUT/ ANS,ICONV
```

```
N=2
```

```
X(1,1)=-.6D0
```

```
X(2,1)=1.1D0
```

```
X(1,2)=-.3D0
```

```
X(2,2)=1.1D0
```

```
X(1,3)=-.6D0
```

```
X(2,3)=1.4D0
```

```
CALL NONLIN(X,N,ANS,ICONV,2,EVAL)
```

```
WRITE(6,OUT)
```

```
STOP
```

```
END
```

```
SUBROUTINE EVAL (XI,Y)
```

```
DOUBLE PRECISION XI(2),Y(2),YS(2),XS(2)
```

```
NAMelist/NAM1/YS,XS
```

```
Y(1)=(XI(1)**2)+XI(1)-(XI(2)**2)+1.
```

```
Y(2)=XI(2)*(1.+(2.*XI(1)))
```

```
YS(1) = Y(1)
```

```
YS(2) = Y(2)
```

```
XS(1) = XI(1)
```

```
XS(2) = XI(2)
```

```
WRITE(6,NAM1)
```

RETURN

END

For the coding shown the successive iterates would be outputted from EVAL (the user may not wish to do this, see Coding Information) and the final answer would be outputted in the calling program (superfluous in this example). This example is also mentioned in [1].

References

- [1] Wolfe, P. (1959): "The secant method for simultaneous non-linear equations, Comm. ACM, 2, 12, pp. 12-13.
- [2] Householder, A. S. (1964): The Theory of Matrices in Numerical Analysis, Ginn and Co.

Authors

R. E. Funderlic, with Computing Technology Center, and J. Rinzel, a summer employee, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE NONLIN(X,N,ANS,ICONV,IR,EVAL)
C  PROGRAM AUTHORS  R. E. FUNDERLIC AND J. RINZEL,
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9066
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSIONX(IR,1),ANS(IR), XBAR(10,2),A(11,11),AINV(11,11),P(11),
      1B(11),XNORM(11),BNORM(10)
      ICONV=0
      NP1=N+1
      DO 1 J=1,NP1
        B(J)=1.
        A(NP1,J)=1.
1     CALL EVAL(X(1,J),A(1,J))
      DO 2 I=1,NP1
      DO 2 J=1,NP1
2     AINV(I,J)=(1/J)*(J/I)
      DO 4 J=1,NP1
        SUM =0.
      DO 3 I=1,N
3     SUM=SUM+DABS(A(I,J))
4     XNORM(J)=SUM
      CALL MATINV(A,AINV,NP1)
      DO 5 I=1,N
5     XBAR(I,1)=X(I,NP1)
      ITER=20*N
      DO 15 K=1,ITER
        XBNUM=0.
        XBDEN=0.
      DO 7 I=1,N
        XB=0.
      DO 6 J=1,NP1
6     XB=XB + AINV(J,NP1)*X(I,J)
        XBAR(I,2)=XB
        XBNUM=XBNUM + (XBAR(I,2)-XBAR(I,1))**2
7     XBDEN =XBDEN + XBAR(I,2)**2
      ICONV=ICONV+1
      IF((XBNUM/XBDEN).LT.1.D-28) GO TO 16
      DO 8 I=1,N
8     XBAR(I,1)=XBAR(I,2)
      BIGNOR =0.
      DO 9 J=1,NP1
        BNORM(J)=XNORM(J)*DABS(B(J))
        IF(BNORM(J).LE.BIGNOR)GO TO 9
        BIGNOR=BNORM(J)
      JCGL = J
9     CONTINUE
      CALL EVAL(XBAR,P)
      P(NP1)=1.

```

```

NONL0001
NONL0002
NONL0003
NONL0004
NONL0005
NONL0006
NONL0007
NONL0008
NONL0009
NONL0010
NONL0011
NONL0012
NONL0013
NONL0014
NONL0015
NONL0016
NONL0017
NONL0018
NONL0019
NONL0020
NONL0021
NONL0022
NONL0023
NONL0024
NONL0025
NONL0026
NONL0027
NONL0028
NONL0029
NONL0030
NONL0031
NONL0032
NONL0033
NONL0034
NONL0035
NONL0036
NONL0037
NONL0038
NONL0039
NONL0040
NONL0041
NONL0042
NONL0043
NONL0044
NONL0045
NONL0046
NONL0047
NONL0048
NONL0049
NONL0050

```

XNORM(JCOL)=0.	NONL0051
DO 10 I=1,N	NONL0052
X(I,JCOL)=XBAR(I,1)	NONL0053
10 XNORM(JCOL)=XNORM(JCOL)+DABS(P(I))	NONL0054
DO 11 I=1,NP1	NONL0055
B(I)=0.	NONL0056
DO 11 J=1,NP1	NONL0057
11 B(I)=B(I)+AINV(I,J)*P(J)	NONL0058
DO 13 I=1,NP1	NONL0059
IF(I.EQ.JCOL) GO TO 13	NONL0060
DO 12 J=1,NP1	NONL0061
12 AINV(I,J)=AINV(I,J)-AINV(JCOL,J)*(B(I)/B(JCOL))	NONL0062
13 CONTINUE	NONL0063
DO 14 I=1,NP1	NONL0064
14 AINV(JCOL,I)=AINV(JCOL,I)/B(JCOL)	NONL0065
15 CONTINUE	NONL0066
DO 17 I=1,N	NONL0067
17 ANS(I)=XBAR(I,2)	NONL0068
RETURN	NONL0069
END	NONL0070
SUBROUTINE MATINV(A,B,III)	NONL0071
IMPLICIT REAL*8 (A-H,O-Z)	NONL0072
DIMENSION A(11,11), B(11,11)	NONL0073
KK=III	NONL0074
NV=III	NONL0075
KKM=KK-1	NONL0076
DO9I=1,KKM	NONL0077
S=0.0	NONL0078
DO1J=1,KK	NONL0079
R=DABS(A(J,I))	NONL0080
IF(R.LT.S) GO TO 1	NONL0081
S=R	NONL0082
L=J	NONL0083
1 CONTINUE	NONL0084
IF (L.EQ.I) GO TO 5	NONL0085
IF(L-I)2,5,2	NONL0086
2 DO3J=1,KK	NONL0087
S=A(I,J)	NONL0088
A(I,J)=A(L,J)	NONL0089
3 A(L,J)=S	NONL0090
DO4J=1,NV	NONL0091
S=B(I,J)	NONL0092
B(I,J)=B(L,J)	NONL0093
4 B(L,J)=S	NONL0094
5 IF(A(I,I).EQ.0.) GO TO 9	NONL0095
IPO=I+1	NONL0096
DO8J=IPO,KK	NONL0097
IF(A(J,I).EQ.0.) GO TO 8	NONL0098
S=A(J,I)/A(I,I)	NONL0099
A(J,I)=0.0	NONL0100

```
      DO6K=IPO, KK
6  A(J,K)=A(J,K)-A(I,K)*S
      DO7K=1, NV
7  B(J,K)=B(J,K)-B(I,K)*S
8  CONTINUE
9  CONTINUE
      KMG=KK-1
      DO11K=1, NV
      B(KK,K)=B(KK,K)/A(KK, KK)
      DO11I=1, KMO
      N=KK-I
      DO10J=N, KMO
10 B(N,K)=B(N,K)-A(N, J+1)*B(J+1, K)
11 B(N,K)=B(N,K)/A(N, N)
      RETURN
      END
```

```
NONL0101
NONL0102
NONL0103
NONL0104
NONL0105
NONL0106
NONL0107
NONL0108
NONL0109
NONL0110
NONL0111
NONL0112
NONL0113
NONL0114
NONL0115
NONL0116
```

F
SOLUTION OF
MATRIX EQUATIONS AND
DETERMINANTS

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Matrix equation solver.

Purpose

DMATEQ and CMATEQ solve the matrix equation $AX = B$ and/or compute the determinant of the square real or complex matrix A. This of course provides the user with the capability of inverting matrices and solving simultaneous linear algebraic equations.

Usage

The calling sequences are:

CALL DMATEQ (A,B,N,NV,ID) or DET = DMATEQ (A,B,N,NV,ID)

CALL CMATEQ (A,B,N,NV,ID) or DET = CMATEQ (A,B,N,NV,ID)

where

A & B are the input doubly-dimensioned arrays. B will contain the answer X on returning from CMATEQ or DMATEQ. When DMATEQ is used, A and B are double precision; and when CMATEQ is used, A and B are double precision complex (COMPLEX*16). A is destroyed.

N is the integer order of the square matrix A.

|NV| is the integer number of columns in B. If NV = 0, B need not be defined and only the determinant of A will be calculated. If NV < 0, det(A) will not be calculated and will be returned as zero.

ID is the integer first dimension of A and B in the calling program. If $|NV| = 1$, then B may be dimensioned B(ID).

Coding Information

DMATEQ and CMATEQ are written in Fortran IV and use double precision and double precision complex arithmetic, respectively. For DMATEQ, A, B, DMATEQ and DET must appear in a DOUBLE PRECISION or a REAL*8 type statement in the calling program. For CMATEQ, A, B, CMATEQ, and DET must appear in a COMPLEX*16 type statement in the calling program. The matrix A is overwritten by a triangular matrix. It was found that, although $AX = B$ was a well-conditioned problem, overflows and underflows sometimes occurred in the calculation of $\det(A)$ even though mathematically $\det(A)$ would have a standard representation in the machine. A sorting strategy can sometimes eliminate this difficulty. Other possible strategies can be found in [3] and pg. 55 of [1].

To calculate the inverse of A, B must be initially set equal to the identity matrix.

Restrictions

The matrix A must be at least nonsingular and N must be greater than 1.

Timing

DMATEQ was timed on an IBM 360/75 in July 1967 by Doyle Turner of CTC. A half second was required to solve a random system of 50 linear equations on an IBM compiler using OPT=2. When B was a square matrix of order 50, 2.26 seconds were required. Times vary not only with the sizes of A and B but with the individual matrices, model of IBM 360, the compiler, fixed vs. variable dimensions, optimization level, time-sharing foreground, moods of operators, etc.

Accuracy

For full matrices A, the accuracy of X depends strongly on $\kappa = ||A|| \cdot ||A^{-1}||$ and to a much lesser extent on N, [1], [5], [6]. For $\kappa > 10^{16}$ when the spectral radius is used as the norm, the user can expect no significant digits. For example, a single precision version of DMATEQ yielded no significant digits in inverting a scaled Hilbert section of order 8 on a 7090.

There is no scaling provision in these routines and equilibration may be desirable for matrices which are out of balance [1], [5].

In fact, it is suggested that as a matter of course the user scale the rows of A and B by using the routine DSCALE which appears in section Z of this compendium.

Method

The method used is Gaussian elimination with partial pivoting [1], [2], [6], [7]. The matrix A is reduced to an upper triangular

matrix by equivalence transformations [6]. This consists of matrix multiplication by elementary triangular and permutation matrices, pg. 4 of [2]. The elementary permutation matrices essentially interchange rows of A and in general aid numerical stability [1]. The process of "back substitution" is then applied to obtain the solution. This amounts to solving the N_{th} equation of the equivalent triangular system, then the N-1 equation, etc.

Gaussian elimination is probably the best practical method to solve systems of equations on a machine like the IBM 360 and has successfully solved systems that filled the 7090 with non-zero numbers. Much larger systems have been solved using auxiliary storage. If the user wishes a single precision routine, he should investigate the use of the Crout algorithm followed by iterative refinement [1]. From the standpoint of storage and computer time, there is probably no advantage to this alternative on a model 75 IBM 360.

References

- [1] Forsythe, George S., and Moler, Cleve B. (1967): Computer Solution of Linear Algebraic Equation, Prentice-Hall.
- [2] Householder, A. S. (1964): The Theory of Matrices in Numerical Analysis, Ginn and Co.
- [3] Klyuyev, V. V., and Kokovkin, N. I. - Schevbak (1965): On the minimization of the number of arithmetic operations for the solution of linear algebraic systems of equations

(translated by G. J. Tee) Computer Science Department,
Technical Report CS 24. Stanford, Calif.: Stanford
University.

- [4] Martin, R. S., and Wilkinson, J. H. (1967): "Solution of symmetric and nonsymmetric band equations and the calculation of eigenvectors of band matrices" Numer. Math., vol. 9, 279-301.
- [5] Osborne, E. E. (1960): "On pre-conditioning of matrices," J. ACM, 7, 358.
- [6] Wilkinson, J. H. (1965): The Algebraic Eigenvalue Problem, Clarendon Press.
- [7] Wilkinson, J. H. (1963): Rounding Errors in Algebraic Processes, Prentice-Hall.

Authors

R. G. Edwards, formerly with Computing Technology Center, and
R. E. Funderlic, Computing Technology Center, Union Carbide
Corporation, Nuclear Division, Oak Ridge, Tennessee.

DOUBLE PRECISION FUNCTION DMATEQ(A,B,III,JJJ,ID)	DMTQ0001
C PROGRAM AUTHORS R. E. FUNDERLIC AND R. G. EDWARDS,	DMTQ0002
C COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	DMTQ0003
C OAK RIDGE, TENN.	DMTQ0004
C	DMTQ0005
C CTC ORD PROGRAM NO. 9067.1	DMTQ0006
DIMENSION A(ID,1),B(ID,1)	DMTQ0007
DOUBLE PRECISION A,B,R,S,D	DMTQ0008
KK=III	DMTQ0009
NV=IARS(JJJ)	DMTQ0010
D=1.	DMTQ0011
IF(JJJ.LT.0)D=0.	DMTQ0012
KKM=KK-1	DMTQ0013
DO9I=1,KKM	DMTQ0014
S=0.0	DMTQ0015
DO1J=1,KK	DMTQ0016
R=DABS(A(J,I))	DMTQ0017
IF(R.LT.S) GO TO 1	DMTQ0018
S=R	DMTQ0019
L=J	DMTQ0020
1 CONTINUE	DMTQ0021
IF(L.EQ.1) GO TO 5	DMTQ0022
DO 2 J=I,KK	DMTQ0023
S=A(I,J)	DMTQ0024
A(I,J)=A(L,J)	DMTQ0025
2 A(L,J)=S	DMTQ0026
IF(NV.LE.0) GO TO 4	DMTQ0027
DO 3 J=1,NV	DMTQ0028
S=R(I,J)	DMTQ0029
B(I,J)=B(L,J)	DMTQ0030
3 B(L,J)=S	DMTQ0031
4 D=-D	DMTQ0032
5 IF(A(I,I).EQ. 0.) GO TO 9	DMTQ0033
IPO=I+1	DMTQ0034
DO8J=IPO,KK	DMTQ0035
IF(A(J,I).EQ. 0.) GO TO 8	DMTQ0036
S=A(J,I)/A(I,I)	DMTQ0037
A(J,I)=0.0	DMTQ0038
DO 6 K=IPO,KK	DMTQ0039
6 A(J,K)=A(J,K)-A(I,K)*S	DMTQ0040
IF(NV.LE. 0) GO TO 8	DMTQ0041
DO 7 K=1,NV	DMTQ0042
7 B(J,K)=B(J,K)-B(I,K)*S	DMTQ0043
8 CONTINUE	DMTQ0044
9 CONTINUE	DMTQ0045
DO10I=1,KK	DMTQ0046
10 D=D*A(I,I)	DMTQ0047
IF(NV.LE.0) GO TO 13	DMTQ0048
KMD = KK-1	DMTQ0049
DO12K=1,NV	

```

      B(KK,K)=B(KK,K)/A(KK,KK)
      DO12I=1,KMO
      N=KK-I
      DO11J=N,KMO
11  B(N,K)=B(N,K)-A(N,J+1)*B(J+1,K)
12  B(N,K)=B(N,K)/A(N,N)
13  DMATEQ=D
      RETURN
      END
      COMPLEX FUNCTION CMATEQ(A,R,III,JJJ,IO)
C  PROGRAM AUTHORS  R. E. FUNDERLIC AND R. G. EDWARDS,
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9067.2
      DIMENSION A(IO,1),B(IO,1)
      COMPLEX*16 A,B,S,D,CMATEQ
      REAL*8 R
      KK=III
      NV=IABS(JJJ)
      D=(1.,0.)
      IF(JJJ.LT.0)D=(0.,0.)
      KKM=KK-1
      DO9I=1,KKM
      S=(0.,0.)
      DO1J=1,KK
      R=CDABS(A(J,I))
      IF(R.LT.CDABS(S)) GO TO 1
      S=R
      L=J
1  CONTINUE
      IF(L.EQ.1) GO TO 5
      DO 2 J=I,KK
      S=A(I,J)
      A(I,J)=A(L,J)
2  A(L,J)=S
      IF(NV.LE.0) GO TO 4
      DO3J=1,NV
      S=B(I,J)
      B(I,J)=B(L,J)
3  B(L,J)=S
4  D=-D
5  IF(CDABS(A(I,I)).EQ. 0.) GO TO 9
      IPO=I+1
      DO8J=IPO,KK
      IF(CDABS(A(J,I)).EQ.0.) GO TO 8
      S=A(J,I)/A(I,I)
      A(J,I)=(0.,0.)
      DO 6 K=IPO,KK
6  A(J,K)=A(J,K)-A(I,K)*S
      DMTQ0050
      DMTQ0051
      DMTQ0052
      DMTQ0053
      DMTQ0054
      DMTQ0055
      DMTQ0056
      DMTQ0057
      DMTQ0058
      CMTQ0001
      CMTQ0002
      CMTQ0003
      CMTQ0004
      CMTQ0005
      CMTQ0006
      CMTQ0007
      CMTQ0008
      CMTQ0009
      CMTQ0010
      CMTQ0011
      CMTQ11.1
      CMTQ0012
      CMTQ0013
      CMTQ0014
      CMTQ0015
      CMTQ0016
      CMTQ0017
      CMTQ0018
      CMTQ0019
      CMTQ0020
      CMTQ0021
      CMTQ0022
      CMTQ0023
      CMTQ0024
      CMTQ0025
      CMTQ0026
      CMTQ0027
      CMTQ0028
      CMTQ0029
      CMTQ0030
      CMTQ0031
      CMTQ0032
      CMTQ0033
      CMTQ0034
      CMTQ0035
      CMTQ0036
      CMTQ0037
      CMTQ0038
      CMTQ0039
      CMTQ0040

```



```
IF(NV.LE.0) GO TO 8
DO 7 K=1,NV
7 B(J,K)=B(J,K)-B(I,K)*S
8 CONTINUE
9 CONTINUE
DO10 I=1, KK
10 D=D*A(I,I)
IF(NV.LE.0) GO TO 13
KMO=KK-1
DO12 K=1,NV
B(KK,K)=B(KK,K)/A(KK,KK)
DO12 I=1,KMO
N=KK-I
DO11 J=N,KMO
11 B(N,K)=B(N,K)-A(N,J+1)*B(J+1,K)
12 B(N,K)=B(N,K)/A(N,N)
13 CMATEQ=D
RETURN
END
```

```
CMTQ0041
CMTQ0042
CMTQ0043
CMTQ0044
CMTQ0045
CMTQ0046
CMTQ0047
CMTQ0048
CMTQ0049
CMTQ0050
CMTQ0051
CMTQ0052
CMTQ0053
CMTQ0054
CMTQ0055
CMTQ0056
CMTQ0057
CMTQ0058
CMTQ0059
```

Identification

Real symmetric matrix inverter.

Purpose

SYMIN obtains the inverse of a real symmetric matrix A. Numerical stability is assured only when using positive definite matrices. For solving systems of linear equations for which A is positive definite [4] should be consulted.

Usage

The calling sequences are:

```
CALL SYMIN1 (A,N,IFAIL)
```

```
CALL SYMIN2 (A,N,IFAIL,ID)
```

where

A is a double precision array (singly-dimensioned or doubly-dimensioned for SYMIN1 or SYMIN2 respectively) containing the input matrix to be inverted (see Coding Information). Upon return from SYMIN it contains the inverse.

N an integer, is the order of the matrix.

IFAIL is a logical variable which upon return has the value .TRUE. if at some stage in the reduction no non-zero pivot could be found and .FALSE. otherwise.

ID is the integer first dimension of A in the calling program.

Coding Information

SYMIN1 and SYMIN2 are two separate double-precision subprograms, and each may be easily converted to single precision.

SYMIN1 has A as a singly-dimensioned array with elements in the order $A(1,1), A(1,2), \dots, A(1,N), A(2,2), A(2,3), \dots, A(2,N), A(3,3), \dots, A(3,N), \dots, A(N,N)$. This program optimizes the use of storage allocation. The dimension of A in the subroutine is irrelevant, but the size of A is limited to a 200 x 200 matrix unless obvious program changes are made. IFAIL should appear in a LOGICAL type statement in the calling program.

SYMIN2 has A as a doubly-dimensioned array with variable dimensions in the subroutine. The lower triangle of A is not referenced by SYMIN2.

Restrictions

All pivots are chosen on the diagonal, and if all further diagonal elements which are eligible as pivots vanish (this is impossible for a positive definite matrix A), then exit with IFAIL set to .TRUE. occurs.

Timing

SYMIN1 and SYMIN2 have been timed on the IBM 7090 using single precision. The matrices used for timing purposes were positive definite matrices generated from the subroutine MATGT, a subprogram found elsewhere in this compendium. SYMIN2 was used with fixed dimensions with the following results.

<u>ORDER</u>	<u>TIME</u>
10	.0547 sec.
20	.334 sec.
40	2.568 sec.
60	7.3 sec.
80	17.0 sec.

Accuracy

Accuracy [2] is strongly dependent on $||A|| \cdot ||A^{-1}||$.

Method

The Gaussian type method of Rutishauser [3] is similar to that of Busing and Levy [1], but requires no interchanges of rows and columns. The procedure requires the upper triangular part of A to be given and overwrites it by the upper triangular part of the inverse which is again denoted by A.

References

- [1] Busing and Levy (1962): "A procedure for inverting large symmetric matrices." Comm. ACM 5, 8, p. 446.
- [2] Forsythe, George E., and Moler, Cleve B. (1967): Computer Solution of Linear Algebraic Equations, Prentice-Hall.
- [3] Rutishauser (1963): Algorithm 150. Comm. ACM 6, 2, 66.
- [4] Martin, R. S., Peters, G., and Wilkinson, J. H.: "Iterative refinement of the solution of a positive definite system of equations," Numer. Math. (to be published).

Author

This program was translated from the ALGOL program of H.

Rutishauser [3] by S. Levin, formerly a co-op with the Computing
Technology Center, Union Carbide Corporation, Nuclear Division,
Oak Ridge, Tennessee.

```

      SUBROUTINE SYMIN1(A,NN,IFAIL)
C   PROGRAM AUTHOR H.RUTISHAUSER,EIDG. TECHNISCHE HOCHSCHULE,ZURICH,
C   SWITZERLAND. TRANSLATED FROM ALGOL TO FORTRAN IV BY S.LEVIN, CDCP.
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC ORD PROGRAM NO. 9068.1
      DOUBLE PRECISION A(1),P(200),Q(200),R(200),BIGA
      LOGICAL R,IFAIL
      IFAIL=.FALSE.
      N = NN
      NP1=N+1
      DO1I=1,N
1    R(I)=.FALSE.
      DO8I=1,N
      BIGA=0.
      JJ=1
      DO2J=1,N
      IF (R(J).OR.DABS(A(JJ)).LT.BIGA) GO TO 2
      BIGA=DABS(A(JJ))
      K=J
      KK=JJ
2    JJ=JJ+NP1-J
      IF (BIGA.EQ.0.) GO TO 9
      R(K)=.TRUE.
      Q(K)=1./A(KK)
      P(K)=1.
      A(KK)=0.
      IF(K.EQ.1)GOTO4
      KM1=K-1
      JK=K
      DO3J=1,KM1
      P(J)=A(JK)
      IF(.NOT.R(J)) A(JK)=-A(JK)
      Q(J)=A(JK)*Q(K)
      A(JK)=0.
3    JK=JK+N-J
      IF(K.EQ.N)GOTO6
4    KP1=K+1
      KJ=KK
      DO5J=KP1,N
      KJ=KJ+1
      Q(J)=-A(KJ)*Q(K)
      IF(.NOT.R(J)) A(KJ)=-A(KJ)
      P(J)=-A(KJ)
5    A(KJ)=0.
6    JJ=0
      DO8J=1,N
      JK=JJ
      DO7K=J,N

```

```

SYM10001
SYM10002
SYM10003
SYM10004
SYM10005
SYM10006
SYM10007
SYM10008
SYM10009
SYM10010
SYM10011
SYM10012
SYM10013
SYM10014
SYM10015
SYM10016
SYM10017
SYM10018
SYM10019
SYM10020
SYM10021
SYM10022
SYM10023
SYM10024
SYM10025
SYM10026
SYM10027
SYM10028
SYM10029
SYM10030
SYM10031
SYM10032
SYM10033
SYM10034
SYM10035
SYM10036
SYM10037
SYM10038
SYM10039
SYM10040
SYM10041
SYM10042
SYM10043
SYM10044
SYM10045
SYM10046
SYM10047
SYM10048
SYM10049
SYM10050

```

```

      JK=JK+1
      7 A(JK)=A(JK)+P(J)*Q(K)
      8 JJ=JJ+NP1-J
      GO TO 10
      9 IFAIL=.TRUE.
10  RETURN
      END
      SUBROUTINE SYMIN2(A,NN,IFAIL,ID)
C  PROGRAM AUTHOR H.RUTISHAUSER,EIDG.TECHNISCHE HOCHSCHULE,ZURICH,
C  SWITZERLAND.TRANSLATED FROM ALGOL TO FORTRAN IV BY S.LEVIN, COUP.
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9068.2
      DOUBLE PRECISION A(ID,1),P(200),Q(200),R(200),BIGA
      LOGICAL R,IFAIL
      IFAIL=.FALSE.
      N = NN
      DO 1 I=1,N
1    R(I)=.FALSE.
      DO 7 I=1,N
      BIGA=0.0
      DO 2 J=1,N
      IF (R(J).OR.DABS(A(J,J)).LT.BIGA) GO TO 2
      BIGA=DABS(A(J,J))
      K=J
2    CONTINUE
      IF (BIGA.EQ.0.) GO TO 8
      R(K)=.TRUE.
      Q(K)=1.0/A(K,K)
      P(K)=1.0
      A(K,K)=0.0
      IF ((K-1).EQ.0) GO TO 4
      KM1=K-1
      DO 3 J=1,KM1
      P(J)=A(J,K)
      IF (.NOT.R(J)) A(J,K)=-A(J,K)
      Q(J)=A(J,K)*Q(K)
3    A(J,K)=0.0
      IF (K.EQ.N) GO TO 6
4    J1=K+1
      DO 5 J=J1,N
      Q(J)=-A(K,J)*Q(K)
      IF (.NOT.R(J)) A(K,J)=-A(K,J)
      P(J)=-A(K,J)
5    A(K,J)=0.0
6    DO 7 J=1,N
      DO 7 K=J,N
7    A(J,K)=A(J,K)+P(J)*Q(K)
      GO TO 9
      SYM10051
      SYM10052
      SYM10053
      SYM10054
      SYM10055
      SYM10056
      SYM10057
      SYM20001
      SYM20002
      SYM20003
      SYM20004
      SYM20005
      SYM20006
      SYM20007
      SYM20008
      SYM20009
      SYM20010
      SYM20011
      SYM20012
      SYM20013
      SYM20014
      SYM20015
      SYM20016
      SYM20017
      SYM20018
      SYM20019
      SYM20020
      SYM20021
      SYM20022
      SYM20023
      SYM20024
      SYM20025
      SYM20026
      SYM20027
      SYM20028
      SYM20029
      SYM20030
      SYM20031
      SYM20032
      SYM20033
      SYM20034
      SYM20035
      SYM20036
      SYM20037
      SYM20038
      SYM20039
      SYM20040
      SYM20041
      SYM20042
      SYM20043

```

```
8 IFAIL=.TRUE.  
9 RETURN  
END
```

```
SYM2C044  
SYM20045  
SYM2C046
```


**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Band matrix solver.

Purpose

To solve a system of linear algebraic equations $Ax=b$ when the matrix A is of very large order and sparse in the sense that the only non-zero elements are in a band centered on the main diagonal.

Usage

The calling sequence is:

```
CALL BANSØL(C,N,M,V,ID)
```

where

N	an integer, is the order of A.
M	is the width of the band which must be an odd integer.
C	is the doubly-dimensioned, double precision $N \times M$ array which contains the pertinent elements of A: the band elements of a given row of A appear in the same row of C but shifted such that element $A(I,J)$ becomes $C(I,J-I+(M+1)/2)$.
V	is a singly-dimensioned, double precision array which initially contains the vector b and upon returning from BANSØL contains the answer vector x.
ID	is the integer first dimension of C and V.

Coding Information

BANSØL is a Fortran IV double precision subprogram which may be easily converted to single precision. As an example of how to dimension C in the minimum way, let A be tridiagonal and of order 200. In this case C could be dimensioned C(200,3). The values of undefined elements of C such as C(1,1) and C(N,M) are irrelevant. C is overwritten during the calculation.

Restrictions

A must be at least nonsingular. The integer M must not be less than 3 and N must not be less than 2. BANSØL does not provide for an unequal number of superdiagonal and subdiagonals as more recent programs [2]. These latter programs promise to be superior to BANSØL.

Accuracy

In general less operations will be performed when A is a band matrix rather than a full matrix. Hence systems with a somewhat higher condition number, $||A|| \cdot ||A^{-1}||$, will yield accurate answers..

Timing

The running time will be proportional to MN^2 .

Method

See DMATEQ in this compendium.

References

- [1] Martin, R. S., and Wilkinson, J. H. (1965): "Symmetric decomposition of positive definite band matrices" Numer. Math. vol 7, pp. 355-361.
- [2] Martin, R. S., and Wilkinson, J. H. (1967): "Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices" Numer. Math. vol 9, pp. 279-301.
- [3] Thurnau, D. H. (1963): Algorithm 195, Comm. ACM, vol. 6, 8.

Author

This program was translated from the ALGOL program of D. H. Thurnau [3] by J. Barish, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE BANSOL(C,N,M,V,ID)
C   PROGRAM AUTHOR      J. BARISH
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C   CTC ORD PROGRAM NO. 9069
C   BANSOL OPERATES ONLY ON BAND PORTION OF MATRIX A
C   GIVEN IN N X M ARRAY C. N IS ORDER OF A M IS NO.
C   OF DIAGONALS(MUST BE ODD) IN THE BAND. ELEMENTS OF A
C   GIVEN ROW IN A ARE SHIFTED SUCH THAT A(I,J) BECOMES
C   C(I,J-I+(M+1)/2) IN THE CALLING PROGRAM. SOLUTION
C   X OF AX=V IS STORED IN V AND CONTENTS OF C
C   ARE DESTROYED.
C
      DIMENSION C(ID,1),V(ID)
      DOUBLE PRECISION C,V,T
      L=(M+1)/2
      LI=L-1
      DO 2 IR=1,LI
      LR=L-IR
      DO 2 I=1,LR
      DO 1 J=2,M
1     C(IR,J-1)=C(IR,J)
      NP1=N+1-IR
      MP1=M+1-I
      C(NP1,MP1)=0.0
2     C(IR,M)=C(NP1,MP1)
      N1=N-1
      DO 0 I=1,N1
      IPIV=I
      IRE=I+1
      DO 3 IR=IRE,L
      IF(DABS(C(IR,1)).LE.DABS(C(IPIV,1))) GO TO 3
      IPIV=IR
3     CONTINUE
      IF(IPIV.EQ.I) GO TO 5
      T=V(I)
      V(I)=V(IPIV)
      V(IPIV)=T
      DO 4 J=1,M
      T=C(I,J)
      C(I,J)=C(IPIV,J)
4     C(IPIV,J)=T
5     V(I)=V(I)/C(I,1)
      DO 6 J=2,M
6     C(I,J)=C(I,J)/C(I,1)
      DO 8 IR=IRE,L
      T=C(IR,1)
      V(IR)=V(IR)-T*V(I)
      DO 7 J=2,M
7     C(IR,J-1)=C(IR,J)-T*C(I,J)

```

```

BANS0001
BANS0002
BANS0003
BANS0004
BANS0005
BANS0006
BANS0007
BANS0008
BANS0009
BANS0010
BANS0011
BANS0012
BANS0013
BANS0014
BANS0015
BANS0016
BANS0017
BANS0018
BANS0019
BANS0020
BANS0021
BANS0022
BANS0023
BANS0024
BANS0025
BANS0026
BANS0027
BANS0028
BANS0029
BANS0030
BANS0031
BANS0032
BANS0033
BANS0034
BANS0035
BANS0036
BANS0037
BANS0038
BANS0039
BANS0040
BANS0041
BANS0042
BANS0043
BANS0044
BANS0045
BANS0046
BANS0047
BANS0048
BANS0049
BANS0050

```

151

```
8 C(IR,M)=0.0
  IF(L.EQ.N) GO TO 9
  L=L+1
9 CONTINUE
  V(N)=V(N)/C(N,1)
  JM=2
  DO 11 ICE=1,N1
    IR=N-ICE
    DO 10 J=2,JM
      IRM1=IR-1+J
10 V(IR)=V(IR)-C(IR,J)*V(IRM1)
    IF(JM.EQ.M) GO TO 11
    JM=JM+1
11 CONTINUE
  RETURN
  END
```

```
BANS0051
BANS0052
BANS0053
BANS0054
BANS0055
BANS0056
BANS0057
BANS0058
BANS0059
BANS0060
BANS0061
BANS0062
BANS0063
BANS0064
BANS0065
BANS0066
```

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Tridiagonal solver.

Purpose

TRISØL solves systems of equations $Ax=b$ for tridiagonal matrices A , i.e., $a_{ij}=0, |j-i|>1$. TRISØL is programmed to conserve storage and improve calculation time over more general programs.

Usage

The calling sequence is:

CALL TRISØL(N,D,D1,D2,B)

where

N   an integer is the order of A.

D   is a singly-dimensioned double precision array
         of the main diagonal elements of A, A(1,1),
         A(2,2),...,A(N,N).

D1 & D2 are singly-dimensioned double precision arrays
         which contain the lower and upper diagonals of
         A respectively A(2,1),A(3,2),...,A(N,N-1) and
         A(1,2),A(2,3),...,A(N-1,N).

B   is a singly-dimensioned double precision array
         which initially contains the vector b and upon
         returning from TRISØL contains the answer vector x.

Coding Information

TRISØL is a Fortran IV double precision subprogram which may be easily converted to single precision. All the input arrays

are overwritten during the calculation. If systems of order $N > 100$ are to be solved, the temporary storage array D3 in TRISOL should be dimensioned not less than N .

Restrictions

A must be at least nonsingular.

Accuracy

There seems to be less dependence on $||A|| \cdot ||A^{-1}||$ than for full matrices.

Timing

The running time is proportional to N^2 .

Method

TRISOL uses Gaussian elimination with partial pivoting [1].

Reference

- [1] Forsythe, George E., and Moler, Cleve B., (1967): Computer Solution of Linear Algebraic Equations, Prentice-Hall.

Author

Mark Elson, formerly a summer employee of Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

SUBROUTINE TRISOL(M,D,D1,D2,P)	TRIS0001
C PROGRAM AUTHOR M. ELSON, FORMERLY WITH	TRIS0002
C COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	TRIS0003
C OAK RIDGE, TENN.	TRIS0004
C	TRIS0005
C CTC ORD PROGRAM NO. 9070	TRIS0006
DIMENSION L(1),D1(1),D2(1),D3(100),B(1)	TRIS0007
DOUBLE PRECISION D,D1,D2,D3,R,T	TRIS0008
DO11=1,4	TRIS0009
1 D3(I)=0.	TRIS0010
M1=M-1	TRIS0011
DO31=1,M1	TRIS0012
IF(DABS(D1(I)).LE.DABS(D(I)))GOTO2	TRIS0013
T=D1(I)	TRIS0014
D1(I)=D(I)	TRIS0015
D(I)=T	TRIS0016
T=D2(I)	TRIS0017
D2(I)=D(I+1)	TRIS0018
D(I+1)=T	TRIS0019
D3(I)=D2(I+1)	TRIS0020
D2(I+1)=0.	TRIS0021
T=B(I+1)	TRIS0022
B(I+1)=B(I)	TRIS0023
B(I)=T	TRIS0024
2 D(I+1)=D(I+1)-D2(I)*D1(I)/D(I)	TRIS0025
D2(I+1)=D2(I+1)-D3(I)*D1(I)/D(I)	TRIS0026
3 B(I+1)=B(I+1)-B(I)*D1(I)/D(I)	TRIS0027
B(M)=B(M)/D(M)	TRIS0028
B(M-1)=(B(M-1)-D2(M-1)*B(M))/D(M-1)	TRIS0029
DO4 I=2,M1	TRIS0030
K=M-I	TRIS0031
4 B(K)=(B(K)-D3(K)*B(K+2)-D2(K)*B(K+1))/D(K)	TRIS0032
RETURN	TRIS0033
END	TRIS0034

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Evaluation of complex determinants by Chio's method.

Purpose

CDET evaluates the determinant of a general complex matrix.

Usage

The calling sequence is:

```
CALL CDET(A,D,N,M)
```

where

A   is the doubly-dimensioned, double precision
          complex array containing the square matrix,
D   is the double precision value of the determinant
          of the matrix A,
N   is the integer order of A,
M   is the integer first dimension of A in the calling
          program.

Coding Information

CDET is a Fortran IV double precision, complex subprogram which may be easily converted to single precision complex. A is overwritten in the computation.

Method

The program uses Gaussian elimination with partial pivoting to reduce the matrix to upper triangular form. Then, it proceeds to evaluate the determinant as $\pm$ the product of the diagonal elements.

Reference

See DMATEQ and CMATEQ, this compendium.

Author

R. E. Funderlic, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

	SUBROUTINE CDET(A,D,NN,M)	CDET0001
C	PROGRAM AUTHOR R. E. FUNDERLIC,	CDET0002
C	COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	CDET0003
C	OAK RIDGE, TENN.	CDET0004
C		CDET0005
C	CTC ORD PROGRAM NO. 9071	CDET0006
	DIMENSION A(M,M)	CDET0007
	COMPLEX*16 A,D,T	CDET0008
	RFAL*8 R,S	CDET0009
	N = NN	CDET0010
	NM1 = N-1	CDET0011
	D = (1.,0.)	CDET0012
	DO 5 I=1,NM1	CDET0013
	S = 0.	CDET0014
	DO 1 J = I,N	CDET0015
	R = CDABS(A(J,I))	CDET0016
	IF (R.LT.S) GO TO 1	CDET0017
	S = R	CDET0018
	L = J	CDET0019
1	CONTINUE	CDET0020
	IF(L.EQ.I) GO TO 3	CDET0021
	DO 2 J = I,N	CDET0022
	T = A(I,J)	CDET0023
	A(I,J) = A(L,J)	CDET0024
2	A(L,J) = T	CDET0025
	D = -D	CDET0026
3	IF(CDABS(A(I,I)).EQ.0.) GO TO 8	CDET0027
	IP1 = I+1	CDET0028
	DO 5 J = IP1,N	CDET0029
	T = A(J,I)/A(I,I)	CDET0030
	DO 4 K = IP1,N	CDET0031
4	A(J,K) = A(J,K)-A(I,K)*T	CDET0032
5	CONTINUE	CDET0033
	DO 6 I = 1,N	CDET0034
6	D = D*A(I,I)	CDET0035
7	RETURN	CDET0036
8	D = (0.,0.)	CDET0037
	GO TO 7	CDET0038
	END	CDET0039

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Evaluation of determinants of Hermitian matrices.

Purpose

HDET calculates determinants of Hermitian matrices and the method guarantees numerical stability for positive definite matrices.

Usage

The calling sequence is:

```
CALL HDET(A,N,DET,ID,IFAIL)
```

where

A is the doubly-dimensioned double-precision complex array containing the Hermitian matrix whose determinant is to be calculated,

N is the integer order of A,

DET is the double precision real variable which returns the value for $\det(A)$ calculated by HDET,

ID is the (integer) first dimension of A in the calling program,

IFAIL is a return integer which will be set to 1 if HDET is unable to compute $\det(A)$ and 0 otherwise.

Coding Information

HDET is a Fortran IV double-precision complex subprogram which can easily be converted to single precision complex. The complex array A must be typed double precision (COMPLEX*16) in the calling program.

Restrictions

In the determinant calculation for some matrices, underflows or overflows may be encountered. Possible methods to avoid this are suggested in [2]. Although the method may work for some non-definite matrices, the user should realize that numerical stability is guaranteed only for positive definite matrices. The contents of the array A are overwritten in HDET.

Method

The matrix A is diagonalized by a sequence of congruence transformations in which the symmetry of A is used to advantage. The procedure is described in detail for the case of real symmetric positive definite matrices in [1].

Timing

The ratios of running times of HDET to CDET (a routine found elsewhere in this compendium which calculates determinants of arbitrary complex matrices) using single precision have been determined for a few orders of A using an IBM 360 model.

Order of A	Ratio of Run Times
10	.771
20	.622
40	.567

Accuracy

Some pertinent comments on accuracy of determinant calculations are available in [3].

References

- [1] Busing and Levy (1962): "A procedure for inverting large symmetric matrices," Comm. ACM 5, 8, pp. 445-6.
- [2] Forsythe, G. E., and Moler, C. B., (1967): Computer Solution of Linear Algebraic Systems, Prentice-Hall, pg. 55.
- [3] Wilkinson, J. H., (1965): The Algebraic Eigenvalue Problem, Oxford, pp. 228-9.

Author

John Rinzel, Summer Employee, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE HDET(A,N,DET,ID,IFAIL)
C  PROGRAM AUTHOR  JOHN RINZEL, A SUMMER EMPLOYEE WITH
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9072
      COMPLEX*16A(ID,1),FACT,TEMP
      REAL*8 BIG,BIGA,DET
      IFAIL=0
      NM1=N-1
      DO 7 II=2,NM1
        IIM1=II-1
        IIP1=II+1
        BIG = A(IIM1,IIM1)
        BIG = DABS(BIG)
        INT=IIM1
        DO 1 J=II,N
          BIGA = A(J,J)
          IF(DABS(BIGA).LE.BIG) GO TO 1
          BIG = DABS(BIGA)
          INT=J
1       CONTINUE
        IF(BIG.EQ.0.0)GO TO 11
        IF(INT.EQ.IIM1)GO TO 4
        DO 2 K=IIM1,INT
          TEMP=DCONJG(A(K,IIM1))
          A(K,IIM1)=DCONJG(A(INT,K))
2       A(INT,K)=TEMP
          TEMP=A(IIM1,IIM1)
          A(IIM1,IIM1)=DCONJG(A(INT,IIM1))
          A(INT,IIM1)=TEMP
          IF(INT.EQ.N)GO TO 4
          INTP1=INT+1
          DO 3 K=INTP1,N
            TEMP=A(K,IIM1)
            A(K,IIM1)=A(K,INT)
3       A(K,INT)=TEMP
4       DO 5 J=IIP1,N
5       A(1,J) =-A(J,IIM1)/A(IIM1,IIM1)
          DO 7 J=II,N
            JP1=J+1
            A(J,J)=A(J,J)-A(J,IIM1)*DCONJG(A(J,IIM1))/A(IIM1,IIM1)
            IF(J.EQ.N)GO TO 7
            FACT=DCONJG(A(J,IIM1))
            DO 6 I=JP1,N
6          A(I,J)=A(I,J)+A(1,I) *FACT
7       CONTINUE
          BIGA = A(N,N)
          BIG = A(NM1,NM1)
          IF (DABS(BIGA).LE.DABS(BIG)) GO TO 8
      HDET0001
      HDET0002
      HDET0003
      HDET0004
      HDET0005
      HDET0006
      HDET0007
      HDET0008
      HDET0009
      HDET0010
      HDET0011
      HDET0012
      HDET0013
      HDET0014
      HDET0015
      HDET0016
      HDET0017
      HDET0018
      HDET0019
      HDET0020
      HDET0021
      HDET0022
      HDET0023
      HDET0024
      HDET0025
      HDET0026
      HDET0027
      HDET0028
      HDET0029
      HDET0030
      HDET0031
      HDET0032
      HDET0033
      HDET0034
      HDET0035
      HDET0036
      HDET0037
      HDET0038
      HDET0039
      HDET0040
      HDET0041
      HDET0042
      HDET0043
      HDET0044
      HDET0045
      HDET0046
      HDET0047
      HDET0048
      HDET0049
      HDET0050

```

```
      TEMP=A(N,N)
      A(N,N)=A(NM1,NM1)-A(N,NM1)*DCCNJG(A(N,NM1))/A(N,N)
      A(NM1,NM1)=TEMP
      GO TO 9
P   A(N,N)=A(N,N)-A(N,NM1)*DCCNJG(A(N,NM1))/A(NM1,NM1)
S   DET=1.0
      DO 10 I=1,N
10  DET=DET*A(I,I)
      RETURN
11  IFAIL=1
      RETURN
      END
```

```
HDETC051
HDETC052
HDETC053
HDETC054
HDETC055
HDETC056
HDETC057
HDETC058
HDETC059
HDETC060
HDETC061
HDETC062
```

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

G

SOLUTION OF
EIGENSYSTEMS

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Test matrix generator.

Purpose

MATG1 and MATGT are used to generate real symmetric test matrices with known eigenvalues and eigenvectors. These test matrices have as entries numbers which can be represented exactly in an IBM 360 (0.1 does not have an exact binary representation and 1/3 doesn't even have an exact finite decimal representation).

Usage

The calling sequences are:

CALL MATGT(N,NEIG,MAT,ID)

and

CALL MATG1(N,NEIG,MAT)

where

N is the integer order of the matrix to be generated.

NEIG is a singly-dimensioned integer array supplied by the user which governs the eigenvalues of the output matrix.

MAT is the output matrix with integer elements. The eigenvalues of MAT are $N \cdot \text{NEIG}(I)$, $I=1,2,\dots,N$. If MATGT is used, MAT is a standard doubly-dimensioned integer array and if MATG1 is used, MAT is a singly-dimensioned integer array with elements in the order MAT(1,1), MAT(1,2), ..., MAT(1,N), MAT(2,2), ..., MAT(2,N), MAT(3,3), ..., MAT(3,N), ..., MAT(N,N). Each of the eigenvectors

of MAT will mathematically have $N-1$ components equal in magnitude to $1-(2/N)$.

ID is the integer dimensions of MAT in the calling program, i.e., MAT is dimensioned MAT(ID,ID).

Coding Information

Both MATGT and MATG1 are Fortran IV subprograms. When MATG1 is used, MAT must be dimensioned not less than $N(N+1)/2$. Of course in most applications upon return MAT will be floated. Quite often core storage is a problem and one would be well advised to equivalence MAT with say A and then float MAT onto itself, i.e., A.

Restrictions

The value $4(\text{NEIG}(1)+\dots+\text{NEIG}(N))=B$ must be a multiple of N . Often $\text{NEIG}(I)=I$ is used to satisfy this condition. Sometimes the integers to be produced for MAT are too large to be stored conveniently in an IBM 360, e.g., if $|B/N-2*(\text{NEIG}(I)*\text{NEIG}(J))| > 2^{31}-1$, cf. program list.

Method

The MAT test matrices are produced [1] by performing similarity transformations $H^{-1}DH$ on diagonal matrices D . The matrices H are elementary Hermitian.

Reference

- [1] Ortega, J. M. (1964): "Generation of test matrices by similarity transformations," Comm. ACM, vol. 7, 6, p. 377.

Author

Mark Elson, formerly a summer employee with the Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

C	SUBROUTINE MATGT (N,NEIG,MAT,ID)	MTGT0001
C	PROGRAM AUTHOR M. ELSON, FORMERLY WITH	MTGT0002
C	COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	MTGT0003
C	OAK RIDGE, TENN.	MTGT0004
C	CTC ORD PROGRAM NO. 9073.1	MTGT0005
C		MTGT0006
	DIMENSION NEIG(1),MAT(ID,ID)	MTGT0007
	ISUM=0	MTGT0008
	DO1I=1,N	MTGT0009
	1 ISUM=ISUM+NEIG(I)	MTGT0010
	DO2I=1,N	MTGT0011
	DO2J=1,N	MTGT0012
	MAT(I,J)=(4*ISUM)/N-2*(NEIG(I)+NEIG(J))	MTGT0013
	IF(I.NE.J) GO TO 2	MTGT0014
	MAT(I,J)=MAT(I,J)+N*NEIG(I)	MTGT0015
	2 CONTINUE	MTGT0016
	RETURN	MTGT0017
	END	MTGT0018
	SUBROUTINE MATG1 (NN,NEIG,MATRIX)	MTG10001
C	PROGRAM AUTHOR M. ELSON, FORMERLY WITH	MTG10002
C	COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	MTG10003
C	OAK RIDGE, TENN.	MTG10004
C		MTG10005
C	CTC ORD PROGRAM NO. 9073.2	MTG10006
	DIMENSION NEIG(1),MATRIX(1)	MTG10007
	N=NN	MTG10008
	NP1=N+1	MTG10009
	ISUM=0	MTG10010
	DO1I=1,N	MTG10011
	1 ISUM=ISUM+NEIG(I)	MTG10012
	II=0	MTG10013
	DO3I=1,N	MTG10014
	IJ=II	MTG10015
	DO2J=1,N	MTG10016
	IJ=IJ+1	MTG10017
	MATRIX(IJ)=(4*ISUM)/N-2*(NEIG(I)+NEIG(J))	MTG10018
	IF(I.NE.J)GOTO2	MTG10019
	MATRIX(IJ)=MATRIX(IJ)+N*NEIG(I)	MTG10020
	2 CONTINUE	MTG10021
	3 II=II+NP1-I	MTG10022
	RETURN	MTG10023
	END	MTG10024

Identification

Real symmetric matrix eigensystem solver.

Purpose

The matrix eigensystem solver BIGMAT calculates eigenvalues and eigenvectors of real symmetric matrices. The main entry, SYMMAT, accepts a standard doubly-dimensioned array while the subordinate entry BIGMAT assumes that the input matrix A is a singly-dimensioned compact array (thus economizing storage).

Usage

The calling sequences are:

```
CALL SYMMAT(A,EIGVAL,N,NEIG,NVEC,ID)
```

or

```
CALL BIGMAT(A,EIGVAL,N,NEIG,NVEC)
```

where

A is the input matrix, a singly-dimensioned, double precision array with elements in the order A(1,1), A(1,2),...,A(1,N), A(2,2),..., A(2,N), A(3,3),..., A(3,N),..., A(N,N) if BIGMAT is called. If SYMMAT is called, the array A is a standard doubly-dimensioned, double precision array. However, only the lower part of the matrix A need be defined, i.e. those elements A(I,J) such that I is not less than J.

EIGVAL is a singly-dimensioned, double precision array in which BIGMAT returns the eigenvalues (eigenvectors are written out in BIGMAT).

N is the integer order of the matrix A.

NEIG is the integer number of eigenvalues desired (BIGMAT will calculate the NEIG largest algebraic eigenvalues).

NVEC is the integer number of eigenvectors (of Euclidean length 1) desired (BIGMAT will calculate the vectors corresponding to the largest NVEC eigenvalues).

ID refers to the dimensions of A if SYMMAT is called, i.e. A should be dimensioned A(ID,ID) in the calling program.

Coding Information

The user must have $NVEC \leq NEIG \leq N$. If the user desires the NEIG smallest eigenvalues, he can of course read in $-A$, and then reverse the signs on the returned roots.

Since the dimensions of A and EIGVAL in the subroutine are irrelevant to allocated storage, the dimensions of BIGMAT should not normally be changed. However, BIGMAT makes use of four temporary arrays of dimension 235 and one of dimension 235 x 3 and these may be altered to any dimension not less than N (and N x 3), where N is the order of the matrix. These scratch vectors could be added to the argument list (along with the equivalenced vectors) if extra storage is needed when BIGMAT is not being executed. Thus BIGMAT essentially uses $N^2/2$ storage locations. The matrix A is overwritten. BIGMAT writes the eigenvectors with corresponding eigenvalues as they are calculated (data set 6 is used). No other I/O transmission occurs.

Restrictions

Equal eigenvalues return identical corresponding eigenvectors and close roots may return non-orthogonal but linearly independent eigenvectors. Various orthogonalization techniques [4], [6] have been investigated for clustered roots, but no satisfactory method was found that would be compatible with BIGMAT's storage capabilities. B. Garbow has written a program [4] using $2N^2$ locations that yields good orthonormal vectors for some matrices that have identical roots. Ortega [12] used several of his suggestions. If time and storage are of little consequence, a program using the Jacobi method yields orthonormal eigenvectors [1], [10]. Also see QRVEC in this compendium which is faster than Jacobi routines.

The problem of overflow has been minimized but not entirely eliminated in BIGMAT. As in most eigenvalue routines, if

$A = (a_1, a_2, \dots, a_N)$, then it is required that $a_i^T a_i \leq \beta$ (the largest machine number). Also if $A \neq 0$, it is required that not all $a_i^T a_i < \sigma$ (the smallest positive machine number). In these cases the matrix should be scaled.

BIGMAT solves only systems of order greater than 2. Less refined techniques are available for lower order matrices.

Timing

BIGMAT has been timed on an IBM 7090 32K machine using single precision. Times obviously vary somewhat not only with order, but also with individual matrices. Indicative execution times

for test matrices (all roots and vectors) include .749 seconds for $N = 10$, 90 seconds for $N = 80$, and 17.66 minutes for $N = 200$.

The tridiagonalization time is independent of NEIG and NVEC; time of subsequent determination of roots and vectors, however, varies approximately linearly with these parameters. Also pertinent to execution time is the fact that there are $2N^3/3$ multiplications in the tridiagonalization and $(NVEC)N^2$ for the vectors. Apparently the dominant role of the Sturm sequence calculations for small matrices has led some SHARE contributors to claim the entire calculation time varies as N^2 . The cubic character of the calculation is found in several cubic loops (innermost of triply nested DO statements) and most Fortrans currently compile these loops very inefficiently. Since almost all the computation is due to these loops, the ALTER option of Fortran IV was suggested, and it was found that execution time could be cut by at least 35% for matrices of order 80 on the 7090. Another suggestion was to make the cubic loops into subroutines in a judicious way and this gave almost as much improvement. This is in fact the compromise that is used in BIGMAT. The following table gives some examples of the timing we have experienced on the 7090 using single precision.

ORDER	TRIDIAGONALIZATION	EIGENVALUES	EIGENVECTORS	TOTAL TIME
40	3.8 Sec.	6.5 Sec.	5.9 Sec.	16.2 Sec.
60	11.8 Sec.	13.8 Sec.	18.0 Sec.	43.6 Sec.
80	27.0 Sec.	24.0 Sec.	39.6 Sec.	90.6 Sec.
100	51.7 Sec.	36.8 Sec.	73.2 Sec.	161.7 Sec.
200	6.60 Min.	2.33 Min.	8.73 Min.	17.66 Min.
235	10.67 Min.	3.18 Min.	13.84 Min.	27.69 Min.

Accuracy

For accuracy considerations it was desirable to produce large matrices that had integer entries and known roots and vectors. A method suggested by J. Ortega [7] was used. BIGMAT will usually obtain eigenvalues to an accuracy about $10^{-15} |\lambda_1|$ where λ_1 is the greatest eigenvalue of A. Hence the greater $|\lambda_1|/\min|\lambda_i|$ the fewer significant digits the smaller eigenvalues will have. For $N = 200$ and the above ratio equal to 200, a test matrix yielded roots accurate to 6-8 significant figures (biggest roots calculated most accurately) using a single precision version on a 7090 and corresponding vectors accurate to 3-8 figures, with most components being accurate to at least five figures. For $N = 10$ and a ratio of 10, slightly better accuracy was obtained.

Method

The input matrix A of order N is reduced to a tridiagonal matrix T by a sequence of (real) elementary Hermitian matrices

$$H_i = H_i^T = H_i^{-1}, [8]. \text{ In fact } T = H_{N-2} \dots H_1 A H_1 \dots H_{N-2} = H^T A H$$

Arriving at a symmetric form for the pertinent submatrices takes a little algebraic juggling since H_1A is not in general symmetric. The roots $\lambda_i(A)$ are calculated by bisection using Sturm sequence properties of T , [5]. For each root upper and lower bounds λ_μ and λ_ℓ are refined until $(\lambda_\mu + \lambda_\ell)/2$ equals either λ_μ or λ_ℓ in the computer. A significant amount of computation time could be saved by giving a less stringent convergence criterion (cf. Timing and [10]). Two passes of the Wielandt inverse power method [9] are used to calculate the eigenvectors y_i of T . Very little work is needed for the second pass since the elementary triangular and permutation matrices used in triangularizing $T - \lambda_i I$ are saved in the arrays TMULT and INTER. Finally the eigenvectors $x_i = Hy_i$ of A are calculated. The matrix H resides in storage made available in A during the tridiagonalization.

Acknowledgment

Programs written at Argonne National Laboratory by Burton Garbow [2], [3], [4] have yielded excellent suggestions. J. L. Cox and Ross Burris gave several programming suggestions pertinent to decreasing running time.

The authors would appreciate receiving matrices (on cards) that failed.

References

- [1] Corbató, F. J. (1959): SHARE 705
- [2] Garbow, B. (1959): SHARE 664

- [3] Garbow, B. (1962): SHARE 1385
- [4] Garbow, B. (1964): (private communication)
- [5] Givens, W. (1954): Numerical Computation of the Characteristic Values of a Real Symmetric Matrix, p. 16, ORNL 1574.
- [6] Golub, G., and Kahan, W. (1964): Calculating the Singular Values and Pseudo-Inverse of a Matrix, Stanford Technical Report CS8, 17-30.
- [7] Ortega, J. M. (1964): "Generation of test matrices by similarity transformations," Comm. ACM, 7, 6, p. 377.
- [8] Wilkinson, J. H. (1960): "Householder's method for the solution of the algebraic eigenproblem," Comput. J., 3, pp. 23-27.
- [9] Wilkinson, J. H. (1963): Rounding Errors in Algebraic Processes, Prentice-Hall, Inc., pp. 142-147.
- [10] Funderlic, R. E. (April 1965): Rev. 7311, Computing Reviews, 6, 2, p. 120.
- [11] White, Paul A., and Brown, Robert R. (1964): "A comparison of methods for computing the eigenvalues and eigenvectors of a real symmetric matrix," Math. Computat. 18, 87, pp. 457-463.
- [12] Ortega, James (1967): "The Givens-Householder method for symmetric matrices," Mathematical Methods for Digital Computers, vol. 2, eds. Ralston & Wilf, Wiley, 94-115.
- [13] Wilkinson, J. H. (1965): The Algebraic Eigenvalue Problem, Oxford.

Authors

R. E. Funderlic, Computing Technology Center, and Mark Elson, formerly a summer employee at Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE SYMMAT(A,VALU,N,NEIG,NVEC,ID)
C PROGRAM AUTHORS R. E. FUNDERLIC AND M. ELSON,
C COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C OAK RIDGE, TENN.
C
C CTC OFD PROGRAM NO. 9074
      IMPLICIT REAL*8(A-H,O-Z)
      REAL*4 ENORM1,ENORM2
      DIMENSION A(27730),VALU(1),VALL(235),UPPERD(235),DIAG(235),V(235),
1 SQGAMP(235),TMULT(235),T(235,3),SQGAMR(235),UPPER2(235)
      LOGICAL INTER(235)
      EQUIVALENCE (TMULT,SQGAMP,VALL),(T,SQGAMR),(T(1,2),UPPER2)
      L=0
      DO 1 J=1,N
      DO 1 I=J,N
      L=L+1
      K=I+ID*(J-1)
1 A(L)=A(K)
      ENTRY SYMMAT(A,VALU,N,NEIG,NVEC)
      ONE = 1.
      IF(N.LE.2)GO TO 41
      NP1=N+1
      NM1=N-1
      NM2=N-2
      NT2P1=N*2+1
      IX=0
C
C TRANSFORM A TO TRIDIAGONAL FORM
C
      DO 12 I=1,NM2
      SIGMA2=C.
      IP1=I+1
      DO 2 J=IP1,N
      IJ=IX+J
2 SIGMA2=SIGMA2+A(IJ)**2
      SIGMA=DSQRT(SIGMA2)
      II=IX+I
      DIAG(I)=A(II)
      IIP1=IX+I+1
      UPPERD(I)=SIGMA
      IF(A(IIP1).GT.C.) UPPERD(I) =-SIGMA
      UPPER2(I)=SIGMA2
      IF(DABS(SIGMA).GT.DABS(A(IIP1)))GO TO 3
      UPPERD(I)=A(IIP1)
      A(IIP1)=C.
      GO TO 11
3 A(IIP1)=DSQRT(1.+DABS(A(IIP1))/SIGMA)
      SQTGAM = SIGMA*A(IIP1)
      IF(UPPERD(I).GT.O.) SQTGAM = -(SIGMA*A(IIP1))
      IP2=I+2

```

```

SYMTO001
SYMTO002
SYMTO003
SYMTO004
SYMTO005
SYMTO006
SYMTO007
SYMTO008
SYMTO009
SYMTO010
SYMTO011
SYMTO012
SYMTO013
SYMTO014
SYMTO015
SYMTO016
SYMTO017
SYMTO018
SYMTO019
SYMTO020
SYMTO021
SYMTO022
SYMTO023
SYMTO024
SYMTO025
SYMTO026
SYMTO027
SYMTO028
SYMTO029
SYMTO030
SYMTO031
SYMTO032
SYMTO033
SYMTO034
SYMTO035
SYMTO036
SYMTO037
SYMTO038
SYMTO039
SYMTO040
SYMTO041
SYMTO042
SYMTO043
SYMTO044
SYMTO045
SYMTO046
SYMTO047
SYMTO048
SYMTO049
SYMTO050

```

DO 4 J=IP2,N	SYMT0051
IJ=IX+J	SYMT0052
4 A(IJ)=A(IJ)/SQTGAM	SYMT0053
JK1=(I*(2*N-I-1))/2	SYMT0054
JX=JK1	SYMT0055
IIX=JK1	SYMT0056
DO 6 J=IP1,N	SYMT0057
SQGAMP(J)=0.	SYMT0058
JK=JK1+J	SYMT0059
DO 5 K=IP1,J	SYMT0060
IK=IX+K	SYMT0061
SQGAMP(J)=SQGAMP(J)+A(JK)*A(IK)	SYMT0062
5 JK=JK+N-K	SYMT0063
IF(J.EQ.N)GO TO 7	SYMT0064
CALL LOOP1(J+2,NP1,SQGAMP(J),A(JX),A(IX))	SYMT0065
6 JX=JX+N-J	SYMT0066
7 DELGAM=0.	SYMT0067
DO 8 J=IP1,N	SYMT0068
IJ=IX+J	SYMT0069
8 DELGAM=DELGAM+A(IJ)*SQGAMP(J)	SYMT0070
DGO2=.5*DELGAM	SYMT0071
DO 9 J=IP1,N	SYMT0072
IJ=IX+J	SYMT0073
9 SQGAMR(J)=SQGAMP(J)-DGO2*A(IJ)	SYMT0074
DO 10 II=IP1,N	SYMT0075
III=IX+II	SYMT0076
CALL LOOP2(A(IIX),A(IX),SQGAMR,SQGAMR(II),A(III),II+1,NP1)	SYMT0077
10 IIX=IIX+N-II	SYMT0078
11 IX=IX+N-I	SYMT0079
12 CONTINUE	SYMT0080
M=(N*(N+1))/2	SYMT0081
.C CALCULATE NORM OF T, I.E. MAXIMUM ROW SUM OF ABS(T)	SYMT0082
C	SYMT0083
C	SYMT0084
UPPERD(NM1)=A(M-1)	SYMT0085
UPPER2(NM1)=UPPERD(NM1)**2	SYMT0086
DIAG(NM1)=A(M-2)	SYMT0087
DIAG(N)=A(M)	SYMT0088
ENORM1 =DABS(DIAG(1))+DABS(UPPERD(1))	SYMT0089
ENORM2 =DABS(DIAG(N))+DABS(UPPERD(NM1))	SYMT0090
ENORM = AMAX1(ENORM1,ENORM2)	SYMT0091
DO 13 I=2,NM1	SYMT0092
ENRTMP=DABS(DIAG(I))+DABS(UPPERD(I))+DABS(UPPERD(I-1))	SYMT0093
13 IF(ENRTMP.GT.ENORM)ENORM=ENRTMP	SYMT0094
DO 14 I=1,NEIG	SYMT0095
VALU(I)=ENORM	SYMT0096
14 VALL(I)=-ENORM	SYMT0097
C	SYMT0098
C STURM SEQUENCE CALCULATION OF THE EIGENVALUES.	SYMT0099
C	SYMT0100

DO 26 I=1,NFIG	SYMT0101
15 ROOT=.5*(VALU(I)+VALL(I))	SYMT0102
IF(ROOT.EQ.VALL(I).OR.ROOT.EQ.VALU(I))GO TO 26	SYMT0103
NAGREE=0	SYMT0104
PM2=0.	SYMT0105
PM1=1.	SYMT0106
DO 23 J=1,N	SYMT0107
IF(PM2.NE.0.)GO TO 17	SYMT0108
16 PM1 =DSIGN(ONE,PM1)	SYMT0109
GO TO 19	SYMT0110
17 IF(PM1.NE.0.)GO TO 19	SYMT0111
18 P=-DSIGN(ONE,PM2)	SYMT0112
PM2=0.	SYMT0113
IF(UPPER2(J-1))20,16,20	SYMT0114
19 P=DIAG(J)-ROOT-(UPPER2(J-1)*PM2)/PM1	SYMT0115
PM2=1.	SYMT0116
20 IF(P)23,21,22	SYMT0117
21 PM2=PM1	SYMT0118
IF(PM2)23,22,22	SYMT0119
22 NAGREE=NAGREE+1	SYMT0120
23 PM1=P	SYMT0121
DO 25 J=1,NFIG	SYMT0122
IF(J.LE.NAGREE)GO TO 24	SYMT0123
IF(VALU(J).LE.ROOT)GO TO 15	SYMT0124
VALU(J)=ROOT	SYMT0125
GO TO 25	SYMT0126
24 VALL(J)=ROOT	SYMT0127
25 CONTINUE	SYMT0128
GO TO 15	SYMT0129
26 CONTINUE	SYMT0130
IF(NVEC.EQ.0)GO TO 41	SYMT0131
EPSLON=ENORM*1.E-16	SYMT0132
IF(EPSLON.EQ.0.)EPSLON=1.D-30	SYMT0133
C EIGENVECTOR CALCULATION	SYMT0134
C	SYMT0135
C	SYMT0136
DO 40 I=1,NVEC	SYMT0137
DO 27 J=1,N	SYMT0138
V(J)=1.	SYMT0139
T(J,2)=DIAG(J)-VALU(I)	SYMT0140
IF(J.EQ.N)GO TO 28	SYMT0141
T(J,3)=UPPERD(J)	SYMT0142
27 T(J+1,1)=UPPERD(J)	SYMT0143
28 T(N,3)=0.	SYMT0144
DO 31 J=1,N	SYMT0145
IF(DABS(T(J,2)).LT.1.D-30)T(J,2)=EPSLON	SYMT0146
T(J,1)=T(J,2)	SYMT0147
T(J,2)=T(J,3)	SYMT0148
T(J,3)=0.	SYMT0149
IF(J.EQ.N)GO TO 32	SYMT0150

```

INTER(J)=.FALSE.
IF(DABS(T(J+1,1)).LE.DABS(T(J,1)))GO TO 30
INTER(J)=.TRUE.
DO 29 K=1,3
TEMP=T(J,K)
T(J,K)=T(J+1,K)
29 T(J+1,K)=TEMP
30 TMULTP=T(J+1,1)/T(J,1)
TMULT(J)=TMULTP
T(J+1,2)=T(J+1,2)-TMULTP*T(J,2)
31 T(J+1,3)=T(J+1,3)-TMULTP*T(J,3)
32 ITER=1
DO 34 J=1,N
L=N+1-J
34 V(L)=(V(L)-T(L,2)*V(L+1)-T(L,3)*V(L+2))/T(L,1)
VNORM=0.
DO 35 L=1,N
35 VNORM=VNORM+V(L)**2
VNORM=DSQRT(VNORM)
DO 36 J=1,N
36 V(J)=V(J)/VNORM
IF(ITER.EQ.2)GO TO 38
ITER=2
DO 37 L=2,N
IF(.NOT.INTER(L-1))GO TO 37
VTEMP=V(L-1)
V(L-1)=V(L)
V(L)=VTEMP
37 V(L)=V(L)-TMULT(L-1)*V(L-1)
GO TO 33
38 IF(VNORM.EQ.0.)V(I)=1.
IIX=(N*N-N-6)/2
DO 39 KK=1,NM2
IIP1=N-KK
UTV=C.
CALL LOOP3(UTV,A(IIX),V,IIP1+1,NP1)
CALL LOOP4(A(IIX+1),V,N,IIP1,UTV)
39 IIX=IIX+IIP1-N-2
40 WRITE(6,100)VALU(I),(V(J),J=1,N)
100 FORMAT(30H EIGENVECTOR CORRESPONDING TO 1PD25.16/
1 (1H 1P5D25.16))
41 RETURN
END
SUBROUTINE LOOP1(JP2,NP1,SGAMPJ,AJX,AIX)
C
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION AJX(235),AIX(235)
DO 1 L=JP2,NP1
1 SGAMPJ=SGAMPJ+AJX(L)*AIX(L)
RETURN

```

```

SYMTO151
SYMTO152
SYMTO153
SYMTO154
SYMTO155
SYMTO156
SYMTO157
SYMTO158
SYMTO159
SYMTO160
SYMTO161
SYMTO162
SYMTO163
SYMTO164
SYMTO165
SYMTO166
SYMTO167
SYMTO168
SYMTO169
SYMTO170
SYMTO171
SYMTO172
SYMTO173
SYMTO174
SYMTO175
SYMTO176
SYMTO177
SYMTO178
SYMTO179
SYMTO180
SYMTO181
SYMTO182
SYMTO183
SYMTO184
SYMTO185
SYMTO186
SYMTO187
SYMTO188
SYMTO189
SYMTO190
SYMTO191
SYMTO192
SYMTO193
SYMTO194
SYMTO195
SYMTO196
SYMTO197
SYMTO198
SYMTO199
SYMTO200

```

END	SYMT 201
SUBROUTINE LOOP2(AIIX,AIX,S,SI,AIII,IP1,NP1)	SYMT 202
IMPLICIT REAL*8(A-H,O-Z)	SYMT 203
DIMENSION AIIX(235),AIX(235),S(235)	SYMT 204
DO 1 JJ=IP1,NP1	SYMT 205
1 AIIX(JJ)=AIX(JJ)-AIII*S(JJ-1)-SI*AIX(JJ)	SYMT 206
RETURN	SYMT 207
END	SYMT 208
SUBROUTINE LOOP3(UTV,AIIX,V,IIP2,NP1)	SYMT 209
IMPLICIT REAL*8(A-H,O-Z)	SYMT 210
DIMENSION AIIX(235),V(235)	SYMT 211
DO 1 J=IIP2,NP1	SYMT 212
1 UTV=UTV+AIIX(J)*V(J-1)	SYMT 213
RETURN	SYMT 214
END	SYMT 215
SUBROUTINE LOOP4(AIIX,V,N,IIP1,UTV)	SYMT 216
IMPLICIT REAL*8(A-H,O-Z)	SYMT 217
DIMENSION AIIX(235),V(235)	SYMT 218
DO 1 K=IIP1,N	SYMT 219
1 V(K)=V(K)-AIIX(K)*UTV	SYMT 220
RETURN	SYMT 221
END	SYMT 222

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Real symmetric QR eigensystem solver.

Purpose

QRVEC calculates the eigenvalues and orthogonal eigenvectors of a real symmetric matrix using the QR technique. The eigenvectors are calculated by carrying along the effects of the rotations used to compute the eigenvalues in a manner similar to the Jacobi method. This method insures that eigenvectors corresponding to repeated eigenvalues will be orthogonal.

Usage

The calling sequence for QRVEC is:

```
CALL QRVEC (A,D,T,N,M,NITER,C)
```

where

A is an M x M dimensioned double precision array containing the N x N matrix whose eigenvalues and eigenvectors are to be computed. Upon return A will contain the eigenvectors. The original matrix is destroyed by the calculations.

D is a singly dimensioned, double precision array used for temporary storage and upon return from the program contains the eigenvalues of A with the i^{th} eigenvalue corresponding to the eigenvector in the i^{th} column of A.

T is a singly dimensioned, double precision array used for temporary storage.

- N is the integer order of the matrix A whose eigenvalues are to be calculated.
- M is the integer variable dimension of the array A.
- NITER is the maximum integer number of QR iterations to be performed in attempting to compute any one eigenvalue of A. (50 should be sufficient.)
- C is a logical variable which upon return will have the value .FALSE. if and only if the maximum number of iterations (NITER) has been exceeded in attempting to find an eigenvalue of A.

Coding Information

QRVEC is a Fortran IV double-precision subprogram which may be easily converted to single precision. The variables D and T should be dimensioned not less than N and N-1, respectively.

Accuracy and Timing

The program will usually determine eigenvalues to an accuracy of about $10^{-15}|\lambda_1|$, where λ_1 is greatest eigenvalue of A. The accuracy of the eigenvectors will depend on their condition [2]; in particular, eigenvectors corresponding to close eigenvalues will be poorly determined. However, the set of approximate eigenvectors returned will be very nearly orthogonal.

QRVEC seems to be somewhat faster than routines employing the Jacobi method and somewhat slower than methods using inverse iteration to calculate the eigenvectors (see SYMMAT/BIGMAT in this

compendium). It is to be preferred over the latter when the user expects multiple eigenvalues and requires a set of orthogonal eigenvectors.

Method

The matrix A is reduced to tridiagonal form using the Householder reduction. The diagonal elements of the result are stored in D and the off diagonal elements in T. The transformations used in the reduction are then multiplied out and the resulting matrix, which is a first approximation to the eigenvectors of the original matrix, is stored in the array A. The eigenvalues are calculated one by one using the QR iteration (Wilkinson [1]) on the matrix contained in D and T. After each eigenvalue has been found, the problem is deflated. The transformations used in the iteration are applied to the matrix contained in A, so that after all the eigenvalues have been found, A contains the eigenvectors of the original matrix.

References

- [1] Wilkinson, J. H. (1965): "The QR algorithm for real symmetric matrices with multiple eigenvalues," Comput. J., 8, 1, 85-87.
- [2] Wilkinson, J. H. (1965): The Algebraic Eigenvalue Problem, Oxford.

Author

G. W. Stewart III, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE QRVEC(A,D,T,NN,M,NITER,CC)
C  PROGRAM AUTHOR  G. W. STEWART,III
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9075
C  QRVEC IS A FORTRAN IV SUBROUTINE TO CALCULATE THE EIGENVALUES
C  AND EIGENVECTORS OF A REAL SYMMETRIC MATRIX USING THE QR
C  ITERATION. ITS CALLING SEQUENCE IS
C      CALL QRVEC(A,D,T,N,M,NITER,C)
C  WHERE
C      A      IS A DOUBLY SUBSCRIPTED ARRAY (DIMENSIONED MXM)
C              CONTAINING THE NXN MATRIX WHOSE EIGENVALUES
C              AND EIGENVECTORS ARE TO BE FOUND. ON RETURN
C              THE COLUMNS OF A CONTAIN THE EIGENVECTORS.
C      D      IS A SINGLY SUBSCRIPTED ARRAY WHICH ON RETURN
C              CONTAINS THE EIGENVALUES.
C      T      IS A SINGLY SUBSCRIPTED ARRAY OF DIMENSION AT
C              LEAST N-1 USED FOR TEMPORARY STORAGE.
C      N      IS THE ORDER OF THE MATRIX CONTAINED IN A.
C      M      IS THE VARIABLE DIMENSION OF THE ARRAY A.
C      NITER  IS THE MAXIMUM NUMBER OF QR ITERATIONS TO BE
C              PERFORMED IN THE COURSE OF FINDING ANY ONE
C              EIGENVALUE.
C      CC     IS A LOGICAL VARIABLE WHICH ON RETURN WILL BE
C              SET .FALSE. IF AND ONLY IF THE QR ITERATION
C              FOR ANY EIGENVALUE FAILED TO CONVERGE AFTER
C              NITER ATTEMPTS.
C
C  IMPLICIT REAL*8 (A-H,O-Z)
C  REAL*8 KS
C  DIMENSION A(M,1),D(1),T(1)
C  LOGICAL CC
C  CC = .TRUE.
C  N = NN
C  NM1 = N-1
C  NM2 = N-2
C
C  REDUCE THE MATRIX TO TRIDIAGONAL FORM
C
C  DO 100 L=1,NM2
C  L1 = L+1
C  SS = 0.
C  DO 25 I=L1,N
25  SS = SS + A(I,L)*A(I,L)
C  IF(SS.NE.0.) GO TO 30
C  D(L) = A(L,L)
C  T(L) = 0.
C  A(L,L) = .5
C  DO 27 I=L1,N

```

```

QRVC0001
QRVC0002
QRVC0003
QRVC0004
QRVC0005
QRVC0006
QRVC0007
QRVC0008
QRVC0009
QRVC0010
QRVC0011
QRVC0012
QRVC0013
QRVC0014
QRVC0015
QRVC0016
QRVC0017
QRVC0018
QRVC0019
QRVC0020
QRVC0021
QRVC0022
QRVC0023
QRVC0024
QRVC0025
QRVC0026
QRVC0027
QRVC0028
QRVC0029
QRVC0030
QRVC0031
QRVC0032
QRVC0033
QRVC0034
QRVC0035
QRVC0036
QRVC0037
QRVC0038
QRVC0039
QRVC0040
QRVC0041
QRVC0042
QRVC0043
QRVC0044
QRVC0045
QRVC0046
QRVC0047
QRVC0048
QRVC0049
QRVC0050

```

27	A(I,L) = 0.	QRVC0051
	A(L1,L) = 1.	QRVC0052
	GO TO 32	QRVC0053
30	S = DSQRT(SS)	QRVC0054
	IF(A(L1,L).LT.0.) S=-S	QRVC0055
	D(L) = A(L,L)	QRVC0056
	T(L) = -S	QRVC0057
	A(L1,L) = S + A(L1,L)	QRVC0058
	A(L,L) = S*A(L1,L)	QRVC0059
32	SS = 0.	QRVC0060
	DO 50 I=L1,N	QRVC0061
	PP = 0.	QRVC0062
	DO 35 J=L1,I	QRVC0063
35	PP = PP + A(I,J)*A(J,L)	QRVC0064
	IF(I.EQ.N) GO TO 45	QRVC0065
	II = I+1	QRVC0066
	DO 40 J=II,N	QRVC0067
40	PP = PP + A(J,I)*A(J,L)	QRVC0068
45	A(L,I) = PP/A(L,L)	QRVC0069
50	SS = SS + A(I,L)*A(L,I)	QRVC0070
	ALPHA = .5*SS/A(L,L)	QRVC0071
	DO 75 I=L1,N	QRVC0072
	A(L,I) = A(L,I) - ALPHA*A(I,L)	QRVC0073
	DO 75 J=L1,N	QRVC0074
75	A(I,J) = A(I,J) - A(I,L)*A(L,J) - A(J,L)*A(L,I)	QRVC0075
100	CONTINUE	QRVC0076
	D(NM1) = A(NM1,NM1)	QRVC0077
	D(N) = A(N,N)	QRVC0078
	T(NM1) = A(N,NM1)	QRVC0079
	A(NM1,NM1) = 1.	QRVC0080
	A(N,N) = 1.	QRVC0081
	A(NM1,N) = 0.	QRVC0082
	A(N,NM1) = 0.	QRVC0083
C		QRVC0084
C	CALCULATE THE TRANSFORMATION BY BACK MULTIPLICATION	QRVC0085
C		QRVC0086
	DO 200 LL=1,NM2	QRVC0087
	L = NM2-LL+1	QRVC0088
	L1 = L+1	QRVC0089
	DO 150 I=L1,N	QRVC0090
	PP = 0.	QRVC0091
	DO 125 J=L1,N	QRVC0092
125	PP = PP + A(J,L)*A(J,I)	QRVC0093
150	A(L,I) = PP/A(L,L)	QRVC0094
	DO 175 I=L1,N	QRVC0095
	DO 175 J=L1,N	QRVC0096
175	A(I,J) = A(I,J) - A(I,L)*A(L,J)	QRVC0097
	DO 185 I=L1,N	QRVC0098
	A(L,I) = 0.	QRVC0099
185	A(I,L) = 0.	QRVC0100

```

200 A(L,L) = 1.
C
C   START THE QR DIAGONALIZATION
C
  DO 500 LL=1,NM1
    L = N-LL
    L1 = L+1
    DO 400 KKK=1,NITER
C
C   PERFORM ONE QR CYCLE
C
  KS = D(L1)
  DD = D(1) - KS
  TT = T(1)
  DO 300 I=1,L
C
C   CALCULATE THE ROTATION
C
  Q = DSQRT(DD**2 + TT**2)
  IF(Q.NE.0.) GO TO 225
  C = 1.
  S = 0.
  GO TO 235
225 C = DD/Q
  S = TT/Q
C
C   PREMULTIPLY
C
235 A11 = C*D(I) + S*T(I)
  A12 = C*T(I) + S*D(I+1)
  A21 = S*D(I) - C*T(I)
  A22 = S*T(I) - C*D(I+1)
  DD = S*T(I) - C*(D(I+1)-KS)
  TT = T(I+1)
  IF(I.NE.1) T(I-1) = C*T(I-1) + S*TT
C
C   POSTMULTIPLY
C
  D(I) = C*A11 + S*A12
  D(I+1) = S*A21 - C*A22
  T(I) = S*A11 - C*A12
C
C   ROTATE THE EIGENVECTORS
C
  DO 250 J=1,N
    TEMP = C*A(J,I) + S*A(J,I+1)
    A(J,I+1) = S*A(J,I) - C*A(J,I+1)
250 A(J,I) = TEMP
    IF(I.EQ.L) GO TO 300
    TTT = S*T(I+1)

```

```

QRVC0101
QRVC0102
QRVC0103
QRVC0104
QRVC0105
QRVC0106
QRVC0107
QRVC0108
QRVC0109
QRVC0110
QRVC0111
QRVC0112
QRVC0113
QRVC0114
QRVC0115
QRVC0116
QRVC0117
QRVC0118
QRVC0119
QRVC0120
QRVC0121
QRVC0122
QRVC0123
QRVC0124
QRVC0125
QRVC0126
QRVC0127
QRVC0128
QRVC0129
QRVC0130
QRVC0131
QRVC0132
QRVC0133
QRVC0134
QRVC0135
QRVC0136
QRVC0137
QRVC0138
QRVC0139
QRVC0140
QRVC0141
QRVC0142
QRVC0143
QRVC0144
QRVC0145
QRVC0146
QRVC0147
QRVC0148
QRVC0149
QRVC0150

```

```
      T(I+1) = -C*T(I+1)
300  CONTINUE
      IF(DABS(T(L)/D(L1)) .LE. 1.D-8) GO TO 500
400  CONTINUE
      CC = .FALSE.
500  CONTINUE
      RETURN
      END
```

```
QRVC0151
QRVC0152
QRVC0153
QRVC0154
QRVC0155
QRVC0156
QRVC0157
QRVC0158
```


**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

The Jacobi iterative method for eigensystems of real symmetric matrices.

Purpose

MATDAG solves for eigenvalues and orthogonal eigenvectors of real symmetric matrices by the use of a cyclic Jacobi method. MATDAG has speed advantages over other methods, [1], when the matrix is almost diagonal. In this regard, if the user has a program iterative system in which a Matrix A_i only changes slightly from A_{i-1} , usually the eigenvalues of A_i will not be very different from those of A_{i-1} . Hence if X is a matrix consisting of the eigenvectors of A_{i-1} , then $X^T A_i X$ will be almost diagonal. Also in the case that orthogonal eigenvectors are desired for clustered roots, MATDAG has advantages over other routines [1], [2].

Usage

The calling sequence is:

```
CALL MATDAG (A,X,N,YR,NITER,NITT,ID)
```

where

A is the input matrix, a doubly-dimensioned, double precision array which upon return from MATDAG has as diagonal elements the eigenvalues of A.

- X is the doubly-dimensioned, double precision output array which returns the eigenvectors of A in its columns with the i^{th} column corresponding to the i^{th} diagonal element (eigenvalue) of the A array.
- N is the integer order of A.
- YR is the user's positive single precision convergence criterion. The input matrix will be made similar to one whose maximum off-diagonal element will be less in absolute value than YR. A suggested YR is 10^{-16} times the expected greatest eigenvalue.
- NITER is the integer output number of iterations (full sweeps) used.
- NITT is the maximum integer input number of iterations allowed by the program. A suggested value is 10.
- ID refers to the adjustable dimensions of A and X, i.e. they are dimensioned A(ID,ID),X(ID,ID).

Coding Information

MATDAG is a double precision Fortran IV subprogram which may be easily converted to single precision. NITER will usually be about 5 or 6. If NITER=NITT, it should be assumed that the matrix did not converge with respect to YR.

Restrictions

MATDAG solves only systems of order greater than one. Overflows and underflows sometimes occur and can sometimes be prevented by scaling.

Timing

See [3]. Also it should be remarked that HDIAG [1] has proven to be consistently faster but more involved than MATDAG. If there are several close or equal eigenvalues, the Jacobi method converges at a faster rate [7]. For the large majority of matrices, the Givens-Householder method should be preferred [1], [2], [3].

Accuracy

See [1], [3], [4], and [8].

Method

See [5]. This should be compared with HDIAG [1], [6].

References

- [1] Funderlic, R. E. (1965): Rev. 7311, Computing Reviews 6, 2, p. 120.
- [2] Elson, Mark, and Funderlic, R. E. (1964): SHARE 3202 BIGMAT, also this compendium.
- [3] Funderlic, R. E. (1964): "Eigensystems of Real Symmetric Matrices," KDP-320.
- [4] White, Paul A., and Brown, Robert R. (1964): "A comparison of methods for computing the eigenvalues and eigenvectors of a real symmetric matrix," Math. Comp. 18, 87, pp. 457-463.
- [5] Corbato, F. J. (1963): "On the coding of Jacobi's method for computing eigenvalues and eigenvectors of real symmetric matrices," J. ACM 10, p. 123.

- [6] Corbato, F. J. (1959): SHARE 705.
- [7] Shroder, Gerhard (1964): "Über die Konvergenz einiger Jacobi-Verfahren zur Bestimmung der Eigenwerte Symmetrischer Matrizen," Forschungsberichte des Landes Nordrhein-Westfalen, Nr. 1291, Westdeutscher Verlag, Opladen.
- [8] Wilkinson, J. H. (1965): The Algebraic Eigenvalue Problem, Oxford.

Author

Anon., The editor was told that the original form of MATDAG was coded by someone at the University of Rochester.

```

      SUBROUTINE MATDAG(B,D,N,YR,NITER,NITT,ID)
C PROGRAM AUTHOR ANJN,U.ROCHESTER
C
C CTC GRD PROGRAM NO. 9076
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION B(ID,1),D(ID,1)
      REAL*4 YR
      NITER = 0
      DO2I=1,N
      DO2J=1,N
      IF(I.NE.J) GO TO 1
      D(I,I)=1.0
      GOTO2
1 D(I,J)=0.0
2 CONTINUE
      NN=N-1
3 DO3K=1,NN
      MO=K+1
      DO3M=MO,N
      IF(DABS(B(K,M)).LE.YR) GO TO 9
      BY=(B(K,K)-B(M,M))/(2.0*B(K,M))
      IF (BY.GE.C.) GO TO 4
      SIG=-1.0
      GOTO5
4 SIG=1.0
5 COT=RY+(SIG*DSQRT(BY*BY+1.0))
      S=SIG/DSQRT(COT*COT+1.0)
      C=S*COT
6 DO7I=1,N
      BKI=B(K,I)
      BMI=B(M,I)
      B(K,I)=C*BKI+S*BMI
7 B(M,I)=-S*BKI+C*BMI
      DO8I=1,N
      BIK=B(I,K)
      BIM=B(I,M)
      DIK=D(I,K)
      DIM=D(I,M)
      B(I,K)=C*BIK+S*BIM
      B(I,M)=-S*BIK+C*BIM
      D(I,K)=C*DIK+S*DIM
8 D(I,M)=-S*DIK+C*DIM
9 CONTINUE
      BMAX = 0.
      DO10I=1,NN
      IO=I+1.
      DO10J=IO,N
      BM=DABS(B(I,J))
      IF (BMAX.GE.BM) GO TO 10
      BMAX=BM

```

```

MTDGG0001
MTDGG0002
MTDGG0003
MTDGG0004
MTDGG0005
MTDGG0006
MTDGG0007
MTDGG0008
MTDGG0009
MTDGG0010
MTDGG0011
MTDGG0012
MTDGG0013
MTDGG0014
MTDGG0015
MTDGG0016
MTDGG0017
MTDGG0018
MTDGG0019
MTDGG0020
MTDGG0021
MTDGG0022
MTDGG0023
MTDGG0024
MTDGG0025
MTDGG0026
MTDGG0027
MTDGG0028
MTDGG0029
MTDGG0030
MTDGG0031
MTDGG0032
MTDGG0033
MTDGG0034
MTDGG0035
MTDGG0036
MTDGG0037
MTDGG0038
MTDGG0039
MTDGG0040
MTDGG0041
MTDGG0042
MTDGG0043
MTDGG0044
MTDGG0045
MTDGG0046
MTDGG0047
MTDGG0048
MTDGG0049
MTDGG0050

```

```
10 CONTINUE
   IF (BMAX.LE.YR) GO TO 11
   NITER = NITER + 1
   IF (NITER .LT. NITT) GO TO 3
11 RETURN
   END
```

```
MTDG0051
MTDG0052
MTDG0053
MTDG0054
MTDG0055
MTDG0056
```

Identification

The cyclic Jacobi method for eigensystems of Hermitian matrices.

Purpose

JACØBH calculates the eigenvalues and orthogonal eigenvectors of Hermitian matrices by the use of the cyclic Jacobi method.

Usage

The calling sequence is:

```
CALL JACØBH(A,X,N,M,ICONV)
```

where

- A is a double precision complex Hermitian matrix.
The eigenvalues of A will be stored in the real parts of the diagonal elements of A upon return from JACØBH.
- X is a double precision complex unitary matrix which returns the eigenvectors in its columns with the i^{th} column corresponding to the eigenvalue located in the i^{th} diagonal position in the A array.
- M is the integer first dimension of A and X in the calling program.
- N is the integer order of A.
- ICONV is a logical variable which will have the value .TRUE. if A converged to a diagonal matrix and .FALSE. otherwise.

Coding Information

JACØBH is a double precision Fortran IV subprogram which may be easily converted to single precision. A and X should appear in a COMPLEX*16 type statement and ICONV in a LOGICAL type statement in the calling program. If the user does not desire the eigenvectors of A, he can remove X from the argument list and complex type statement, remove the four cards at the beginning of the 10 loop and also the four cards that make up the 4 loop. JACØBH usually requires about five sweeps through the matrix before convergence and no matrices have been tested which did not converge in the maximum number of sweeps allowed (eight).

Restrictions

The eigenvalues of A are not necessarily in any algebraic order on the diagonal of A and hence no convenient way is available to calculate only a few of them as in the Givens-Householder method [3]. If overflows or underflows occur, scaling A so that its largest element is in modulus about one will probably eliminate the difficulty. The matrix A is assumed to be of order greater than one.

Timing

A single precision version of JACØBH was tested and timed on an IBM 7090 machine. Times vary not only with order but also with individual matrices. This latter point is much more important for the Jacobi method than for the Givens-Householder method

which is in general faster [4]. In fact, the QR method followed by Wilkinson's method for vectors is in general faster (see ALLMAT in this compendium).

Since the method is based on increasing the sum of the squares of the moduli of the diagonal elements, JACØBH runs faster for matrices which are "almost diagonal." Also, if there are several close or equal eigenvalues, the Jacobi method converges at a faster rate [5]. Pertinent to the execution time is that the number of operations depends on N^3 . The following table gives some examples of the timing we have experience on the 7090 for random Hermitian matrices using single precision. Fixed dimensions were used in these tests.

ORDER	EIGENVALUES ONLY	EIGENVALUES AND EIGENVECTORS
10	6.8 sec.	9.8 sec.
20	60.5 sec.	84.3 sec.
30	209.0 sec.	309.0 sec.

Accuracy

The program will usually determine eigenvalues to an accuracy of about $10^{-15} \rho(A)$, where $\rho(A)$ is the spectral radius of A. Hence, the greater $\rho(A)/\min |\lambda_i|$ the fewer significant digits the smaller eigenvalues may have. In most cases JACØBH yields roots and vectors accurate to about fourteen significant digits.

Method

The input matrix A is reduced to a diagonal matrix by a sequence of simple unitary matrices $X_j^H = X_j^{-1}$, [2]. The sequence $A_j = X_j^H \dots X_1^H A X_1 \dots X_j$ converges to a diagonal matrix. The eigenvalues of A_j are the same as the eigenvalues of A so that when A_j is effectively diagonal, the diagonal elements are essentially the eigenvalues of A . At this stage if $X = X_1 X_2 \dots X_j$, we have $X^H A X = \Lambda$, the diagonal matrix of eigenvalues, and X has as columns the eigenvectors of A . The convergence criterion is chosen as $\|A\|_E$, the Euclidean length of A , times 10^{-16} , i.e., the process is stopped when all the off-diagonal elements of A have magnitude not greater than $\|A\|_E 10^{-16}$. A more lenient convergence criterion would probably not be worthwhile because of the quadratic convergence character of the Jacobi method [6]. Each X_i annihilates two off-diagonal elements of A but may introduce non-zero elements where zero elements previously appeared. The elements are annihilated in a cyclic fashion which is probably faster than searching for the largest off-diagonal element as in the classical Jacobi method. However, the search time need depend only linearly on the order of the matrix [1], [4] and JACØBH's speed could be improved.

References

- [1] Corbato, F.J. (1963): "On the coding of Jacobi's method for computing eigenvalues and eigenvectors of real symmetric matrices," J. ACM, 10, 123-125.

- [2] Householder, A. S. (1953): Principles of Numerical Analysis, McGraw-Hill, pp 160-162.
- [3] Elson, M., and Funderlic, R. E. (1965): "Real Symmetric Matrix Eigensystem Solver," SHARE 3202-01.
- [4] Funderlic, R. E. (1965): Rev. 7311, Computing Reviews, 6, 2, 120.
- [5] Schröder, Gerhard (1964): "Über die Konvergenz einiger Jacobi-Verfahren zur Bestimmung der Eigenwerte symmetrischer Matrizen," Forschungsberichte des Landes Nordrhein-Westfalen, Nr. 1291, Westdeutscher Verlag, Opladen.
- [6] Wilkinson, J. H. (1960): Advanced numerical analysis notes, summer session, U. of Michigan, p. 115.
- [7] Wilkinson, J. H. (1965): The Algebraic Eigenvalue Problem, Clarendon.

Author

R. E. Funderlic, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE JACDBH(A,X,NN,M,ICONV)
C  PROGRAM AUTHOR  R.E. FUNDERLIC
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9077
      IMPLICIT REAL*8 (A-H,O-Z)
      COMPLEX*16 A(M,M),X(M,M),GAMMA,SIGMA,AINU,AJNU,ANUI,ANUJ,XNU1,XNUJ
      LOGICAL ICONV
      SQR2D2 = DSQRT(2.D0)/2.
      N=NN
      NM1=N-1
      UPER=0.
      RELAI = A(N,N)
      DIAG=RELAI**2
      DO 2 I=1,NM1
        IP1=I+1
        DO 1 J=IP1,N
          RELAI = A(I,J)
          AIMG = (0.,-1.)*A(I,J)
1        UPER=UPER+RELAI**2+AIMG**2
          RELAI = A(I,I)
2      DIAG=DIAG+RELAI**2
      ANORM = DSQRT(DIAG+2.*UPER)*1.D-16
      DO 4 I=1,N
        DO 3 J=1,N
3      X(I,J)=0.
4      X(I,I)=1.
      DO 12 ITER=1,8
        ICONV=.TRUE.
        DO 11 I=1,NM1
          IP1=I+1
          DO 11 J=IP1,N
            BETA=CDABS(A(I,J))
            IF(BETA.LE.ANORM)GO TO 11
            ICONV=.FALSE.
            RELAI = A(I,I)
            RELAJ = A(J,J)
            TAU = RELAI-RELAJ
            ZETA=DSIGN(2.*BETA,TAU)
            SQRZT=DSQRT(TAU**2+ZETA**2)
            COSTHE=DSQRT((1.+DABS(TAU)/SQRZT)/2.)
            SIN THE=ZETA/(2.*SQRZT*COSTHE)
            RELAI = A(I,J)
            COSPHI=RELAJ/BETA
            AIMG = (0.,-1.)*A(I,J)
            HSINP=AIMG/(2.*BETA)
            TEMP=DSQRT((1.+DABS(COSPHI))/2.)
            IF(COSPHI)5,7,6
5          SINPD2=TEMP
          JACH0001
          JACH0002
          JACH0003
          JACH0004
          JACH0005
          JACH0006
          JACH0007
          JACH0008
          JACH0009
          JACH0010
          JACH0011
          JACH0012
          JACH0013
          JACH0014
          JACH0015
          JACH0016
          JACH0017
          JACH0018
          JACH0019
          JACH0020
          JACH0021
          JACH0022
          JACH0023
          JACH0024
          JACH0025
          JACH0026
          JACH0027
          JACH0028
          JACH0029
          JACH0030
          JACH0031
          JACH0032
          JACH0033
          JACH0034
          JACH0035
          JACH0036
          JACH0037
          JACH0038
          JACH0039
          JACH0040
          JACH0041
          JACH0042
          JACH0043
          JACH0044
          JACH0045
          JACH0046
          JACH0047
          JACH0048
          JACH0049
          JACH0050

```

COSPD2=HSINP/SINPD2	JACH0051
GO TO 8	JACH0052
6 COSPD2=DSIGN(TEMP,HSINP)	JACH0053
SINPD2=HSINP/COSPD2	JACH0054
GO TO 8	JACH0055
7 COSPD2 = DSIGN(SQR2D2,HSINP)	JACH0056
SINPD2 = SQR2D2	JACH0057
8 SIGMA=DCMPLX(COSPD2*SIN THE,SINPD2*SIN THE)	JACH0058
GAMMA=DCMPLX(COSPD2*COSTHE,-SINPD2*COSTHE)	JACH0059
DO 9 NU=1,N	JACH0060
AINU=A(I,NU)	JACH0061
AJNU=A(J,NU)	JACH0062
A(I,NU)=AINU*GAMMA+AJNU*SIGMA	JACH0063
9 A(J,NU)=-AINU*DCONJG(SIGMA)+AJNU*DCONJG(GAMMA)	JACH0064
DO 10 NU=1,N	JACH0065
XNUI=X(NU,I)	JACH0066
XNUJ=X(NU,J)	JACH0067
X(NU,I)=XNUI*DCONJG(GAMMA)+XNUJ*DCONJG(SIGMA)	JACH0068
X(NU,J)=-XNUI*SIGMA+XNUJ*GAMMA	JACH0069
ANUI=A(NU,I)	JACH0070
ANUJ=A(NU,J)	JACH0071
A(NU,I)=ANUI*DCONJG(GAMMA)+ANUJ*DCONJG(SIGMA)	JACH0072
10 A(NU,J)=-ANUI*SIGMA+ANUJ*GAMMA	JACH0073
11 CONTINUE	JACH0074
12 IF(ICONV)GO TO 13	JACH0075
ICONV=.FALSE.	JACH0076
13 RETURN	JACH0077
END	JACH0078

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Arbitrary real matrix eigensystem solver.

Purpose

The subroutine RILMAT (main entry) uses the QR algorithm to calculate the eigenvalues and eigenvectors of arbitrary real matrices. A subordinate entry, EIGS, allows the user to calculate eigenvalues only.

Usage

The calling sequences are:

```
CALL RILMAT(A,H,R00TR,R00TI,N,ID,IPRNT,NCAL)
```

or

```
CALL EIGS(A,N,R00TR,R00TI,ID,IPRNT,NCAL)
```

where

A is the double precision input matrix, a doubly-dimensioned array. On return from RILMAT, A contains the real parts of the computed eigenvectors.

H is a doubly-dimensioned double precision array to which RILMAT returns the imaginary parts of the computed eigenvectors.

R00TR is a singly-dimensioned double precision array to which RILMAT returns the real parts of the computed eigenvalues with the i^{th} eigenvalue corresponding to the eigenvector in the i^{th} column of A and of H.

R00TI is a singly-dimensioned double precision array to which RILMAT returns the imaginary parts of the computed eigenvalues.

N is the integer order of the matrix A.

ID is the integer first dimension of A in the calling program.

IPRNT is an integer which controls the print out of eigenvalues. If IPRNT $\neq$ 0, then the eigenvalues and the number of iterations required in calculating each eigenvalue will be output. If IPRNT = 0, there will be no printed output. Also see "Restrictions."

NCAL is an integer which upon return from RILMAT contains the number of eigenvalues with corresponding number of eigenvectors which RILMAT calculated. Also see "Restrictions."

Coding Information

A and H must be dimensioned at least NxN and R00TR and R00TI by at least N in the calling program. Also, the variables A, H, R00TR, and R00TI must appear in a double precision (REAL*8) type statement in the calling program.

RILMAT makes use of two temporary double precision arrays of dimension 30x30 and several singly-dimensioned arrays. If eigen-systems of matrices of order greater than 30 are desired, the dimensions in RILMAT should be altered accordingly. The only references to arrays with fixed dimensions, i.e. 30, occur at the

beginning of RILMAT.

If only eigenvalues are desired and the user calls EIGS, the dimensions of the arrays HLR, HLI, VECTR, VECTI, RTR, MULTI, and INTH may be changed to "1". IPRNT $\neq$ 0 yields printout on data set 6.

Restrictions

If IPRNT $\neq$ 0, the eigenvalues are printed out. However, the eigenvalues are not printed out in the same order in which they are stored in the arrays, R00TR and R00TI. Therefore, the i^{th} eigenvalue as printed out, using the IPRNT variable, in RILMAT will not correspond to the eigenvector stored in the i^{th} column of A and H.

Fifty iterations per eigenvalue are allowed. In the case of non-convergence, RILMAT will return a value for NCAL less than N and will calculate NCAL eigenvectors. There are known cases of matrices for which the QR method fails. See ALLMAT, "Restrictions."

Equal computed eigenvalues return identical corresponding eigenvectors even though linearly independent vectors may exist.

Timing

On the 360/75 (IBM Fortran IV Compiler, OPT=2), the time for calculation of eigenvalues and eigenvectors of a matrix of order 50 was 16.84 seconds. The matrix was obtained using a pseudo random number generator.

Accuracy

In most cases RILMAT has yielded eigenvalues and eigenvectors accurate to about 12 digits. However, in some cases run on the IBM 7090 the eigenvalues and eigenvectors were accurate to only 8 digits because of underflows. This difficulty was not present on the IBM 360.

Method

The subroutine RILMAT transforms the matrix to upper Hessenberg form using a series of similarity transformations. The calculation of eigenvalues is done by the subroutine EIGQR, which is called by RILMAT. The QR transformation is performed in the subroutine DQRT, which is called by EIGQR. The eigenvectors are calculated in RILMAT using the Wielandt inverse power method. See ALLMAT, "Method."

Acknowledgment

Two subroutines, EIGQR and DQRT, called in RILMAT, are modifications of SHARE 3006.01. The authors are P. Imiad Fawzi and J. E. Van Ness, Northwestern University.

SHARE 3006.01 was programmed prior to the publication of several suggested improvements [1], [3], [4]; also see reviews in the SSD's of SHARE 3006. We hope to incorporate several improvements for the next edition of this compendium.

References

- [1] Wilkinson, J. H. (1965): The Algebraic Eigenvalue Problem, Oxford.

- [2] Householder, A. S. (1964): The Theory of Matrices in Numerical Analysis, Blaisdell.
- [3] Ortega, James (1967): "The Givens-Householder method for symmetric matrices," pp. 94-115, Mathematical Methods for Digital Computers, vol. 2, eds. Ralston & Wilf, Wiley.
- [4] Parlett, Beresford (1967): "The LU and QR algorithms," pp. 116-130, in same book as [3].

Authors

Miss Jane Cope, a summer employee of the Computing Technology Center, and Miss Jane Dunphey, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

SUBROUTINE RILMAT(A,H,ROOTR,ROOTI,M,ID,IP,NCAL)      RILM0001
C PROGRAM AUTHORS MISS JANE COPE AND MISS JANE DUNPHEY RILM0002
C COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV., RILM0003
C GAK RIDGE, TENN. RILM0004
C CTC OFD PROGRAM NO. 9078 RILM0005
C RILM0006
DOUBLE PRECISION A(ID,ID), H(ID,ID),HLK(30,30),HLI(30,30),ROOTR(ID,RILM0007
1 ),ROOTI(ID),VECTR(30),VECTI(30),RTR(30),MULTR(30),MULTI(30),ABIG,RILM0008
1 ABS1,RI,SUM,VTI,BIG,TEMPI,VTR,EPS,TEMPR,TEMP,RR RILM0009
LOGICALINTH(30),TWICE RILM0010
INTEGRINT(30),R,RP1,PP2 RILM0011
IPET = 1 RILM11.1
GO TO 49 RILM11.2
ENTRY EIGS (A,M,ROOTR,ROOTI,ID,IP,NCAL) RILM11.3
IPET = 0 RILM11.4
49 CONTINUE RILM11.5
N=M RILM0012
IF(N-2)1,1,3 RILM0013
1 DO2I=1,N RILM0014
DO2J=1,N RILM0015
2 H(I,J)=A(I,J) RILM0016
CALL FIGOR (A, M, ROOTR, ROOTI, IP, ID, NCAL) RILM0017
GO TO 16 RILM0018
C RILM0019
C REDUCE MATRIX A TO UPPER HESSENBERG FORM RILM0020
C RILM0021
3 NM2=N-2 RILM0022
DO14R=1,NM2 RILM0023
RP1=P+1 RILM0024
RP2=R+2 RILM0025
ABIG=0.00 RILM0026
INT(R)=RP1 RILM0027
DO4I=RP1,N RILM0028
ABS1=DABS(A(I,R)) RILM0029
IF(ABS1.LE.ABIG)GOTO4 RILM0030
INT(R)=I RILM0031
ABIG=ABS1 RILM0032
4 CONTINUE RILM0033
INTER=INT(R) RILM0034
IF(ABIG.EQ.0.00)GOTO14 RILM0035
IF(INTER.EQ.RP1)GOTO7 RILM0036
DO5I=R,N RILM0037
TEMP=A(RP1,I) RILM0038
A(RP1,I)=A(INTER,I) RILM0039
5 A(INTER,I)=TEMP RILM0040
DO6I=1,N RILM0041
TEMP=A(I,RP1) RILM0042
A(I,RP1)=A(I,INTER) RILM0043
6 A(I,INTER)=TEMP RILM0044
7 DO8I=RP2,N RILM0045

```

```

      MULTR(I)=A(I,R)/A(RP1,R)
8  A(I,R)=MULTR(I)
      DO10I=1,RP1
      TEMP=0.D0
      DO9J=PP2,N
9  TEMP=TEMP+A(I,J)*MULTR(J)
10 A(I,RP1)=A(I,RP1)+TEMP
      DO12I=RP2,N
      TEMP=0.D0
      DO11J=RP2,N
11 TEMP=TEMP+A(I,J)*MULTR(J)
12 A(I,RP1)=A(I,RP1)+TEMP-MULTR(I)*A(RP1,RP1)
      DO13I=RP2,N
      DO13J=RP2,N
13 A(I,J)=A(I,J)-MULTR(I)*A(RP1,J)
14 CONTINUE
C  SAVE A IN H
      DO15I=1,N
      DO15J=1,N
15 H(I,J)=A(I,J)
C
C  CALCULATE EIGENVALUES
C
      CALL FIGQR (A, M, ROOTR, ROOTI, IP, ID, NCAL)
16 IF ( NCAL .EQ. 0 .OR. IRET .EQ. 0) GO TO 48
C
C  CALCULATE VECTORS
C
      EPS=0.D0
      DO 17 I=1,N
17 EPS=EPS+DABS(H(1,I))
      DO 19 I=2,N
      SUM=0.D0
      IM1=I-1
      DO 18J=IM1,N
18 SUM=SUM+DABS(A(I,J))
19 IF(SUM.GT.EPS)EPS=SUM
      EPS=DSQRT(DBLE(FLOAT(N)))*EPS*1.D-19
      IF(EPS.EQ.0.D0)EPS=1.D-19
      N=M
      NM1=N-1
      ITAG1=0
      ITAG2=0
      NMT = NCAL
20 DO 44 L = 1,NCAL
      LMT=L-ITAG1
      DO22I=1,N
      DO21J=1,N
      HLR(I,J)=H(I,J)
21 HLI(I,J)=0.D0

```

```

RILM0046
RILM0047
RILM0048
RILM0049
RILM0050
RILM0051
RILM0052
RILM0053
RILM0054
RILM0055
RILM0056
RILM0057
RILM0058
RILM0059
RILM0060
RILM0061
RILM0062
RILM0063
RILM0064
RILM0065
RILM0066
RILM0067
RILM0068
RILM0069
RILM0070
RILM0071
RILM0072
RILM0073
RILM0074
RILM0075
RILM0076
RILM0077
RILM0078
RILM0079
RILM0080
RILM0081
RILM0082
RILM0083
RILM0084
RILM0085
RILM0086
RILM0087
RILM0088
RILM0089
RILM0090
RILM0091
RILM0092
RILM0093
RILM0094
RILM0095

```

```

HLR(I,I)=HLR(I,I)-ROOTR(L)
22 HLI(I,I)=-1.D0*ROOTI(L)
DO26I=1,NM1
  MULTR(I)=0.D0
  MULTI(I)=0.D0
  INTH(I)=.FALSE.
  IP1=I+1
  IF((HLR(I+1,I))**2+(HLI(I+1,I))**2.LE.(HLR(I,I))**2+(HLI(I,I))**2)
1GU TO 24
  INTH(I)=.TRUE.
  DO23J=1,N
    TEMPR=HLR(I+1,J)
    TEMPI=HLI(I+1,J)
    HLR(I+1,J)=HLR(I,J)
    HLI(I+1,J)=HLI(I,J)
    HLR(I,J)=TEMPR
23 HLI(I,J)=TEMPI
24 IF(HLR(I,I).EQ.0.D0.AND.HLI(I,I).EQ.0.D0)GOTO26
  MULTR(I)=- (HLR(I+1,I)*HLR(I,I)+HLI(I+1,I)*HLI(I,I))/(HLR(I,I)
1**2+HLI(I,I)**2)
  MULTI(I)=(HLR(I+1,I)*HLI(I,I)-HLI(I+1,I)*HLR(I,I))/(HLR(I,I)
1**2+HLI(I,I)**2)
  DO25J=IP1,N
    TEMPR=HLR(I+1,J)+MULTR(I)*HLR(I,J)-MULTI(I)*HLI(I,J)
    TEMPI=HLI(I+1,J)+MULTR(I)*HLI(I,J)+MULTI(I)*HLR(I,J)
    HLR(I+1,J)=TEMPR
25 HLI(I+1,J)=TEMPI
26 CONTINUE
DO27I=1,N
  VECTR(I)=1.D0
27 VECTI(I)=0.D0
  TWICE=.FALSE.
28 IF(HLR(N,N).EQ.0.D0.AND.HLI(N,N).EQ.0.D0)HLR(N,N)=EPS
  TEMPR=(VECTR(N)*HLR(N,N)+VECTI(N)*HLI(N,N))/(HLR(N,N)**2
1+HLI(N,N)**2)
  TEMPI=(VECTI(N)*HLR(N,N)-VECTR(N)*HLI(N,N))/(HLR(N,N)**
12+HLI(N,N)**2)
  VECTR(N)=TEMPR
  VECTI(N)=TEMPI
DO30I=1,NM1
  K=N-I
DO29J=K,NM1
  TEMPR=VECTR(K)-HLR(K,J+1)*VECTR(J+1)+HLI(K,J+1)*VECTI(J+1)
  TEMPI=VECTI(K)-HLR(K,J+1)*VECTI(J+1)-HLI(K,J+1)*VECTR(J+1)
  VECTR(K)=TEMPR
29 VECTI(K)=TEMPI
  IF(HLR(K,K).EQ.0.D0.AND.HLI(K,K).EQ.0.D0)HLR(K,K)=EPS
  TEMPR=(VECTR(K)*HLR(K,K)+VECTI(K)*HLI(K,K))/
1(HLR(K,K)**2+HLI(K,K)**2)
  TEMPI=(VECTI(K)*HLR(K,K)-VECTR(K)*HLI(K,K))/

```

```

RILM0086
RILM0097
RILM0098
RILM0099
RILM0100
RILM0101
RILM0102
RILM0103
RILM0104
RILM0105
RILM0106
RILM0107
RILM0108
RILM0109
RILM0110
RILM0111
RILM0112
RILM0113
RILM0114
RILM0115
RILM0116
RILM0117
RILM0118
RILM0119
RILM0120
RILM0121
RILM0122
RILM0123
RILM0124
RILM0125
RILM0126
RILM0127
RILM0128
RILM0129
RILM0130
RILM0131
RILM0132
RILM0133
RILM0134
RILM0135
RILM0136
RILM0137
RILM0138
RILM0139
RILM0140
RILM0141
RILM0142
RILM0143
RILM0144
RILM0145

```

1(HLR(K,K)**2+HLI(K,K)**2)	RILMO146
VECTR(K)=TEMPR	RILMO147
30 VECTI(K)=TEMPI	RILMO148
BIG=0.00	RILMO149
DO31I=1,N	RILMO150
SUM=DABS(VECTR(I))+DABS(VECTI(I))	RILMO151
31 IF(SUM.GT.BIG)BIG=SUM	RILMO152
DO32I=1,N	RILMO153
VECTR(I)=VECTR(I)/BIG	RILMO154
32 VECTI(I)=VECTI(I)/BIG	RILMO155
IF(TWICE)GOTO35	RILMO156
DO34I=1,NMI	RILMO157
IF(.NOT.INTH(I))GOTO33	RILMO158
TEMPR=VECTR(I)	RILMO159
TEMPI=VECTI(I)	RILMO160
VECTR(I)=VECTR(I+1)	RILMO161
VECTI(I)=VECTI(I+1)	RILMO162
VECTR(I+1)=TEMPR	RILMO163
VECTI(I+1)=TEMPI	RILMO164
33 VTR=VECTR(I+1)+MULTR(I)*VECTR(I)-MULTI(I)*VECTI(I)	RILMO165
VTI=VECTI(I)+MULTI(I)*VECTR(I)+MULTR(I)*VECTI(I)	RILMO166
VECTR(I+1)=VTR	RILMO167
34 VECTI(I+1)=VTI	RILMO168
TWICE=.TRUE.	RILMO169
GOTO28	RILMO170
35 IF(N.EQ.2)GOTO38	RILMO171
NM2=N-2	RILMO172
DO37I=1,NM2	RILMO173
N1I=N-1-I	RILMO174
N1I=N-1+1	RILMO175
DO36J=N1I,N	RILMO176
VECTR(J)=H(J,N1I)*VECTR(N1I+1)+VECTR(J)	RILMO177
36 VECTI(J)=H(J,N1I)*VECTI(N1I+1)+VECTI(J)	RILMO178
INDEX=INT(N1I)	RILMO179
TEMPR=VECTR(N1I+1)	RILMO180
TEMPI=VECTI(N1I+1)	RILMO181
VECTR(N1I+1)=VECTR(INDEX)	RILMO182
VECTI(N1I+1)=VECTI(INDEX)	RILMO183
VECTR(INDEX)=TEMPR	RILMO184
37 VECTI(INDEX)=TEMPI	RILMO185
38 DO39I=1,N	RILMO186
IF(VECTI(I).NE.0.00)GO TO 40	RILMO187
39 CONTINUE	RILMO188
GOTO42	RILMO189
40 ITAG1=ITAG1+1	RILMO190
IF(VECTI(I).LT.0.00)GOTO44	RILMO191
ITAG2=ITAG2+1	RILMO192
NMT = NCAL-2*ITAG2	RILMO193
ITAG1 TELLS HOW MANY COMPLEX VECTORS, ITAG2 HOW MANY STORED IN A	RILMO194
DO41I=1,N	RILMO195


```

      A(I,NMT+1)=VECTR(I)
      A(I,NMT+2)=VECTI(I)
41  CONTINUE
      RTR(NMT+1)=ROOTR(L)
      RTR(NMT+2)=ROOTI(L)
      GOTQ44
42  DO43I=1,N
      A(I,LMT)=VECTR(I)
43  CONTINUE
      RTR(LMT)=ROOTR(L)
44  CONTINUE
C    PUT REAL PARTS OF EIGENVECTORS IN A, IMAGINARY PARTS IN H
      DO 45 I=1,N
      DO 45JJ=1,NMT
      ROOTR(JJ)=RTR(JJ)
      ROOTI(JJ)=0.00
45  H(I,JJ)=0.00
      IF (NMT.EQ. NCAL) GO TO 48
      NMTP1=NMT+1
      NM1 = NCAL - 1
      DO46I=1,N
      DO46J=NMTP1,NM1,2
      ROOTI(J)=-RTR(J+1)
      ROOTI(J+1)=RTR(J+1)
      H(I,J)=-A(I,J+1)
46  H(I,J+1)=A(I,J+1)
      DO 47 I=1,N
      DO 47 J=NMTP1,NM1,2
      ROOTR(J)=RTR(J)
      ROOTR(J+1)=ROOTR(J)
47  A(I,J+1)=A(I,J)
48  RETURN
      END
C
C    SUBROUTINE FIGQR (A,M,ROOTR,ROOTI,IPRNT,ID, NCAL)
C    PROGRAM AUTHORS  MISS JANE COPE AND MISS JANE DUNPHEY
C    COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C    OAK RIDGE, TENN. MODIFICATION OF SHARE 3C06.01.
C
C    PROGRAM TO CALL DOUBLE PRECISION QR TRANSFORMATION
C
C    DOUBLE PRECISION A(ID,ID)
C    DOUBLE PRECISION ROOTR(ID),ROOTI(ID),AA,B,C,DD,E,F,G,H,R,SIG,X,S,
1  ZERO,XNN,XN2,Z1,Z2,VQ,SQ,D
      N=M
      NCAL = N
      IF(IPRNT) 1,2,1
1  WRITE (6,100)
2  ZERO=0.00
      JJ=1

```

```

RILM0196
RILM0197
RILM0198
RILM0199
RILM0200
RILM0201
RILM0202
RILM0203
RILM0204
RILM0205
RILM0206
RILM0207
RILM0208
RILM0209
RILM0210
RILM0211
RILM0212
RILM0213
RILM0214
RILM0215
RILM0216
RILM0217
RILM0218
RILM0219
RILM0220
RILM0221
RILM0222
RILM0223
RILM0224
RILM0225
RILM0226
RILM0227
RILM0228
RILM0229
RILM 230
RILM 231
RILM 232
RILM 233
RILM 234
RILM 235
RILM 236
RILM 237
RILM 238
RILM 239
RILM 240
RILM 241
RILM 242
RILM 243
RILM 244
RILM 245

```

3	XNN=0.00	RILM 246
	XN2=0.00	RILM 247
	AA=0.00	RILM 248
	B=0.00	RILM 249
	C=0.00	RILM 250
	DD=0.00	RILM 251
	R=0.00	RILM 252
	SIG=0.00	RILM 253
	ITER=0	RILM 254
4	IF (N-2) 5,9,10	RILM 255
5	IF (IPRNT) 6,7,6	RILM 256
6	WRITE (6,200)A(1,1)	RILM 257
7	ROOTR(1)=A(1,1)	RILM 258
	ROOTI(1)=0.00	RILM 259
8	RETURN	RILM 260
9	JJ=-1	RILM 261
10	X=(A(N-1,N-1)-A(N,N))*2	RILM 262
	S=4.00*A(N,N-1)*A(N-1,N)	RILM 263
	ITER=ITER+1	RILM 264
	IF(X.EQ.0.00.OR.DABS(S/X).GT.1.0-16)GO TO 15	RILM 265
11	IF(DABS(A(N-1,N-1))-DABS(A(N,N)))13,13,12	RILM 266
12	E=A(N-1,N-1)	RILM 267
	G=A(N,N)	RILM 268
	GO TO 14	RILM 269
13	G=A(N-1,N-1)	RILM 270
	E=A(N,N)	RILM 271
14	F=0.00	RILM 272
	H=0.00	RILM 273
	GO TO 20	RILM 274
15	S=X+S	RILM 275
	X=A(N-1,N-1)+A(N,N)	RILM 276
	IF(S)19,16,16	RILM 277
16	SQ=DSQRT(S)	RILM 278
	F=0.00	RILM 279
	H=0.00	RILM 280
	IF(X)17,17,18	RILM 281
17	E=(X-SQ)/2.000	RILM 282
	G=(X+SQ)/2.000	RILM 283
	GO TO 20	RILM 284
18	G=(X-SQ)/2.000	RILM 285
	E=(X+SQ)/2.000	RILM 286
	GO TO 20	RILM 287
19	F=DSQRT(-S)/2.000	RILM 288
	E=X/2.000	RILM 289
	G=F	RILM 290
	H=-F	RILM 291
20	CONTINUE	RILM 292
	IF (JJ) 22,21,21	RILM 293
21	D=1.00-17*(DABS(G)+F)	RILM 294
	IF(DABS(A(N-1,N-2)).GT.D)GO TO 25	RILM 295

22 IF (IPRNT)23,24,23	RILM 296
23 WRITE (6,200)E,F,ITER	RILM 297
WRITE (6,200)G,H	RILM 298
24 RQCTR(N)=E	RILM 299
RCOTI(N)=F	RILM 300
RQCTR(N-1)=G	RILM 301
RCOTI(N-1)=H	RILM 302
N=N-2	RILM 303
IF(JJ)8,3,3	RILM 304
25 IF(DABS(A(N,N-1)).GT.1.(D-17*DARS(A(N,N)))GO TO 29	RILM 305
26 IF(IPRNT)27,28,27	RILM 306
27 WRITE(6,200)A(N,N),ZERO,ITER	RILM 307
28 RQCTR(N)=A(N,N)	RILM 308
RCOTI(N)=0.D0	RILM 309
N=N-1	RILM 310
GO TO 3	RILM 311
29 IF(DABS(DABS(XNN/A(N,N-1))-1.D0)-1.D-13)31,31,30	RILM 312
30 IF(DABS(DABS(XN2/A(N-1,N-2))-1.D0)-1.D-13)31,31,40	RILM 313
31 VQ=DARS(A(N,N-1))-DARS(A(N-1,N-2))	RILM 314
IF(ITER-15)45,32,35	RILM 315
32 IF(VQ)33,33,34	RILM 316
33 R=A(N-1,N-2)**2*1.D0	RILM 317
SIG=2.D0*A(N-1,N-2)	RILM 318
GO TO 50	RILM 319
34 R=A(N,N-1)**2	RILM 320
SIG=2.D0*A(N,N-1)	RILM 321
GO TO 50	RILM 322
35 IF(VQ)38,38,36	RILM 323
36 IF(IPRNT)37,24,37	RILM 324
37 WRITE(6,300)A(N-1,N-2)	RILM 325
GO TO 23	RILM 326
38 IF(IPRNT)39,28,39	RILM 327
39 WRITE(6,300)A(N,N-1)	RILM 328
GO TO 27	RILM 329
40 IF(ITER.LE.50)GO TO 42	RILM 330
NCAL = M - N	RILM 331
MNC = M - NCAL + 1	RILM 332
DO 41 K = 1,NCAL	RILM 333
II = MNC + K - 1	RILM 334
RQCTR(K) = RQCTR(II)	RILM 335
41 RCOTI(K) = RCOTI(II)	RILM 336
GO TO 8	RILM 337
42 IF(ITER.GT.5)GO TO 45	RILM 338
43 Z1=((E-AA)**2+(F-B)**2)/(E*E+F*F)	RILM 339
Z2=((G-C)**2+(H-DD)**2)/(G*G+H*H)	RILM 340
IF(Z1-0.25D0)44,44,47	RILM 341
44 IF(Z2-0.25D0)45,45,46	RILM 342
45 R=E*G-F*H	RILM 343
SIG=E+G	RILM 344
GO TO 50	RILM 345

46 R=E*E	RILM 346
SIG=E+E	RILM 347
GO TO 50	RILM 348
47 IF(Z2-0.2500)48,48,49	RILM 349
48 R=G*G	RILM 350
SIG=G+G	RILM 351
GO TO 50	RILM 352
49 R=0.00	RILM 353
SIG=C.00	RILM 354
50 XNN=A(N,N-1)	RILM 355
XN2=A(N-1,N-2)	RILM 356
CALL DQRT(A,N,R,SIG,D,ID)	RILM 357
AA=E	RILM 358
B=F	RILM 359
C=G	RILM 360
DD=H	RILM 361
GO TO 10	RILM 362
100 FORMAT(////1X,9HREAL PART,14X,14HIMAGINARY PART,26X,	RILM 363
113HTAKEN AS ZERO,6X,4HITER//)	RILM 364
200 FORMAT(1X,D23.16,3X,D23.16,26X,I3)	RILM 365
300 FORMAT(64X,D23.16)	RILM 366
END	RILM 367
SUBROUTINE DQRT(A,N,R,SIG,D,ID)	RILM 368
DOUBLE PRECISION A(ID,ID),PSI(2),G(3),SIG,DEN,XK,AL,	RILM 369
1C,D,E,R,GMAX,G1,G2	RILM 370
N1=N-1	RILM 371
IA=N-2	RILM 372
IP=IA	RILM 373
IF(N-3)41,5,1	RILM 374
1 DO4J=3,N1	RILM 375
J1=N-J	RILM 376
IF(DABS(A(J1+1,J1))-D)5,5,2	RILM 377
2 DEN=A(J1+1,J1+1)*(A(J1+1,J1+1)-SIG)+A(J1+1,J1+2)*A(J1+2,	RILM 378
1J1+1)+R	RILM 379
IF(DEN)3,4,3	RILM 380
3 IF(DABS(A(J1+1,J1)*A(J1+2,J1+1)*(DABS(A(J1+1,J1+1)+A(J1+2,	RILM 381
1J1+2)-SIG)+DABS(A(J1+3,J1+2)))/DEN)-D)5,5,4	RILM 382
4 IP=J1	RILM 383
5 DO6J=1,IP	RILM 384
J1=IP-J+1	RILM 385
IF(DABS(A(J1+1,J1))-D)7,7,6	RILM 386
6 IQ=J1	RILM 387
7 DO40I=IP,N1	RILM 388
IF(I-IP)9,8,9	RILM 389
8 G(1)=A(IP,IP)*(A(IP,IP)-SIG)+A(IP,IP+1)*A(IP+1,IP)+R	RILM 390
G(2)=A(IP+1,IP)*(A(IP,IP)+A(IP+1,IP+1)-SIG)	RILM 391
G(3)=A(IP+1,IP)*A(IP+2,IP+1)	RILM 392
A(IP+2,IP)=0.00	RILM 393
GO TO 12	RILM 394
9 G(1)=A(I,I-1)	RILM 395

G(2)=A(I+1,I-1)	RILM 396
IF(I-IA)10,10,11	RILM 397
10 G(3)=A(I+2,I-1)	RILM 398
GO TO 12	RILM 399
11 G(3)=0.00	RILM 400
12 CONTINUE	RILM 401
IF (G(1) .EQ. 0.00 .AND. G(2) .EQ. 0.00 .AND. G(3) .EQ. 0.00)	RILM 402
1 GO TO 19	RILM 403
GMAX = G(1)	RILM 404
IF (DABS(G(2)) .GT. DABS(GMAX)) GO TO 13	RILM 405
G1 = G(2)	RILM 406
GO TO 14	RILM 407
13 GMAX = G(2)	RILM 408
G1 = G(1)	RILM 409
14 IF (DABS(G(3)) .GT. DABS(GMAX)) GO TO 15	RILM 410
G2 = G(3)	RILM 411
GO TO 16	RILM 412
15 G2 = GMAX	RILM 413
GMAX = G(3)	RILM 414
16 CONTINUE	RILM 415
XK = DABS(GMAX)*DSQRT((G1/GMAX)**2 + (G2/GMAX)**2 + 1.00)	RILM 416
IF (G(1) .GE. 0.00) GO TO 17	RILM 417
XK = -XK	RILM 418
17 CONTINUE	RILM 419
IF (XK) 18,20,18	RILM 420
18 AL=G(1)/XK+1.00	RILM 421
PSI(1)=G(2)/(G(1)+XK)	RILM 422
PSI(2)=G(3)/(G(1)+XK)	RILM 423
GO TO 21	RILM 424
19 XK = 0.00	RILM 425
20 AL=2.00	RILM 426
PSI(1)=0.00	RILM 427
PSI(2)=0.00	RILM 428
21 IF(I-IQ)22,25,22	RILM 429
22 IF(I-IP)24,23,24	RILM 430
23 A(I,I-1)=-A(I,I-1)	RILM 431
GO TO 25	RILM 432
24 A(I,I-1)=-XK	RILM 433
25 DO30J=I,N	RILM 434
IF(I-IA)26,26,27	RILM 435
26 C=PSI(2)*A(I+2,J)	RILM 436
GO TO 28	RILM 437
27 C=C.0	RILM 438
28 E=AL*(A(I,J)+PSI(1)*A(I+1,J)+C)	RILM 439
A(I,J)=A(I,J)-E	RILM 440
A(I+1,J)=A(I+1,J)-PSI(1)*E	RILM 441
IF(I-IA)29,29,30	RILM 442
29 A(I+2,J)=A(I+2,J)-PSI(2)*E	RILM 443
30 CONTINUE	RILM 444
IF(I-IA)31,31,32	RILM 445

31 L=I+2	RILM 446
GO TO 33	RILM 447
32 L=N	RILM 448
33 DO38J=IQ,L	RILM 449
IF(I-IA)34,34,35	RILM 450
34 C=PSI(2)*A(J,I+2)	RILM 451
GO TO 36	RILM 452
35 C=C.O	RILM 453
36 F=AL*(A(J,I)+PSI(1)*A(J,I+1)+C)	RILM 454
A(J,I)=A(J,I)-E	RILM 455
A(J,I+1)=A(J,I+1)-PSI(1)*E	RILM 456
IF(I-IA)37,37,38	RILM 457
37 A(J,I+2)=A(J,I+2)-PSI(2)*E	RILM 458
38 CONTINUE	RILM 459
IF(I-N+3)39,39,40	RILM 460
39 E=AL*PSI(2)*A(I+3,I+2)	RILM 461
A(I+3,I)=-E	RILM 462
A(I+3,I+1)=-PSI(1)*E	RILM 463
A(I+3,I+2)=A(I+3,I+2)-PSI(2)*E	RILM 464
40 CONTINUE	RILM 465
41 RETURN	RILM 466
END	RILM 467

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

An arbitrary matrix eigensystem solver.

Purpose

The subroutine ALLMAT uses the QR algorithm to calculate the eigenvalues and eigenvectors of arbitrary complex matrices.

Usage

The calling sequence is:

```
CALL ALLMAT (A,EIGVAL,N,IA,NCAL)
```

where

A is the input matrix, a doubly-dimensioned double precision complex array. On return from ALLMAT, A contains the complex eigenvectors of the input matrix.

EIGVAL is a singly-dimensioned double precision complex array to which ALLMAT returns the eigenvalues with the i^{th} eigenvalue corresponding to the eigenvector in the i^{th} column of A.

N is the integer order of the matrix A.

IA is the integer first dimension of A in the calling program ($IA \leq 30$ unless program is changed).

NCAL is an integer which upon return from ALLMAT contains the number of eigenvalues with corresponding eigenvectors which ALLMAT calculated. Also see "Restrictions".

Coding Information

ALLMAT is a Fortran IV double precision complex subprogram which can be easily converted to single precision complex. ALLMAT makes use of two temporary complex arrays of dimension 30 x 30 and several singly-dimensioned arrays. These scratch arrays can be added to the argument list if extra storage is needed when ALLMAT is not being executed. If eigensystems of matrices of order greater than 30 are desired, the dimensions in ALLMAT should be altered appropriately. The second dimension of A, and the dimension of EIGVAL are irrelevant in ALLMAT, but A must be dimensioned by, at least, $N \times N$, and EIGVAL by at least N in the calling program. Also, the variables A and EIGVAL must appear in a COMPLEX*16 type statement in the calling program.

Simple programming modifications can be affected if the eigenvectors are not desired. This calculation begins at statement 37.

Restrictions

The eigenvalues of A are not necessarily calculated in any absolute algebraic order. Ten iterations per eigenvalue are allowed, and examples exist for which convergence will not occur in ALLMAT (e.g., most lower triangular matrices with all equal eigenvalues; a matrix with ones on the lower diagonal, one as the N^{th} component of the first row and zeros elsewhere). In the case of nonconvergence, ALLMAT will return a value for NCAL less than N, and it is suggested that the user experiment with arbitrary shifts of the input matrix

(i.e., add a constant to the diagonal of A) which will sometimes eliminate the difficulty (e.g., second example just stated). If overflows or underflows occur, scaling A such that its largest element is in modulus about one will probably eliminate the difficulty. Equal computed eigenvalues return identical corresponding eigenvectors even though linearly independent vectors may exist.

Timing

A single precision version of ALLMAT has been timed on an IBM 7090. Times vary not only with order but also with individual matrices. For full matrices execution time is approximately proportional to N^3 . An interesting comparison is that ALLMAT runs in less time than a Jacobi routine for Hermitian matrices. The following table gives some examples of the timing we have experienced on the 7090 for single precision complex matrices obtained from a pseudo random number generator. Fixed dimensions were used in these tests.

ORDER	EIGENVALUES ONLY	EIGENVALUES AND EIGENVECTORS
10	3.47 sec.	4.76 sec.
20	21.9 sec.	29.5 sec.
30	66.5 sec.	93.4 sec.

Accuracy

The accuracy obtainable in computing the eigenvalues of A, $\lambda_i(A)$, is usually related to the spectral radius, $\rho(A)$, of the matrix A

or more generally to some norm of A times the norm of its inverse. Hence, the greater $\rho(A)/\min|\lambda_i|$ the fewer significant digits, the smaller eigenvalues may have. Accuracy also decreases as the order of the matrix increases. Close eigenvalues are usually calculated with less accuracy than well separated ones. In most cases ALLMAT has yielded roots and vectors accurate to about fourteen significant digits.

Method

The input matrix A is reduced to an upper Hessenberg matrix, H , by a sequence of elementary triangular and permutation matrices which make up a matrix P such that $P^{-1}AP = H$. The QR algorithm is made use of in ALLMAT by applying unitary similarity transformations to Hessenberg matrices, H_i : $H_1 = P^{-1}AP$, $H_s = (h_{ij}^{(s)}) = Q_s T_s$, $H_{s+1} = Q_s^H H_s Q_s = Q_s^H Q_s T_s Q_s = T_s Q_s$ where Q_s^H is the product of plane rotations, chosen so that T_s is upper triangular. This process makes $h_{n,n-1}^{(s)}$ converge to zero and therefore $h_{nn}^{(s)}$ converges to an eigenvalue of A . When convergence is met ($h_{n,n-1}^{(s)}$ negligible), the Hessenberg matrix, H_s , is deflated (i.e., last row and column eliminated) and ALLMAT proceeds with its leading principal submatrix (a new H_1) of order one less. If $h_{n-1,n-2}^{(s)}$ becomes negligible, the eigenvalues of the lower right hand matrix of order two are calculated, and ALLMAT proceeds with the leading principal submatrix of order two less. It can be shown that convergence is accelerated by judiciously subtracting scalar matrices from the H_s matrices. ALLMAT actually replaces H_s by $H_s - k_s I$ such that k_s is one of the eigenvalues

p_s or q_s of the lower right hand 2×2 matrix of H_s . The choice of p_s or q_s is made on the basis of whether $|h_{nn}^{(s)} - p_s|$ or $|h_{nn}^{(s)} - q_s|$ is a minimum. The shift technique is applied at each iteration.

Two passes of the Wielandt inverse power method are used to calculate the eigenvectors, y_i , of H . Very little work is required for the second pass since the necessary elementary triangular and permutation matrices are stored in MULT and INTH. Finally, the eigenvectors of A , Py_i , are calculated. The matrix P resides in JNT and the lower part of H .

The theory and a complete description of the algorithms appear in [1]. Both [1] and [3] point out several improvements and refinements that could be incorporated into later versions of ALLMAT.

References

- [1] Wilkinson, J. H., (1965): The Algebraic Eigenvalue Problem, Oxford.
- [2] Householder, A. S., (1964): The Theory of Matrices in Numerical Analysis, Blaisdell.
- [3] Parlett, B. N., (1967): "The LU and QR Algorithms," chapter 5, Mathematical Methods for Digital Computers, vol. 2, (Eds. Ralston and Wilf), Wiley.

Authors

R. E. Funderlic and J. Rinzel, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      SUBROUTINE ALLMAT(A,LAMBDA,M,IA,NCAL)
C   PROGRAM AUTHORS   JOHN RINZEL AND R. E. FUNDERLIC,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C
C   CTC ORD PROGRAM NO. 9079
      IMPLICIT REAL*8 (A-H,O-Z)
      COMPLEX*16 A(IA,1),H(30,30),HL(30,30),LAMBDA(1),VECT(30),
      IMULT(30),SHIFT(3),TEMP,SIN,COS,TEMP1,TEMP2
      LOGICAL INTH(30),TWICE
      INTEGER JNT(30),R,RP1,RP2
C   DIMENSIONS   CHANGED
      N=M
      NCAL=N
      IF(N.NE.1)GO TO 1
      LAMBDA(1)=A(1,1)
      A(1,1)=1.
      GO TO 57
1   ICOUNT=0
      SHIFT(1) = 0.
      IF(N.NE.2)GO TO 4
2   TEMP=(A(1,1)+A(2,2)+CDSQRT((A(1,1)+A(2,2))**2-
      14.*(A(2,2)*A(1,1)-A(2,1)*A(1,2))))/2.
      RELTEM = TEMP
      AMGTEM = (0.,-1.)*TEMP
      IF (RELTEM.NE.0..OR.AMGTEM.NE.0.) GO TO 3
      LAMBDA(M)=SHIFT(1)
      LAMBDA(M-1)=A(1,1)+A(2,2)+SHIFT(1)
      GO TO 37
3   LAMBDA(M)=TEMP+SHIFT(1)
      LAMBDA(M-1)=(A(2,2)*A(1,1)-A(2,1)*A(1,2))/(LAMBDA(M)-SHIFT(1))+
      1SHIFT(1)
      GO TO 37
C
C   REDUCE MATRIX A TO HESSENBERG FORM
C
4   NM2=N-2
      DO 15 R=1,NM2
        RP1=R+1
        RP2=R+2
        ABIG=0.
        JNT(R)=RP1
        DO 5 I=RP1,N
          RELAIR = A(I,R)
          AMGAIR = (0.,-1.)*A(I,R)
          ABSSQ = RELAIR**2 + AMGAIR**2
          IF(ABSSQ.LE.ABIG)GO TO 5
          JNT(R)=I
          ABIG=ABSSQ
5   CONTINUE

```

```

ALLM0001
ALLM0002
ALLM0003
ALLM0004
ALLM0005
ALLM0006
ALLM0007
ALLM0008
ALLM0009
ALLM0010
ALLM0011
ALLM0012
ALLM0013
ALLM0014
ALLM0015
ALLM0016
ALLM0017
ALLM0018
ALLM0019
ALLM0020
ALLM0021
ALLM0022
ALLM0023
ALLM0024
ALLM0025
ALLM0026
ALLM0027
ALLM0028
ALLM0029
ALLM0030
ALLM0031
ALLM0032
ALLM0033
ALLM0034
ALLM0035
ALLM0036
ALLM0037
ALLM0038
ALLM0039
ALLM0040
ALLM0041
ALLM0042
ALLM0043
ALLM0044
ALLM0045
ALLM0046
ALLM0047
ALLM0048
ALLM0049
ALLM0050

```

INTER=JNT(R)	ALLM0051
IF(ABIG.EQ.0.)GO TO 15	ALLM0052
IF(INTER.EQ.RP1)GO TO 8	ALLM0053
DO 6 I=R,N	ALLM0054
TEMP=A(RP1,I)	ALLM0055
A(RP1,I)=A(INTER,I)	ALLM0056
6 A(INTER,I)=TEMP	ALLM0057
DO 7 I=1,N	ALLM0058
TEMP=A(I,RP1)	ALLM0059
A(I,RP1)=A(I,INTER)	ALLM0060
7 A(I,INTER)=TEMP	ALLM0061
8 DO 9 I=RP2,N	ALLM0062
MULT(I)=A(I,R)/A(RP1,R)	ALLM0063
9 A(I,R)=MULT(I)	ALLM0064
DO 11 I=1,RP1	ALLM0065
TEMP=0.	ALLM0066
DO 10 J=RP2,N	ALLM0067
10 TEMP=TEMP+A(I,J)*MULT(J)	ALLM0068
11 A(I,RP1)=A(I,RP1)+TEMP	ALLM0069
DO 13 I=RP2,N	ALLM0070
TEMP=0.	ALLM0071
DO 12 J=RP2,N	ALLM0072
12 TEMP=TEMP+A(I,J)*MULT(J)	ALLM0073
13 A(I,RP1)=A(I,RP1)+TEMP-MULT(I)*A(RP1,RP1)	ALLM0074
DO 14 I=RP2,N	ALLM0075
DO 14 J=RP2,N	ALLM0076
14 A(I,J)=A(I,J)-MULT(I)*A(RP1,J)	ALLM0077
15 CONTINUE	ALLM0078
C	ALLM0079
C CALCULATE EPSILON	ALLM0080
C	ALLM0081
EPS=0.	ALLM0082
DO 16 I=1,N	ALLM0083
16 EPS=EPS+CDABS(A(1,I))	ALLM0084
DO 18 I=2,N	ALLM0085
SUM=0.	ALLM0086
IM1=I-1	ALLM0087
DO 17 J=IM1,N	ALLM0088
17 SUM=SUM+CDABS(A(I,J))	ALLM0089
18 IF(SUM.GT.EPS)EPS=SUM	ALLM0090
EPS=DSQRT(DFLOAT(N))*EPS*1.D-20	ALLM0091
IF(EPS.EQ.0.)EPS=1.D-20	ALLM0092
DO 19 I=1,N	ALLM0093
DO 19 J=1,N	ALLM0094
19 H(I,J)=A(I,J)	ALLM0095
20 IF(N.NE.1)GO TO 21	ALLM0096
LAMBDA(M)=A(1,1)+SHIFT(1)	ALLM0097
GO TO 37	ALLM0098
21 IF(N.EQ.2)GO TO 2	ALLM0099
22 MN1=M-N+1	ALLM0100

```

RELANN = A(N,N)
AMGANN = (0.,-1.)*A(N,N)
RLNNM1 = A(N,N-1)
AMNNM1 = (0.,-1.)*A(N,N-1)
RLNDN1 = A(N,N-1)/A(N,N)
AMNDN1 = (0.,-1.)*(A(N,N-1)/A(N,N))
IF (RELANN.NE.0..OR.AMGANN.NE.0.)
  1 IF (DABS(RLNDN1) + DABS(AMNDN1) - 1.0-18) 24,24,23
23 IF (DABS(RLNNM1) + DABS(AMNNM1).GE.EPS) GO TO 25
24 LAMBDA(MN1)=A(N,N)+SHIFT(1)
   ICGUNT=0
   N=N-1
   GO TO 21
C
C   DETERMINE SHIFT
C
25 SHIFT(2)=(A(N-1,N-1)+A(N,N)+CDSQRT((A(N-1,N-1)+A(N,N))**2
  1 -4.*(A(N,N)*A(N-1,N-1)-A(N,N-1)*A(N-1,N))))/2.
   RELSHF = SHIFT(2)
   AMGSHF = (0.,-1.)*SHIFT(2)
   IF (RELSHF.NE.0..OR.AMGSHF.NE.0.) GO TO 26
   SHIFT(3)=A(N-1,N-1)+A(N,N)
   GO TO 27
26 SHIFT(3)=(A(N,N)*A(N-1,N-1)-A(N,N-1)*A(N-1,N))/SHIFT(2)
27 IF (CDABS(SHIFT(2)-A(N,N)).LT.CDABS(SHIFT(3)-A(N,N))) GO TO 28
   INDEX=3
   GO TO 29
28 INDEX=2
29 IF (CDABS(A(N-1,N-2)).GE.EPS) GO TO 30
   LAMBDA(MN1)=SHIFT(2)+SHIFT(1)
   LAMBDA(MN1+1)=SHIFT(3)+SHIFT(1)
   ICGUNT=0
   N=N-2
   GO TO 20
30 SHIFT(1)=SHIFT(1)+SHIFT(INDEX)
   DO 31 I=1,N
31 A(I,1)=A(I,1)-SHIFT(INDEX)
C
C   PERFORM GIVENS ROTATIONS, QR ITERATES
C
   IF (ICGUNT.LE.10) GO TO 32
   NCAL=M-N
   GO TO 37
32 NM1=N-1
   TEMP1=A(1,1)
   TEMP2=A(2,1)
   DO 36 R=1,NM1
   RP1=R+1
   RELTM1 = TEMP1
   AMGMT1 = (0.,-1.)*TEMP1

```

```

ALLM0101
ALLM0102
ALLM0103
ALLM0104
ALLM0105
ALLM0106
ALLM0107
ALLM0108
ALLM0109
ALLM0110
ALLM0111
ALLM0112
ALLM0113
ALLM0114
ALLM0115
ALLM0116
ALLM0117
ALLM0118
ALLM0119
ALLM0120
ALLM0121
ALLM0122
ALLM0123
ALLM0124
ALLM0125
ALLM0126
ALLM0127
ALLM0128
ALLM0129
ALLM0130
ALLM0131
ALLM0132
ALLM0133
ALLM0134
ALLM0135
ALLM0136
ALLM0137
ALLM0138
ALLM0139
ALLM0140
ALLM0141
ALLM0142
ALLM0143
ALLM0144
ALLM0145
ALLM0146
ALLM0147
ALLM0148
ALLM0149
ALLM0150

```

```

RELTM2 = TEMP2
AMGTM2 = (0.,-1.)*TEMP2
RHO = DSQRT(RELTM1**2 + AMGTM1**2 + RELTM2**2 + AMGTM2**2)
IF(RHO.EQ.0.)GO TO 36
COS=TEMP1/RHO
SIN=TEMP2/RHO
INDEX=MAX0(R-1,1)
DO 33 I=INDEX,N
TEMP=DCONJG(COS)*A(R,I)+DCONJG(SIN)*A(RP1,I)
A(RP1,I)=-SIN*A(R,I)+COS*A(RP1,I)
33 A(R,I)=TEMP
TEMP1=A(RP1,RP1)
TEMP2=A(R+2,R+1)
DO 34 I=1,R
TEMP=COS*A(I,R)+SIN*A(I,RP1)
A(I,RP1)=-DCONJG(SIN)*A(I,R)+DCONJG(COS)*A(I,RP1)
34 A(I,R)=TEMP
INDEX=MIN0(R+2,N)
DO 35 I=RP1,INDEX
A(I,R)=SIN*A(I,RP1)
35 A(I,RP1)=DCONJG(COS)*A(I,RP1)
36 CONTINUE
ICOUNT=ICOUNT+1
GO TO 22

C
C CALCULATE VECTORS
C
37 IF(NCAL.EQ.0)GO TO 57
N=M
NM1=N-1
IF(N.NE.2)GO TO 38
EPS=DMAX1(CDABS(LAMBDA(1)),CDABS(LAMBDA(2)))*1.D-16
IF(EPS.EQ.0.)EPS=1.D-12
H(1,1)=A(1,1)
H(1,2)=A(1,2)
H(2,1)=A(2,1)
H(2,2)=A(2,2)
38 DO 56 L=1,NCAL
DO 40 I=1,N
DO 39 J=1,N
39 HL(I,J)=H(I,J)
40 HL(I,I)=HL(I,I)-LAMBDA(L)
DO 44 I=1,NM1
MULT(I)=0.
INTH(I)=.FALSE.
IP1=I+1
IF(CDABS(HL(I+1,I)).LE.CDABS(HL(I,I)))GO TO 42
INTH(I)=.TRUE.
DO 41 J=I,N
TEMP=HL(I+1,J)

```

```

ALLM0151
ALLM0152
ALLM0153
ALLM0154
ALLM0155
ALLM0156
ALLM0157
ALLM0158
ALLM0159
ALLM0160
ALLM0161
ALLM0162
ALLM0163
ALLM0164
ALLM0165
ALLM0166
ALLM0167
ALLM0168
ALLM0169
ALLM0170
ALLM0171
ALLM0172
ALLM0173
ALLM0174
ALLM0175
ALLM0176
ALLM0177
ALLM0178
ALLM0179
ALLM0180
ALLM0181
ALLM0182
ALLM0183
ALLM0184
ALLM0185
ALLM0186
ALLM0187
ALLM0188
ALLM0189
ALLM0190
ALLM0191
ALLM0192
ALLM0193
ALLM0194
ALLM0195
ALLM0196
ALLM0197
ALLM0198
ALLM0199
ALLM0200

```


HL(I+1,J)=HL(I,J)	ALLM0201
41 HL(I,J)=TEMP	ALLM0202
42 RELHII = HL(I,I)	ALLM0203
AMGHII = (0.,-1.)*HL(I,I)	ALLM0204
IF(RELHII.EQ.0..AND.AMGHII.EQ.0.) GO TO 44	ALLM0205
MULT(I)=-HL(I+1,I)/HL(I,I)	ALLM0206
DO 43 J=I+1,N	ALLM0207
43 HL(I+1,J)=HL(I+1,J)+MULT(I)*HL(I,J)	ALLM0208
44 CONTINUE	ALLM0209
DO 45 I=1,N	ALLM0210
45 VECT(I)=1.	ALLM0211
TWICE=.FALSE.	ALLM0212
46 RELHNN = HL(N,N)	ALLM0213
AMGHNN = (0.,-1.)*HL(N,N)	ALLM0214
IF(RELHNN.EQ.0..AND.AMGHNN.EQ.0.) HL(N,N) = EPS	ALLM0215
VECT(N)=VECT(N)/HL(N,N)	ALLM0216
DO 48 I=1,NM1	ALLM0217
K=N-I	ALLM0218
DO 47 J=K,NM1	ALLM0219
47 VECT(K)=VECT(K)-HL(K,J+1)*VECT(J+1)	ALLM0220
RELHKK = HL(K,K)	ALLM0221
AMGHKK = (0.,-1.)*HL(K,K)	ALLM0222
IF(RELHKK.EQ.0..AND.AMGHKK.EQ.0.) HL(K,K) = EPS	ALLM0223
48 VECT(K)=VECT(K)/HL(K,K)	ALLM0224
BIG=0.	ALLM0225
DO 49 I=1,N	ALLM0226
RELVEC = VECT(I)	ALLM0227
AMGVEC = (0.,-1.)*VECT(I)	ALLM0228
SUM = DABS(RELVEC)+DABS(AMGVEC)	ALLM0229
49 IF(SUM.GT.BIG)BIG=SUM	ALLM0230
DO 50 I=1,N	ALLM0231
50 VECT(I)=VECT(I)/BIG	ALLM0232
IF(TWICE)GO TO 52	ALLM0233
DO 51 I=1,NM1	ALLM0234
IF(.NOT.INTH(I))GO TO 51	ALLM0235
TEMP=VECT(I)	ALLM0236
VECT(I)=VECT(I+1)	ALLM0237
VECT(I+1)=TEMP	ALLM0238
51 VECT(I+1)=VECT(I+1)+MULT(I)*VECT(I)	ALLM0239
TWICE=.TRUE.	ALLM0240
GO TO 46	ALLM0241
52 IF(N.EQ.2)GO TO 55	ALLM0242
NM2=N-2	ALLM0243
DO 54 I=1,NM2	ALLM0244
N1I=N-1-I	ALLM0245
N1I=N-I+1	ALLM0246
DO 53 J=N1I,N	ALLM0247
53 VECT(J)=H(J,N1I)*VECT(N1I+1)+VECT(J)	ALLM0248
INDEX=JNT(N1I)	ALLM0249
TEMP=VECT(N1I+1)	ALLM0250

```
      VECT(N1I+1)=VECT(INDEX)
54  VECT(INDEX)=TEMP
55  DO 56 I=1,N
56  A(I,L)=VECT(I)
57  RETURN
    END
```

```
ALLM0251
ALLM0252
ALLM0253
ALLM0254
ALLM0255
ALLM0256
```

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

H
LEAST SQUARES

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Linear least squares routine.

Purpose

To form and solve the "normal equations" associated with least squares fits of discrete data. The models are assumed to be linear (in the unknown coefficients) functions which, for example, provides the user with the capability to fit polynomials to his data.

Usage

The calling sequence is:

CALL LSQFIT(M,N,A,B,P,ID)

where

- M is the integer number of points in the sample;
i.e., the number of components of B.
- N is the integer number of parameters in the equation being fitted, e.g., in the system

$$y(x) = P_1 f_1(x) + P_2 f_2(x) + \dots + P_N f_N(x);$$
the introduction of "x" is somewhat artificial and is not meant to restrict applicability to curve fitting.
- A is the doubly-dimensioned double precision input
MxN matrix with ith row:

$$(f_1(x_i), f_2(x_i), \dots, f_N(x_i))$$
- B is the singly-dimensioned double precision input
column vector

$$(y(x_1), y(x_2), \dots, y(x_M))^T.$$

P is the singly-dimensioned double precision array which returns estimates for the parameters $P_1, P_2, \dots, P_N$.

ID is the integer first dimension of A in the calling program.

Coding Information

LSQFIT is a double precision Fortran IV subprogram that may be easily converted to single precision. However, the arrays DA and DB should remain in double precision and inner products should be accumulated in double precision (see AXTY elsewhere in this compendium). LSQFIT uses DMATEQ found elsewhere in this compendium and should be provided by the user. If the user requires a solution with N greater than 10, the entries of 10 in the DOUBLE PRECISION statement should be changed to a number not less than N. Also the 10 entry in the statement following statement number 4 should be changed in the same way. The dimension of B and P and the second dimension of A are of no concern to LSQFIT. Also the input arrays A and B are not altered by LSQFIT. However, B and P may be identified in the calling sequence.

Restrictions

The columns of A must at least be linearly independent. It is required that $2 \leq N \leq M$.

Method

LSQFIT minimizes the sum of the squares of the residuals; i.e., it minimizes $(AX - B)^T(AX - B)$, or in other terminology, it minimizes

$$\sum_{k=1}^M \left[\left(\sum_{i=1}^N p_i f_i(x_k) \right) - y(x_k) \right]^2$$

Author

Miss Jane Cope, a summer employee with Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

	SUBROUTINE LSQFIT(M,N,A,B,X,ID)	LSQF0001
C	PROGRAM AUTHOR MISS JANE COPE, A SUMMER EMPLOYEE AT	LSQF0002
C	COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	LSQF0003
C	OAK RIDGE, TENN.	LSQF0004
C		LSQF0005
C	CTC ORD PROGRAM NO. 9080	LSQF0006
C		LSQF0007
	DOUBLE PRECISION A(ID,1),B(1),DA(10,10),DB(10),D,DMATEQ,X(1)	LSQF0008
	DO1I=1,N	LSQF0009
	DB(I)=0.	LSQF0010
	DO1J=1,M	LSQF0011
1	DB(I)=DB(I)+A(J,I)*B(J)	LSQF0012
	DO3I=1,N	LSQF0013
	DO2J=1,N	LSQF0014
	DA(J,I)=0.	LSQF0015
	DO2K=1,M	LSQF0016
2	DA(J,I)=DA(J,I)+A(K,I)*A(K,J)	LSQF0017
3	CONTINUE	LSQF0018
	DO4I=1,N	LSQF0019
	DO4J=1,N	LSQF0020
4	DA(I,J)=DA(J,I)	LSQF0021
	D=DMATEQ(DA,DB,N,-1,10)	LSQF0022
	DO5I=1,N	LSQF0023
5	X(I)=DB(I)	LSQF0024
	RETURN	LSQF0025
	END	LSQF0026

Identification

Linear least squares solution.

Purpose

ALSQ and BLSQ are double precision Fortran IV subroutines to solve the linear least squares problem by applying the Householder reduction to the least squares matrix [1]. Both programs use the same technique in solving the least squares problem; however, BLSQ returns additional information useful in statistical applications. Both programs have subordinate entries, ALSQ1 and BLSQ1, which enable the user to perform additional fits at very little cost once the least squares matrix has been reduced.

Method

The linear least squares problem to be solved may be formulated as follows: given $n \times m$ ($m < n$) matrix A of rank m and an n -vector y find an m -vector b such that

$$\|Ab - y\|^2 = \min. \quad (1)$$

The programs, in effect, use the Householder reduction of the matrix A to triangular form to find an $n \times n$ orthogonal matrix H such that

$$HA = \begin{pmatrix} T \\ 0 \end{pmatrix}, \quad (2)$$

where T is an $m \times m$ upper triangular matrix and "0" is an $(n - m) \times m$ null matrix. If we let U be the $n \times m$ matrix consisting of the first m columns of H^T (in order) and V be the $n \times (n - m)$

matrix consisting of the remaining columns and consider the new least squares problem of minimizing $||Uc - y||^2$, then

$$c = U^T y \quad (3)$$

and

$$b = T^{-1}c. \quad (4)$$

ALSQ performs the Householder reduction (2) and uses equations (3) and (4) to calculate b. BLSQ calculates and returns the vector c and the matrices T and T^{-1} which are useful in statistical applications. The vector b may subsequently be calculated by the user from equation (4). Both programs return the approximating vector $Ab = Uc$ and the residual sum of squares $||Ab - y||^2$.

Under the usual statistical assumptions of regression analyses, it can be shown that the number of runs in $V^T y$ minus one has the binomial distribution with parameters $n - m - 1$ and .5. If the problem being solved involves fitting continuous functions of a real variable, a statistical test based on the above may be expected to be powerful when the fit inadequately represents the data. Because of this, BLSQ returns the number of runs in $V^T y$.

Usage

The calling sequence for ALSQ is:

```
CALL ALSQ(A, Y, B, R2, N, M, NA),
```

where

A is a doubly subscripted double precision array containing the least squares matrix. The elements of A are altered by the program.

- Y is a singly subscripted, double precision array containing the vector to be fit. The elements of Y are unchanged by the program.
- B is a singly subscripted, double precision array which upon return contains the coefficients of the fit.
- R2 contains, upon return, the double precision residual sum of squares of the fit.
- N is the integer number of rows in the least squares matrix.
- M is the integer number of columns in the least squares matrix.
- NA is the integer first dimension of the array A.

The program requires that the array A be dimensioned so that it will hold one extra row and one extra column in addition to the least squares matrix, i.e., the dimension of A must be at least $N + 1$ by $M + 1$. Upon return from the program, the locations $A(I, M + 1)$ contain the components of the approximating vector Ab (see Method).

A second entry, ALSQ1, allows the user to obtain additional fits without repeating the reduction of the least squares matrix. ALSQ must be called before calling ALSQ1, and the contents of the array A must not be changed between these calls.

The calling sequence is:

CALL ALSQ1(Y, B, R2, K)

where Y, B, and R2 are the same as above and

K is the number of columns of the least squares matrix to fit to the vector Y; e.g., if $K = 4$, the first four columns of A will be fit to Y returning four coefficients in the array B. K must be less than or equal to M in the call ALSQ.

This entry also returns the approximating vector in the locations $A(I, M + 1)$.

The calling sequence for BLSQ is:

CALL BLSQ(A, Y, C, R2, T, T1, N, M, NA, NT),

where A, Y, R2, N, M, and NA are the same as in the calling sequence for ALSQ and

C is a singly subscripted, double precision array which upon return contains the coefficients of the transformed problem given by (3).

IRUN is an integer array whose i^{th} element contains upon return the number of runs in $V^T y$ after i columns of the least squares matrix have been fitted.

T is a doubly subscripted, double precision array which upon return contains the upper triangular transformation matrix of equations (2).

T1 is a doubly subscripted, double precision array which upon return contains the inverse of the transformation matrix.

NT is the first dimension of the arrays T and T1.

BLSQ also has a second entry, BLSQ1, to calculate new fits. The same restrictions apply to the use of BLSQ1 as apply to ALSQ1. The calling sequence is:

CALL BLSQ1(Y, C, R2, IRUN, K),

where Y, C, R2, IRUN are the same as in the calling sequence of BLSQ and K is the number of columns of the least squares matrix to fit to the data. Both BLSQ and BLSQ1 return the components of the approximating vector in the locations A(I, M + 1).

Coding Information

Although these programs are coded in double precision arithmetic, they may be easily converted to single precision. Additional accuracy may be obtained in the single precision program by accumulating inner products in double precision. The places where this should be done are signaled by the appearance of the variables SS and PP.

Reference

- [1] Golub, G. (1965): "Numerical methods for solving linear least squares problems," Numer. Math. 7, 206-216.

Author

G. W. Stewart III, formerly with Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

SUBROUTINE ALSQ(A,Y,B,R2,NN,MM,NA)
C PROGRAM AUTHOR G. W. STEWART, III.
C COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C OAK RIDGE, TENN.
C
C CTC ORD PROGRAM NO. 9081.1
C ALSQ IS A FORTRAN IV SUBROUTINE TO SOLVE THE LINEAR LEAST SQUARES
C PROBLEM  $\text{NORM}(AB - Y) = \text{MIN.}$  ITS CALLING SEQUENCE IS
C CALL ALSQ(A,Y,B,R2,N,M,NA)
C WHERE
C A IS AN ARRAY CONTAINING THE LEAST SQUARES MATRIX.
C UPON RETURN THE (M+1)-TH COLUMN CONTAINS THE
C APPROXIMATING VECTOR AB.
C Y IS THE VECTOR TO BE FIT.
C B CONTAINS UPON RETURN THE COEFFICIENTS OF THE FIT.
C R2 CONTAINS UPON RETURN THE RESIDUAL SUM OF SQUARES.
C N IS THE NUMBER OF ROWS IN THE LEAST SQUARES MATRIX.
C M IS THE NUMBER OF COLUMNS IN THE LEAST SQUARES
C MATRIX.
C NA IS THE FIRST DIMENSION OF THE ARRAY A.
C ALSQ HAS AN ADDITIONAL ENTRY, ALSQ1, TO PERMIT THE FITTING OF
C NEW VECTORS Y AFTER AN INITIAL FIT HAS BEEN PERFORMED VIA ALSQ.
C ITS CALLING SEQUENCE IS
C CALL ALSQ1(Y,B,R2,K),
C WHERE Y,B, AND R2 ARE THE SAME AS IN THE CALLING SEQUENCE FOR
C ALSQ, AND
C K IS THE NUMBER OF COLUMNS OF A TO FIT TO Y. K MUST
C BE LESS THAN OR EQUAL TO M IN THE CALLING SEQUENCE
C OF ALSQ.
C UPON RETURN FROM ALSQ1, THE (M+1)-TH COLUMN OF THE ARRAY A IN
C THE CALL TO ALSQ WILL CONTAIN THE APPROXIMATING VECTOR.
C
C IMPLICIT REAL*8 (A-H,O-Z)
C DIMENSION A(NA,1),Y(1),B(1)
C N = NN
C N1 = N+1
C M = MM
C M1 = M+1
C MM1 = M-1
C
C REDUCE THE LEAST SQUARES MATRIX TO UPPER TRIANGULAR FORM
C
C DO 100 L=1,M
C SS = 0.
C DO 25 I=L,N
25 SS = SS + A(I,L)**2
C S2 = SS
C S = DSORT(S2)
C IF(A(L,L).LT.0.) S=-S
C D = S2 + S*A(L,L)

```

A(L,L) = A(L,L) + S	ALSQ0051
IF(L.EQ.M) GO TO 80	ALSQ0052
L1 = L+1	ALSQ0053
DO 50 J=L1,M	ALSQ0054
PP = 0.	ALSQ0055
DO 35 I=L,N	ALSQ0056
35 PP = PP + A(I,L)*A(I,J)	ALSQ0057
50 A(N1,J) = PP/D	ALSQ0058
DO 75 J=L1,M	ALSQ0059
DO 75 I=L,N	ALSQ0060
75 A(I,J) = A(I,J) - A(I,L)*A(N1,J)	ALSQ0061
80 A(N1,L) = -S	ALSQ0062
100 CONTINUE	ALSQ0063
GO TO 105	ALSQ0064
C	ALSQ0065
C REDUCE THE VECTOR Y	ALSQ0066
C	ALSQ0067
ENTRY ALSQ1(Y,B,R2,MMM)	ALSQ0068
M = MMM	ALSQ0069
MM1 = M-1	ALSQ0070
105 DO 110 I=1,N	ALSQ0071
110 A(I,M1) = Y(I)	ALSQ0072
DO 150 L=1,M	ALSQ0073
PP = 0.	ALSQ0074
DO 120 I=L,N	ALSQ0075
120 PP = PP + A(I,L)*A(I,M1)	ALSQ0076
D = PP/(-A(L,L)*A(N1,L))	ALSQ0077
DO 150 I=L,N	ALSQ0078
150 A(I,M1) = A(I,M1) - D*A(I,L)	ALSQ0079
C	ALSQ0080
C CALCULATE THE COEFFICIENT VECTOR B	ALSQ0081
C	ALSQ0082
B(M) = A(M,M1)/A(N1,M)	ALSQ0083
IF(M.EQ.1) GO TO 210	ALSQ0084
DO 200 LL=1,MM1	ALSQ0085
L = M-LL	ALSQ0086
L1 = L+1	ALSQ0087
PP = A(L,M1)	ALSQ0088
DO 175 I=L1,M	ALSQ0089
175 PP = PP - A(L,I)*P(I)	ALSQ0090
200 B(L) = PP/A(N1,L)	ALSQ0091
C	ALSQ0092
C CALCULATE R2	ALSQ0093
C	ALSQ0094
210 SS = 0.	ALSQ0095
MP1 = M+1	ALSQ0096
DO 225 I=MP1,N	ALSQ0097
SS = SS + A(I,M1)**2	ALSQ0098
225 A(I,M1) = 0.	ALSQ0099
R2 = SS	ALSQ0100


```

C
C   PERFORM THE PACK CALCULATIONS
C
C   DO 300 LL=1,M
C   L = M-LL+1
C   PP = 0.
C   DO 250 I=L,N
250 PP = PP + A(I,L)*A(I,M1)
C   D = PP/(-A(L,L)*A(N1,L))
C   DO 275 I=L,N
275 A(I,M1) = A(I,M1) - D*A(I,L)
300 CONTINUE
C   RETURN
C   END
C   SUBROUTINE BLSQ(A,Y,C,R2,IRUN,T,T1,NN,MM,NA,NT)
C   PROGRAM AUTHOR G. W. STEWART,III,
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C   CTC ORD PROGRAM NO. 9081.2
C   BLSQ IS A FORTRAN-IV SUBROUTINE TO SOLVE THE LINEAR LEAST SQUARES
C   PROBLEM  $\text{NORM}(AB - Y) = \text{MIN}$ , RETURNING ADDITIONAL INFORMATION
C   USEFUL IN STATISTICAL APPLICATIONS. ITS CALLING SEQUENCE IS
C   CALL BLSQ(A,Y,C,R2,IRUN,T,T1,N,M,NA,NT),
C   WHERE
C   A   IS AN ARRAY CONTAINING THE LEAST SQUARES MATRIX.
C   UPON RETURN THE (M+1)-TH COLUMN CONTAINS THE
C   APPROXIMATING VECTOR AB.
C   Y   IS THE VECTOR TO BE FIT.
C   C   CONTAINS UPON RETURN THE COEFFICIENTS OF THE FIT
C   OF THE COLUMNS OF A*1 TO Y.
C   R2  CONTAINS UPON RETURN THE RESIDUAL SUM OF SQUARES.
C   IRUN IS AN ARRAY WHOSE I-TH COMPONENT CONTAINS UPON
C   RETURN THE NUMBER OF RUNS IN THE TRUNCATED
C   RESIDUAL VECTOR AFTER I COLUMNS OF A HAVE BEEN FIT.
C   T   CONTAINS UPON RETURN THE MXM UPPER TRIANGULAR
C   SQUARE ROOT OF THE PRODUCT OF A TRANSPOSE AND A.
C   T1  CONTAINS UPON RETURN THE INVERSE OF T.
C   N   IS THE NUMBER OF ROWS IN THE LEAST SQUARES MATRIX.
C   M   IS THE NUMBER OF COLUMNS IN THE LEAST SQUARES
C   MATRIX.
C   NA  IS THE FIRST DIMENSION OF THE ARRAY A.
C   NT  IS THE FIRST DIMENSION OF THE ARRAYS T AND T1.
C
C   BLSQ HAS AN ADDITIONAL ENTRY, BLSQ1, TO PERMIT THE FITTING OF
C   NEW VECTORS Y AFTER AN INITIAL FIT HAS BEEN PERFORMED VIA BLSQ.
C   ITS CALLING SEQUENCE IS
C   CALL BLSQ1(Y,C,R2,IRUN,K)
C   WHERE Y,C,R2, AND IRUN ARE THE SAME AS IN THE CALLING SEQUENCE
C   FOR BLSQ, AND
C   K   IS THE NUMBER OF COLUMNS OF A TO FIT TO Y. K MUST

```

```

ALSQ0101
ALSQ0102
ALSQ0103
ALSQ0104
ALSQ0105
ALSQ0106
ALSQ0107
ALSQ0108
ALSQ0109
ALSQ0110
ALSQ0111
ALSQ0112
ALSQ0113
ALSQ0114
BLSQ0001
BLSQ0002
BLSQ0003
BLSQ0004
BLSQ0005
BLSQ0006
BLSQ0007
BLSQ0008
BLSQ0009
BLSQ0010
BLSQ0011
BLSQ0012
BLSQ0013
BLSQ0014
BLSQ0015
BLSQ0016
BLSQ0017
BLSQ0018
BLSQ0019
BLSQ0020
BLSQ0021
BLSQ0022
BLSQ0023
BLSQ0024
BLSQ0025
BLSQ0026
BLSQ0027
BLSQ0028
BLSQ0029
BLSQ0030
BLSQ0031
BLSQ0032
BLSQ0033
BLSQ0034
BLSQ0035
BLSQ0036

```

```

C          BE LESS THAN OR EQUAL TO M IN THE CALLING SEQUENCE BLSQ0037
C          OF ALSQ. BLSQ0038
C          UPON RETURN FROM ALSQ1, THE (M+1)-TH COLUMN OF THE ARRAY A IN BLSQ0039
C          THE CALL TO ALSQ WILL CONTAIN THE APPROXIMATING VECTOR. BLSQ0040
C BLSQ0041
C      IMPLICIT REAL*8 (A-H,O-Z) BLSQ0042
C      DIMENSION A(NA,1),Y(1),C(1),RUN(1),T(NT,1),T1(NT,1) BLSQ0043
C      INTEGER RUN BLSQ0044
C      DOUBLE PRECISION SS,PP BLSQ0045
C      N = NN BLSQ0046
C      N1 = N+1 BLSQ0047
C      M = MM BLSQ0048
C      M1 = M+1 BLSQ0049
C      MM1 = M-1 BLSQ0050
C BLSQ0051
C      REDUCE THE LEAST SQUARES MATRIX TO UPPER TRIANGULAR FORM BLSQ0052
C BLSQ0053
C      DO 100 L=1,M BLSQ0054
C      SS = 0. BLSQ0055
C      DO 25 I=L,N BLSQ0056
25  SS = SS + A(I,L)**2 BLSQ0057
C      S2 = SS BLSQ0058
C      S =DSQRT(S2) BLSQ0059
C      IF(A(L,L).LT.0.) S=-S BLSQ0060
C      D = S2 + S*A(L,L) BLSQ0061
C      A(L,L) = A(L,L) + S BLSQ0062
C      IF(L.EQ.M) GO TO 80 BLSQ0063
C      L1 = L+1 BLSQ0064
C      DO 50 J=L1,M BLSQ0065
C      PP = 0. BLSQ0066
C      DO 35 I=L,N BLSQ0067
35  PP = PP + A(I,L)*A(I,J) BLSQ0068
50  A(N1,J) = PP/D BLSQ0069
C      DO 75 J=L1,M BLSQ0070
C      DO 75 I=L,N BLSQ0071
75  A(I,J) = A(I,J) - A(I,L)*A(N1,J) BLSQ0072
80  A(N1,L) = -S BLSQ0073
100 CONTINUE BLSQ0074
C BLSQ0075
C      FORM THE MATRIX T AND INVERT IT BLSQ0076
C BLSQ0077
C      DO 110 I=1,M BLSQ0078
C      DO 105 J=I,M BLSQ0079
C      T(I,J) = A(I,J) BLSQ0080
C      T(J,I) = 0. BLSQ0081
105 T1(J,I) = 0. BLSQ0082
110 T(I,I) = A(N1,I) BLSQ0083
C      DO 150 LL=1,M BLSQ0084
C      L = M-LL+1 BLSQ0085
C      T1(L,L) = 1./T(L,L) BLSQ0086

```

252

IF(L.EQ.1) GO TO 155	BLSQ0087
DO 150 I1=2,L	BLSQ0088
I = L-I1+1	BLSQ0089
I1 = I+1	BLSQ0090
PP = 0.	BLSQ0091
DO 130 J=I1,L	BLSQ0092
130 PP = PP - T(I,J)*T1(J,L)	BLSQ0093
150 T1(I,L) = PP/T(I,I)	BLSQ0094
GO TO 155	BLSQ0095
C	BLSQ0096
C REDUCE THE VECTOR Y	BLSQ0097
C	BLSQ0098
ENTRY BLSQ1(Y,C,R2,PUN,MMM)	BLSQ0099
M = MMM	BLSQ0100
MM1 = M-1	BLSQ0101
155 DO 160 I=1,N	BLSQ0102
160 A(I,M1) = Y(I)	BLSQ0103
DO 200 L=1,M	BLSQ0104
L2 = L+2	BLSQ0105
PP = 0.	BLSQ0106
DO 170 I=L,N	BLSQ0107
170 PP = PP + A(I,L)*A(I,M1)	BLSQ0108
D = PP/(-A(L,L)*A(N1,L))	BLSQ0109
DO 180 I=L,N	BLSQ0110
180 A(I,M1) = A(I,M1) - D*A(I,L)	BLSQ0111
C(L) = A(L,M1)	BLSQ0112
RUN(L) = 1	BLSQ0113
DO 200 I=L2,N	BLSQ0114
200 IF(A(I-1,M1)*A(I,M1).LT.C.) RUN(L) = RUN(L) + 1	BLSQ0115
C	BLSQ0116
C CALCULATE R2	BLSQ0117
C	BLSQ0118
210 SS = 0.	BLSQ0119
MP1 = M+1	BLSQ0120
DO 225 I=MP1,N	BLSQ0121
SS = SS + A(I,M1)**2	BLSQ0122
225 A(I,M1) = 0.	BLSQ0123
R2 = SS	BLSQ0124
C	BLSQ0125
C PERFORM THE BACK CALCULATIONS	BLSQ0126
C	BLSQ0127
DO 300 LL=1,M	BLSQ0128
L = M-LL+1	BLSQ0129
PP = 0.	BLSQ0130
DO 250 I=L,N	BLSQ0131
250 PP = PP + A(I,L)*A(I,M1)	BLSQ0132
D = PP/(-A(L,L)*A(N1,L))	BLSQ0133
DO 275 I=L,N	BLSQ0134
275 A(I,M1) = A(I,M1) - D*A(I,L)	BLSQ0135
300 CONTINUE	BLSQ0136

RETURN
END

BLSG0137
BLSG0138

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

I

STATISTICS

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Pseudo-random number generation subroutines.

Purpose

These routines produce sequences of numbers randomly drawn from various probability distributions.

Usage

The eight generators with their calling sequences are:

1. Uniform distribution

CALL FLAT(X,A,B)

where

X is a generated pseudo-random real number uniformly distributed between A and B.

2. Gaussian distribution

CALL GAUSS (X,A,S)

where

X is a generated pseudo-random real number from the Gaussian distribution with mean A and variance S^2 ,

$$\frac{\exp(-(X-A)^2/S^2)}{S\sqrt{2\pi}}$$

3. Exponential distribution

CALL EXPON(X,P)

where

X is a generated pseudo-random real number from the exponential distribution $P \exp(-PX)$ on the interval $0 \leq X \leq \infty$.

4. Cauchy distribution

CALL CAUCH(X,A,P)

where

X is a generated pseudo-random real number from the cauchy distribution $P/(P^2 - (X-A)^2)$.

5. Uniform distribution of integers

CALL INTER(IX,N)

where

IX is a generated pseudo-random integer uniformly distributed between 0 and N-1 inclusive.

6. Binomial distribution

CALL IBINOM(IX,P,N)

where

IX is a generated pseudo-random integer from the Binomial distribution with parameters P and N.

7. Poisson distribution

CALL IPOIS(IX,FL)

where

IX is a generated pseudo-random integer from the Poisson distribution with parameter FL.

8. The assembly coded function subprogram

X = FLTRN(DUM)

where

DUM is a dummy argument neither used nor changed by FLTRN.

The value of the function is a Real*4 number from a uniform distribution between zero and one non-inclusive at both limits.

Coding Information

The subroutines are written in Fortran IV (except FLTRN) and all real arguments are single precision (REAL*4).

The pseudo-random number generators in this package may be used independently with the following exceptions:

1. All of the generators require the assembly coded FLTRN (see 8 above).
2. The generators GAUSS and IPOIS use the generator EXPON.

Initialization

The sequences of numbers all originate in FLTRN. Unless the procedure described in the following paragraph is taken, FLTRN will always produce the same sequence of numbers. Thus, if a program is run twice under the same conditions, the results will be the same.

A look at the coding for FLTRN will reveal a control section named RNDOM, which contains two constants. By initializing these constants to different values, different sequences of random numbers may be produced.

Acknowledgment

The assembly routine FLTRN was coded by J. G. Sullivan, Oak Ridge National Laboratory.

Author

Alfred Hume, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

      FUNCTION FLAT(RES,A,B)
C  PROGRAM AUTHOR  A. HUME
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C  GENERATES UNIFORMLY DISTRIBUTED RANDOM NUMBERS
C  ON THE CLOSED INTERVAL (A,B).
C  RES -- RESULT LEFT HERE
C  A -- LOWER BOUND
C  B -- UPPER BOUND
C  CTC ORD PROGRAM NO. 9082.1
      RES=FLTRN(1.)*(B-A)+A
      FLAT=RES
      RETURN
      END
      FUNCTION GAUSS(RES,A,S)
C  PROGRAM AUTHOR  A. HUME
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C  GENERATES NORMALLY DISTRIBUTED RANDOM NUMBERS
C  RES -- RESULT LEFT HERE
C  A -- MEAN OF DISTRIBUTION
C  S -- STANDARD DEVIATION OF DISTRIBUTION
C  CTC ORD PROGRAM NO. 9082.2
      1 CALL EXPON(EXPR1,1.)
      CALL EXPON(EXPR2,1.)
      IF((EXPR2-1.)*2.GT(EXPR1*2.) GOTO 1
      IF(FLTRN(1.).LT..5) EXPR2=-EXPR2
      RES=EXPR2*S+A
      GAUSS=RES
      RETURN
      END
      FUNCTION EXPON(RES,P)
C  PROGRAM AUTHOR  A. HUME
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C  GENERATES EXPONENTIALLY DISTRIBUTED RANDOM NUMBERS
C  RES -- RESULT LEFT HERE
C  P -- THE EQUATION IS  $P \cdot \exp(-RES \cdot P)$ 
C  CTC ORD PROGRAM NO. 9082.3
      I=0
      1 X=FLTRN(0.)
      Z=X
      2 Y=FLTRN(0.)
      IF(Y.GT.Z) GOTO 3
      Z=FLTRN(0.)
      IF(Z.LT.Y) GOTO 2
      I=I+1
      GOTO 1
      3 RES=(X+FLOAT(I))/P
      EXPON=RES

```

```

FLAT0001
FLAT0002
FLAT0003
FLAT0004
FLAT0005
FLAT0006
FLAT0007
FLAT0008
FLAT0009
FLAT0010
FLAT0011
FLAT0012
FLAT0013
FLAT0014
GAUS0001
GAUS0002
GAUS0003
GAUS0004
GAUS0005
GAUS0006
GAUS0007
GAUS0008
GAUS0009
GAUS0010
GAUS0011
GAUS0012
GAUS0013
GAUS0014
GAUS0015
GAUS0016
GAUS0017
EXP00001
EXP00002
EXP00003
EXP00004
EXP00005
EXP00006
EXP00007
EXP00008
EXP00009
EXP00010
EXP00011
EXP00012
EXP00013
EXP00014
EXP00015
EXP00016
EXP00017
EXP00018
EXP00019

```

```

      RETURN
      END
      FUNCTION CAUCH(RES,A,S)
C   PROGRAM AUTHOR  A. HUME
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C   GENERATES NUMBERS FROM THE DISTRIBUTION  $S/(S^2+(X-A)^2)$ 
C   RES -- RESULT LEFT HERE
C   A -- SEE EQUATION
C   S -- SEE EQUATION
C   CTC ORD PROGRAM NO. 9082.4
      1 R1=FLTRN(1.)
      R2=FLTRN(1.)
      IF(R1**2+R2**2.GT.1.) GOTO 1
      IF(FLTRN(1.).LT..5) R2=-R2
      RES=(R1/R2)*S+A
      CAUCH=RES
      RETURN
      END
      FUNCTION INTER(I,N)
C   PROGRAM AUTHOR  A. HUME
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C   GENERATES UNIFORMLY DISTRIBUTED INTEGERS FROM 1 THROUGH N-1
C   I -- RESULT LEFT HERE
C   N -- RANGE OF I WILL BE FROM 0 THROUGH N-1
C   CTC ORD PROGRAM NO. 9082.5
      I=FLTRN(1.)*FLOAT(N)
      IF(I.EQ.N) I=N-1
      INTER=I
      RETURN
      END
      FUNCTION IBINOM(I,P,N)
C   PROGRAM AUTHOR  A. HUME
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C   GENERATES INTEGERS FROM A BINOMIAL DISTRIBUTION.
C   I -- RESULT LEFT HERE
C   P -- PARAMETER OF BINOMIAL DISTRIBUTION
C   N -- PARAMETER OF BINOMIAL DISTRIBUTION
C   CTC ORD PROGRAM NO. 9082.6
      I=0
      DO 1 J=1,N
      1 IF(FLTRN(0.).LT.P) I=I+1
      IBINOM=I
      RETURN
      END
      FUNCTION IPOIS(I,FL)
C   PROGRAM AUTHOR  A. HUME
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,

```

```

EXP00020
EXP00021
CAUC0001
CAUC0002
CAUC0003
CAUC0004
CAUC0005
CAUC0006
CAUC0007
CAUC0008
CAUC0009
CAUC0010
CAUC0011
CAUC0012
CAUC0013
CAUC0014
CAUC0015
CAUC0016
CAUC0017
INTR0001
INTR0002
INTR0003
INTR0004
INTR0005
INTR0006
INTR0007
INTR0008
INTR0009
INTR0010
INTR0011
INTR0012
INTR0013
IBIN0001
IBIN0002
IBIN0003
IBIN0004
IBIN0005
IBIN0006
IBIN0007
IBIN0008
IBIN0009
IBIN0010
IBIN0011
IBIN0012
IBIN0013
IBIN0014
IBIN0015
IP0S0001
IP0S0002
IP0S0003

```

C	OAK RIDGE, TENN.	IPOS0004
C	GENERATES NUMBERS FROM A POISSON DISTRIBUTION	IPOS0005
C	I -- RESULT LEFT HERE	IPOS0006
C	FL -- PARAMETER OF POISSON DISTRIBUTION	IPOS0007
C	CTC ORD PROGRAM NO. 9082.7	IPOS0008
	E1=EXP(-FL)	IPOS0009
	I=0	IPOS0010
	E2=1.	IPOS0011
1	E2=E2*FLTRN(1.)	IPOS0012
	IF(E2.LT.E1) GOTO 2	IPOS0013
	I=I+1	IPOS0014
	GOTO 1	IPOS0015
2	IPOIS=I	IPOS0016
	RETURN	IPOS0017
	END	IPOS0018
FLTRN	START	FLRN0001
*	FUNCTION FLTRNF(Q)	FLRN0002
*	PROGRAM AUTHOR J. G. SULLIVAN	FLRN0003
*	OAK RIDGE NATIONAL LABORATORY, OAK RIDGE, TENNESSEE	FLRN0004
	USING *,15	FLRN0005
	STM 0,4,SAVE	FLRN0006
	L 4,0	FLRN0007
	USING RANDOM,4	FLRN0008
	L 1,RANDOM+4	FLRN0009
	M 0,GENERA	FLRN0010
	LTR 0,0	FLRN0011
	BC 10,PLUS	FLRN0012
	A 0,GENERA	FLRN0013
PLUS	ST 1,RANDOM+4	FLRN0014
	ST 0,SAVE	FLRN0015
	L 1,RANDGM	FLRN0016
	M 0,GENERA	FLRN0017
	AL 1,SAVE	FLRN0018
	STH 1,RANDOM+2	FLRN0019
	LD 0,ZERO	FLRN0020
	AD 0,RANDOM	FLRN0021
	LM 0,4,SAVE	FLRN0022
	BR 14	FLRN0023
Q	DC A(RANDOM)	FLRN0024
	DS 0D	FLRN0025
ZERO	DC X'420000000000000000'	FLRN0026
SAVE	DS 7F	FLRN0027
RNDOM	CSECT	FLRN0028
RANDOM	DC X'4200000071AFD498D'	FLRN0029
GENERA	DC X'1AFD498D'	FLRN0030
	END	FLRN0031

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Z

MISCELLANEOUS

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Inner product of two single precision vectors.

Purpose

AXTY is a function subprogram to calculate the inner product of two vectors. The sum of the double precision products of the components of the two single precision vectors is accumulated in double precision.

Usage

The calling sequence is:

$XIP = AXTY(XF, YF, N, INCX, INCY)$

where

AXTY will contain the inner product of two single precision vectors, one of which is stored in the X array. The other vector is stored in the Y array. Either or both of the X and Y arrays may be singly- or doubly-dimensioned.

XF is the element of the X array, e.g. $X(3,1)$, which is the first component of the vector stored in the X array.

YF is the element of the Y array which is the first component of the vector stored in the Y array.

N is the integer number of components of each of the vectors.

INCX is the integer "distance" in full words from one component of the vector stored in the X array to the next. If X is a singly-dimensioned array, or if X is a doubly-dimensioned array and the vector is stored in a column of X, then INCX=1. If X is a doubly-dimensioned array, and the vector is stored in a row of X, then INCX equals the first dimension of the array X.

INCY is the integer "distance" in full words from one component of the vector stored in the Y array to another (see explanation for INCX).

Coding Information

The X and Y arrays must be single precision in the calling program.

The value of the inner product, AXTY, may be returned to the calling program as either single precision or double precision, depending on how AXTY is typed in the calling program.

AXTY is written in 360 Assembly Language because at the present time it is not possible to obtain efficiently the double precision product of two single precision numbers (the components of the two vectors) using the IBM Fortran IV compiler.

Author

Jane Dunphey, Computing Technology Center, Union Carbide Corporation, Oak Ridge, Tennessee.

```

      ENTRY AXTY
*      FUNCTION AXTY (X, Y, N, INCX, INCY)
*      PROGRAM AUTHOR MISS JANF DUNPHEY
*      COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
*      OAK RIDGE, TENN.
*      CTC ORD PROGRAM NO. 9083
*
      USING *,15
AXTY   STM    14,12,12(13)
      LR      2,13
      LA      13,AREA
      ST      13,8(0,2)
      ST      2,4(0,13)
      B       SAVE
AREA   DS      16F
FPSAV2 DS      0
SAVE   STD     2,FPSAV2
      L       7,8(1)
      L       7,0(7)      R7 = NN
      L       2,0(1)
      L       3,4(1)
      L       4,12(1)
      L       4,0(4)
      SLA     4,2          R4 = INCX
      L       5,16(1)
      L       5,0(5)
      SLA     5,2          R5 = INCY
SER    0,0
LOOP   LE      2,0(2)
      ME      2,0(3)
      ADR     0,2
      AR      2,4
      AR      3,5
      RCT     7,LOOP
      LD      2,FPSAV2
      L       13,AREA+4
      LM      2,12,28(13)
      L       14,12(13)
      MVI     12(13),X'FF'
      BCR     15,14
      END

```

00000001
00000002
00000003
00000004
00000005
00000006
00000007
00000008
00000009
00000010
00000011
00000012
00000013
00000014
00000015
00000016
00000017
00000018
00000019
00000020
00000021
00000022
00000023
00000024
00000025
00000026
00000027
00000028
00000029
00000030
00000031
00000032
00000033
00000034
00000035
00000036
00000037
00000038
00000039
00000040
00000041

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Scaling of a double precision vector.

Purpose

DSCALE scales a vector of double precision numbers (REAL *8) so that all of its non-zero elements lie in absolute value between 1 and 1/16. The main purpose of DSCALE is to scale equations [1] prior to their solution by routines like DMATEQ (see this compendium). The routine allows for the convenient scaling of rows of a matrix even though such double precision numbers to be scaled do not normally lie in consecutive locations.

Usage

The calling sequence is:

```
CALL DSCALE (A,N,INC,IP16,NUNDER)
```

where

A is the double precision first element of the collection of elements to be scaled. It is assumed that the double precision "adjacent" elements to be scaled are "equidistant" in storage.

N is the integer number of double precision numbers to be scaled.

INC is the integer "distance" in double precision words from one element of A to the next, i.e.,

if A is a standard singly-dimensional array, INC equals one. If A is a doubly-dimensioned array and the elements of A to be scaled are stored in a row of A, then INC is the first dimension of A.

IP16 is the returned integer exponent of 16 by which each element of the vector was multiplied.

NUNDER is the integer number of elements which underflowed. These elements are set to zero.

Coding Information

DSCALE is an IBM 360 assembly language routine. On performing the scaling, only the exponents of the numbers are manipulated; the mantissas (i.e., the fraction parts) are completely unaltered.

Accuracy

Since the mantissas are unaltered, no accuracy is lost unless underflows occur. Underflows are set to zero.

Reference

- [1] Forsythe, George, and Moler, Cleve B. (1967): Computer Solution of Linear Algebraic Systems, Prentice-Hall.

Author

Alfred Hume, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

```

* SUBROUTINE DSCALE(A,N,INC,IP16,NUNDER) 00000001
* PROGRAM AUTHOR A. HUME 00000002
* COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV., 00000003
* OAK RIDGE, TENN. 00000004
* SCALES THE VECTOR A OF DOUBLE WORDS. SCALES THEM SO THAT ALL 00000005
* NUMBERS LIE BETWEEN 1 AND 1/16. UNDERFLOWS ARE SET TO ZERO. THERE 00000006
* IS NO OVERFLOW PROBLEM. ZEROS ARE NOT COUNTED AS UNDERFLOWS. 00000007
* IF VECTOR IS ALL ZEROS THEN IP16 IS SET TO ZERO AND NUNDER IS SET 00000008
* TO N. NOTE THAT NUNDER WILL NEVER OTHERWISE BE ZERO. 00000009
* A IS THE D. P. FIRST ELEMENT OF THE VECTOR TO BE SCALED 00000010
* N IS THE INTEGER STORAGE DISTANCE IN DOUBLE WORDS BETWEEN 00000011
* ELEMENTS OF A, 00000012
* INC IS THE INTEGER STORAGE DISTANCE IN DOUBLE WORDS BETWEEN 00000013
* ELEMENTS OF A, 00000014
* IP16 IS THE RETURNED INTEGER POWER OF 16 BY WHICH EACH ELEMENT 00000015
* OF THE VECTOR WAS MULTIPLIED, 00000016
* NUNDER IS THE INTEGER NUMBER OF UNDERFLOWS WHICH OCCURRED IN 00000017
* COMPUTATION. 00000018
* CTC ORD PROGRAM NO. 9C84 00000019
* 00000020
DSCALE DK CSFCT 00000021
ENTRY DSCALE 00000022
USING *,15 00000023
DSCALE LM R2,R8,SAVE 00000024
STD FPR0,FPP0SAVE 00000025
STD FPR2,FPR2SAVE 00000026
* GETS N (SECOND ARG) INTO N 00000027
L RT1,4(R1) 00000028
L RT1,0(RT1) 00000029
ST RT1,N 00000030
* GETS BYTE INCR (THIRD ARG TIMES 8) INTO RINC. 00000031
L RINC,8(R1) 00000032
L RINC,0(RINC) 00000033
SLA RINC,3 00000034
* GETS MASK WORD THAT ZEROS ALL BUT LAST 7 BYTES INTO RMASK. 00000035
L RMASK,=X'0000007F' 00000036
* SCAN THE ROW TO FIND THE BIGGEST CHARACTERISTIC 00000037
L RCT,N 00000038
L RARRY,0(R1) A OF A(1) INTO RARRY 00000039
L RMAX,=F'-1' MAXIMUM CHARACTERISTIC WILL BE PUT 00000040
INTO RMAX. 00000041
* LOOP1 LD FPR2,0(RARRY) 00000042
LTDR FPR2,FPR2 00000043
RZ LOOP1A DISREGARD IF ELEMENT IS ZERO 00000044
IC RT1,0(RARRY) 00000045
NR RT1,RMASK 00000046
CR RMAX,RT1 00000047
BH LOOP1A RMAX GT FT1 00000048
LR RMAX,RT1 00000049
LOOP1A AR RARRY,RINC 00000050

```

```

      BCT   RCT,LOOP1
*   IF RMAX EQ -1 THEN ALL VECTORS WERE ZERO.
*   SET NUNDEK (FIFTH ARG) TO N AND IP16 (FOURTH ARG) TO ZERO.
      C     RMAX,=F'-1'
      BNE   XXX
      L     RUND,N
      SP    RMAX,RMAX
      B     ARGSTOR
*   MAKE RMAX=64-RMAX. THIS IS WHAT EACH CHARACTERISTIC WILL BE
*   MODIFIED BY.
XXX    LCR   RMAX,RMAX
      A     RMAX,=F'64'
*   ADD RMAX TO EACH CHARACTERISTIC.
      L     RCT,N
      L     RARRY,0(R1)
*   IF NUMBER IS ZERO THEN DISREGARD IT.
      SDR   FPRO,FPRO
      SR    RUND,RUND
LOOP2   LD   FPR2,0(RARRY)
      LTDR  FPR2,FPR2
      BZ    LOOP2B
      IC    RT1,0(RARRY)
      NR    RT1,RMASK
      AR    RT1,RMAX
*   IF NEGATIVE (UNDERFLOW) THEN SET THE ELEMENT TO ZERO ELSE
*   PUT BACK THIS MODIFIED CHARACTERISTIC.
      BP    LOOP2A
      STD   FPRO,0(RARRY)    STORES A ZERO INTO THE WORD.
      LA    RUND,1(RUND)    INCR RUND
      B     LOOP2B
LOOP2A  TM   0(RARRY),B'10000000'
      BZ    LOOP2C          WORD WAS POSITIVE
      STC   RT1,0(RARRY)
      OI    0(RARRY),B'10000000' PUTS MINUS SIGN ON
      B     LOOP2B
LOOP2C  STC   RT1,0(RARRY)    WORD WAS POSITIVE
LOOP2B  AR    RARRY,RINC
      RCT   RCT,LOOP2
*   SETS FOURTH ARG (IP16) TO RMAX
ARGSTOR L    RT1,12(R1)
      ST    RMAX,0(RT1)
*   SETS FIFTH ARG TO RUND
      L     RT1,16(R1)
      ST    RUND,0(RT1)
*   EPILOGUE
      LD    FPRO,FPROSAVE
      LD    FPR2,FPR2SAVE
      LM    R1,R8,SAVE
      BR    14
*   WORKING AREAS ETC.

```

R1	EQU	1
RCT	EQU	2
RARRY	EQU	3
RT1	EQU	4
RMASK	EQU	5
RINC	EQU	6
RMAX	EQU	7
RUND	EQU	8
FPRO	EQU	0
FPR2	EQU	2
R2	EQU	2
R8	EQU	8
FPROSAVE	DS	1D
FPR2SAVE	DS	1D
SAVE	DS	7F
N	DS	1F
END		

00000101
00000102
00000103
00000104
00000105
00000106
00000107
00000108
00000109
00000110
00000111
00000112
00000113
00000114
00000115
00000116
00000117

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Identification

Matrix listing routines.

Purpose

ARRAY, ARRAYD, ARRAYC, and ARAYDC are output subroutines which write respectively a single precision, a double precision, a complex, or a double precision complex matrix on the standard Fortran output tape 6. Each of these routines labels the matrix, which may be square or non-square, and its rows and columns.

Usage

The calling sequences are:

CALL ARRAY (A,NR,NC,ID,4H----

or

CALL ARRAYD (A,NR,NC,ID,4H----

where

A is the singly- or doubly-dimensioned array to be listed,

NR is the integer number of rows of the matrix A desired in the output,

NC is the integer number of columns of the matrix A desired in the output,

ID is the integer first dimension of A in the calling program; i.e., A(ID,*) or A(ID),

4H---- is a four-character Hollerith argument which will be the heading for the matrix A. For example, if the

user would like "AMAT" written as a heading, this argument would be 4HAMAT.

or

```
CALL ARRAYC (A,NR,NC,ID,IRC,4H----
```

or

```
CALL ARAYDC (A,NR,NC,ID,IRC,4H----
```

where

A is the same as in the preceding calling sequence,

NR same,

NC same,

ID same,

IRC is an integer which indicates whether the real or the imaginary part of the array is to be listed.

For IRC=1, the real part of A is listed; for IRC=2, the imaginary part of A is listed,

4H---- same.

Coding Information

All four of the listing routines are written in Fortran IV. The first argument in the calling sequences, A, the matrix to be listed, must be typed in the calling program according to the matrix listing routine used. ARRAY lists a single precision (REAL*4) matrix; ARRAYD, a double precision (REAL*8) matrix; ARRAYC, a complex (COMPLEX*8) matrix; and ARAYDC, a double precision complex (COMPLEX*16) matrix.

All four matrix listing routines use output tape 6. In using ARRAYC and ARAYDC, for both the real and imaginary parts of a complex matrix to be listed, these routines must be called twice (see calling sequence for ARRAYC and ARAYDC).

ARRAY and ARRAYC both list a maximum of 50 rows of 7 columns per output page while ARRAYD and ARAYDC list 50 rows of 5 columns per output page. If the dimensions of the matrix exceed these numbers, on successive pages of output the routine continues to list the remaining rows of the first set of 7 or 5 columns (depending on routine used). It then lists the next set of 7 or 5 columns entirely, etc., until all columns of the matrix have been listed.

Authors

Jane Dunphey, Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee, and Lee Feldman, formerly a co-op with Computing Technology Center.

```

      SUBROUTINE ARRAY (B,NRR,NCC,LVV,NAME)
C   PROGRAM AUTHORS  J. DUNPHEY OF COMPUTING TECHNOLOGY CENTER AND
C                       L. FELDMAN FORMERLY WITH
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C   CTC ORD PROGRAM NO. 9085.1
C
      DOUBLE PRECISION COLUMN
      DIMENSION B(1)
      DATA COLUMN /8HCOLUMN /
      NR = NRR
      NC = NCC
      LV = LVV
      DO 3 ML = 1,NC,7
      MH = ML + 6
      IF (NC .GE. MH) GO TO 1
      MH = NC
1    MLM1 = (ML - 1)*LV
      MMH1 = (MH - 1)*LV
      DO 3 LI = 1,NR,50
      NL = LI
      NH = NL + 49
      IF (NR .GE. NH) GO TO 2
      NH = NR
2    WRITE (6,100) NAME
100  FORMAT (1H1,A4)
      WRITE (6,200) (COLUMN,J,J=ML,MH)
200  FORMAT (4HCRQW,4X,7(3X,A8,I3,3X))
      WRITE (6,300)
300  FORMAT (1H0)
      DO 3 L = NL,NH
      MML = MLM1 + L
      MMH = MMH1 + L
3    WRITE (6,400) L,(B(J),J=MML,MMH,LV)
400  FORMAT (1H ,I3,4X,7E17.8)
      RETURN
      END
      SUBROUTINE ARRAYD (B,NRR,NCC,LVV,NAME)
C   PROGRAM AUTHORS  J. DUNPHEY OF COMPUTING TECHNOLOGY CENTER AND
C                       L. FELDMAN FORMERLY WITH
C   COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C   OAK RIDGE, TENN.
C   CTC ORD PROGRAM NO. 7085.2
      DOUBLE PRECISION COLUMN,B(1)
      DATA COLUMN /8HCOLUMN /
      NR = NRR
      NC = NCC
      LV = LVV
      DO 3 ML = 1,NC,5

```

```

      ARAY0001
      ARAY0002
      ARAY0003
      ARAY0004
      ARAY0005
      ARAY0006
      ARAY0007
      ARAY0008
      ARAY0009
      ARAY0010
      ARAY0011
      ARAY0012
      ARAY0013
      ARAY0014
      ARAY0015
      ARAY0016
      ARAY0017
      ARAY0018
      ARAY0019
      ARAY0020
      ARAY0021
      ARAY0022
      ARAY0023
      ARAY0024
      ARAY0025
      ARAY0026
      ARAY0027
      ARAY0028
      ARAY0029
      ARAY0030
      ARAY0031
      ARAY0032
      ARAY0033
      ARAY0034
      ARAY0035
      ARAY0036
      ARAY0037
      ARYD0001
      ARYD0002
      ARYD0003
      ARYD0004
      ARYD0005
      ARYD0006
      ARYD0007
      ARYD0008
      ARYD0009
      ARYD0010
      ARYD0011
      ARYD0012
      ARYD0013

```

```

      MH = ML + 4
      IF (NC .GE. MH) GO TO 1
      MH = NC
1     MLM1 = (ML-1)*LV
      MHM1 = (MH-1)*LV
      DO 3 LI = 1,NR,50
      NL = LI
      NH = NL + 49
      IF (NR .GE. NH) GO TO 2
      NH = NR
2     WRITE (6,100) NAME
100  FORMAT (1H1,A4)
      WRITE (6,200) (COLUMN,J,J=ML,MH)
200  FORMAT (4HROW,3X,5(3X,A8,I5,9X))
      WRITE (6,300)
300  FORMAT (1H0)
      DO 3 L=NL,NH
      MML = MLM1 + L
      MMH = MHM1 + L
3     WRITE (6,400) L,(B(J),J=MML,MMH,LV)
400  FORMAT (1H,I3,3X,5D25.16)
      RETURN
      END
      SUBROUTINE ARRAYC (B,NRR,NCC,LVV,IRC,NAME)
C     PROGRAM AUTHORS J. DUNPHEY OF COMPUTING TECHNOLOGY CENTER, AND
C                       L. FELDMAN FORMERLY WITH
C     COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C     OAK RIDGE, TENN.
C
C     CTC ORD PROGRAM NO. 7085.3
      COMPLEX B(1)
      REAL RC(8)
      DOUBLE PRECISION COLUMN
      DATA COLUMN /8HCOLUMN /
      NR = NRR
      NC = NCC
      LV = LVV
      DO 4 ML = 1,NC,7
      MH = ML + 6
      IF (NC .GE. MH) GO TO 1
      MH = NC
1     MLM1 = (ML-1)*LV
      MHM1 = (MH-1)*LV
      DO 4 LI = 1,NR,50
      NL = LI
      NH = NL + 49
      IF (NR .GE. NH) GO TO 2
      NH = NR
2     WRITE (6,100) NAME
100  FORMAT (1H1,A4)

```

```

ARYD0014
ARYD0015
ARYD0016
ARYD0017
ARYD0018
ARYD0019
ARYD0020
ARYD0021
ARYD0022
ARYD0023
ARYD0024
ARYD0025
ARYD0026
ARYD0027
ARYD0028
ARYD0029
ARYD0030
ARYD0031
ARYD0032
ARYD0033
ARYD0034
ARYD0035
ARYD0036
ARYC0001
ARYC0002
ARYC0003
ARYC0004
ARYC0005
ARYC0006
ARYC0007
ARYC0008
ARYC0009
ARYC0010
ARYC0011
ARYC0012
ARYC0013
ARYC0014
ARYC0015
ARYC0016
ARYC0017
ARYC0018
ARYC0019
ARYC0020
ARYC0021
ARYC0022
ARYC0023
ARYC0024
ARYC0025
ARYC0026
ARYC0027

```

```

      WRITE (6,200) (COLUMN,J,J=ML,MH)
200  FORMAT (4HROW,4X,7(3X,A8,I3,3X))
      WRITE (6,300)
300  FORMAT (1H0)
      DO 4 L = NL,NH
      MML = MLM1 + L
      MMH = MHM1 + L
      II = 0
      DO 3 J = MML,MMH,LV
      II = II + 1
      3  RC(II) = FLOAT(2-IRC)*REAL(B(J)) + FLOAT(IRC-1)*AIMAG(B(J))
      4  WRITE (6,400) L,(RC(J),J=1,II)
400  FORMAT (1H ,I3,4X,7E17.8)
      RETURN
      END
      SUBROUTINE ARAYDC(B, NRR, NCC, LVV, IRC,NAME)
C  PROGRAM AUTHORS  J. DUNPHEY OF COMPUTING TECHNOLOGY CENTER AND
C                   L. FELDMAN FORMERLY WITH
C  COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,
C  OAK RIDGE, TENN.
C
C  CTC ORD PROGRAM NO. 9085.4
      COMPLEX*16 B(1)
      DOUBLE PRECISION COLUMN
      REAL*8 RC(8)
      DATA COLUMN/8HCOLUMN /
      NR = NRR
      NC = NCC
      LV = LVV
      DO 7 ML = 1,NC,5
      MH = ML + 4
      IF(NC.GE.MH)GOTO1
      MH=NC
1  MLM1 = (ML-1)*LV
      MHM1 = (MH-1)*LV
      DO 7 LI = 1,NR,50
      NL = LI
      NH = NL + 49
      IF(NR.GE.NH)GO TO 2
      NH = NR
2  WRITE(6,3) NAME
3  FORMAT(1H1,A4)
      WRITE(6,4)(COLUMN,J,J=ML,MH)
4  FORMAT(4HROW,3X,5(3X,A8,I5,9X))
      WRITE(6,5)
5  FORMAT (1H0)
      DO 7 L = NL,NH
      MML = MLM1 + L
      MMH = MHM1 + L
      II=0
      ARYC0028
      ARYC0029
      ARYC0030
      ARYC0031
      ARYC0032
      ARYC0033
      ARYC0034
      ARYC0035
      ARYC0036
      ARYC0037
      ARYC0038
      ARYC0039
      ARYC0040
      ARYC0041
      ARYC0042
      AYDC0001
      AYDC0002
      AYDC0003
      AYDC0004
      AYDC0005
      AYDC0006
      AYDC0007
      AYDC0008
      AYDC0009
      AYDC0010
      AYDC0011
      AYDC0012
      AYDC0013
      AYDC0014
      AYDC0015
      AYDC0016
      AYDC0017
      AYDC0018
      AYDC0019
      AYDC0020
      AYDC0021
      AYDC0022
      AYDC0023
      AYDC0024
      AYDC0025
      AYDC0026
      AYDC0027
      AYDC0028
      AYDC0029
      AYDC0030
      AYDC0031
      AYDC0032
      AYDC0033
      AYDC0034
      AYDC0035

```

```
DO 6 J=MML,MMH,LV
  II=II+1
6 RC(II) = (2-IRC)*B(J) + (IRC-1)*(G.,-1.)*B(J)
7 WRITE(6,8)L,{RC(J),J=1,II}
8 FORMAT(1H ,I3,3X,5D25.16)
RETURN
END
```

```
AYDC0036
AYDC0037
AYDC0038
AYDC0039
AYDC0040
AYDC0041
AYDC0042
```

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

A sorting routine.

Purpose

To sort an array of one word items in ascending algebraic order.

Usage

The calling sequence is:

CALL SORT(A,ITEMS)

where

A is a singly-dimensioned single precision array to be sorted in ascending order.

ITEMS is the number of words to be sorted, i.e., SORT will sort the first ITEMS words of A.

Coding Information

SORT is a Fortran IV single precision (REAL*4) subprogram which may be easily converted to double precision or integer. Also an alphabetic sort could be affected by assuming an integer array in SORT.

Restrictions

The only restriction is that the size of the array must not exceed the amount of core storage available.

Timing:

24,000 one-word items from a normal distribution (mean of 500) were sorted in 1.18 minutes on an IBM 7090.

Method

Generally, the method [1] is as follows:

1. Divide the array into halves and compare corresponding items in each half, exchanging them when the item in the lower half is greater.
2. The table is then divided into quarters, first comparing corresponding items in the first quarter with those in the second quarter and then those in the second quarter with those in the third quarter, etc. When an item is switched from third quarter to second quarter, it is then compared with the corresponding item in the first quarter to determine if it should be moved again. In general, when an item is moved from one section to another, it is compared with the corresponding item in the next higher section until a move is not made or until it reaches the top section.
3. Continue dividing the table and following the above procedure until a pass is completed comparing adjacent items.

Reference

- [1] Shell, D. L., (July 1959): "A high speed sorting procedure,"
Comm. ACM 2, 30-32.

Author

John Vaughan, formerly with Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.

	SUBROUTINE SORT(A,ITEMS)	SORT0001
C	PROGRAM AUTHOR J. VAUGHN, FORMERLY WITH	SORT0002
C	COMPUTING TECHNOLOGY CENTER, UNION CARBIDE CORP., NUCLEAR DIV.,	SORT0003
C	OAK RIDGE, TENN.	SORT0004
C		SORT0005
C	CTC ORD PROGRAM NO. 9086	SORT0006
	DIMENSION A(ITEMS)	SORT0007
	M=ITEMS	SORT0008
	1 M=M/2	SORT0009
	IF(M.EQ.0)RETURN	SORT0010
	K=ITEMS-M	SORT0011
	J=1	SORT0012
	2 I=J	SORT0013
	3 IM=I+M	SORT0014
	IF(A(I).LE.A(IM))GO TO 4	SORT0015
	TEMP=A(I)	SORT0016
	A(I)=A(IM)	SORT0017
	A(IM)=TEMP	SORT0018
	I=I-M	SORT0019
	IF(I.GE.1)GO TO 3	SORT0020
	4 J=J+1	SORT0021
	IF(J.GT.K)GO TO 1	SORT0022
	GO TO 2	SORT0023
	END	SORT0024

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Identification

Finding greatest common divisors and least common multiples.

Purpose

The function subprograms, NGCD and LCM, calculate greatest common divisors and least common multiples of integer arrays.

These routines are of use in such problems as generating scaled exact Hilbert matrices. For the Hilbert matrix H of order n , αH will have integer elements for $\alpha = \text{lcm}(n+1, n+2, \dots, 2n)$.

Usage

The calling sequences are:

$I = \text{NGCD}(N, \text{NUM})$

$J = \text{LCM}(N, \text{NUM})$

where

N is the integer size of the integer array NUM .

NUM is the singly-dimensioned integer array which contains the integers of which you wish to find the greatest common divisor or least common multiple.

I is the integer greatest common divisor of the first N integers in NUM .

J is the integer least common multiple of the first N integers in NUM .

Coding Information

These Fortran IV function subprograms, NGCD and LCM, both call another function IGCD(M,N) which calculates the g.c.d. of the two integers M and N by the Euclidean algorithm.

The procedures are inductive and utilize the following principles:

1. $\text{lcm}(a,b) = ab/\text{gcd}(a,b)$
2. $\text{gcd}(a,b,c) = \text{gcd}(\text{gcd}(a,b),c)$
3. $\text{lcm}(a,b,c) = \text{lcm}(\text{lcm}(a,b),c)$

Author

Mark Elson, former summer employee with Computing Technology Center, Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee.


```
N=NN
IF(M-N)2,2,1
1 I=M
M=N
N=I
2 IGCD=M
IGCD1= MOD (N,M)
IF (IGCD1)4,4,3
3 N=M
M=IGCD1
GOTO2
4 RETURN
END
```

```
LCM 19
LCM 20
LCM 21
LCM 22
LCM 23
LCM 24
LCM 25
LCM 26
LCM 27
LCM 28
LCM 29
LCM 30
LCM 31
```

BIBLIOGRAPHY*

The following list of references may prove useful in the research for numerical methods and in the critical evaluation of methods.

A. Elementary Textbooks for Beginners

1. Arden, B. W. An Introduction to Digital Computing. Addison-Wesley, 1963.
2. Conte, S. D. Elementary Numerical Analysis. McGraw-Hill, 1965.
3. Fröberg, Carl-Erik Introduction to Numerical Analysis.
(translated from Swedish), Addison-Wesley, 1965.
4. Macon, N. Numerical Analysis. Wiley, 1963.
5. Stanton, R. G. Numerical Methods for Science and Engineering.
Prentice-Hall, 1961.
6. Stiefel, E. Einfuehrung in die numerische Mathematik.
B. G. Teubner, Stuttgart, 1961. (English translation published
by Academic Press). First four chapters, plus some of chapters
6 and 7.

---

*Most of these references are from a list in Comm.ACM, 7, 1964, p. 214.

B. Senior Textbooks

1. Buckingham, R. A. Numerical Methods. Pitman, rev. ed. 1962.
2. Hamming, R. W. Numerical Methods for Scientists and Engineers. McGraw-Hill, 1962.
3. Hartree, D. R. Numerical Analysis. Clarendon Press, Oxford, 2nd ed. 1958.
4. Henrici, P. Elements of Numerical Analysis. Wiley, 1964.
5. Hildebrand, F. B. Introduction to Numerical Analysis. McGraw-Hill, 1956.
6. Kunz, K. S. Numerical Analysis, McGraw-Hill, 1957.
7. Modern Computing Methods. National Physical Laboratory, Teddington (see C29 below).

C. Selected Reference Books

1. Berezin, I. S., and Zhidkow, N. P. Metody Vychislenii. 2, Moscow, 1959 (English transl. Computing Methods, Addison - Wesley, 1965).
2. Bodewig, E. Matrix Calculus. Interscience, 2nd ed. 1959.
3. Collatz, L. Application of functional analysis to numerical analysis, Dept. of Math., U. of California, Berkeley, 1960 (notes).

4. Collatz, L. The Numerical Treatment of Differential Equations,
3rd ed., Springer-Verlag, Berlin, 1960.
5. Durand, E. Solution Numériques des Équations Algébriques,
2 vols. Masson, Paris, 1960-61.
6. Faddeev, D. K., and Faddeeva, V. N. Computational Methods of
Linear Algebra (translated from Russian) Freeman, 1963.
(Also a German translation of the 2nd (revised) edition is
available.)
7. Faddeeva, V. N. Computational Methods of Linear Algebra.
Dover, 1959.
8. Forsythe, G. E., and Wasow, W. R. Finite-Difference Methods
for Partial Differential Equations, Wiley, 1960.
9. Forsythe, George, and Moler, Cleve B., Computer Solution of
Linear Algebraic Systems, Prentice-Hall, 1967.
10. Fox L. The Numerical Solution of Two-Point Boundary Problems
in Ordinary Differential Equations. Clarendon Press, 1957.
11. _____. Numerical Solution of Ordinary and Partial Differential
Equations. Pergamon, 1962.
12. _____. Introduction to Numerical Linear Algebra. Oxford, 1964.
13. Hastings, C., Jr., Hayward, J. T., and Wong, J. P., Jr.
Approximations for Digital Computers. Princeton U. Press,
1955.

14. Henrici, P. Discrete-Variable Methods in Ordinary Differential Equations. Wiley, 1962.
15. _____. Error Propagation for Difference Methods. Wiley, 1963
(sequel to previous title; the two together are the definitive reference in the field).
16. Householder, A. S. Principles of Numerical Analysis. McGraw-Hill, 1953.
17. _____. The Theory of Matrices in Numerical Analysis. Ginn & Co., 1964.
18. Krylov, V. I. Approximate Calculation of Integrals, (translated from Russian), Macmillan, 1962.
19. Lanczos, C. Applied analysis. Prentice-Hall, 1956.
20. _____. Linear Differential Operators. Van Nostrand, 1961.
21. Meyer, H. A. (Ed.) Symposium on Monte Carlo Methods. Wiley, 1956.
22. Milne, W. E. Numerical Calculus. Princeton U. Press, 1949.
(entirely based on desk calculation)
23. Ostrowski, A. Solution of Equations and Systems of Equations. Academic Press, 1966, 2nd edition.
24. Ralston, A., and Wilf, H. S. (Eds.) Mathematical Methods for Digital Computers. Wiley, 1960.

25. Ralston, A., and Wilf, H. S. (eds.) Mathematical Methods for Digital Computers, volume 2, Wiley, 1967.
26. Richtmyer, R. D. Difference Methods for Initial-Value Problems. Interscience, 1957.
27. Sangren, W. C. Digital Computers and Nuclear Reactor Calculations. Wiley, 1960.
28. Sard, A. Linear Approximation, Amer. Math. Soc., 1963.
29. Modern Computing Methods. National Physical Laboratory. Teddington, 2nd ed., HMSO, London, and Philosophical Library, New York, 1961 (excellent annotated bibliography included).
30. Survey of Numerical Analysis. J. Todd (Ed.). McGraw-Hill, 1962.
31. Varga, R. S. Matrix Iterative Analysis. Prentice-Hall, 1962.
32. Wilde, J. Optimum Seeking Methods. Prentice-Hall, 1964.
33. Wilkinson, J. H. The Algebraic Eigenvalue Problem. Oxford Press, 1965.
34. _____. Rounding Errors in Algebraic Processes. HMSO, London, and Prentice-Hall, 1963.

D. Valuable Sources of Scientific Subprograms

1. Collected Algorithms from Communications of the ACM*,
ed. John G. Herrick, 1965 (available from ACM, 211 E. 43rd
St., New York, New York, 10017, (each month new algorithms
appear in Comm ACM).
2. Handbook for Automatic Computation*, eds. F. L. Bauer et al.,
(to be published), Springer-Verlag, (the programs are pre-
published in Numer. Math.).
3. A Collection of Mathematical Computer Routines, Courant Insti-
tute report NY.-1480-14, eds. M.S. Shapiro and Max Goldstein,
1965.
4. System/360 Scientific Subroutine Package, IBM H20-0166.

E. Selected Journals

1. Communication of the ACM.
2. Journal of the ACM.
3. Computing Reviews (ACM).
4. Permuted (KWIC) Index to Computing Reviews.
5. Computer Journal.

---

* We have found the following article useful for translating programs from ALGOL to Fortran: B. Higman (1963): "What EVERYBODY should know about ALGOL," Computer J., 6, pp. 50-56.

6. Mathematics of Computation.
7. Numerische Mathematik.
8. SIAM Journal on Numerical Analysis.
9. SIAM Review.
10. SIAM Journal on Applied Mathematics.

F. Useful Notes

1. Advanced Numerical Analysis Notes, Summer Sessions, University of Michigan, 1958-1967, etc.